Yellowbrick Data Warehouse

Product Documentation

Version 7.3.0

7.3.0-74895.181ea900

7/25/2025

Contents

Introduction

Yellowbrick Documentation > Introduction

Platforms: All platforms

Parent topic: Yellowbrick Documentation

Yellowbrick is an elastic Data Warehouse designed for the most demanding batch, real-time, *ad hoc*, and mixed workloads. It provides an ACID-compliant, Massively Parallel Processing (MPP) SQL relational database that enables thousands of concurrent users to run complex SQL queries against highly normalized schemas and petabytes of data.

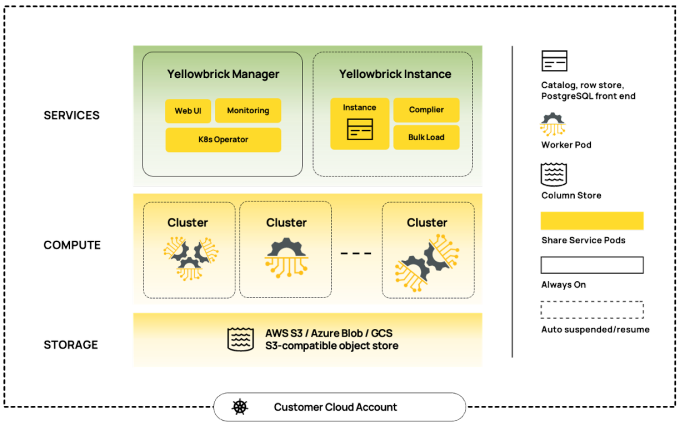
Yellowbrick deploys as an appliance in a data center or as software running in a public cloud account. It presents a PostgreSQL interface to the ecosystem and supports the standard ODBC and JDBC PostgreSQL open source drivers, enabling compatibility with a wide range of PostgreSQL-compatible tools and services.

Architecture

While the database software is the same version in all deployments, the architecture of Yellowbrick on an appliance differs from the architecture in the public cloud.

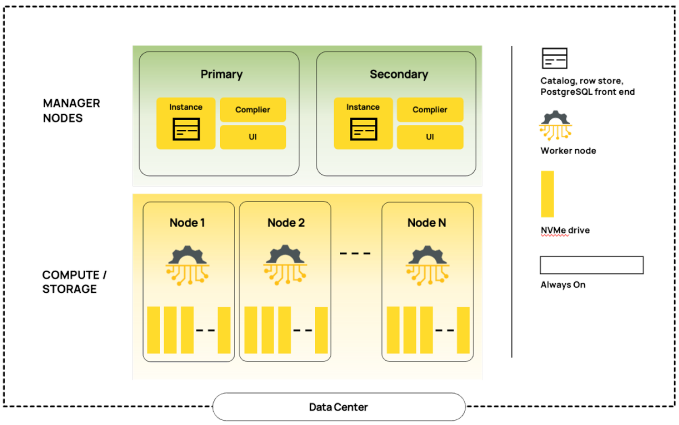
Kubernetes in Public Cloud

Yellowbrick Enterprise deploys on AWS, Azure, and GCP as a Kubernetes app. In the public cloud, the architecture separates compute from storage and supports multiple elastic compute clusters, each containing one or more compute nodes.



Appliance in a Data Center

The appliance combines the storage and compute into a single multi-node compute cluster. For high availability, all hardware components are redundant — this includes the power, cooling, network, storage, and compute elements. If you purchase an appliance, the Yellowbrick team takes care of the data center installation for you.



Document Types

Tutorials

If you're looking to build something specific or are more of a hands-on learner, check out the tutorials section. This is the best place to get started.

These are the best ones to get started with:

- Walkthrough Tour
- Loading Data
- Connecting with DBeaver

Explore the full list of Yellowbrick tutorials here.

How-to Guides

Here you'll find short answers to "How do I….?" types of questions. These how-to guides don't cover topics in-depth — you'll find that material in the Tutorials and the Reference. However, these guides help you quickly accomplish common tasks using Yellowbrick.

Concepts Guide

Introductions to all the key parts of Yellowbrick you need to know. Here you'll find details of the Yellowbrick architecture and concepts.

Reference

Head to the reference section for full documentation on all SQL data types, functions, and commands in Yellowbrick.

Ecosystem Integration

Yellowbrick supports a rich ecosystem of tools via native and PostgreSQL-compatible connectors. If you're looking to build solutions involving other components, check out the growing list of Yellowbrick integrations.

Concepts

Yellowbrick Documentation > Concepts

Platforms: All platforms

Parent topic: Yellowbrick Documentation

In this section:

- General
- Cloud

This guide provides explanations of the key concepts behind Yellowbrick and its architecture.

The concepts guides do not cover step-by-step instructions or specific implementation examples — those are in the How-to guides and Tutorials. For detailed reference material, please visit the Reference guide.

Architecture and Concepts Common to All Deployments

Platforms: All platforms

Parent topic: Concepts

In this section:

This section details architectural concepts of Yellowbrick that apply to both appliance and cloud deployments.

Shared Services

Platforms: All platforms

Parent topic: General

Each Yellowbrick instance consists of compute clusters, which process data and execute SQL queries, and shared services which comprise the remaining platform funtionality. Some examples of shared services functionality:

- Database connection management and termination
- Query planning and compilation
- Cluster management
- The row-oriented storage engine ("row store")
- Diagnostics, alerting and logging functions
- Authentication software
- Coordinating bulk operations
- Maintaining the Metering database

On appliance platforms, the shared services run on the manager nodes, with the scalability dictated by the compute, memory and CPU available on those particular servers. Typically, the appliance manager nodes are sized to support large, complex workloads with high concurrency.

On cloud platforms, the shared services can run on a choice of different instance profiles, Standard and Scaled, described below. By default all new installations run with Standard shared services to minimise cost. Since Yellowbrick only bills for vCPU used by compute clusters, the Yellowbrick cost won't change depending on the shared services type you select, but the cloud vendor cost will vary.

To view or alter the current shared services type using Yellowbrick Manager, navigate to *Instance Management → Shared Services*. Note that changing shared service type will incur a few minutes or more of instance downtime.

Standard Shared Services

Standard shared services are recommended for smaller installations with few concurrent users and small data sets, or workloads where the SQL queries in use are simple. Standard shared services are the best choice to minimise cloud expenditure since they run on hardware instance types with less vCPU and slower IO volumes. A simple rule would be to run standard shared services for environments with simple SQL queries and less than 100 logged-in users, or when a small number of compute cluster nodes are being used.

Scaled Shared Services

Scaled shared services are recommended for larger enterprise deployments. Although the stack consumes more cloud infrastructure, it will compile queries faster, provide faster insertion rates into the row store, plan more complex SQL and support thousands of logged-in users. Large compute clusters, consisting of tens or hundreds of nodes, will require Scaled shared services.

Compute Clusters

Yellowbrick database queries run on "compute clusters." On Tinman and Andromeda appliance platforms, there's one compute cluster where each node is a blade server, and the number of nodes in use in the cluster is a function of how many blades are inserted. All user queries run against this default cluster, and the columnar data is written locally to storage devices within the cluster. The size of the cluster is constrained by how many physical blades can be inserted into the appliance.

On cloud platforms, we implement an architecture that's fully elastic, with separate storage and compute. Multiple compute clusters are managed through SQL enabling workload isolation. Different users or roles are assigned to compute clusters, and a cluster load balancer allows intelligent distribution of work across to enable higher concurrency. Data is written to shared object storage and cached on local ephemeral storage devices. The total sizes of all clusters is effectively constrained by the availability zone the software is running in.

Creating New Clusters (cloud platforms)

Compute clusters are created using the SQL command `CREATE CLUSTER`. Similar commands exist to alter clusters or drop clusters. Alternatively, a simple wizard in Yellowbrick Manager provides a graphical way of creating clusters: To use it, navigate to *Instance Management*, choose your instance and click on the + *Cluster* button.

Examining Clusters

To see what clusters are currently in operation, and the status of nodes in the cluster, use the `sys.cluster` system view. On appliance platforms you'll only see one cluster present called `yellowbrick` consisting of all the blades in the appliance.

Cluster Node Acquisition (cloud platforms)

The nodes in the cluster are acquired in four stages, as reflected in the node counts in `sys.cluster`. The stages are as follows:

1. The number of desired nodes for a cluster is written into the `nodes` column when the cluster is created or modified, and the cloud provider is requested to provide more nodes.
2. As nodes are received from the cloud provider, the counter `prepared_nodes` is incremented.
3. Yellowbrick provisions the Kubernetes pods on the node, forcing the software to be downloaded. The counter `configured_nodes` is incremented when done.
4. The Yellowbrick database software component called the "compute node" is launched, and the counter in `ready_nodes` is updated.
5. The compute node database software comes online and the node is ready to process database queries, and the `ready_workers` counter is incremented.

The Yellowbrick Manager application, when creating clusters, reflects these last four stages as Preparing, Configuring, Starting and Running respectively. Until all nodes requested have been acquired and started up (`nodes` = `ready_workers`) the `state` of the cluster will not be marked as `RUNNING`.

If the process of starting a cluster takes a long time it may be because of account quota issues or a shortage of the requested hardware instance type in the given region and availability zone. Check account quotas, then consider using ODCRs or Reserved Nodes to acquire nodes in the background. Unfortunately the public cloud providers do not provide public interfaces for monitoring node availability, but often support staff or engineers from the cloud provider can do so. If you regularly suffer from a lack of node availability, we suggest talking to your cloud account team for their advice on how to best ameliorate.

Default and System Clusters

A Yellowbrick instance must have one and only one *default cluster*, which remains running as the default resource for executing queries and other operations. If an instance has only a single cluster, it is by definition the default cluster. If an instance has only a single cluster, it is also implied to be the *system cluster*. The system cluster is a cluster designated to do all system work (background operations that run under internal system accounts, such as flushing data from the row store, updating statistics or garbage collection). The default cluster and system cluster may be the same.

To alter the default and system clusters, use the ALTER SYSTEM SET CLUSTER commands. To see which are the current default and system clusters, see the `is_default` and `is_system` columns in `sys.cluster`.

A user may have a designated default cluster where all queries submitted by that user are run. The default cluster for a user may be set with an ALTER USER SET DEFAULT_CLUSTER command.

Cluster Policy Options

Every cluster has an operating policy, as defined by the following options:

Auto-suspend:
This option ensures that a cluster does not consume resources when it is not being used.

Auto-resume:
This option starts up a cluster automatically when its use is requested. The cluster will resume on submission of the first query that attempts to use it. If a query takes a long time to run, this is often why.

Workload Management Profile:
Each cluster can be assigned a specific workload management profile, or it can run with the `default` profile.

Operations That Do Not Require a Running Cluster

A running cluster is not needed for *all* queries and system operations. Users can run "front-end" queries, and the system can run commands that manage the instance. An instance is operational in this way before its first (default) cluster is created, and that cluster can be managed (suspended/resumed).

Examples of front-end queries include `CREATE DATABASE`, commands that create external objects, and other high-level commands do not require a running cluster. You can run `COUNT(*)` queries, and return the results of certain system functions, such as `CURRENT_DATABASE()` and `CURRENT_USER`, when no `FROM` clause is present in the query. You can also query certain system views, but not the `sys.log_query` set of views. You can insert rows into the row store.

Note: If you are creating your own primary storage for the databases in a given data warehouse instance, you should create that storage before creating any clusters or databases. See CREATE EXTERNAL LOCATION.

Modifying Clusters

After creating a cluster, you can modify it in a few ways:

- `SUSPEND` / `RESUME` the cluster and apply a specified wait time.
- Set the automatic `SUSPEND` / `RESUME` policy.
- `RENAME` the cluster.
- Scale the cluster up or down by adding or subtracting from the node count.
- Associate the cluster with a different WLM profile.

See also ALTER CLUSTER.

Licensing

Yellowbrick software licensing varies by platform. For more information on supported platforms, see the platform support guide. The details are as follows:

Enterprise Edition (EE) on appliances
: The software is licensed together with the appliance hardware, according to contractual terms agreed upon during purchasing. You don't need to install a software license to use it.

Enterprise Edition (EE) on cloud platforms
: The software requires a software license in order to activate. You'll need to acquire and install one. See information on software licensing below.

About Software Licenses

Software licenses are issued by Yellowbrick Data. Licenses are tied to a cloud account/subscription/project (choose your terminology depending on your cloud provider) and a given region and instance. Broadly, there are two classes of licenses: Trial licenses and full licenses, described below.

Licenses are delivered via email in a scrambled text form; to activate the license, cut and paste the text into Yellowbrick Manager's licensing page by navigating to *Instance Management → Instance Management → License*.

An unlicensed Yellowbrick installation can't create computer clusters, severely limiting its utility by making it unable to execute most kinds of SQL queries. You can still log in, poke around and run some very simple SQL queries that don't involve computation on stored data, but that's about it.

Trial licenses

Trial licenses can be requested for an instance via Yellowbrick Manager or, if you prefer, by contacting us at `licensing@yellowbrick.com`. Automatically issued trial licenses are currently valid for 30 days and cover up to 128 vCPUs; if you'd like a longer term, assistance from one of our engineers in a PoC, or to run clusters with more vCPUs, please contact us via email at `licensing@yellowbrick.com`.

To request a trial license in Yellowbrick Manager, navigate to *Instance Management → Instance Management → License*. *Instance Management* is the cube icon in the leftmost pane. The trial license will be delivered by email in a couple of minutes. Remember to check your Junk/Spam folder.

Full licenses

A trial license can be replaced by a full license. To request a full license, please contact us at `licensing@yellowbrick.com`. A full license will require you to commit to on-demand pricing, a subscription or another class of commercial agreement. Full licenses typically don't contain vCPU limits.

Technical Implementation

The scrambled license text that you paste into Yellowbrick Manager is actually a Base64URL encoded JSON Web Token (JWT). You can decode the JWT and look inside its contents, if you're interested, by pasting the text into jwt.io.

The license includes the following key fields:

Trial

Whether or not the license is a trial license. A new trial license can't be installed when one has already been installed. If you need to extend your trial, please reach out and contact us via email at `licensing@yellowbrick.com`.

Soft expiry date

The date at which Yellowbrick Manager will start to warn you the license is about to expire. After this date, operation of the instance will not be restricted in any way.

Hard expiry date

The date at which the instance will become unlicensed. After this date, running compute clusters will be suspended, and you won't be able to start new ones.

vCPU limit

An optional limit on the maximum number of vCPUs that can be used. For paid enterprise licenses, this is typically unlimited.

Contact information

Contact details for your business or organization, in case we at Yellowbrick need to get in touch.

This JWT is also used as the authentication token for instances to automatically report metering data when enabled in Yellowbrick Manager.

Databases

Platforms: All platforms

Parent topic: General

Databases are logical containers of schema objects and data. A new instance contains the default `yellowbrick` database. New databases can be created using Yellowbrick Manager or via standard SQL commands. Since databases can be completely isolated from one another, some multi-tenant architectures like to have one separate database per customer.

Databases and the objects they contain are in no way associated with compute clusters. You can run queries and other operations against a database using any available running cluster.

Yellowbrick supports read-only cross-database queries.

Data Distribution

Platforms: All platforms

Parent topic: General

Introduction to Data Distribution in MPP Data Warehouses

In Massively Parallel Processing (MPP) data warehouses like Yellowbrick, data distribution is a core architectural concept. It determines how rows of a table are spread across multiple compute nodes in the cluster.

Each compute node in an MPP system operates independently and processes its portion of the data in parallel with others. Effective data distribution ensures balanced workloads, minimal data movement during query execution, and fast, scalable performance.

MPP databases typically distribute data using one of the following strategies:

Hash Distribution

Hash distribution assigns rows to nodes based on the hash of one column. This is the most common and effective distribution method for large fact tables.

Benefits:

- Enables *co-location* of related rows across tables when the same distribution key is used.
- Minimizes *runtime data movement* during joins and aggregations.
- Distributes data evenly *when the key has high cardinality and low skew*.

Example:

```sql
CREATE TABLE sales (
  customer_id INT,
  order_id INT,
  amount DECIMAL(10,2)
)
DISTRIBUTE ON (customer_id);
```

Random (Round-Robin) Distribution

Random distribution assigns rows to nodes arbitrarily, typically in a round-robin fashion.

When to use:

- When there's no obvious column with high cardinality and low skew.
- For small dimension tables where even distribution is more important than co-location.

Tradeoffs:

- Even data distribution, but less efficient joins, since related rows may be on different nodes.
- May lead to redistribution at runtime if joining to a hash-distributed table.

Replicated Distribution

Replicated distribution copies the entire table to every compute node.

Best for:

- Small lookup or reference tables that are frequently joined to large fact tables.
- Avoiding data movement during joins, since each node has local access.

Important considerations:

- Should only be used for small tables, as replication adds memory and network overhead.
- Inefficient and potentially harmful for large datasets.

> **INFO**
>
> There's an interesting edge-case in Yellowbrick when every table involved in a query is replicated. In this instance, the query execution is not parallelized across nodes. Instead, execution takes place on a single node only. For workloads involving small tables and requiring high query throughput, the use of all-replicated tables may increase query concurrency.

Data Distribution in Kubernetes Deployments

Data distribution is subtly different in Kubernetes deployments, where compute is separated from storage, compared to appliance deployments. In Kubernetes deployments the shard files are stored centrally in object storage and are copied to the local NVMe caches on each compute node at query execution time. Which compute node receives which shard is determined when the data is loaded into Yellowbrick in the case of a hash-distributed table, and is related to the number of nodes in the cluster. This means that if the cluster size changes after data is loaded, the data becomes mis-distributed with respect to the new cluster configuration, and extra overhead may be incurred during query. We recommend that data is always loaded and queried by clusters containing the same number of nodes.

Why Distribution Strategy Matters

Choosing the right distribution strategy impacts:

- Query performance – Co-located joins are faster than distributed joins.
- Workload balance – Poor distribution can lead to execution skew where one node does more work.
- Storage efficiency – At-rest data skew can cause uneven disk usage and maintenance challenges.
- Network overhead – Mismatched distribution keys result in runtime redistribution, which slows down execution.

In summary, thoughtful table distribution is essential for optimizing performance and resource utilization in Yellowbrick and other MPP systems. Always consider the size, join patterns, and cardinality of your data when selecting a distribution method.

Read the Distributing Data How-To Guide for practical guidance and examples.

Data Ordering

Platforms: All platforms

Parent topic: General

Introduction

In analytic MPP databases such as Yellowbrick, data ordering is a critical table design consideration, second only to data distribution. Proper data ordering enhances SQL statement performance by optimizing scan efficiency and reduces storage requirements through optimized compression.

- Tables with effective ordering allow the database to skip entire shards or blocks during scans when a filtering predicate involves an order column or a column with high correlation to the order column.
- Compression efficiency is directly influenced by data order, as most compression algorithms benefit from sequential or repeated values and contiguous byte sequences.

The order of data within a table is determined by several factors, including table definition attributes, the method of data loading, data modifications such as inserts, updates, and deletes, and built-in database storage maintenance and management.

Data Storage in Yellowbrick

In Yellowbrick, user data is stored on compute nodes in a columnar block format, with data organized in sets of rows by column. Understanding the mechanics of data storage is key to comprehending the impact of data ordering on scanning efficiency and compression.

The fundamental unit of storage in Yellowbrick is a **shard**, which:

- Holds up to 1GB of data (before compression).
- Contains data and metadata for rows from a single table.
- Stores data into blocks per column, with up to 32,000 rows per block.
- It is immutable and never updated in place.

INFO

It is important not to confuse a Yellowbrick data shard with the "shard" concept from sharded databases (such as MongoDB, for example), where a shard refers to a distribution partition.

Shard Components in Yellowbrick

A shard in Yellowbrick is composed of five key parts:

Component	Description
Shard Metadata	Includes per-column metadata such as the number of rows, number of nulls, and minimum and maximum values, etc. This metadata acts as an alternative to traditional indexes and is necessary for enabling shard skipping, which significantly enhances scan performance.
Block Data	Contains compressed data for a column, with up to 32,000 rows per block.

Component	Description
Block Metadata	Similar to shard metadata, it includes per-column information such as the number of rows, number of nulls, and minimum and maximum values, and others. This metadata enables block skipping within shards, further improving scan performance.
Parity Blocks	Yellowbrick makes use of a RAID 6-like parity mechanism striped across compute nodes rather than disks within a compute node. Parity data is also stored within shards.
Boundary alignment	All shards are aligned to end on a 2 MB boundary.

Shard and Block Illustration

The following provides a logical illustration of a shard layout. Although block data is only one of the five components of a shard, it occupies most of the space in a full shard (i.e., often more than 99%).

Metadata Type / Data	Row Scope / Data	
Shard metadata	Rows 1 - ...	
col1 (min, max, nulls,...)	col2 (min, max, nulls,...)	...
Block metadata	Rows 1 to 32k	
col1:block1 (min, max, ...)	col2:block1 (min, max, ...)	...
Block Data	Rows 1 to 32k	
col1: block1	col2:block1	...
Block metadata	Rows 32k + 1 to 64k	
col1: block2 (min, max, ...)	col2: block2 (min, max, ...)	...
Block Data	Rows 32k + 1 to 64k	
col1: block2	col2: block2	...
...	...	...
Data Parity Blocks	(All)	
node1:shard1:parityBlk1	node1:shard1:parityBlk2	...
...	...	...
Boundary Alignment	(up to 2 MB)	

Shard and Block Scan and Skipping

To optimize data scans, the goal is to read as few shards as possible and, within those shards, as few blocks as possible. Shard and block skipping (also known as range-restricted scans) uses shard and block metadata to exclude unnecessary shards and blocks from the scan — the Yellowbrick columnar datastore does not use indexes.

For a practical guide to optimizing data ordering, see the Data Ordering How-To Guide.

Data Loading

Platforms: All platforms

Parent topic: General

Yellowbrick supports loading data through standard SQL `INSERT` and PostgreSQL `\copy` commands. This works well for small quantities of data, especially where the data needs to be inserted and available to query immediately. However, like other columnar databases, when tens of megabytes through terabytes of data need to be loaded efficiently, a bulk mode is supported that moves data directly to a compute cluster, bypassing the shared services.

Bulk loads insert data directly into database tables from source files in object storage, source files on NFS servers or source files on local disc. There are several ways to bulk load data, outlined below.

Bulk Loading with `ybload`

Data can be loaded through a command-line tool called `ybload` which is part of the client tools distribution. It supports a wide variety of file formats and protocols, as well as third party integrations. For more information see the the `ybload` documentation.

Bulk loading via SQL

On cloud platforms, a full SQL grammar supports loading data from external object storage. See SQL-Based Loads from External Storage and the LOAD TABLE command.

Using Yellowbrick Manager Load Assistant

Yellowbrick Manager contains a simple load assistant that can be used for importing data sets with minimal SQL knowledge. See Loading a Table via the Load Assistant for a walkthrough.

Workload Management Concepts

Yellowbrick Documentation > Concepts > General > Workload Management

Platforms: All platforms

Parent topic: General

In this section:

- How Queries Are Executed
- How WLM Works

Data warehouse resources can be shared in various ways. The main resources that can be allocated to database queries and other operations are:

- CPU
- Memory
- Temporary spill space

For example, on a system with a high degree of concurrency, you can allocate more CPU to specific queries by increasing their *priority*. For complex queries that require more memory or spill space, you can request more of these resources at different points during execution. Long-running queries can be queued behind very fast queries to avoid starving short-running queries of adequate resources.

Workload management (WLM) refers generally to the process of sharing system resources in an optimal way so that database operations can be done efficiently and with respect to some order of priority. The goal is to meet service-level agreements (SLAs) between users and database administrators. These SLAs typically define performance requirements for both end-user queries and other database operations, such as bulk loads, backups, and system maintenance.

A *workload* is a set of queries or other database requests that is to some extent a known quantity. For example, if a group of users run ad hoc queries against the same set of tables every day, that set of queries, though somewhat arbitrary, may be thought of as a known and expected workload. A resource-intensive report that is run by one user at the same time every morning may also be anticipated as a separate (and probably high-priority) workload. A third example is database administration work, such as bulk loads and backups, which may occur during a "maintenance window," when end users have minimal access to the system.

Workloads may be defined across many different dimensions: in terms of when they are run, the application or user that runs them, the type of work, their expected duration, whether they are resource-intensive, and so on. These variables are typical for MPP database systems, which are rarely used for one type of query or by one type of user. Some familiar use cases that workload management can address are as follows:

- *Runaway queries*: identify and stop long-running queries that, for example, select all of the rows from a very large table (whether issued naively, by mistake, or at a "bad time")
- *Short-query bias*: give priority to queries that run very fast (subsecond speed) and prevent them from being queued behind longer-running queries for which an instant response is neither expected nor required
- *Ad hoc queries*: place "browsing" or "discovery" queries at a lower priority in the queue than more critical queries that are needed to run the business
- *Time-sensitive queries*: apply different rules at different times of the day or week. For example, weekly business roll-ups have the highest priority until they are done. All other queries have lower priority.
- *Admin queries*: allocate resources to run `sysadmin` queries immediately, especially internally generated queries that maintain the database (for example, operations that flush and analyze new table rows).
- *Loads and updates*: write queries that do batch loads, deletes, and updates must not starve read queries.
- *Logging, auditing, and reporting*: log user-defined messages and tag queries as they are executed; learn about system usage in order to adjust future workload management behavior; create audit trails for separate applications and user groups.

To optimize resource allocation based on workloads, Yellowbrick administrators create WLM objects called *rules*, *resource pools*, and *profiles*. These objects define a flexible set of heuristics to translate typical WLM use cases into an optimal strategy for resource allocation and scheduling. You can set up WLM objects either in Yellowbrick Manager or by using SQL commands.

How Queries Are Executed

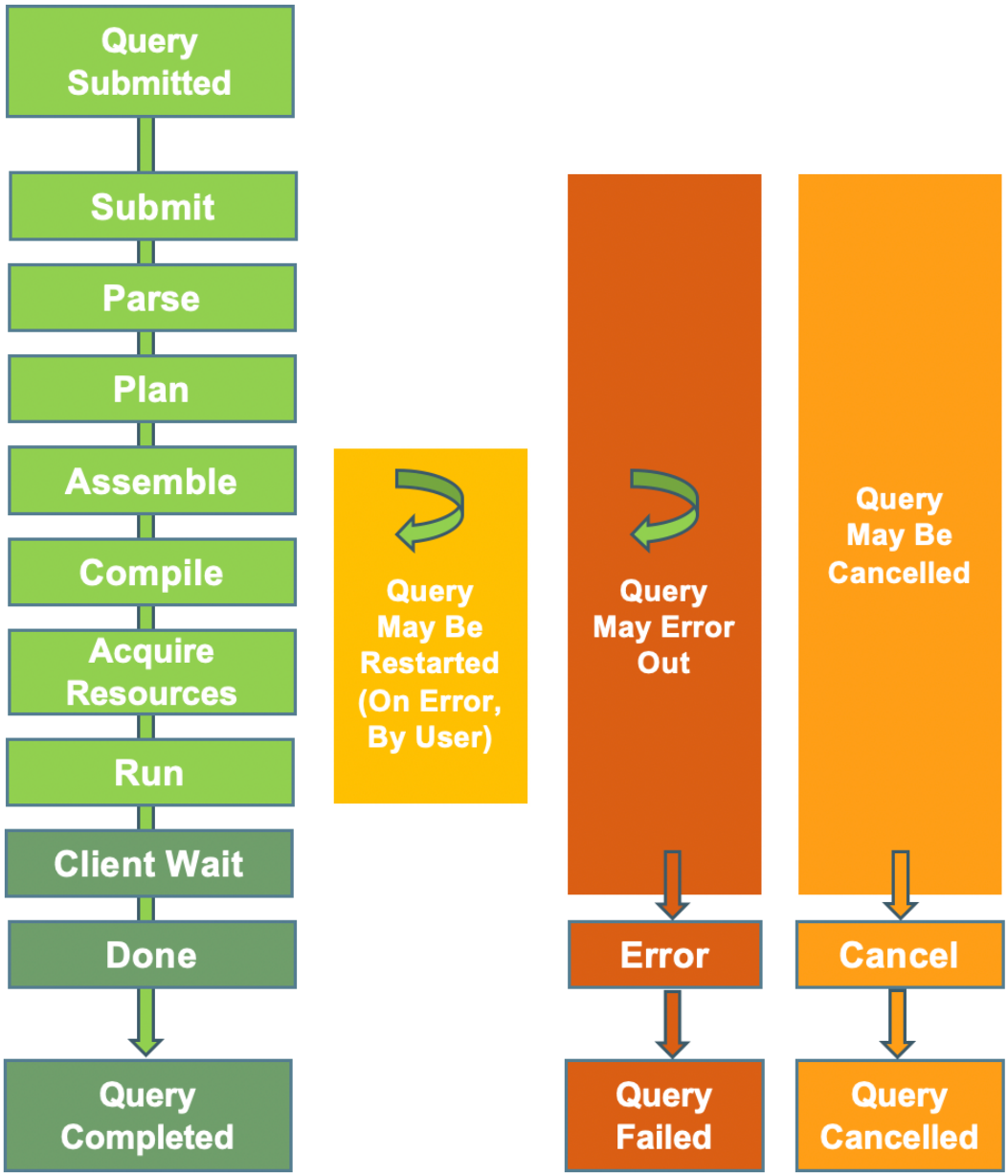
Platforms: All platforms

Parent topic: Workload Management

An important facet of making decisions about system resource allocation is an understanding of query execution. Yellowbrick queries move through several finite states from submission to completion. A basic understanding of this process will help you develop an effective WLM strategy, especially with respect to the creation of WLM rules. You can define WLM rules that are applied as queries enter certain states and use these rules to develop a WLM profile that optimizes resource allocation for a given workload.

As a query passes through each state in its life cycle, runtime statistics are captured and logged. These statistics provide a measure of the time spent in each phase of query execution, giving administrators a means of monitoring and analyzing query performance. Wait times and actual processing times are measured at each stage.

The following diagram shows the life cycle of a query. Each query passes through several states while it is being prepared for execution, then it starts running (executing) on the compute nodes.



The diagram also identifies when queries can be canceled or restarted. Once submitted, a query runs to completion, is canceled, or fails with an error (`Done` , `Cancel` , and `Error` states). If a query is restarted or returns an error, it may re-enter the cycle in the `Assemble` state, but ultimately, all queries finish in one of the three completion states. (If a query is canceled, it cannot be restarted.)

A query has the following states:

Submit

The query is received by the front-end database and enters this state first. The query is validated to make sure it contains well-formed SQL. This state is the earliest phase of a query where you can apply WLM throttle rules (limit concurrent queries). You can also reject queries when they are in this state. The `submit_time`, as logged in `sys.log_query`, is the wall-clock time when the front-end database starts the query.

Parse

The query is parsed into an initial abstract syntax tree (AST) and the locks required to run the query are acquired. Hinting rules get applied at this stage. A trivial query (such as a single-row insert) may jump straight from this state to the `Run` state. The time the query spends waiting for locks is tracked in the `wait_lock_ms` column in `sys.query`. If a query is blocked on the acquisition of a lock, the description of the lock appears in the `blocked` column in `sys.query`. Likewise, the time a query spent processing hinting rules is tracked in the `hinting_ms` column in `sys.query`.

Plan

The query plan is generated. Details about the tables referenced in the query are available to WLM (from the AST), as well as the `type` of the query. You cannot apply rules to queries in this state, but a broad range of rule definitions is available in the subsequent `assemble` and `compile` states.

Assemble

The query plan is turned into an artifact that can be compiled. Additional optimizations are applied at this point, and filtering code for scans is generated.

Compile

In this state, the assembled query is turned into a compiled binary that can be sent to the workers. Depending on the complexity of the query, it can remain in this state for a significant amount of time. This phase of the query lifecycle is highly parallelized. Information available to rules includes memory requirements, confidence in memory estimates, and the number of workers. You can write rules to throttle compile resources and make decisions based on memory information.

Acquire Resources

Resources are acquired for the query. Compile statistics are now available to WLM, as well as the size of the loader cache. You cannot apply rules to queries in this state.

Run

The query is sent to the compute nodes and starts executing on one or more nodes. Detailed query statistics are now available. Runtime WLM rules can be applied.

Client Wait

The query is sending rows to the client and waiting for the client to receive or request all the rows. You cannot apply rules to queries in this state.

Completion states: Done, Error, Cancel

`Done` state is reached after the last result rows have been received by the client. `Error` state is reached when a query fails for some reason (and is not restarted). `Cancel` state is reached when the query is canceled by the user. Completion rules can be applied when queries reach any of these states, to take simple actions such as logging messages.

Queries remain in the `sys.query` view (with state `Done`) until they are asynchronously written to the `sys.log_query` view. At that point they no longer appear in `sys.query`. The same query ID may appear in both views for a brief period of time.

Restart states: On Error, On User (via Rule or Administrator)

Queries may be restarted by the user or by the system.

Several known query error codes will trigger an automatic restart, causing query execution to be retried. For example, a query that runs out of memory is typically restarted. A query that is moved from one resource pool to another may also be restarted by the system if the move does not provide adequate resources.

A query may also be restarted based on the application of a specific rule or when an administrator runs a SQL `RESTART` command.

Queries are restartable from the `Assemble`, `Compile`, `Acquire Resources`, and `Run` states and may cycle through those states multiple times. Restarted queries always restart from the `Assemble` state. When a query restarts, it may run under different conditions because the workload is changing as other queries are submitted or completed. Additional resources may or may not become available to a restarted query. Note that when a query restarts, it is subject to the same sequence of

rules that were evaluated and applied when it was first submitted (as well as any specific restart rules when it goes into a restart state). A rule that was evaluated when a query first entered the `Compile` state, for example, may have a different outcome the second time it enters that state.

A small set of WLM actions can be applied when a query enters a `Restart` state, mainly for logging and audit purposes.

See Rule Types for more details.

How WLM Works

Platforms: All platforms

Parent topic: Workload Management

This section explains how WLM works and walks through the setup process for profiles, resource pools, and rules.

Profiles and Configurations

At any given time, a single WLM configuration is active *per virtual compute cluster*. This active state is based on a specific WLM *profile*. A profile describes one or more workloads that the system needs to manage and a set of operating conditions, as defined by associated *resource pools* and *rules*.

Rules and resource pools are the main mechanisms for defining the conditions under which concurrent queries are executed and scheduled. WLM rules trigger work to go to different pools and consume resources appropriate to those pools.

When you want to modify your WLM strategy, you can switch to a different profile and change the active state. You may have different configurations set up based on the workloads that are run by different compute clusters, and you may have a WLM configuration that is intended to satisfy requirements for "business hours" and a configuration that meets the requirements for a "maintenance window." You can also activate changes by adding or changing rules and pools in the current profile, rather than by switching profiles completely.

For more details, see Creating WLM Profiles.

Resource Pools

Every profile is associated with one or more *resource pools*, which define attributes for work that gets assigned to those pools. These attributes define maximum memory allocation, minimum and maximum concurrency levels, queue length, and so on. To reduce the need to maintain and switch between too many different profiles, "flex pools" have adjustable concurrency limits, rather than fixed boundaries on the number of queries that can run at any given time. These pools support flexible and dynamic sharing of system resources.

Upon activation, a profile inherits a fixed (and shared) `system` pool that you cannot edit or remove. The presence of this pool guarantees that background system operations and Yellowbrick Manager operations are not starved of resources. Every profile *requires* another pool that is marked as the *default pool*; the `system` pool cannot be the default pool. You can add any number of other pools to the profile based on your requirements.

For more details, see Creating WLM Resource Pools.

Rules

Work is assigned to resource pools based on *rules*. These rules qualify work for execution in specific pools, based on various criteria and actions triggered by those criteria. For every query that comes in, all of the rules associated with the active configuration are evaluated. For example, the user that runs a particular query, the database in which it runs, or the SQL content of the query itself might determine which pool the query runs in, impose limits on or expand its memory use, or decrease or increase its priority. Rules also influence concurrency, providing a means to throttle the number of queries that a given user, role, or application can execute at the same time.

For more details, see Rule Types and Creating WLM Rules.

Summary of WLM Setup Steps

To set up a simple WLM configuration and test it, follow these steps:

1. Create a profile.

2. Create one or more resource pools.
3. Set one pool as the default pool.
4. Create some rules.
5. Activate the profile for work done by a specific target compute cluster.
6. Run some queries to test the application of rules.
7. Monitor rule processing and resource pool selection.

For a specific example of these steps, see A Simple WLM Example.

FIPS 140-2

Platforms: EE: All cloud platforms, CE

Parent topic: General

FIPS 140-2 is the Federal Information Processing Standard that deals with standards for cryptographic modules. It mandates that cryptographic modules meet minimum security standards for protecting sensitive information. Yellowbrick uses FIPS-compliant, certified OpenSSL and Java libraries, and enables "FIPS mode" for these libraries by default.

Cloud Architecture and Concepts

Yellowbrick Documentation > Concepts > Cloud

Platforms: All platforms

Parent topic: Concepts

In this section:

This section details architectural concepts of Yellowbrick in the cloud.

Inside the Yellowbrick Data Warehouse

Summary

This white paper describes how Yellowbrick rebuilt the cloud database software stack by leveraging Kubernetes and modern computer architecture to implement a modern, fully elastic SQL analytic database with separate storage and compute. By revisiting key assumptions in database architecture, Yellowbrick delivers the most efficient data warehouse in the industry with flexible deployment in customers' own cloud accounts or on-premises.

Introduction

The Yellowbrick Data Warehouse is a cloud native, parallel SQL database designed for the most demanding batch, ad hoc, real-time and mixed workloads. Fully elastic clusters with separate storage and compute run complex queries at multi-petabyte scale with sub-second response times.

Yellowbrick innovates in three key areas:

- Enabling a more efficient business model by allowing customers to consume a modern, elastic, SaaS user experience in their own cloud account with predictable cost.
- Optimizing price/performance for new use cases requiring more concurrent users and more ad hoc analytics.
- Providing deployment flexibility by offering an identical data warehouse for on-premises use cases.

This document explains the architectural approaches that support these innovations. They cross everything from the creative usage of Kubernetes to cluster management, to data storage, to query planning and execution, and even the user interface itself.

Yellowbrick is designed to run tier-1 enterprise-grade data warehouses with the following characteristics:

Modern, elastic architecture	Elasticity, separate storage & compute, driven completely through SQL.
Run in the customer's cloud account	Yellowbrick customers pay for their own cloud infrastructure, make use of their own cloud storage and control their own data security . Yellowbrick doesn't see or store user data or queries.
Run across all public clouds	At the time of writing, Yellowbrick supports AWS, Azure, and GCP public clouds.
Run on-premises	Provide the same elastic user experience as in the cloud on-premises.
Hardened SQL support	Yellowbrick can reliably execute complex ANSI-standard SQL queries across massive schemas, supporting complex join hierarchies, correlated subqueries, deeply nested CTEs, stored procedures, etc.
Execute complex workloads	Yellowbrick handles highly concurrent mixed workloads with continual bulk and real-time ingest, merging, and highly concurrent queries with guaranteed quality of service through workload management. Full ACID transaction semantics are present throughout the stack.
Reliability	Yellowbrick is highly available and resilient to node, storage, and network failure. Workloads across compute clusters are isolated from one another.
Support for disaster recovery	Asynchronous replication of both data and DDL with read-only hot standby instances is built-in.

Support for data retention and business continuity	A mature enterprise-level backup scheme for off-site data retention supports incremental, cumulative, and full backups and object-level restore.
Mature ecosystem	Yellowbrick has enterprise support agreements with all major BI, ETL, data mining, CDC, and machine learning vendors.
Best query efficiency in the industry	Yellowbrick executes ad-hoc queries against large data sets incredibly efficiently, making use of many technical innovations in query execution.
Flexible pricing and consumption	Support for subscription contracts as well as workload management allows variable numbers of concurrent users with a predictable price.
Open interfaces	By making use of PostgreSQL's wire protocols, *DBC drivers, and metadata schema, Yellowbrick is comfortable for developers and DBAs to work with. Open-source integrations for tools like Kafka and Spark are standard.

Table 1: Yellowbrick characteristics

Yellowbrick isn't a SQL-on-Hadoop type of product, or a solo query engine running on top of other open source infrastructure. It's a database that organizations can trust to store and be the system of record for their most valuable enterprise data, and to generate business-critical, auditable financial reports that their businesses depend on. It requires almost no management, tuning, diagnosing, or handholding and is familiar to modern developers accustomed to working with PostgreSQL.

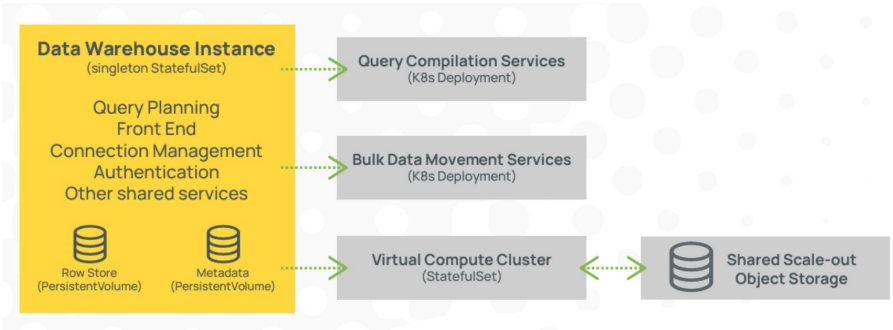
Cloud Native Cluster Management

Full elasticity with separate storage and compute is now a table stakes feature for any modern data warehouse. BigQuery and Snowflake set the bar here, the latter allowing explicit control of clusters ("virtual warehouses") within the database through an expressive SQL grammar. Both technologies were conceived before Kubernetes was invented. Kubernetes has been a double-edged sword: On one hand, it provides a declarative, standard, and portable interface for running scale-out applications across many different cloud vendors and stacks ("the new cluster operating system"); but on the other, it places a burdensome and daunting management overhead onto the teams that run such clusters due to its extreme complexity.

At Yellowbrick, Kubernetes is used internally to gain portability across cloud platforms without requiring end users to know anything about it. Users never touch or see Kubernetes, a helm chart, pod, node, operator, or configuration file and aren't aware they even exist. The user experience largely mirrors that of Snowflake: creating, growing, and destroying compute clusters through SQL utility statements, and viewing the state of the scale-out data warehouses through SQL tables. Users simply provision a warehouse and create an elastic cluster through one line of SQL (or a few clicks of the UI) and off they go.

Data Warehouse Instance Topology

Each data warehouse instance is a fully independent database supporting multiple elastic compute clusters with separate storage and compute. Data warehouse instances are independent of one another but can be managed through a single pane of glass with the Yellowbrick Manager. There is no coupling of availability between instances, and an error in one instance cannot affect another instance. Each instance is deployed into one Kubernetes namespace. Figure 1 shows the components that make up each instance:



- **The instance pod** is the front end of the database. It provides shared services such as connection management, metadata schema, and query planning. The entire instance can be suspended and resumed. Yellowbrick's row-oriented storage for real-time ingest is also located in the instance pod to provide the lowest possible commit latency. No query execution occurs on the instance pod; query execution is deferred to elastic compute clusters.
- **Elastic compute clusters** are required to write data and execute SQL queries. Each compute cluster operates independently of one another. SQL utility statements support creating clusters, assigning users to clusters, suspending, resuming, and resizing. Since multiple compute clusters work on the same shared data, which is cached locally and persisted in object storage, different clusters can be used for different purposes. For example, spinning up a cluster on demand to perform ETL, having a different cluster for business-critical reports vs. ad-hoc queries, or simply creating more clusters to support higher levels of concurrency.
- **Query compilation services** turn SQL statements into vectorized, executable code. Query plans and compiled code are aggressively cached for re-use. Query compilation services pods scale up and down automatically based on CPU consumption.
- **Bulk data movement services** support bulk loading and unloading of data and scale out automatically, depending on the number of concurrent requests.

System tables are provided for querying the current state of all the above services. For example, `sys.cluster` shows the current state of all elastic compute clusters.

Implementation Using Kubernetes

SQL-based management of elastic compute clusters is implemented using Kubernetes while hiding details from the users and DBAs. When SQL commands to scale compute clusters or supporting services are issued, the Instance pod uses the Kubernetes API to create or alter underlying API objects (for example, pods, namespaces, configmaps, and events) and monitor progress in affecting the changes. Similarly, when querying system views to examine the state of clusters, the Instance pod executes Kubernetes API requests to list the relevant objects, join the Kubernetes API objects with locally stored metadata, and return the view to the user.

The very existence of Kubernetes is never exposed to users, but by leveraging this containerized, cloud-native architecture, Yellowbrick gains trivial portability across all the public clouds and modern on-premises infrastructure. This "SQL user interface for Kubernetes" is a Yellowbrick innovation that is not available in other data warehouse platforms; most don't support the management of elasticity and scaling through SQL, and those that do aren't implemented on top of Kubernetes. It also becomes possible to embed and scale Yellowbrick inside other microservices-based, cloud-native applications, even to the point of automating instance creation and provisioning through provided GraphQL APIs.

Tracking Consumption and Auditing Changes

All changes in the level of billable resources are captured in system tables. The core unit of consumption in Yellowbrick is the vCPU/second of cluster compute resources used; this is aggregated hourly and daily and can be queried through system tables, summarizing by user or by cluster, etc.

Direct Data Path Architecture

The algorithms used by most databases built in the last two decades have several consistent core assumptions:

- Databases executors should cache data in a buffer cache in memory to avoid reading too much data from disk.
- Core operators, such as joins, sorts, and aggregates, should use a buffer pool to transparently support spilling data to disk in case memory capacity is exceeded.
- Keeping more data "in memory" is the way to better performance – after all, CPUs can copy memory far faster than they can move it from the storage or network.
- Tight integration with Linux provides performant storage and network IO and threading.

Yellowbrick turns most of these assumptions on their heads because on modern servers available in the cloud and on-premises, it's now the case that:

- The maximum effective bandwidth of reading data from high performance, direct attached storage can be the same as from main memory.
- Moving data across the network can be less resource-intensive than copying it from/to main memory.
- The CPU can run 10x faster if processing data from caches, rather than from main memory.
- Linux simply can't scale IO effectively at the levels of concurrency offered by today's servers.

These advances in core computer architecture have allowed us to build the Yellowbrick database engine in a manner that supports much more efficient query execution than other databases. It's not uncommon to find Yellowbrick using 1/4 or less vCPU resources per query than competitors, resulting in substantial cost savings for customers.

To take advantage of the latest technology, Yellowbrick developed a software data path that's much more efficient than that available in Linux. Linux's limitations are quite widely known in the network security and low-latency stock trading community, where the most advanced software developers will talk about using OS bypass technologies such as DPDK or OpenOnload with Linux hugepages to fix such problems. However, these leading-edge technologies from Intel and others are far too restrictive to be sufficiently general purpose for running a parallel database engine.

Yellowbrick's own OS bypass technology is specifically designed for analytic database engines and – due to recent advances in Kubernetes and container architecture – can be used in any standard container environment. Yellowbrick has developed:

- A networking stack for high-speed data exchange between parallel processing nodes that's up to 20x more efficient than built-in Linux.
- A storage I/O stack and file system with 1/100 of the cost per IO compared to standard Linux.
- An S3-style object storage stack that supports all three cloud providers at 1/10 of the CPU usage compared to the vendors' own software.
- A process and thread management subsystem with no measurable overhead.
- A cluster-based scheduler to allow synchronized query execution across cluster nodes.
- A memory management approach optimized for the above.

We'll discuss some of these innovations below.

The Need to Optimize for Modern Computer Architecture

Today's servers are routinely available with over a terabyte of memory and over 100 CPU cores, and the trend of more cores and memory is set to continue. Running generic software on these instances does not work well because operating system schedulers were built to wait for events and "context switch." Threads wait for events – such as a keypress, a network packet arriving, a storage I/O completing or synchronization primitives becoming available – and switch between competing threads and processes to try to be as fair as possible and use buffers efficiently. As a result, it's not uncommon for modern databases to do tens of thousands of context switches per second per CPU core, and millions of them per second in aggregate.

Conventional wisdom states that if you're not spending much CPU time context switching – under 10% – you're in good shape; context switches are cheap with a good operating system. However, this assumption is outdated. Modern CPUs get their performance from processing data from their caches, typically called L1, L2, and L3. The L1 cache contains data pertinent to the most recent processing, the L2 cache is larger but slower to access, and likewise the L3 cache. The L1 cache per CPU core is measured in tens of kilobytes, the L2 cache in hundreds of kilobytes, and the L3 cache in single-digit megabytes. If you were to look at the speed at which you can access data on a modern CPU, you'd see ratios like those shown in Table 1.

Memory Type	Access latency (in nanoseconds)
L1 cache	1.5
L2 cache	5.3
L3 cache	24.5
Main memory	120

Table 2: Memory access latency by memory tier

When the CPU context-switches inside a database, the different contexts may be doing tasks such as:

- Running a hash join of two tables for Tim.
- Sorting data for Torben.
- Running a hash join of two tables for Marc.
- Waiting to look up a block in the filesystem for Mark to scan a table.
- Preparing data to send over the network in TCP packets for Jason.

Each of these tasks is accessing its own large data structures and dealing with its own cached data. Each context switch requires moving new data in and out of the CPU caches, invalidating data that was there before.

One might ask, "Well, what if we were to be able to do these tasks sequentially: Finish Tim's join, then Marc's join, then Torben's sort, then Mark's work, then Jason's? Doing things one at a time, we'd do less context switching and use our caches more efficiently, right?" This is partly true, but it doesn't end there: The minute Joe's join needs to send data over the network, the CPU enters the Linux networking stack – around 100,000 lines of complex code that will do a great job changing memory mappings, filling the caches, and evicting Joe's hash tables, only to force them to be reloaded from main memory when the network processing is done. Similarly, if Joe's join needs to read a new disk block, it will enter the Linux I/O stack and filesystem, and hundreds of thousands of lines of code and complex data structures will also handily displace his hash tables.

When this context switching and bouncing in and out of complex Linux kernel subsystems is happening continuously across dozens of cores, any modern CPU will struggle to work efficiently. The DBAs will be none the wiser because the CPU will be 100% utilized, but under the covers, the database is achieving only a fraction of its theoretical maximum efficiency. Therefore, most data warehouses today support very low levels of concurrency.

Design of the Direct Data Path

Yellowbrick's Direct Data Path implements a new execution model to work around the issues above by eliminating measurable context switching overhead and penalties associated with accessing storage, the network, and other hardware devices. This is done with a new, reactive programming model for the entire data path. Some of the principles of this new programming model are described below.

Threading, Processes, and Scheduling

Yellowbrick has a new threading model based on reactive concepts such as futures and co-routines. Small, individual units of work called tasks are scheduled and run to completion without preemptive context switching. Tasks do not have stacks associated with them. Tasks in Yellowbrick never block (as far as the OS is concerned) and no stacks are preserved when waiting for futures; instead, it's up to the caller to optimally preserve state in either per-thread data structures or lambdas. Although tough for general-purpose programming, in a constrained environment such as a database, this model is tractable and offers massive efficiency improvements.

Yellowbrick implements synchronization primitives as well as parallel iterators to protect access to shared resources. Awareness of multiple CPU cores and NUMA nodes is intrinsic to the system from top to bottom: The author of code, written in assembly or C++, makes deliberate decisions as to the parts of the compute complex on which to run. Optimal memory allocations, from the closest or most recently cached data structures, will be provided automatically to the runtime, and even the handling of bizarre modern CPU artifacts (such as cache aliasing of stacks) is built-in.

In a traditional OS, a process comprises threads that execute. Yellowbrick implements a different type of process model: A work comprises tasks that execute in a fully asynchronous, reactive manner. Works have their own memory arenas from which to allocate, which are torn down together, and all resource consumption of the work is bounded and isolated by the kernel: what fraction of CPU it can use, how much memory, how much compute, how much disk storage, and so on.

The scheduler is aware of works and tasks and will try not to intermix the execution of tasks from different works to further avoid cache displacement. It even goes one step further and tries to ensure that, when database queries are exchanging large amounts of data (such as re-distributing data for a large join), the same work is running on peers in the cluster at exactly the same time, such that received data may be processed immediately. Multi-tasking is cooperative, rather than preemptive.

Works are created programmatically from built-in code, or dynamically loaded and unloaded at runtime, just like processes in Linux. The latter approach is used by the database for runtime query loading and unloading.

Device Access

The database needs to access network devices, storage devices, and hardware accelerators when available. Traditional device drivers run in the Linux kernel and interrupt execution whenever something happens. In contrast, all Yellowbrick device drivers are asynchronous and polling in nature. Access to drivers is always via queue pairs – command and completion queues – with well-defined interfaces. Drivers are present for general PCIe devices, NVMe SSDs, various network adapters, and so on, all of which work without Linux's involvement. In cases where Yellowbrick is running without bypass being available, emulated drivers for each class (network, storage, etc.) are present that fall back on the Linux kernel or software emulation.

Local File System

Yellowbrick implemented a local file system called BBFS (Big Block File System), that builds on top of the device access layer to provide a full namespace on top of raw storage devices. It provides most expected file system semantics – directory management, creating files, open, close, read/write, readv/writev, delete, rename, etc. – but is implemented completely asynchronously. Metadata and metadata locks are heavily sharded to maximize concurrency. The file system metadata itself is tiny, fitting in indexed in-memory structures for fast lookups. A read() operation on a file can be submitted through the entire file switch, file system, and device drivers in only a few hundred CPU cycles.

Network Data Exchange

Like many modern microservices-based software stacks, Yellowbrick is implemented in a variety of different languages: Primarily C, C++, and Java, with a sprinkling of Go and Python where necessary. These services need to talk to each other. For maximum efficiency, Yellowbrick contains a highly efficient communication framework called YBRPC that's optimized for the latest server instances. Several different underlying YBRPC transports are in use:

- **YRD (Yellowbrick Reliable Datagram):** This is built using Intel's DPDK library for OS bypass. It's a datagram-oriented protocol that implements checksums and re-transmissions as needed to guarantee reliability and be able to route across subnets. Due to wide support for DPDK, we use this protocol on all cloud-based instances. YRD enables zero-copy sends and single-copy receives with a fraction of the CPU overhead of TCP.
- **RDMA (Remote Direct Memory Access):** A networking technology used to move data directly between the memory of servers as efficiently as possible. It works over both Ethernet (where supported) and InfiniBand fabrics, primarily for on-premises instances. Sends and receives are both zero-copy and latency is measured in nanoseconds.
- **Linux TCP:** A transport layer for generic kernel-based TCP which is used when other more efficient transports are not available.
- **Unix Domain Socket:** When processes are known to execute on the same node, this is more efficient than using a TCP socket.

The custom network stacks pay substantial dividends in query execution efficiency: Yellowbrick benchmarks have clocked a single CPU core sending and receiving 16GB/sec of data across the network in the MPP fast path, with time to spare. When using the Linux kernel, around 1.5GB/sec is the limit and the CPU core is fully loaded, leaving no time whatsoever for data processing. YBRPC allows expensive parts of database queries – such as re-distribution of data for joins, aggregates (GROUP BY), and sorting – to run 10x more efficiently than competing databases, using a fraction of the resources.

Object Store Access

In cloud architecture, local storage is ephemeral, so the only way to reliably persist data is by writing it to some form of remote storage, and Amazon S3/Azure ADLS/GCS object storage is the most cost effective. Remote storage has issues with latency, though, so to maximize bandwidth and IOPS, large IO queue depths across many targets must be correctly pipelined. The client libraries from the cloud vendors are incompatible and all third-party libraries are incredibly inefficient, performing gratuitous data copying and dealing poorly with pipelining the massive numbers of outstanding operations needed to drive high bandwidth. By developing a custom asynchronous user-space HTTP stack and object store library, CPU consumption is reduced to around 3% of Amazon's library. Direct attached storage is used as a cache for blocks on object storage to further increase processing efficiency in the cloud.

Cluster Parity Filesystem

To lower costs while improving availability and reliability for on-premises deployments, Yellowbrick built a stacked, higher-level cluster filesystem called ParityFS. It sits on top of BBFS, implementing the same POSIX-subset interface in an asynchronous, reactive fashion, but provides the system with resilience against data loss in case of node failure. In a traditional database, such resilience is provided by "mirroring" or "replicating" multiple copies of data: Typical database deployments will store two or three copies of data across nodes so that, in case of node failure, another node in the cluster can replace the work of the failed node. However, doing so will lead to that node performing twice as much computation (since 2x more data needs to be processed), leading to substantially slower query processing performance due to the added "skew"' in query processing.

Yellowbrick has implemented erasure coding similar to RAID-5 or RAID-6 in a disk-drive configuration but at file level rather than block level. The scheme provides the same level of redundancy as writing three copies of data but without the storage overhead. As files are written in parallel across the cluster, reconstruction data is written. If a node is lost, the files it was storing are virtually reassigned to all the other nodes in the cluster and reconstructed dynamically when read – such that all nodes in the cluster share the processing work of the failed node. The data writing and data reconstruction processes use the massive amount of parallel computation and high network throughput available, such that in the rare event of nodes failing, the overhead added to typical database queries is under 5%. At the time of writing, Yellowbrick production customers that have experienced node failures never even noticed performance degradation.

Query Execution

The Yellowbrick database engine is responsible for taking query plans, performing computation on data – reading and writing as necessary – and returning answers. There are four major software components, all built from scratch by Yellowbrick: The Storage Engine, the Execution Engine, the Workload Manager, and the Query Compiler. There's also a fifth, optional hardware accelerator (the Kalidah processor) for customers using the Andromeda optimized instance in private clouds, which offloads much computation.

Storage Engine

The Storage Engine stores structured data (two-dimensional tables with rows and columns). It's designed to scale from tables with as few as one row to tables with trillions of rows and thousands of columns, occupying petabytes of storage space. Almost all the data stored in the database is stored in a column-oriented store, and large loads of data are written directly to this store. Recent real-time or streaming data is stored in a row-oriented store instead. When the row-oriented store reaches a certain size, data is automatically moved to the column-oriented store. The storage engine is ACID compliant; transactions semantics are consistent between both the row-oriented store and column-oriented store, and queries automatically look at all the data present in both stores.

Row-oriented Store

The row-oriented store (row store) is a scale-up storage engine. It's optimized for low commit latency for real-time streams, such as those from Kafka or CDC tools. When streaming data (such as INSERT statements in small transactions) arrives in the database, the most important thing for it to do is commit the data as fast as possible and return control to the client so it can continue. The number of real-time, streaming commits per second is thus a function of commit latency, and the fastest path to the lowest possible latency is to avoid networking and commit to disk.

Rows are stored in a log-oriented structure in which multiple files are kept per table per CPU core, and new rows of data are appended in real time. It's stored on mirrored, highly available storage volumes: For cloud instances, on EBS-like volumes; for on-premises instances, on four-way replicated SSDs. When a given table's row store approaches a size where it will have a measurable impact on query performance, it is flushed in the background to the column-oriented store. Users and admins need not concern themselves with this flushing process.

The row store contains a high-bandwidth streaming optimization: When an incoming stream is copying many rows in a transaction without committing, the row store can transparently switch its operating mode to one where it passes the data directly through to the columnstore without writing intermediate data to disk. Upon commit, the data will be persisted to the columnstore instead.

Column-oriented Store

The column-oriented store (columnstore) is where most data in Yellowbrick resides. Columnar databases are nothing new, and the benefits for analytic workloads – improving compression ratios and reducing the amount of scanned data – are well known, so much so that most analytic databases now store their data in columns rather than rows.

Yellowbrick has been creative about how the column-store indexes and finds data efficiently on SSD media and cloud storage. The columnstore runs entirely within the Yellowbrick Direct Data Path. Static partitioning is not required to achieve acceptable performance: As tabular data is written, rows are distributed across storage nodes in the cluster into multiple partitioned units called shards, which represent around 200MB of compressed data. Data within each shard is laid out in a columnar fashion, so data from individual columns can be read and processed without having to read the whole shard. Rows may be reordered before the shard is written by "clustering" data on as many as four columnar keys. Clustering allows the database to group related rows, such that if, for example, queries often select rows by time and by customer, clustering on both columns will make accessing such data more efficient. Clustering is rarely necessary due to the staggering throughput of Yellowbrick, and won't make much of a difference to large, intensive queries, but it can help you find small amounts of data ("needle-in-a-haystack" queries) even more efficiently by making built-in indexes work better.

As shards are built and written, Yellowbrick builds granular indexes and compresses data. Yellowbrick uses several compression techniques, with a primary focus on the CPU efficiency of decompression rather than using the minimum amount of storage because storage costs are cheaper than compute costs. For each column within the shard, the indexes store common distinct values, a data structure for computing cardinality statistics, and the ranges of values present. These values are stored per shard, and hierarchically within each shard, per 4KB-to-32KB disk block, and within that, per small group of rows. This is a far more granular index structure than that used by traditional analytic databases. That's possible because the storage engine has been designed for SSD storage, which excels at large numbers of random disk operations (IOPS). When one can execute tens of millions of IOPS while consuming a barely measurable amount of CPU, table scans can be turned into massive numbers of random I/O operations to avoid reading as much data as possible. Doing so allows Yellowbrick to read only the minimal amount of pertinent disk blocks needed for each table scan, but this is done carefully to keep queue depths low enough to minimize L3 cache usage. Achieving massive bandwidth with tiny queue depths requires an extraordinarily efficient IO framework.

Since statistics are stored along with every shard and automatically recombined, there's no need to worry about keeping statistics up to date. Yellowbrick deletes rows in the columnstore by writing deleted row identifiers to new files alongside the shards. It handles row updates by deleting the old rows and inserting new ones into a new shard. A consequence of this design approach is that Yellowbrick is incredibly fast at running large bulk updates and deletes but less efficient at small random ones. A built-in garbage collector periodically sweeps up fragmented shards, removes deleted rows, and recombines them efficiently. In Yellowbrick, it does this incrementally in very small units of storage, so that – unlike in older databases where performance suffers greatly during vacuuming – there is no measurable impact. These processes do not need to be initiated by a DBA.

The architectural approach means that over time, for update-oriented and delete-oriented workloads, shards will retain garbage (old rows of deleted data) and storage efficiency will drop. However, the benefits of writing and recombining immutable shards of data enable us to implement functionality like data snapshots and time travel relatively easily. The former

is already productized for the backup and restore functions, and the latter is a committed roadmap item.

Locking and Transaction Management

Yellowbrick, being an enterprise-class database, implements full ACID transactions. The isolation level provided universally is READ COMMITTED. Locking is performed at the table level. The Yellowbrick transaction log comes from PostgreSQL and is solid, with decades of production use. Locking is built using a multi-version concurrency control (MVCC) approach. Regardless of how much updating, deleting, or loading is taking place, readers still see the data as of the transaction they are in. Readers never block each other. Generally, writers block other writers and will queue behind each other. However, there are isolated cases where more than one writer is allowed into a table:

- Multiple "bulk loads" can run concurrently into one table.
- Bulk loads can run concurrently with other update and delete operations.
- Data flushes from the row store into the columnstore can run concurrently with bulk loads.

Hybrid Execution Engine

Just as the database stores data in both rows and columns, Yellowbrick also executes queries in a row-oriented or column-oriented fashion. The execution engine, otherwise known as the "EE," is modeled after packet-processing frameworks, just like networks. SQL operators are like nodes in the network, and packets flow over the links between the nodes.

Query Topology and Flow Control

A query planner turns a SQL query such as:

```SQL
SELECT a,b FROM foo INNER JOIN bar on foo.p=bar.f
```

into a hierarchy of SQL operators. In the case of this simple query, the query plan contains two table scan nodes and a join node. Data flows from the bottom of the tree up to the top of the tree, where it's returned to the user.

In the EE, queries are represented by graphs rather than trees. The nodes in the query graph are the operators – such as table scan, join, or sort – and the edges connecting the operators are links. Graphs allow us to plan and execute complex query topologies, such as a table scan that feeds multiple consumers of data at the same time. The job of the EE is to execute the query graph optimally. Data flows from the leaves of the graph toward the root of the graph. As data is handed between nodes in the graph, it's placed into packets that may contain row-oriented or column-oriented data. Packets are sized to make optimal use of L1 and L2 cache memory, so they can be kept core local as they move between nodes. Flow control is required, just like in networks, to ensure that memory usage and queue depths of packets are bounded to stay cache-resident. This is because some nodes may produce far more packets than they consume, such as an outer or cross join, whereas others may produce far fewer, such as an inner join with few matching rows.

In contrast to traditional databases that use "iterators" to pull data from the top of a tree (the "volcano model"), Yellowbrick uses a more sophisticated approach that lets us tightly control resources, where grants are handed toward the leaves and data flows in the reverse direction.

The Distribution operator, which moves data packets across the physical network between MPP nodes, also uses the same flow control and backpressure approaches – in essence, extending the EE graph to be global across MPP nodes. The network buffers are the packets themselves and they can be transmitted and received in place with no data copying when supported by the underlying YRPC transport.

The entire EE framework is fully multi-core and NUMA-aware. Wherever possible, packet processing is kept primarily core-local and secondarily NUMA-node-local; but in case of skew, reallocation of packets across cores on a NUMA node will take place first, followed by reallocation across NUMA nodes if necessary. This affinity of packets and operators to cores and NUMA nodes also extends across the MPP network.

Row-oriented and Column-oriented Execution

It's optimal to process data in different ways depending on the type of SQL operator node in question. The EE supports row-oriented and column-oriented packets. For example, the Distribution operator, which moves data packets across the physical network between nodes to implement MPP execution, wants to operate in a row-oriented fashion because rows of data will be transmitted to different MPP nodes depending on the hash of a column in the row. Likewise, the Join operator combines rows from two different tables and concatenates them.

However, the table scan node prefers to operate on columnar data straight from the Storage Engine, where it can take advantage of vectorized execution. Vectorization allows us to apply efficient SIMD (AVX) instructions to build incredibly efficient predicate filters, expression calculators, or bloom filters that can operate on multiple values in one CPU instruction, but it requires that data is laid out in a manner that is amenable to the instructions.

Partitioning

Like many OLTP and OLAP databases, Yellowbrick supports a SQL syntax for partitioning tables. Admins can define partitioning schemes on a per-table basis. Currently, hash partitioning and range partitioning are the available schemes.

Most MPP databases implement partition pruning at the planner level to find data more rapidly by sub-setting the data that needs to be scanned. In Yellowbrick there's no need for partition pruning because shard indexing is so efficient. Instead, all knowledge of partitions is pushed down to the executor itself. DBAs configure Yellowbrick to use partitioning solely to reduce memory usage for massive joins or aggregates. Consider two examples:

1. A hash join of two massive tables with perhaps hundreds of billions of rows (a fact-to-fact join). The traditional approach would be to build a hash table on the smaller side of the relation and scan the larger side, looking up each row and emitting the joined result. The hash table is still huge, occupying a massive amount of memory and likely forcing the query to spill.
2. A billing query for complex call data record (CDR) time-series data, with more than 1 billion subscribers, summing call costs by telephone number and call. For a large telecom operator with hundreds of millions of subscribers making large numbers of calls per day, the hash table for the GROUP BY operator would surely be gigantic and spill to disk.

In the case of example 1, if both hash tables are partitioned identically, the join can be executed a partition at a time because it's known that the data in the partitions is non-overlapping. If 1,000 partitions were chosen, 1,000 smaller joins would be performed rather than one massive join, using 1/1000 of the necessary memory. The query will execute faster and will not spill. Similarly, for example 2, rather than running one giant aggregate, if it is known that the CDRs have been partitioned by hour, the query can run one aggregate per hour and incrementally emit the result.

Yellowbrick accomplishes this by adding partition iterator nodes to the query plan. Partition iterators repeatedly reset and re-run all downstream operators, once for each possible partition value, incrementally emitting results. This is a far more sophisticated approach than naive planner partition pruning and yields substantial benefits in runtime efficiency by reducing memory utilization and eliminating spilling for many complex queries.

Storage Predicate Pushdown

The Yellowbrick Storage Engine maintains various granular indexing structures to efficiently find data that's necessary and avoid data that isn't. For example, if you run a simple query:

```SQL
SELECT * FROM person WHERE age<24;
```

The executor will pass the predicate `age<24` to the storage engine to enable it to return only rows matching the criteria.

Yellowbrick does both static predicate pushdown – identifying predicates derived during query planning such as described above – as well as dynamic predicate pushdown, which adds additional predicates to queries at runtime. Static predicates are identified at plan time, but the actual values are injected at runtime to enable parameterized queries to make use of pushdown functionality.

Runtime predicates are generated by joins as well as SQL constructs that perform similar operations to joins, such as large IN lists or sequences of OR criteria, which are internally rewritten to semi-joins. The runtime predicates typically take the form of bloom filters, pushed down after generating the build sides of hash tables. Yellowbrick also creates BETWEEN predicates from the minimum and maximum values present in the build sides of joins, which helps in surprisingly common cases where there is correlation between the two tables.

Query Compilation

All queries in Yellowbrick are aggressively compiled to CPU instructions to run as fast as possible, in their entirety. Yellowbrick contains no interpreter and no just-in-time compiler for queries. Memory management in queries is explicit, with no garbage collection.

Yellowbrick contains a SQL compiler built from scratch called Lime. Lime's job is to consume the output of the query planner and generate code to execute the query. It understands the Execution Engine and the reactive programming model. Lime's processing consists of multiple phases:

- Produce an abstract syntax tree (AST) for the incoming query plan, converting query plan nodes to execution engine operators.
- Perform a type optimization phase.
- Apply several optimization passes on the AST: rearranging data to be contiguous in memory, static evaluation, reordering of memory accesses, fast-path deduction, and so on.
- Emit code (including inline assembler) corresponding to the AST.
- Compile the code using the optimizing LLVM compiler infrastructure.

LLVM Compilation

Lime uses a modified version of the LLVM compiler infrastructure to generate machine code. LLVM itself can generate highly optimized code. The EE framework and SQL operator templates are built in C++. They use inline-able injector functions: It's not just the "core loops" of the operators that are compiled; the code implementing the entire operator is ultimately expanded to one set of code that can be aggressively optimized by LLVM.

This is a relatively expensive process, especially in a single-threaded compiler such as LLVM. To provide interactive queries quickly, Lime will split each query into multiple exclusive segments – segments for which in-lining code from other segments would add no value – that can be both generated and then compiled in parallel across multiple CPU cores. To further lower latency, we have modified LLVM to allow it to remain memory-resident in a state with its own ASTs pre-loaded. If compilation becomes a bottleneck, the compile service itself is elastic and horizontally scalable, and more pods will be spun up.

This parallelization allows simple queries to be compiled and executed in milliseconds, while very large, complex queries can still compile within a couple of seconds, fast enough for interactive analytic queries.

Pattern Compiler (Regular Expressions and Friends)

Regular expressions and LIKE operations have historically been slow when run against large data sets. To improve execution speed, Yellowbrick implements a special compilation framework called the pattern compiler. The pattern compiler currently supports the following input patterns:

- **SQL LIKE**
- **SQL SIMILAR TO**
- **POSIX-compatible regular expressions**
- **Date/time parsing**

The pattern compiler generates highly optimized, deterministic finite state machines for each unique pattern. The set of state transitions is optimized and compiled into optimal machine code by the LLVM compiler infrastructure and then loaded as needed into running SQL queries. All POSIX regular expression functionality is supported, including all character classes, operators, and capture groups. The pattern compiler supports backtracking and analyzes subexpressions to determine whether a faster deterministic, or slower nondeterministic, finite automaton, is necessary on a per-operator basis. Storage predicate pushdown is performed for the starting characters of all patterns.

Yellowbrick has one of the fastest database regular expression implementations ever created, if not the fastest. At the time of this writing, the Yellowbrick database contains no support for interpreting or doing JIT compilation of patterns so it's not possible to store a regular expression in a table column and use it in a query; however, you may supply patterns as runtime parameters in queries.

Code Caching

Although code compilation is fast, latency is still minimized by saving milliseconds of compile time for short, tactical queries and seconds of compile time for large, complex ones. To do that, the query compiler contains multiple layers of caching. Efficient and reliable caching requires care and attention: A given query has a huge quantity of dependencies, from execution engine templates to the version of the C++ runtime, versions of libraries, the query plan in use, the statistics of the query, and so on. Yellowbrick factors all these dependencies to try to reuse as much previously compiled object code as possible.

Kalidah Processor

The Kalidah processor is new accelerator intellectual property developed by Yellowbrick for optimized on-premises hardware instances. Designed for implementation in FPGA or ASIC, it accelerates bandwidth-oriented data processing tasks used during table scans such as data validation, decompression, filtering, and compaction. Kalidah supports all data types currently supported by the Yellowbrick database, including decimal numbers and variable-length strings. For more details about Kalidah, see the "Andromeda Optimized Instances" white paper.

PostgreSQL Compatibility and Query Planning

We use open-source technology for external access to the database. PostgreSQL is the logical choice: Its SQL support is very close to ANSI standard, with a track record of adoption in other data warehousing platforms (Amazon Redshift, IBM Netezza, Vertica, and so forth). Furthermore, PostgreSQL has become perhaps the coolest and fastest-growing relational database, with a thriving ecosystem of developers and users contributing to its success.

Compatibility

The front end of the Yellowbrick database derives from PostgreSQL 9.5.x. Yellowbrick periodically merges fixes and enhancements from newer PostgreSQL versions where it makes sense. Yellowbrick is not a PostgreSQL fork; if you write an analytic query against Yellowbrick, more than 99% of the machine instructions will be running against the new Yellowbrick code. The core parts of PostgreSQL – many parts of the query optimizer, the storage engine, and the execution engine – have been entirely replaced in Yellowbrick.

Access to Yellowbrick is typically accomplished via standard PostgreSQL ODBC/JDBC/ADO drivers. This is a deliberate choice because it allows the database to interact with numerous ecosystem tools. That said, PostgreSQL's wire protocols are not particularly efficient, so Yellowbrick is actively developing custom higher performing *DBC drivers while continuing to maintain Postgres compatibility. PostgreSQL's metadata catalogs are also supported for interoperability with standard tools and will be familiar to admins and developers alike.

SQL Dialect

At the time of this writing, Yellowbrick supports the following SQL data types: Booleans; integer types; decimal types; floating point types; UUID, CHAR, and VARCHAR; date and time types; and some new data types for IP address and MAC addresses. INTERVAL is supported as an intermediate data type and cannot be stored. TEXT is present for PostgreSQL compatibility and internally aliases to VARCHAR(64000).

PostgreSQL's SQL dialect has been extended with functions for Oracle, Microsoft, and Teradata compatibility, and a new, SQL-based user-defined function (UDF) grammar. Stored procedures written in PL/pgSQL are supported. Partitioning is supported. Advanced PostgreSQL functionality such as XML, JSON, geospatial SQL, and other programming languages for server-side programming (Python, Perl, TCL, and so forth) are roadmap items.

Numerous utility commands have been added to the SQL grammar for tasks such as managing the workload management subsystem, controlling elastic clustering, and loading and unloading data from external sources. You can find a full description of Yellowbrick SQL support in the product documentation.

Query Planning

Yellowbrick has replaced, extended, and altered many parts of the PostgreSQL query planner. Query planning is a large and complex subject with entire papers devoted to individual optimizations. Here's a partial list of examples in Yellowbrick:

- MPP awareness, with cost-based estimation of network data exchange.
- MPP plans for all standard primitives (joins, aggregates, sorts, distinct counts, and so on) and data-distribution nodes for hash-distributed, replicated, and randomly distributed data sources.
- Join algorithm replacement, with a new estimator and costing model that uses metadata, histograms, implied equalities, and uniqueness inference.
- Scan selectivity algorithm replacement.
- Support for various forms of correlated subqueries.
- Support for using and combining incremental statistics using big data algorithms.
- Query auto-parameterization to support plan re-use.
- Support for partitioning and adding partition iterators to query plans.
- New data type hierarchy to match other enterprise databases and remove the need for excessive explicit type casting in expressions.
- Late-bound views, wherein views are evaluated when accessed, so database objects referenced by views can be dropped and recreated.
- Support for expression aliases.
- Normalizing in-lists, semi-joins, and OR-lists.
- Support for planning graphs rather than just trees.

- Reimplementation of estimation for aggregates.
- Filter push-down improvements for table scans.
- New EXPLAIN interface.
- Constant folding.
- Static elimination of relational operators and expressions.

Changes to the security model to eliminate "super users" have also affected the query planner to some degree; see the Security, Systems Management, and Monitoring section for details.

Query Processing and Workload Management

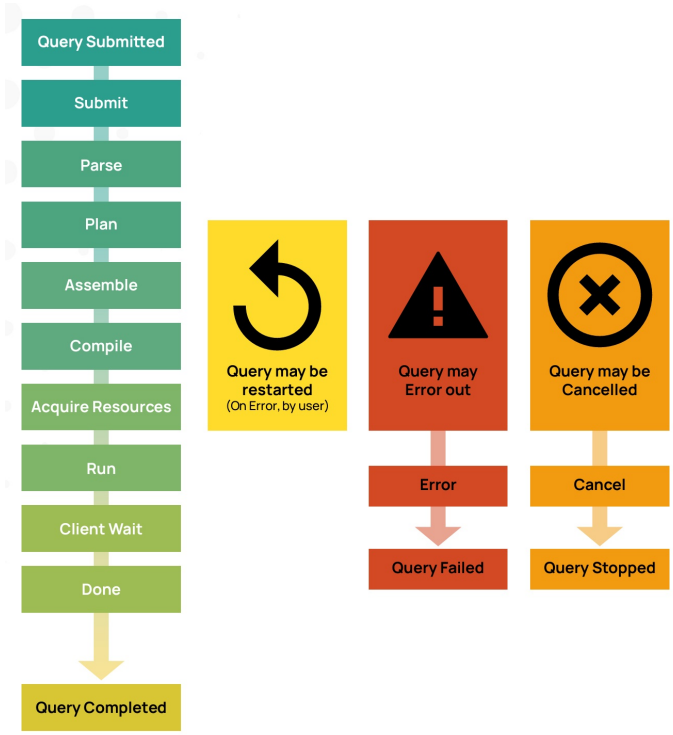
When a user submits a SQL command that involves query processing – henceforth referred to as a "query" – to Yellowbrick via ODBC/JDBC/ADO.NET, the query is passed through several subsystems in its journey through the Yellowbrick database software stack. At the bottom of the stack, the query ends up in the Storage Engine and Execution Engine, at which point it generates results. The results are then passed back up the stack and returned to the user.

During this journey, what starts as a simple piece of SQL text is enriched with more metadata and information. The workload management subsystem is present throughout the lifetime of the query, allowing it to be measured, monitored, routed, prioritized, and adjusted as it travels down and back up the software stack. The adjustments relate to both the query itself, as well as to limits on resource consumption. The Workload Manager allows DBAs and users (with prerequisite permission) to see what's going on as well as to change the nature of the routing to ensure business goals are met. For example:

- Certain business-critical reports may be a higher priority than anything else going on in the system.
- "Bad actor" rogue queries, perhaps written by users with poor working knowledge of SQL, should not affect other users.
- ETL tasks shouldn't consume more resources than necessary.

Query Processing Flow

Each query goes through the state transitions shown in Figure 2. The meanings of these stages are generally self-explanatory. When a query is submitted, little is known about it: its text, who submitted it, which machine and client it came from, and when. After parsing, a query is planned by the query planner, at which time much more information is known: which tables it looks at, its cost, and estimates of resource consumption. It's at this point that any table-level locks necessary to execute the query are acquired. Query fragments are assembled and then compiled to native code by the LLVM compiler infrastructure, at which point the query may wait to acquire the necessary resources (memory, disk, processing power) to execute. The query then runs, at which point it sends results back to the client and is then marked as done. Queries can be canceled or moved into an error condition at any point in the flow.



Monitoring and Introspection

The Workload Manager is the single source of truth for everything executing in the database. It records, for every query in the system, when that query entered a given processing state and how much time it spent there. Through system tables, users can view this information for all queries currently executing in the database as well as previously completed ones.

As one would expect for an enterprise-class database, runtime statistics are measured for every query. Statistics include how much CPU was consumed, how much IO was driven, how many rows were processed, and the sizes of data structures such as hash tables in joins. These statistics are logged on a per-query as well as a per-query-plan-node basis and can be consumed raw in the database or graphically in a web browser via Yellowbrick Manager.

Internal Details

Yellowbrick's Workload Manager can proactively allocate and manage the following system resources:

- Persistent storage space – through disk quotas.
- Spill space.
- Memory.
- CPU – through priorities.

Resource Pools

Each compute cluster's available resources are divided into pools. Queries are routed to a given cluster's pool by rules executing within the Workload Manager (see Rules below). There may be, for example, a small pool for DBAs to make sure they are always able to acquire sufficient resources to kill queries, a pool for short-running, tactical queries, and a pool for long-running ones.

A pool has associated with it a certain level of concurrency it can support, which may be fixed or flexible. Fixed concurrency is useful when it is known that only a certain amount of concurrency is possible or desired, whereas flexible concurrency is a good idea for random queries by data scientists. Queries slow down as they spill more, and it is desirable to have queries run faster when there are fewer users on the system, and slower when there are more users on the system.

Rules

At various points in the execution of a query, Workload Manager rules can introspect what's going on, and perform various actions: cancel the query, route the query to a different pool, prioritize it, or alter its resource consumption. A query can also be throttled according to a named semaphore – for example, to limit every business user to submitting at most two concurrent queries or ensure a misconfigured ETL tool doesn't overload the system with concurrent single-row retrievals.

Workload Manager rules are written in JavaScript, providing incredible flexibility. Properties about the system and each executing query are available via standard JavaScript properties, and actions such as pool assignments, throttling, logging, resource assignments, and recording errors can be accomplished through JavaScript methods. To reduce the runtime overhead of evaluating rules, all the JavaScript rules are compiled into native code for rapid execution.

Some DBAs are not comfortable writing JavaScript, so within the browser-based Yellowbrick Manager is an intuitive point-and-click IF/ELSE rule builder for the Workload Manager. The JavaScript generated by the rule builder can be seen alongside and tweaked if desired. For users who prefer not to use a GUI, a full SQL utility grammar is present for creating, modifying, and controlling workload management rules.

Control Points

Many different types of rules operate at different control points in the system. Typical deployments don't need to implement numerous rules or sophisticated behaviors, but some customers with massive, operational warehouses that support multiple lines of business tend to make maximum use of the flexibility:

- Submit rules are run before queries enter the parser. They allow queries to be rejected by rules that allow matching by SQL text or source IP address.
- Assemble rules are evaluated when the query is building the artifacts required for query compilation before it is submitted for compilation. A common task at this stage is to set the initial priority of the query which helps govern progress and resource usage in later phases.
- Compile rules run during the pre-execution phase for a query, before a resource pool being selected. Rules written for this stage of execution have more information from the resource planner and are commonly created at this phase for pre-execution concerns. Various actions can be taken at this stage, including imposing limits on execution time and memory, setting query priority, or selecting a resource pool.
- Restart rules are evaluated when a query encounters an error condition that allows it to be restarted automatically – for example, to re-run it with more resources or log the event.
- Runtime rules execute periodically during query execution and can introspect the state of the query along with its runtime statistics: how long it's been executing, what resources it's consumed, or the state of its query plan. At runtime, queries that haven't returned data to the user can be transparently moved between pools – perhaps with differing priority or resource limits – or rejected. SNMP traps and alerts can be generated for external systems-management tools. This is how you can be be sure that misestimated, long-running queries don't get in the way of tactical ones, or that "bad" queries are placed in a penalty box such that they don't affect other users.
- Completion rules are evaluated once when execution steps are completed or stopped, due to the query running to completion, being canceled, or erroring out.

Logging capabilities are available to trace rule execution. Rules have priorities and are evaluated in priority order.

Profiles

A container of rules is called a profile; profiles may be imported and exported to/from JSON to move them between instances. Each elastic compute cluster has an active profile that can be changed at any time. If, for example, an insurance company needs to prioritize workloads differently for closing the books at the end of the month, it can switch profiles to enable this only for the days necessary.

Availability and Business Continuity

Yellowbrick has been designed for high availability and a degree of fault tolerance, and with support for backing up and restoring data, replicating data, and storing data on cloud object stores. No third-party tools, services, or "enterprise editions" are needed.

Yellowbrick is designed for data warehousing and analytics rather than online transaction processing, so "five nines" of availability – 99.999% uptime – is not a requirement. Five nines allow only about five minutes of downtime per year and necessitate no maintenance windows, fully online upgrades, and the like. Yellowbrick's architecture is centered around achieving "three-and-a-half nines" per instance – 99.95% availability – allowing about 4.5 hours of downtime per year. This allows time for scheduled downtime for short quarterly

maintenance windows to support software upgrades, along with a small amount of unscheduled downtime due to infrastructure failures or occasional software bugs. Yellowbrick doesn't claim to have mainframe-class or Oracle-class stability and availability, but its track record is excellent: Yellowbrick routinely backs production, online, ad hoc 24x7x365 business-critical financial applications for many Fortune 500 customers with minimal unscheduled downtime.

Yellowbrick's storage and compute can scale elastically; resizing clusters, creating new ones, or adding storage capacity (for on-premises instances only) are all online activities that do not require maintenance windows.

High Availability

Yellowbrick is clustered software, running on multiple server nodes. The software is redundant to component failure, drive failure, and entire node failure with minimum user disruption. Running in the cloud, failed nodes are replaced by new ones as they become available; on-premises, failed nodes are left out of the cluster until a replacement is installed. For more details about on-premises hardware instances, see the "Andromeda Optimized Instances " white paper.

Protection for Data Stored in Cloud Object Stores

For instances running in public clouds, as well as on-premises instances accessing cloud storage, all column store shard files are persisted to cloud object stores. In Amazon, this means S3; in Azure, ADLS; and in GCP, GCS. A fraction of local direct attached SSD storage is used as a block-level (not file level) cache for data persisted in object stores. The block cache is scan-resistant such that large table scans where data is only used once will not evict hot data from the cache. Cache misses result in the given block/s being requested from the cloud object store directly. Due to the massive parallelism needed to achieve good read performance from object stores, table scans will do a substantial read-ahead. For writing table data, shard files are written through to the cloud object store, and the transaction is only committed once the object store has acknowledged the write.

Protection for Data Stored in On-premises Instances

Yellowbrick's cluster filesystem ParityFS protects against data loss due to node failure. It implements n+2 erasure coding, such that two nodes in every parity group can be lost in their entirety due to node failure, or partially due to SSD failure. Data is reconstructed dynamically, and when drives or nodes are replaced, the original data is rebuilt. The difference in query execution performance and throughput is unnoticeable; failure of a node or SSD results in a cluster reconfiguration event which causes momentary service degradation.

Backup and Restore

The Yellowbrick Storage Engine can take transactionally consistent snapshots of all the data in a database at any time. These snapshots are low overhead, leverage the transactional nature of the database, and are completed in a fraction of a second. The snapshot metadata itself occupies very little storage; however, active snapshots impose constraints on garbage collection such that the system will use more storage space until the snapshots are dropped.

The snapshots are per-database and include all tables, including system catalog tables: Those that store the database schema objects such as tables, columns, and constraints; as well as roles, workload management rules, and other system configurations. In the roadmap, the snapshot functionality will be exposed to support "time travel" queries for general SQL use.

SQL-native Backup and Restore

Backup and restore operations are implemented as SQL operations to leverage the high throughput of the Yellowbrick database engine. By performing delta queries against transaction snapshots, the system can select only the rows that have been added, updated, or deleted between two transactions for backup. Changed rows are then compressed in parallel on all nodes for increased performance; backups are not simply rigid copies of files in the filesystem.

Restore is also implemented as a SQL operation and is analogous to bulk data loading. All nodes in the cluster receive backup data. It is decompressed and decoded in parallel on all workers and new data/changed data is written to the Yellowbrick Storage Engine.

Table Delete Horizon

The history of different types of database backups – full, incremental, or cumulative – together forms the backup chain for that database. Each chain is a logical grouping of snapshots, each representing a discrete point in time. As a result, each backup chain has an implicit "delete horizon," a transactionally consistent point in time for the database after which deleted space cannot be fully reclaimed. Deletes made after the horizon point in time leave behind "tombstone" markers in the storage engine that occupy tens of bytes per row. Backups move the delete horizon forward in a way such that deltas of previous backups can still be taken.

Types of Backup and Restore

Backups work one database at a time. Each backup requires the creation of a new snapshot. Yellowbrick supports three types of backups:

- **Full backups** are complete backups of an entire database. A full backup is typically done once, to initiate a new backup chain and rarely thereafter. Full backups tend to be very large and can be thought of as a set of all changes between the first transaction and the backup transaction snapshot.
- **Cumulative backups** capture all the changes since the last cumulative or full backup snapshot, whichever is more recent, and advance the table delete horizon.
- **Incremental backups** capture all the changes since the last incremental, cumulative, or full backups, whichever is more recent, but do not advance the table delete horizon.

Yellowbrick also supports incremental restores so that deltas captured by an incremental or cumulative backup can be applied without having to restore the entire database. For the data and schema, this is straightforward; for other parts of the system catalog, various merge strategies are used. More advanced backup strategies are possible but have not been implemented.

Readable Replicas

A hot standby database is a database that is in read-only mode while receiving continuously applied incremental restores. Yellowbrick allows hot standby databases to be queried and used, including the creation and deletion of temporary tables as needed by reporting and BI tools. In addition, a database can be placed in a purely read-only mode to "freeze" an environment.

Asynchronous Replication for Disaster Recovery (DR)

Yellowbrick contains full support for unidirectional asynchronous replication, to be used for establishing a DR site. Replication can be between on-premises and cloud instances, and across cloud vendors, as desired. No third-party utilities are required for replication. Both DDL and data are replicated. Replication takes place over TLS-secured TCP sockets and is interruptible: In case of socket failure, the replication process will pick up roughly where it left off and continue from there. The target databases for replication must be hot standby databases, enabling them to be actively used for queries while receiving writes.

The replication process does place some additional query load on both databases. Replication is transactionally consistent, with data written to the target in one transaction to guarantee consistency for users. Where possible, an initial replication target should be seeded using backup-and-restore functionality. Replication is configured, managed, and monitored through full SQL utility grammar and system tables. Currently, replication is supported from one source database to one target database. Multi-target support is not currently productized.

Failover and Fail-back

In Yellowbrick, failover is not automated because it is considered a rare event. Individual instances are highly available and protected, so failover is only necessary in case of a mass loss of connectivity or a true natural disaster. The database will ensure that no split-brain scenario has occurred whenever replication takes place by checking transaction sequences and database configuration between the source and target. If inconsistency is detected, manual intervention is required. Fail-back after failover is fully supported but note that bi-directional replication, where both databases are accepting changes, is not supported.

High-throughput, Parallel Data Movement

Data warehouses need to be able to ingest and unload large amounts of data rapidly. Sometimes data ingest and extraction needs to be done in batch mode with as much throughput as possible; whereas other times, the data movement needs to be done in a streaming fashion for real-time queries. Sometimes the data movement is best initiated through SQL. Sometimes it’s taken care of by modern pipelines built with Kafka and/or Spark. And in traditional deployments, it’s initiated as part of a complex set of scripts involving ETL tools. Yellowbrick caters to all these scenarios.

Bulk Data Load and Unload

Yellowbrick contains an efficient binary protocol for parallel bulk loading and unloading. Data is streamed to or from the database in parallel, across multiple network sockets (typically one socket per node in the cluster). The bulk protocol is row-oriented and supports compression.

These protocols are designed for batch operation. Large unloads complete in one shot, and large loads will commit a few massive transactions periodically. Format transformations such as parsing and formatting are done by the sender (for loads) and receiver (for unloads) to reduce load on the database. Bulk operations pump data directly from/to the Yellowbrick Storage Engine. These operations typically are bound by the speed of the network. The ways of initiating bulk data movement with Yellowbrick are:

- Through SQL: Using the built-in utility statements such as CREATE EXTERNAL STORAGE, CREATE EXTERNAL FORMAT, CREATE EXTERNAL LOCATION and LOAD TABLE.
- Through pre-existing integrations with products such as Kafka, Spark, Informatica, Oracle Golden Gate, etc.
- Through traditional cross-platform client tools that run on everything from Windows to AIX: ybload and ybunload.
- Through Java integrations with a Yellowbrick-provided data loading library.

Progress of loads and unloads, regardless of whether they are initiated using client tools or SQL, can be monitored through system views. A variety of file formats are supported including Apache Parquet as well as traditional delimited data and BCP files. Yellowbrick Manager also enables browsing and loading of data through a web browser.

Streaming Data Movement

Because bulk data is committed in large chunks, and socket negotiation across clusters is required to initiate the process, it's an inefficient way to load and unload small numbers of rows. For small unloads, the ODBC/JDBC/ADO.NET (henceforth "*DBC") drivers will suffice, and PostgreSQL's built-in \copy command is a good shortcut.

For loading small numbers of rows in a streaming fashion, you can use *DBC INSERT statements, or alternatively, \copy. With a few parallel clients, when loading data directly into the row-oriented store, rates of several million rows/sec are achievable. Unlike bulk loads, which are committed in huge batches of hundreds of millions of rows, row-store transactions can be committed frequently – even once per row, if desired.

Unlike other data warehouses that struggle to do streaming ingest efficiently, the hybrid nature of the Yellowbrick Storage Engine allows it to easily ingest large numbers of small transactions and query the most recent data, along with historical data, in a transactionally consistent fashion. This is ideal for integrations with CDC tools (Oracle Golden Gate, HVR) or an enterprise message bus (Kafka). There's no need to worry about micro-batching or writing custom code to combine reporting from transactional and analytic databases.

Concurrent Loading and Querying

Yellowbrick can efficiently query tables as data is being written to them. Large bulk loads or data merges running in parallel will not alter query performance in any unexpected ways. Consequently, it's possible to have databases with a very high level of churn that can handle large numbers of concurrent, ad hoc queries at the same time. Some Yellowbrick customers have databases hundreds of terabytes in size where each day 30% of the data continuously changes.

Security, Systems Management, and Observability

Enterprise-class databases must be secure, and easy to manage, monitor, and configure. Yellowbrick has a variety of mechanisms that cater to the needs of enterprise customers running in private and public clouds, in both HIPAA-compliant and regular deployments.

Security

Yellowbrick is built assuming everything is "private by default." Default access controls to areas of the product are limited. Private S3 buckets are enabled by default, with no public access to data. No built-in guest user accounts are present, and strict access must be granted to all data and management functionality. Because the Yellowbrick Data Warehouse runs in a customer's own cloud account, Yellowbrick by default has no access to customer data, customer queries, or logs. Companies have full control of who has access and what they can access.

Authentication

Yellowbrick's authentication is based on OpenID Connect (OIDC) and supports authentication with all OIDC-compliant identity providers such as Azure Active Directory (Office365), Okta, and Ping. OIDC authentication allows companies to continue to maintain identities and roles in their existing identity providers making the management of users easier to maintain. On-premises Yellowbrick instances can authenticate with LDAP and Active Directory, also synchronization of roles between external identity providers and the database is supported. For business-critical activities, the Yellowbrick database supports local user authentication to allow operations to run in case connectivity to external authentication is unavailable.

Manageability Without "Super Users"

PostgreSQL relies on having a "super user" for administration and regular users for everything else. The super user can administer anything whatsoever, much like the "root" user on a Unix system. Although the front end of the database is based on PostgreSQL, privileges afforded to the super user have been deliberately split into dozens of different grants to allow users to manage subsections of the database in a far more fine-grained manner – for example, the ability of a role to manage other roles, view SQL query text of other users, initiate backups, control LDAP integrations, and even the ability to grant privileges themselves all can be granted or revoked individually.

Role-based Access Control

Role-based access control allows mapping roles to specific database access controls. Roles are granted and revoked using already existing identity provider management tools, such as adding a role to a user in Active Directory. Access to all database schema objects is fully role-based. Roles can be used to grant access down to column granularity. PostgreSQL administrators will be familiar with this concept.

Encryption of Data at Rest

Encryption of data on cloud object stores is managed by the object store itself and ephemeral cloud storage is also encrypted and crypto-erased. Yellowbrick on-premises instances store all data fully encrypted with AES-256 using keys stored in HashiCorp Vault.

Column-level Encryption and Functions

Yellowbrick provides a variety of SQL functions for data encryption, decryption, and hashing using several algorithms. Individual VARCHAR columns within tables can be designated as encrypted so that Yellowbrick will encrypt the data for the column as it’s inserted. When a user with access to the corresponding encryption key executes a query, they will see the decrypted data; however, users without access to the encryption key will see only the encrypted, scrambled data. Keys are stored in and referenced from the built-in HashiCorp Vault.

Yellowbrick has a production-quality integration with partner Protegrity which provides sophisticated, policy-based data protection and masking functionality that goes beyond Yellowbrick column-level encryption.

TLS Support

All communication over the wire is encrypted. TLS 1.2 is required for all traffic in and out of Yellowbrick. This is true for *DBC access and web access and all external connectivity, loading, unloading, backup, and so on. TLS mutual authentication is used for authentication of cross-database replication sessions.

Observability

Instrumentation and statistics gathering in the database are visible through system views. Several additional integrations are available to allow the database to be observed, monitored, and alerted upon in customer environments.

Instance Observability

Yellowbrick Manager includes an observability stack consisting of Prometheus, Loki, and Grafana. These cloud-native technologies are used to aggregate logs from the database, delivered via FluentBit, and raise alerts in case of issues being found. Prometheus alertmanager is used to deliver alerts to such typical receivers as Slack, Opsgenie, PagerDuty, etc. On-premises installations of Yellowbrick are also able to deliver alerts via SNMP and logs through remote syslog.

User-configured alerts can be raised by the Workload Manager and are raised automatically when user disk quotas are exceeded. Other alerts pertain to unexpected errors in the database, such as software crashes or replication falling behind.

Remote "Phone Home" Support

Yellowbrick is designed with a SaaS user experience in mind, regardless of whether the database is running in a private data center or the public cloud. Yellowbrick optionally maintains remote, unidirectional connectivity to Yellowbrick’s internal SaaS monitoring platform. Any crash mini-dumps, key telemetry, and key logs are sent back over this phone-home connection to enable Yellowbrick’s Customer Success team to monitor customer systems 24/7. In the event of any issues, Yellowbrick will know about them first. Different levels of data scrubbing guard personal identifiable information (PII) and other confidential data. No customer data is ever shared. Customers with strict security requirements can disable Phone Home completely.

Compute Cluster Load Balancer

The compute cluster load balancer enhances the flexibility and performance of compute cluster management. This feature allows users or roles to distribute workloads across multiple compute clusters, ensuring optimized resource utilization and availability.

Overview

The cluster load balancer enables users or roles to define a set of compute clusters to be used for query execution. The specific compute cluster chosen for a query depends on various factors, including:

1. **Transactional State:** If the current session has written data in its transaction, the cluster load balancer setting is ignored. Instead, the compute cluster where the data was written is used to maintain data residency.
2. **Least Busy Heuristic:** If no data has been written in the current transaction, the system chooses the compute cluster with the fewest running and queued queries. If two or more clusters have the same level of "least busy," one of them is selected at random to ensure fairness.
3. **Cluster Availability:** Clusters that are offline or suspended are excluded from consideration, even if they are part of the configured list.

Users can configure their default cluster behavior with a comma-separated list of clusters using the `ALTER USER` or `ALTER ROLE` commands:

```sql
ALTER USER <username> SET DEFAULT_CLUSTER "<cluster1>,<cluster2>,<cluster3>";
ALTER ROLE <rolename> SET DEFAULT_CLUSTER "<cluster1>,<cluster2>,<cluster3>";
```

Scaling Out Clusters

The cluster load balancer also allows users to scale out the number of clusters they have access to, distributing workloads across them. This capability is particularly beneficial for zero-downtime scaling and for workloads that require higher concurrency rather than more compute, memory, or storage per query. By assigning a user multiple default clusters, the instance automatically directs queries to the least busy cluster. Additionally, this feature supports seamless suspension and resumption of clusters, providing greater elasticity.

Example:

Users with explicit grants on clusters can add new clusters to their default cluster settings:

```sql
GRANT USAGE ON CLUSTER "bi_cluster2" TO john_doe;
ALTER USER "john_doe" SET DEFAULT_CLUSTER "bi_cluster,bi_cluster2";
```

In this example, the user has access to multiple clusters, enabling scalable and elastic query processing without downtime. For example, adding `bi_cluster2` increases concurrency for workloads while avoiding minimal distribution penalties.

Important: Users or roles assigned a load balancer cluster setting must also have `GRANT USAGE ON CLUSTER <name>` privileges for each cluster listed.

Key Concepts

1. **Dynamic Cluster Selection:** The compute cluster is dynamically selected based on the "least busy" heuristic or transactional state.
2. **Transactional Data Residency:** When a transaction includes temporary data, the session remains on the compute cluster where the data was written until the transaction is committed.
3. **Fair Scheduling:** If multiple clusters have equal load, one is randomly chosen to balance query distribution.
4. **Offline or Suspended Clusters:** Compute clusters that are offline or suspended are not considered during query scheduling.
5. **Cluster Name Length Limitation:** The cluster name setting for a role must fit within 128 characters. This limitation restricts the number of concatenated cluster names that can be included in the load balancer setting. To maximize the number of clusters, use short and descriptive names for clusters.

Viewing Default Cluster Settings

To view the current default cluster setting for a user or role, query the `sys.users` or `sys.roles` system tables:

```sql
SELECT default_cluster FROM sys.users WHERE username = 'john_doe';
SELECT default_cluster FROM sys.roles WHERE rolename = 'data_scientists';
```

Interaction with `USE CLUSTER`

The `USE CLUSTER` command can override the default cluster or cluster list for the current session. However, when the session ends, the user or role reverts to their default cluster list setting.

Best Practices

- **List Maintenance:** Ensure the list of clusters is kept up-to-date to avoid routing workloads to retired or decommissioned clusters.
- **Uniform Configuration:** Use the same WLM profile and cluster configuration, including hardware instance type and node count, to achieve predictable workload performance across load balanced compute clusters.
- **Workload Optimization:** Configure cluster lists to prioritize clusters with the appropriate capacity and proximity for intended workloads.
- **Testing and Validation:** Simulate various load and failover scenarios to validate the behavior of the load balancer.
- **Short Cluster Names:** Use short and descriptive names for compute clusters to maximize the number of clusters that can be included in the load balancer setting.
- **Grant Privileges:** Ensure all users or roles assigned a cluster load balancer setting have the necessary `GRANT USAGE ON CLUSTER <name>` privileges.

Further Reading

Refer to the Compute Clusters Documentation for foundational concepts related to compute clusters in Yellowbrick.

See also ALTER USER SET DEFAULT_CLUSTER.

Reserved Nodes

Platforms: EE: All cloud platforms

Parent topic: Cloud

Sometimes certain server instance types in public clouds make take time to acquire; creating a new compute cluster or altering an existing one to add nodes can take a long time, and ultimately fail if instances of the requested type are unavailable.

Reserved Nodes is functionality to allow Yellowbrick to acquire server instances in the background. Reserved Nodes are shared among all the compute clusters that are created for a single Yellowbrick instance, and they are immediately available for use when those clusters start running and doing work. You'll need to pay your cloud vendor for these reserved nodes, but Yellowbrick won't bill you for them unless they are part of a running compute cluster. This approach to reserving nodes is driven by Kubernetes auto-scaling, which results in the cloud vendor spinning up nodes even before any compute clusters have been created.

An alternate approach to solving capacity problems is to use on-demand capacity reservations (ODCRs) available from the cloud vendor. ODCRs typically belong to an account and have a global application, whereas Reserved Nodes are specific to the Yellowbrick platform. While Reserved Nodes don't require any extra IAM role permissions from the cloud vendor, ODCRs may be easier to automate using standard tooling, for example to acquire extra nodes ahead of an end-of-quarter reporting period.

To work with ODCRs, follow the cloud vendor's documentation.

To work with Reserved Nodes, in Yellowbrick Manager navigate to *Configuration → Instance Management → Reservations*.

The **Reserved** field specifies the number of reserved nodes requested. If empty, no background reservations are acquired. The **Limit** field sets a maximum number of compute cluster nodes the instance in question can consume in total. The limit per instance must be greater than or equal to the number of reserved nodes.

Sample Data

Yellowbrick Documentation > Concepts > Cloud > Sample Data

Platforms: EE: All cloud platforms

Parent topic: Cloud

Yellowbrick maintains some sample data sets (source files, DDL, and queries) in GitHub at YmSamples. The DDL scripts in that repository reference AWS S3 files in a public bucket. You can use either the Yellowbrick Manager Load Assistant or SQL commands to load the sample data.

Some of the data sets include:

- `gdelt` : The GDELT Project is the largest, most comprehensive, and highest resolution open database of human society ever created. Just the 2015 data alone records nearly three quarters of a trillion emotional snapshots and more than 1.5 billion location references, while its total archives span more than 215 years, making it one of the largest open-access spatio-temporal data sets in existence and pushing the boundaries of "big data" study of global human society.
- `NOAA GHCN-D` : Weather observations for over 200 years collected from a large number of land-based weather stations. The source files are provided by the Registry of Open Data on AWS (RODA).
- `nyc-taxi` : Yellow and green taxi trip records include fields capturing pick-up and drop-off dates/times, pick-up and drop-off locations, trip distances, itemized fares, rate types, payment types, and driver-reported passenger counts.
- `premdb` : A tiny database that contains actual English Premier League soccer results for about 20 seasons, as well as a few details about each team in the league. You can run a script that creates and loads five small tables in about 5 seconds. These tables are used extensively in the main Yellowbrick documentation to provide simple, reproducible examples of SQL commands and functions.
- `tpcds_sf1000` : A 1TB version of the TPC-DS data set, which is frequently used by database companies for competitive benchmarking. This data set is pre-loaded into the `yellowbrick_trial` database. Scripts are available for re-creating and loading these tables.
- `NetFlow` : a data set generated by a service that sits on network routers and collects information on network traffic as it enters or exits an interface. This information includes source and destination addresses, ports, protocols, and the amount of data transmitted. By analyzing NetFlow data, a network administrator can summarize network activity, find sources of network congestion, and identify potential security threats.

Note: The YmSamples site and the referenced S3 bucket are both public. You do not need an AWS account to access the data.

VPC Peering vs. Network Transit Solutions in Hub/Spoke Architectures Across Cloud Providers

Platforms: EE: All cloud platforms

Parent topic: Cloud

Overview

VPC Peering (or its equivalent in different cloud platforms) is a networking connection between two Virtual Private Clouds (VPCs) or similar constructs that enables private communication between them. This allows resources in one network to access resources in another as if they were within the same network, without using the public internet, VPNs, or gateways. A common use case for VPC Peering is within a **hub/spoke architecture**, where multiple spokes (networks) connect to a centralized hub for improved network efficiency and security.

Hub/Spoke Architecture Across Cloud Platforms

In a hub/spoke model:

- The **hub** is a central network that connects multiple **spoke** networks, facilitating communication and centralized management.
- Each **spoke** hosts specific workloads or applications that need access to shared services in the hub.

Multi-Cloud Considerations

- **AWS**: Uses **VPC Peering** and **AWS Transit Gateway** for network interconnectivity.
- **Azure**: Uses **VNet Peering** and **Azure Virtual WAN** for centralized management.
- **Google Cloud (GCP)**: Uses **VPC Peering** and **Cloud Interconnect** for scalable networking.

For example, a **Yellowbrick SQL Database** can reside on a **spoke network**, while a **BI tool such as Tableau** is hosted in another peered spoke. This setup enables **secure, low-latency access** between analytics tools and data storage without traversing the public internet.

Benefits of Hub/Spoke with VPC Peering

- **Improved Security**: Traffic stays within the cloud provider's private network, avoiding exposure to public internet risks.
- **Centralized Network Management**: The hub provides a single point of control, simplifying routing and access control.
- **Reduced Latency**: Peering enables direct VPC-to-VPC (or VNet-to-VNet) communication, minimizing data transfer delays.
- **Efficient Data Processing**: Spoke networks can handle specialized workloads, such as SQL analytics, without impacting other workloads.

How VPC Peering Works in a Hub/Spoke Model

1. **Establish Peering Connections**: The Yellowbrick database network and the BI application network establish peering connections with the central hub.
2. **Configure Route Tables**: The hub network's route table allows traffic between the spoke networks.
3. **Apply Security Rules**: Network ACLs and security groups allow necessary traffic between the BI tool and the database.

Peering Limitations Across Cloud Providers

- **Non-Transitive Routing**: Peered networks do not automatically share access with other peers. The hub must explicitly connect each spoke.
- **CIDR Overlap Restrictions**: Spoke networks must have unique, non-overlapping IP address ranges.
- **Cross-Region Latency Considerations**: Peering across cloud regions may introduce additional latency.

Comparison: Peering vs. Network Transit Solutions in Hub/Spoke

Feature	Peering Solutions (AWS VPC Peering, Azure VNet Peering, GCP VPC Peering)	Network Transit Solutions (AWS Transit Gateway, Azure Virtual
Connectivity Type	Direct connection between two networks	Centralized hub for multiple networks
Transitive Routing	Not supported	Supported
Scalability	Limited to individual network connections	Highly scalable, supports multiple networks
Cross-Region Support	Requires separate peering connections	Native support for cross-region communication
Multi-Cloud Support	No direct support	Requires VPN or third-party transit solutions
Complexity	Simple to set up for a few networks	More complex, but better suited for large-scale architectures
Cost	Lower cost as it only involves standard cloud data transfer fees	Higher cost due to additional transit fees
Best For	Direct, low-latency communication between specific networks	Large-scale, multi-network environments needing centralized control

---

Use Case Example

A company may set up a **hub/spoke architecture** where:

- **Hub Network**: Centralized networking and security services (e.g., AWS Transit Gateway, Azure Virtual WAN, GCP Cloud Interconnect).
- **Spoke 1**: Yellowbrick SQL Database for high-performance analytics.
- **Spoke 2**: Tableau or other BI tools that query the database.
- **Spoke 3**: Custom data application or front-end.

With **Peering Solutions**, Tableau in **Spoke 2** can securely retrieve data from Yellowbrick in **Spoke 1**, improving efficiency without exposing traffic to the public internet.

Alternatives to Peering Solutions

- **AWS Transit Gateway / Azure Virtual WAN / GCP Cloud Interconnect**: More scalable for complex hub/spoke architectures.
- **PrivateLink (AWS) / Private Endpoint (Azure) / Private Service Connect (GCP)**: Useful for securely accessing cloud services or SaaS applications.
- **VPN or Direct Connect**: Required for hybrid cloud models integrating on-premise infrastructure with multiple cloud providers.

Conclusion

Peering Solutions provide a simple and cost-effective way to enable private, low-latency communication between networks in a **hub/spoke architecture**. However, for large-scale networking needs, **Transit Solutions (AWS Transit Gateway, Azure Virtual WAN, GCP Cloud Interconnect)** offer more flexibility and scalability.

For a Step by Step Tutorial

- AWS VPC Peering Tutorial

For configuration guides:

- Configuring AWS VPC Peering
- Configuring AWS Transit Gateway Peering

For more details, refer to the official documentation of each cloud provider:

- AWS VPC Peering
- Azure VNet Peering
- Google Cloud VPC Peering

How-to Guides

Yellowbrick Documentation > How-to Guides

Platforms: All platforms

Parent topic: Yellowbrick Documentation

In this section:

- Get Started
- Create Tables
- Load Tables
- Run Queries
- Unload Tables
- Work with Tools & Drivers
- Administer Yellowbrick
- Install Airbyte

Here you'll find answers to "How do I….?" types of questions. These guides are *goal-oriented* and *concrete*; they're meant to help you complete a specific task. For conceptual explanations see the Conceptual guide. For end-to-end walkthroughs see Tutorials. For comprehensive descriptions of every command and function see the Reference guide.

How to Get Started with Enterprise Edition in Cloud

Yellowbrick Documentation > How-to Guides > Get Started

Platforms: EE: All cloud platforms

Parent topic: How-to Guides

After installing Yellowbrick, the easiest way to get going is to use Yellowbrick Manager. It provides both instance administration functionality as well as a SQL query editor.

Yellowbrick Manager contains basic tools for instance administration, provisioning, supports the current version and the last two public releases of the following browsers:

- Apple Safari
- Google Chrome
- Chromium-based browsers (such as Microsoft Edge or Opera)
- Mozilla Firefox

To use Yellowbrick Manager, open the URL provided at the end of the installation process and log in with the instance administrator password you provided during installation. See the walkthrough tour of Yellowbrick Manager to help get started.

WARNING

Before loading confidential data or doing business-critical work, you will want to set up single sign-on (SSO).

Using Third-Party Tools

Alternatively you can download third party tools to connect using JDBC and ODBC drivers for PostgreSQL. A set of Yellowbrick-specific command line tools called `ybtools` supports tasks such as bulk data load/unload and database backup/restore.

Creating and Managing Tables and Views

Yellowbrick Documentation > How-to Guides > Create Tables

Platforms: All platforms

Parent topic: How-to Guides

In this section:

- Distributing Data
- Auto-Analyzing Tables
- Disk Quotas
- External Tables
- Generating Values with Sequences
- Ordering Data
- Partitioning Tables
- System Columns in Tables

After creating a database and (optionally) schemas within it, you can create tables and views. You can also create tables and views within the default `yellowbrick` database.

Database Tables

Yellowbrick supports persistent tables and temporary tables.

- **Persistent tables** remain in the database until they are explicitly dropped.
- **Temporary tables** are *local* to the session in which they are created and are dropped when the session ends.

You can use CREATE TABLE, CTAS, and SELECT INTO statements to create tables.

Table Maintenance

Aside from loading tables, you may need to maintain them in a few other ways:

Altering and Dropping Tables

Yellowbrick supports basic ALTER TABLE and DROP TABLE commands. You can add columns, rename a table, and change the owner of the table.

You can also GRANT and REVOKE privileges at the table and column level.

Inserting, Updating, and Deleting Rows

Yellowbrick supports standard SQL INSERT, UPDATE, and DELETE commands.

You can also run the `\copy` command to load smaller tables from files via `ybsql`. In general, you should use the bulk loader (`ybload`) to load large tables.

Note: The Yellowbrick column store is designed to optimally store, compress, and organize data for high-volume read-oriented analytics. As a result, when data is deleted it continues to exist in the system for a period of time until it can be efficiently removed and data can be reorganized. These optimizations are fully automatic and require no administration or manual interaction.

Analyzing Tables

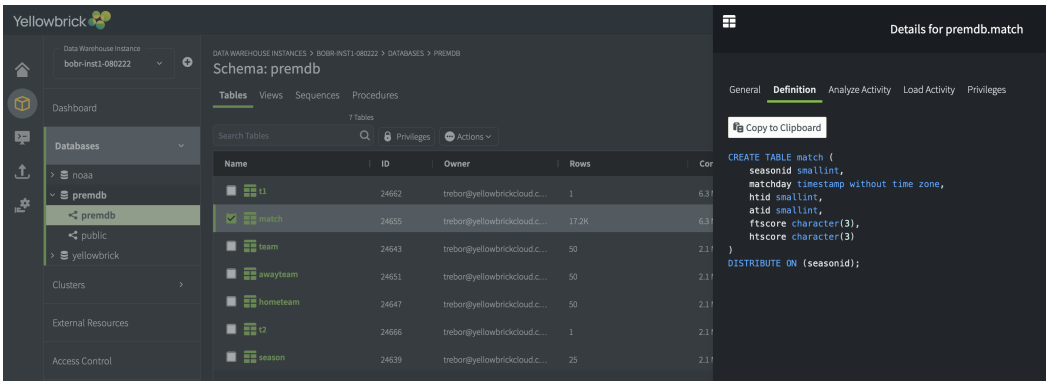
By default, tables are automatically analyzed. This analysis updates column-level statistics that the planner uses to optimize query execution. See Auto-Analyzing Tables.

Flushing Tables

A background flush command periodically updates the Yellowbrick column store with rows that are temporarily stored in the row store. See Managing the Row Store.

Table DDL

You can use Yellowbrick Manager to copy the DDL for a table. Go to **Databases** and select the database and schema. Then click the table name and go to **Details** > **Definition** > **Copy to Clipboard**. For example:



See also Distribution Options.

User-Defined Views

You can create a view based on any valid Yellowbrick query, and you can reference other views in the view definition. Views are not materialized. See CREATE VIEW for syntax details.

Yellowbrick views are persistent objects. They are not dependent on the tables that they reference; therefore, you can drop or rename a table and its associated views persist in the database. For example, you can create table `team`, then create view `teamview` by selecting from `team`. If you drop `team` with `CASCADE`, the view remains in the database. If table `team` is re-created, you can still select from `teamview` if the columns that originally existed in the table also exist in the new instance of the table and have the same data types.

For example:

```
premdb=# create view htv as select * from hometeam;
CREATE VIEW
premdb=# \d htv
           View "public.htv"
 Column |         Type          | Modifiers
--------+-----------------------+-----------
 htid   | smallint              |
 name   | character varying(30) |

premdb=# drop table hometeam cascade;
DROP TABLE
premdb=# select * from htv;
ERROR:  relation "public.hometeam" does not exist
premdb=# create table hometeam (htid smallint, name varchar(30), stadium varchar(50));
CREATE TABLE
premdb=# insert into hometeam values (1, 'Leicester City', 'King Power Stadium');
INSERT 0 1
premdb=# select * from htv;
 htid |      name
------+----------------
    1 | Leicester City
(1 row)
```

Note that a `select *` query against a view whose table was dropped and re-created will only return data for the set of columns that were defined in the original instance of the table. The `select *` view definition is expanded to the specific columns that are in the table when the view is created:

```
premdb=# \d+ htv
                  View "public.htv"
 Column |         Type          | Modifiers | Description
--------+-----------------------+-----------+-------------
 htid   | smallint              |           |
 name   | character varying(30) |           |
View definition:
 SELECT hometeam.htid,
    hometeam.name
    FROM hometeam;
```

System Views

System views contain metadata and statistics for various aspects of the system. You can query these views, but you cannot modify or drop them. System views are based on a set of underlying system tables, but these tables are not intended for general use. See System Views.

Distributing Data

Yellowbrick Documentation > How-to Guides > Create Tables > Distributing Data

Platforms: All platforms

Parent topic: Create Tables

In this section:

Best Practices

When you are creating tables, it is important to define the data distribution scheme for the table, which determines how the rows are distributed across the compute nodes. Good data distribution is critical to optimize parallel processing for queries and other database operations.

Within a `CREATE TABLE` statement, you can define a single-column distribution key and distribute the data by hashing on the values in that column. Alternatively, you can set the distribution to `RANDOM` or `REPLICATE`.

DISTRIBUTE ON (column)

Hash distribution across the compute nodes based on the values in the specified column. This option is recommended for most tables. For example:

```
create table team
(teamid smallint, htid smallint, atid smallint, name varchar(30), nickname varchar(20), city varchar(20), stadium varchar(50)
distribute on (teamid);
```

DISTRIBUTE REPLICATE

Replication of the entire table across all blades. This option is intended for smaller tables. For example:

```
create table season
(seasonid smallint, season_name character(9), numteams smallint, winners varchar(30))
distribute replicate;
```

DISTRIBUTE RANDOM

Random distribution of table rows.

If you do not specify the `DISTRIBUTE` clause, the table is hash-distributed on the first named column in the table.

Data Distribution for CREATE TABLE AS Results

If you do not specify a distribution type for a CTAS table, the resulting data distribution depends on the nature of the query that creates the table.

- A table created from columns in one or more replicated tables is also replicated.
- A table created from a single hash-distributed table is typically hash-distributed on the same column when the distribution column is included in the select list. If the distribution column is not included in the select list, random distribution is used.
- Tables resulting from equality joins over distribution columns typically preserve the hash distribution.
- In general, hash distribution is preserved where possible, but tables created from complex joins or that contain complex select list expressions may produce randomly distributed result sets.

Data Skew with NULLs and NaNs in Distribution Columns

When the distribution column of a table contains a significant number of `NULL` values or `NaN` values, the data may not be distributed evenly. Extreme data skew on the compute nodes may occur. In general, it is best to avoid distributing a table on any column that is nullable or a floating-point column that it is likely to contain a large number of `NaN` values.

For more details about `NaN`, see DOUBLE PRECISION.

Data Distribution Best Practices

Platforms: All platforms

Parent topic: Distributing Data

Introduction

This document provides a guide to designing effective table distributions in Yellowbrick. By following these examples, you can address common challenges such as data skew and workload imbalances, optimizing query execution and resource utilization. The focus is on aligning table distribution with workload patterns to reduce runtime data redistribution and enhance performance.

Read Data Distribution Concepts to understand why data distribution matters in Yellowbrick.

Before You Begin

Effective planning is essential when designing table distributions in Yellowbrick. Before selecting a distribution key, it is important to address common misconceptions and evaluate workload characteristics.

INFO

If a table is already loaded in *Yellowbrick* and requires design optimization, use tools like *YbEasyCli* to query table and column metadata. These tools provide insights into key attributes, including distribution patterns and column characteristics, to help refine table distribution strategies.

Avoid Common Misconceptions

Proper understanding of table distribution in Yellowbrick is critical to avoid common misconceptions that can result in suboptimal performance. The following sections address two key misunderstandings:

Misconception 1: Even Distribution of Data is the Most Important Criteria

SQL workload characteristics determine the effectiveness of a distribution strategy, not just the uniformity of data distribution.

Characteristics	Description
Common Join and Grouping Columns	Distribution should align with frequently joined or grouped columns to minimize redistribution.
Cardinality of Key Columns	High cardinality columns are preferred to ensure balanced data distribution.
Filtering Predicates	Predicates in queries should align with the distribution key to improve data locality.

Uniformly distributed data alone does not guarantee optimal performance. Redistribution during query execution may negate the benefits of uniform storage if the distribution scheme does not align with workload patterns.

Misconception 2: Distribution Only Matters for Large Tables

Small tables are often overlooked during table design, but in reality, distribution on small tables can have a significant performance impact on join order and redistribution in execution plans.

Distribution should be explicitly set for both large and small tables.

Evaluate Your Workload

When determining a table's distribution, the goal is to achieve balanced data and workload distribution across compute nodes while minimizing data redistribution during query execution. A well-planned distribution enhances both performance and resource utilization.

To optimize table distribution, focus on the following principles:

Principle	Description
Avoid Execution Skew	Avoid situations causing "execution skew," where some compute nodes process disproportionately more data than others. A step in a process will only be as fast as that process on the slowest compute node.
Avoid At-Rest Data Skew	Prevent "at-rest data skew," where one or more compute nodes store significantly more data than others. In an MPP system, storage is limited by the compute node with the least available space.
Minimize Runtime Redistribution	Reduce movement of data between compute nodes during statement execution to minimize I/O and performance costs. At-rest distribution is the most common source of runtime redistribution.

Gather the following details about your tables to design an effective distribution strategy:

Consideration	Guidance
Table Type	Determine if the table is a *fact table* or a *dimension table*.
Size	Assess the number of rows and the uncompressed size of the table to gauge its impact on storage and performance.
For Fact Tables	1. Identify frequent join columns, focusing on those commonly used in joins with other fact tables or large dimension tables. 2. Focus on primary and foreign keys relationships in the data. 3. Evaluate the cardinality and nullability of key columns to avoid skew caused by low cardinality or `NULL` values. For a detailed discussion, refer to the section *Distributing by Fact Tables*.

Distributing Small and Frequently Used Dimension Tables with DISTRIBUTE REPLICATE

The `DISTRIBUTE REPLICATE` method, in the use cases below, ensures that all compute nodes maintain a complete copy of the table, eliminating the need for redistributions during query execution.

Use Cases

Use Case	Description
Small Dimension Tables	Suitable for tables under 1 GB (uncompressed) that are used in joins or `GROUP BY` operations.
Larger Dimension Tables	Applicable for tables larger than 1 GB (uncompressed) but frequently joined on different columns. This method should be used cautiously for larger tables, as it increases storage requirements and may impact scalability.

Advantages

- Minimizes Data Redistribution: `DISTRIBUTE REPLICATE` eliminates the need for data movement across compute nodes, reducing query overhead by ensuring all nodes have a complete copy of the table.
- Efficient for Filtering and Non-Filtering Joins: By avoiding runtime redistributions, replicated tables enable low-cost execution of both filtering joins and non-filtering joins, as all required data is already present on each node.

Disadvantages

- Increased Storage Requirements: Each compute node must store a full copy of the table, potentially impacting system scalability, especially for larger datasets.

Distributing Fact Tables with Common Join Columns

For fact tables that are frequently joined to other fact or large dimension tables, selecting the appropriate distribution column is critical for achieving optimal performance.

Below are guidelines to help determine the best approach for your scenario:

Guideline	Description
Focus on Common Joins	Select the column most frequently used in joins with other fact or dimension tables. *Tip: Identify primary key and foreign key relationships in your source data.*
Choose Based on the filtered data set	Select the distribution column considering the size of the data set after filtering, rather than the total size of the table. *Example: If a large dimension table is heavily filtered before a join, focus on the rows that remain after filtering.*
Use High Cardinality Columns	Choose a distribution column with high cardinality in the filtered result set. *Example: A* `sales` *table distributed on a* `date` *column may seem balanced, but frequent filtering for narrow date ranges can cause skew.*
Consider Surrogate Keys for Multi-Column Joins	For multi-column joins, consider a surrogate (synthetic) distribution key that captures join patterns effectively. *Example: If low-cardinality columns like* `customer_id` *and* `product_id` *are part of the join, a synthetic key (e.g., combining these columns) can improve distribution balance.*

Distributing Fact Tables with Common Grouping or Partition Columns

In scenarios where a fact table is not frequently joined with other fact or large dimension tables but is heavily used in `GROUP BY` queries or partitioned window functions, distributing the table on common grouping or partition columns can enhance performance significantly.

The table below demonstrates two possible approaches for this distribution type:

Consideration	Guidance
Choose the Most Common Grouping or Partition Column	Select the column most frequently used in `GROUP BY` or window function partitioning. Ensure the chosen column has high cardinality to avoid data skew.
Use a Surrogate (Synthetic) Key for Multi-Column Grouping	Create a synthetic key for tables requiring grouping or partitioning on multiple columns. This approach helps mitigate issues of low cardinality and high skew in individual columns by combining them into a balanced distribution. **Example:** Combine `customer_id` and `region_id` into a synthetic key: `ALTER TABLE sales ADD COLUMN customer_region_key BIGINT; UPDATE sales SET customer_region_key = customer_id * 1000 + region_id;`

Using DISTRIBUTE RANDOM as Last Option

`DISTRIBUTE RANDOM` is a fallback option for tables that are neither commonly joined nor aggregated on a specific column. This method evenly distributes rows across all compute nodes without considering logical groupings.

When to Use

DISTRIBUTE RANDOM	Description
Uncommon Use Cases	Applicable for tables that lack a suitable column for `DISTRIBUTE ON` and are not part of frequent joins or aggregations.
Last Resort	Use only if distributing on any column would lead to high skew.

Alternative: Using Surrogate (Synthetic) Keys

Before resorting to `DISTRIBUTE RANDOM`, consider creating a surrogate (synthetic) key for distribution. This approach can better align with query patterns while avoiding skew by combining multiple columns or transforming data to achieve a balanced distribution.

Example:

```sql
ALTER TABLE example_table ADD COLUMN synthetic_key BIGINT;
UPDATE example_table SET synthetic_key = col1 * 1000 + col2;
```

Key Takeaways

- Use `DISTRIBUTE REPLICATE` strategically for dimension tables with data size less than 1 GB.
- Distribute tables strategically based on common join, grouping, or partitioning patterns to minimize data movement and optimize query execution. For multi-column groupings, synthetic keys are often the most effective choice for achieving even distribution.
- `DISTRIBUTE RANDOM` should be avoided. Instead, using surrogate keys may align better with query patterns and help avoid skew.

Solutions to Common DISTRIBUTE ON Problems

Here are practical solutions to common challenges faced when using `DISTRIBUTE ON`.

Materialize Joins and Grouping Expressions on the Distribution Column

Joins and groupings on expressions always result in a redistribution, even if the data is redistributed to the same node and the expression is on the distribution column. This leads to costly execution due to both the redistribution of data and the query planner's inability to estimate the cardinality of the expression accurately.

Example Query

```sql
-- ! WARNING !
-- Avoid this query structure as it forces a
-- full redistribution of both tables during execution.
SELECT *
FROM t1
JOIN t2
ON UPPER(t1.c1) = UPPER(t2.c1);
```

Possible Issues

Issues	Description
Data Redistribution	Rows are unnecessarily moved between compute nodes.
Query Planner Limitations	The query planner cannot estimate the cardinality of the `UPPER()` expression, resulting in suboptimal execution plans.

Solution

If a join or grouping on an expression is common, the preferred approach is to materialize the expression into a new column and distribute the table using that column.

Steps For Small Tables

Step	Description	SQL
1	Rename the original table to retain the data.	`ALTER TABLE t1 RENAME TO t1_old;`
2	Create a new table with the materialized expression.	`CREATE TABLE t1 AS SELECT *, UPPER(c1) AS c1_uc FROM t1_old;`
3	Use the materialized column in joins and groupings.	`SELECT * FROM t1 JOIN t2 ON t1.c1_uc = t2.c1_uc;`

Steps For Large Tables

Step	Description	SQL
1	Rename the original table to retain the data.	`ALTER TABLE t1 RENAME TO t1_old;`
2	Create a new table without data.	`CREATE TABLE t1 AS SELECT *, UPPER(c1) AS c1_uc FROM t1_old WITH NO DATA;`
3	Insert data into the new table in chunks.	`INSERT INTO t1 SELECT *, UPPER(c1) AS c1_uc FROM t1_old;`
4	Use the materialized column in joins and groupings.	`SELECT * FROM t1 JOIN t2 ON t1.c1_uc = t2.c1_uc;`

Use a Surrogate Distribution Key for Multi-Column Distribution

For tables with frequent multi-column joins, groupings, or aggregations, using a surrogate distribution key can improve performance by reducing skew and optimizing data placement. This approach is particularly useful when individual columns have low cardinality or high skew but their combination results in a better distribution.

Example Queries

Query Type	Example
Aggregation	`SELECT prod_id, loc_id, SUM(price) AS price FROM line_items GROUP BY 1, 2 ORDER BY 1, 2 LIMIT 10;`
Join	`SELECT prod_id, loc_id, SUM(units) AS units FROM line_items JOIN top_prods USING (prod_id, loc_id);`

Possible Issues

Low Cardinality or High Skew	**Individual columns like `prod_id` and `loc_id` may result in uneven data distribution.**
Complex Distribution for Multi-Column Joins	Low cardinality or high skew in individual columns (such as prod_id and loc_id from the examples above) can result in uneven data distribution across compute nodes. This imbalance affects the efficiency of joins and aggregations.

Solution

Create a surrogate distribution key by combining columns into a composite key.

Steps

Step	Action	SQL
1	Rename the original table to retain the data	`ALTER TABLE line_items RENAME TO line_items_old;`
2	Create a new table with the surrogate key	`CREATE TABLE line_items AS SELECT *, (prod_id * 2^31 + loc_id) AS prod_loc FROM line_items_old;`
3	Update queries to use the surrogate key for aggregation	`SELECT prod_loc, SUM(price) AS price FROM line_items GROUP BY prod_loc ORDER BY prod_loc LIMIT 10;`

Step	Action	SQL
4	Update queries to use the surrogate key for joins	`SELECT prod_loc, SUM(units) AS units FROM line_items JOIN top_prods USING (prod_loc);`

Benefits of Surrogate Keys

Benefit	Description
Reduced Skew	Combines columns to distribute data more evenly across compute nodes.
Optimized Query Performance	Avoids redistributions during joins and groupings.
Simplified Distribution Design	Provides a single key that aligns with query patterns, simplifying data management and optimization.

Use “Fake” Values for High NULL Skew

Columns with a high percentage of `NULL` values can lead to significant at-rest data skew when used as distribution keys. This imbalance occurs because `NULL` values are treated as a single value, causing a large portion of rows to be stored on a single compute node.

Example Query

```sql
SELECT *
FROM line_items li
JOIN location l
ON li.loc_id = l.loc_id;
```

Possible Issues

Issue	Description
High NULL Skew	Columns with many `NULL` values cause uneven data distribution, with most rows stored on one compute node.
At-Rest Skew	Rows with `NULL` values are disproportionately assigned to a single compute node, leading to imbalance.

Solution

Replace `NULL` values with a unique “fake” value to distribute data evenly.

Steps

Step	Description	Example
1	Identify a unique “fake” value that does not match any existing data in the related table.	Example values: `-999999`, negative timestamps, or constants.
2	Replace `NULL` values in the column with the fake value.	`UPDATE line_items SET loc_id = -999999 WHERE loc_id IS NULL;`

Benefits of Using “Fake” Values

Benefit	Description
Eliminates At-Rest Skew	Distributes rows with `NULL` values evenly across compute nodes.

Benefit	Description
Maintains Query Integrity	Ensures queries that exclude `NULL` values remain accurate without additional adjustments.
Avoids Runtime Skew	Prevents performance issues during execution caused by unbalanced data distribution.

Summary

Effective table distribution design is a cornerstone of performance optimization in *Yellowbrick*'s MPP system. By adhering to these best practices, users can address challenges such as data skew, balance storage and workload distribution, and ensure efficient query execution.

Key Takeaways

Takeaway	Description
Analyze Workloads	Define distribution strategies based on common query patterns, join operations, and data characteristics.
Use High-Cardinality Keys	Minimize skew by selecting high-cardinality distribution keys or surrogate keys for multi-column joins.
Manage `NULL` Skew	Replace `NULL` values with unique placeholders to avoid skew where appropriate.
Apply `DISTRIBUTE REPLICATE`	Use for small dimension tables to eliminate redistributions.
Avoid `DISTRIBUTE RANDOM`	Use only when no suitable distribution key exists, as it can lead to inefficiencies in query execution.

By designing table distributions that align with workload requirements and revisiting these strategies over time, users can achieve balanced workloads, reduced data movement, and sustained system performance.

Auto-Analyzing Tables

The ANALYZE command generates updated statistics for the number of distinct values in table columns. This information is useful to the query planner and needs to be kept up-to-date, especially for tables with columns that appear in the WHERE clause of your queries.

The following operations always include an automatic and immediate analyze operation as part of their own post-processing:

- Bulk load operations with `ybload`
- INSERT INTO...SELECT
- CREATE TABLE AS statements (CTAS)
- Flush operations

When these operations complete, their target tables are guaranteed to have up-to-date statistics.

The following operations *do not* include automatic analyze operations:

- INSERT INTO...VALUES
- UPDATE
- DELETE

When these operations complete, the statistics in the target tables may need to be updated. In that case, they wait for the next auto-analyze operation that is scheduled for the system.

By default, all tables are analyzed automatically on a regular schedule. The system checks tables every 5 minutes (300 seconds) and determines which tables require an ANALYZE operation. If at least 10% of a table's rows have changed, a background ANALYZE operation is run.

Tip: To find out when a table was last analyzed, either query the sys.log_analyze view or use Yellowbrick Manager. In Yellowbrick Manager, go to **Databases** and select the specific database, schema, and table. On the **Details** screen, go to **Analyze Activity**.

In general, the default ANALYZE behavior is optimal and is likely to be the best choice for most workloads. However, the following sections describe two distinct use cases that may require immediate ANALYZE operations to be disabled.

Disabling Immediate ANALYZE Operations for INSERTs

If your workload requires frequent small INSERT INTO...SELECT operations into a large fact table and you want to optimize performance, see the `ybd_analyze_after_writes` configuration parameter.

Disabling Immediate ANALYZE Operations for Bulk Loads

If your workload consists of a large number of continuous trickle-feed loads using `ybload` and you want to optimize performance, see the `ybd_analyze_after_loads` configuration parameter.

Disk Quotas

The disk quota feature allows users to limit disk usage for individual databases, tables, and schemas based on quotas. This feature can be implemented to separate use cases and users in terms of disk usage and prevent the accidental consumption of too much space, which could cause serious issues for user applications. If more space is consumed than is allowed by the quota, the transaction is aborted and rolled back when the limit is met.

For example, Database A has a quota of 10TB. If a transaction that would otherwise consume 15TB is run on Database A, it would be stopped and rolled back once it meets the limit.

It is important to note that the quota is not an amount of reserved usage for a given object but rather an upper bound on the given object's disk usage. This means that the total of database quotas can be higher or lower than the total available disk space, and the total of table and schema quotas can be higher or lower than the database quota or the total available space.

For example, Databases A, B, and C have a total of 100TB of disk space. Database A has a quota of 10TB, and Databases B and C have no quotas set. If Database A uses 2TB of data, then Databases B and C can use all remaining 98TB of space.

Disk Quota Behavior

When set, disk quotas apply to all data created, including data that is not yet committed. The disk quota is checked whenever a user query creates data. If a query creates more data than is allocated for the object, an error is thrown, and the transaction that hits the limit is rolled back. When a quota is set, it only affects the given object: the database, schema, or table.

If quotas are set at multiple levels of the database > schema > table hierarchy, whichever quota is set the lowest takes precedence. You can set a smaller quota on a schema than on a table inside that schema, for example. The total of one level of the hierarchy does not need to add up to the next level and can be higher or lower.

Note: Disk quotas must take into account any storage that is being consumed by the row store. When write operations send rows to the row store, and they have not yet been flushed to the column store (or "garbage-collected" in the case of deleted rows), these rows count toward the disk quota. For example, if you set your database quota to 770GB, and 765GB is already being used, a delete of 4GB would put you within 1GB of reaching the quota limit, and you could exceed the limit on the next `INSERT` or `DELETE` that targets a table in the same database.

DELETE and TRUNCATE operations can always be run on objects with a quota, and DROP is allowed on objects in schemas with quotas.

You can query the following system views to monitor disk quota-related behavior:

- sys.database
- sys.table
- sys.schema

Setting and Disabling Disk Quotas

By default, no limits are set on disk usage, which is not impacted by installations and upgrades. Using an ALTER DATABASE, ALTER TABLE, or ALTER SCHEMA command, you can set or disable the maximum amount of space for an individual database, schema, or table, respectively. Quotas on objects can be set by superusers and the owner of the object. Changing a disk quota does not require restarting the database, and the changes should apply as soon as the transaction reads the quota value from the database.

Note: The system has a default maximum of 500k disk quota objects. See `max_quota_objects`.

External Tables

Yellowbrick Documentation > How-to Guides > Create Tables > External Tables

Platforms: EE: All appliance platforms

Parent topic: Create Tables

This section covers user-defined external tables that are stored in, and accessed from, an NFS-mounted file system outside the database. For information about loading tables from object storage via SQL commands, see the documentation for the Yellowbrick Test Drive.

NFS-mounted external tables support use cases for both exporting data and importing it. You can create external tables solely for the purpose of writing data out to external files. You can also read from external tables as part of standard SQL queries, and you can use them to load other tables via `INSERT` and `CTAS` statements.

In the background, external tables use `ybload` and `ybunload` operations to read and write data.

Important: Yellowbrick Data does not recommend the use of external table loads as a general replacement for standard `ybload` operations.

External table operations run `ybload` and `ybunload` processes on the manager node and must share limited compute resources with other critical processes. In addition, external table loads and unloads access source and target files via NFS file systems, which are typically much slower than local file systems.

For these reasons, `ybload` and `ybunload` performance for external table operations may be *significantly* slower than performance for standard loads and unloads. For larger files, the performance difference may be severe.

NFS Mount Points for External Tables

Regardless of their intended use, external tables must be defined in the context of a mount point on a system that is running an NFS server. Before you can create external tables, a Yellowbrick database superuser must create at least one mount point by using the CREATE EXTERNAL MOUNT command. This command defines a short reference name for the mount point and a complete path to the NFS-mounted directory where flat files will be read or written. For example:

```
create external mount '/qumulo/yb100' as 'nfs://qumulo:/data/yb100/' with (gid 0, uid 0);
```

When an external mount is in place, you can read from files that already exist in that location or write to files that will be created in that location. Mounts created for external tables are logged to the sys.mount view.

Definition and Use of External Tables

You can create an external table with a CREATE EXTERNAL TABLE SQL command. Instead of a regular table name, you provide the external file location. The table is not registered in the system catalog, and its schema is derived from a query. (This type of table is also known as a *transient external table*.) A special USING clause defines format choices for writing the data out to the file.

To query an external table, you use special `SELECT...FROM EXTERNAL` syntax and specify the schema of the table in terms of a column list. You can query external tables continuously provided that their storage files remain in the specified location and the associated NFS mount is active. A single query can process a combination of regular table data and external table data. For example, you can join an external table to a regular table, and you can return the union of an external table and a regular table.

You can also write CTAS statements that select from external tables, and you can insert data into other tables by selecting from external tables (using `INSERT INTO SELECT` or `SELECT INTO`).

When external tables are created or queried, background `ybload` and `ybunload` operations generate detailed log files and write them to an NFS directory relative to the mount point you created. See External Table Examples.

For details about external table command syntax and options, see the SQL Commands section (or use the `ybsql help` command).

Examples of Creating and Selecting From External Tables

- To write data to an external file and create an external table, use a CREATE EXTERNAL TABLE statement. This statement runs a query against database tables and writes the results to an external file. For example:

```
create external table '/local_external_tables/ext_season.csv'
using(format text) as select * from season;
```

- To read data from an external table, use a SELECT...FROM EXTERNAL statement. This statement runs a query against an external file and returns the data to the client. For example:

```
select c5, c6 from external '/local_external_tables/ext_match.csv'
(c1 int, c2 date, c3 int, c4 int, c5 char(3), c6 char(3)) using(format text)
order by 1 desc limit 10;
```

- To create a regular table based on data in an external table, use a `CTAS` statement. For example:

```
create table match_results as
select c5, c6
from external '/local_external_tables/ext_match.csv'
(c1 int, c2 date, c3 int, c4 int, c5 char(3), c6 char(3)) using(format text)
order by 1 desc;
```

- To read data from an external table and write it into a regular table, use an INSERT INTO SELECT...FROM EXTERNAL statement. You do not have to declare the column list for the file in this case; it is optional. The `USING` clause is required. For example:

```
premdb=# insert into season
select * from external 'ext_db/season.csv'
using(format csv);
INSERT 0 25
```

The external table in this statement uses the schema of the target table (`season`). The columns in the external data file must be in the same order as they are in the target table, and every column in the target table must also exist in the external file.

Limitations on External Tables

- You cannot create views by selecting from external tables.
- External tables are not backed up or restored.
- External tables cannot be updated, deleted, or truncated.

Generating Values with Sequences

A sequence is a database object that generates unique integer values. You can use sequences to uniquely identify rows in tables when no "real" data is available or appropriate for that purpose. For example, you can use a sequence to generate surrogate primary keys for a large fact table.

Sequences are not tied to specific tables or columns. When you create a sequence, you give it a name and optionally a schema. Once created, a sequence is available for use by any table column in the database. The two main uses for sequences are bulk loads and inserts, where a new sequence value can be generated for each row. You can also generate sequence values within queries and as part of a CTAS statement.

You generate new values in all cases by calling the NEXTVAL function. Each time `NEXTVAL` is called, the sequence advances to its next value.

Sequence values are generated independently on the compute nodes. Each value is stored as a 64-bit `BIGINT` number. The last 10 bits identify the compute node. Increments of 1024 or greater are added to each new generated value to preserve the uniqueness of a specific sequence object across all of the compute nodes on the cluster. Each compute node generates its own distinct set of values.

Specific sequence values reported by system views and in query results depend on which node is the last one to finish executing the `NEXTVAL` function. Sequence values are highly timing-dependent; the increment step is not expected to be uniform between `NEXTVAL` calls.

If you want to reset the starting value for a sequence, you can use the `ALTER SEQUENCE RESTART` command. If you need to protect a sequence from being modified by other users and sessions, you can open a transaction and lock the sequence. Other transactions will not be able to modify the sequence until the transaction with the lock commits or is rolled back.

The lowest sequence number that is ever generated is `1024`.

The manager node also generates its own set of sequence values for use in `INSERT INTO...VALUES` operations. These operations do not execute on the compute nodes.

Note: The maximum supported sequence value is `9223372036854775807` (263-1). When the maximum sequence value is reached on a node, the system returns an error. The database does not cycle through the range automatically; you may need to restart the sequence. Sequence values do not roll back when a transaction is rolled back. Those sequence values are considered used and cannot be reused.

See also Loading Generated Key Values.

Ordering Data Efficiently

Yellowbrick Documentation > How-to Guides > Create Tables > Ordering Data

Platforms: All platforms

Parent topic: Create Tables

The purpose of this guide is to provide best-practices for ordering data in Yellowbrick tables. Efficient data ordering leads to higher query throughput, lower response times and better data compression.

See Data Ordering Concepts to understand the foundational ideas illustrated here.

An Example Shard

Below is a DDL and a logical view of a portion of a shard of a purchases fact table. This example will be used throughout most of the query samples in this document.

```sql
CREATE TABLE purchases (
prch_dt DATE
, geo_id INTEGER
, acct INTEGER
, subacct INTEGER
, prch_amt DECIMAL(9,2)
)
DISTRIBUTE ON ( acct )
SORT ON ( prch_dt )
;
```

This table is ordered within each shard on a single column (`prch_dt`) due to the `SORT ON` attribute. Since fact tables are often loaded in date order, they will likely be naturally ordered by date across shards.

Shard Data and Metadata Illustration

Below is a logical representation of a shard, including its blocks. The table shows shard-level information, such as the range of values (**Min** and **Max**) for each column, and block-level details, as the number of rows and block-specific column data.

		prch_dt	geo_id	acct	subacct	prch_amt
Shard 1						
Shard MD	Min: Max:	2020-01-01 2020-01-03	1 4	10000 99000	10010 99890	100.00 11234.56
Block MD	Min: Max: Block:	2020-01-01 2020-01-01 1	2 4 2	11000 98000 3	11010 98190 4	121.45 4145.09 5
Block Data	Rows: 1-32k	2020-01-01, ..., 2020-01-01	2, 2, 2, 2, 3, ...	12000, 68000, 98000, 34000, 11000,...	12200, 68400, 98190, 34220, 11010,...	340.00, 265.90, 1023.89,...
Block MD	Min: Max: Block:	2020-01-01 2020-01-02 6	1 4 7	19000 97000 8	19030 97770 9	187.60 11001.23 10

		prch_dt	geo_id	acct	subacct	prch_amt
Block Data	Rows: 32k-64k	2020-01-01, ..., 2020-01-02	4, 4, 1, 1, 2, ...	20000, 97000, 56000, 22000, 97000,...	22340, 97600, 56010, 22000, 97070,...	231.45, 668.88, 11001.23,...
Block MD	Min: Max: Block:	2020-01-02 2020-01-02 11	2 4 12	19000 97000 13	13030 97070 14	187.60 11001.23 15
Block Data	Rows: 32k-64k	2020-01-02, ..., 2020-01-02	2, 2, 2, 3, 3, ...	20000, 97000, 56000, 22000, 97000,...	22340, 97600, 56010, 22000, 97000,...	231.45, 668.88, 11001.23,...

For brevity, consider the row range 1-32k to be inclusive, and the range 32k-64k to be inclusive of rows 32,001 and 64,000.

For example, consider the query:

```sql
SELECT prch_amt
FROM purchases
WHERE
prch_dt = '2020-01-01'
AND geo_id = 1;
```

Using the shard and block details from the example table above:

- Shard Skipping

> The shard cannot be skipped because the shard metadata indicates that `prch_dt = '2020-01-01'` falls within the shard's date range (2020-01-01 to 2020-01-03).

- Block Skipping

> Most blocks can be skipped. Block metadata reveals the following:

- Only rows 1-64k have matching `prch_dt` values.
- Within that range, only rows higher than 32k have matching `geo_id` values.

The resulting set includes only the blocks for the queried columns (`prch_dt` , `geo_id` , and `prch_amt`) for rows 32k-64k. Specifically, blocks 6, 7, and 11.

Correlated Columns

It is important to note that the data is well ordered within the shard based on common filtering predicates, starting with `prch_dt` followed by `geo_id` , even though the SORT ON clause only specifies `prch_dt` . In Yellowbrick, correlated columns refer to cases where the ordering of one column reflects or aligns with the ordering of another. Unlike indexes in traditional relational databases, which might involve creating an index on multiple columns, Yellowbrick utilizes these relationships by allowing data to be ordered on a single column. This approach also enhances scan performance and compression efficiency.

Using a Calendar Table as an Example

A calendar table demonstrates how the same date can be represented in multiple formats, such as numeric, textual, or string based. Below is a representation of such a table:

col	dt	yyyymmdd	yyyymm	yyyy	m	dt_vc	dt_str
type	DATE	INT4	INT4	INT2	INT2	VARCHAR	VARCHAR
example	2024-01-23	20240123	202401	2024	1	'024-01-23'	'Jan 23, 2024'

There are many kinds of relationships between columns that we can leverage.

Relationship	Usage
1:1, Same Ordering	For example, a calendar table where a date exists as a DATE, an INTEGER, and a text string. In this case, ordering by the smallest representation is recommended. A date stored as a DATE or INTEGER uses 4 bytes, fitting into a CPU register and allowing efficient numeric sorting. Sorting is significantly faster than using a 10-character string format (e.g., *2024-01-23*).
1:1, Different Ordering	For example, a calendar table with a column containing the spelled-out name, such as *January 1, 2024*. Although there is a *1:1* relationship with the date, their sort orders differ. Sorting by the date provides better scan efficiency and improves compression. Dictionary-based algorithms (e.g., ZIP) effectively compress repeated strings like month names.
Parent-Child, Same Ordering	Ordering by the *child* column (e.g., *yyyymmdd*) ensures its parent columns (e.g., *yyyymm* and *yyyy*) are ordered naturally.
Parent-Child, Different Ordering	For example, columns like *geo_name* and *country_name*. When queries frequently filter on both columns, prioritize ordering by the *lower-cardinality key* (e.g., *geography*) first, followed by the *higher-cardinality key* (e.g., *country*).
Commonly Found Together	For example, *product category* and *product subcategory*. Ordering these columns together *does not greatly assist with shard or block skipping but can significantly improve compression in large datasets*. Ordering by the column with *lower cardinality first* (e.g., *product category*) and *then by the column with higher cardinality* (e.g., *product subcategory*) ensures that *similar values are grouped together*. This grouping allows compression algorithms to achieve better results by reducing redundancy.

Data Compression

Efficient data sorting improves compression. Most compression (or encoding) algorithms rely on or are significantly impacted by the order of the data, particularly when values are repeated in adjacent rows. Two of the most common compression methods are:

Algorithm	Characteristics
Run-Length Encoding (RLE)	Optimized for consecutive repeated values.
Dictionary	Optimized for repeating values across a block of rows.

Ordering data on multiple columns can provide benefits for both compression and scan performance.

Compression becomes increasingly effective as column size grows and cardinality decreases. For example, consider a block of 16k rows containing a 10-character string repeated 100 times in adjacent rows. Using run-length compression, the storage required for these rows in the compressed block would shrink from 1,200 bytes to just 14 bytes.

Table SORT ON and CLUSTER ON

Yellowbrick provides two order-related attributes that can be set during table creation: SORT ON and CLUSTER ON. These attributes are immutable, meaning any changes require dropping and recreating the table.

SORT ON

The `SORT ON` attribute in Yellowbrick defines the order of data within shards, but not across shards, when written to disk. This applies during operations such as *bulk loads*, `INSERT SELECT`, `CREATE TABLE AS SELECT` (CTAS), and when shards are merged by Yellowbrick's storage garbage collector (GC).

INFO

Garbage Collector – In Yellowbrick, GC refers to automated shard maintenance processes such as rewriting shards with high numbers of deleted rows and merging small shards (less than 250MB). It is not related to memory garbage collection in programming languages like Java or C#.

Key Characteristics

SORT ON	Scope
Shards	Works within shards, not across shards. Operations like cluster expansion, GC shard merging, and `INSERT SELECT` *without an* `ORDER BY` clause *may not preserve ordering across shards*. For example, an `INSERT SELECT` operation involving 10 shards per compute node over 50 days without `ORDER BY` date could result in up to 50 different dates in each shard. However, rows within each shard would still be ordered by date.
Correlated Columns	The benefits of `SORT ON` extend beyond the specified column. For example, sorting an invoices table by invoice_id often implies de facto ordering by invoice_date, due to column correlation.
Performance	Can improve the performance of queries *with filtering predicates and, in some cases, joins*. Does not improve the performance of `GROUP BY` or `ORDER BY` operations.
Small Tables	Typically not worth applying to tables with fewer than ~50k rows per compute node.

Limitations

SORT ON	Limitations
Single Column	`SORT ON` can only be applied to one column. For multi-column ordering, consider *creating a composite (synthetic) sort key*. See the Data Distribution How-To Guide for more information.
Encryption	Cannot be applied to encrypted columns.
Immutability	Cannot be added to or changed on an existing table. You must drop and recreate the table to apply a new `SORT ON`.

CLUSTER ON

`CLUSTER ON` is an alternative to `SORT ON`. It *allows you to specify multiple columns but is not equivalent to a multi-column sort* or cube. In *most cases*, it is *not recommended* and should only be used in highly specific scenarios.

Key Characteristics

CLUSTER ON	Scope
Not a Multi-Column Sort	CLUSTER ON does not provide ordered sorting across columns. For cases requiring multi-column order, use a composite sort key instead.
Low Cardinality Keys	Effective only when applied to low cardinality columns of equal weight.
High Cost	It has significant performance overhead during data loads and INSERT operations.

WARNING

In practice, `CLUSTER ON` is *rarely beneficial and carries high costs*. Unless you are certain that your use case justifies it, you should avoid using `CLUSTER ON` and instead explore alternative methods, such as composite sort keys, to achieve your goals.

Ordering Within vs Across Shards

Data ordering is an essential aspect of table design in Yellowbrick. The method used to load data—Bulk Load, Insert-Select, or Trickle Load—determines how data is ordered within and across shards, directly impacting query performance and compression efficiency. The following table compares the effects of these three loading methods on ordering:

Method	Within Shards	Across Shards
Bulk Load	Data within shards is ordered according to the SORT ON column if defined. Without SORT ON, ingestion order is preserved.	No guaranteed ordering across shards due to parallelized data ingestion.
Insert-Select	An ORDER BY clause ensures data is ordered within shards before being written. Without it, the ingestion sequence determines the order.	Guarantees ordering across shards if an ORDER BY clause is included in the query.
Trickle Load	Data retains its streaming order unless reordered by a SORT ON column.	Incremental ingestion does not ensure ordering across shards due to the nature of real-time streaming.

Inherent Data Ordering

Data loaded into shards retains its incoming order unless explicitly reordered by the SORT ON column. However, due to the highly parallelized nature of `ybload`, this order may differ from that of the original source file(s).

Fact data is often loaded in a natural date order, either incrementally throughout the day or in batches at the end of the day. Queries on fact tables also typically include specific date range filters. As a result, it is common to define the SORT ON column as the most frequently used filtering predicate. This approach works well for tables with a small number of shards per compute node and for shards that are rarely updated. For larger tables, consider using composite sort keys and carefully designed ordering. See the Data Distribution How-To Guide for instructions on creating surrogate keys.

The `SORT ON` and `CLUSTER ON` clauses determine the order of data within individual shards but do not impose an order across shards. When data is streamed to a compute node, the stream is divided into 1 GB buffers. Once a buffer is full, it is sorted (if a `SORT ON` or `CLUSTER ON` clause is defined for the table) and written out as a shard. This behavior applies to most backend data appends, including bulk loads, `\COPY`, `INSERT` `VALUES` *operations larger than 30 MB*, and `INSERT` `SELECT` *without an* `ORDER BY` clause.

For cross-shard data ordering, use `INSERT` `SELECT` `ORDER BY`. This method sorts all data before streaming it to the shard writer, ensuring consistent ordering across shards.

Once data is written to shards, its order remains consistent except in the following scenarios:

Case	Description
UPDATE	Updates are processed as a DELETE followed by an append. *Rows are not modified in place*; instead, updated rows are written to new shards. If the update size is small (*less than 250 MB*), these smaller shards will be merged with others during garbage collector operations.
DELETE	When shards have less than 250 MB of undeleted rows, the garbage collector merges them. During this process, data within the new shards is sorted. This typically occurs after large-scale DELETE or UPDATE operations.
Cluster Expansion	Changes to the cluster topology, such as replication to a new configuration, alter the distribution of rows across compute nodes. This inevitably changes the order of rows as well.
Replication / Restore	Backup and restore operations are highly parallelized, meaning rows may not be restored in the same order they originally existed. Additionally, restoring or replicating to a different topology (e.g., with a different number of compute nodes or chassis) guarantees changes in both the row distribution and their order.

Partitioning Tables

Partitioning is a table design technique that significantly reduces resource consumption when queries join or aggregate rows. A partitioned table is stored in separate pieces called *partitions*. Queries read and operate on individual partitions of the table one at a time instead of having to read and process the whole table at once, which reduces memory requirements and the likelihood of spilling. The reduction in memory requirements in turn enables more concurrency and better performance.

You set up the partitioning scheme for a table by declaring partition columns in the CREATE TABLE statement. You can partition on specific ranges of values, such as dates or numbers, or you can use a hash function to partition the data for you.

Partitioning is secondary to data distribution and does not change the distribution scheme defined in the CREATE TABLE statement. Partitions divide data within the compute node where data is already distributed.

For complete syntax and examples, see Partitioning Options.

To see how data is physically partitioned for a specific table that you have created and loaded, query the sys.table_partition view.

General Restrictions on Partitioning Tables

- Up to four columns in a single table can be partitioned.
- The number of partitions produced by a single specification must be no greater than 32767.
- The product of the number of partitions produced by all specifications for the same table must be no greater than 250000. For example, the following table cannot be created:

```
premdb=# create table hashtable(a int, b int)
partition by (hash(a with 3000 partitions), hash(b with 1000 partitions));
ERROR:  the product of the number of partitions produced by all specifications results in 3000000 partitions, 250000 allowed
```

- Replicated tables cannot be partitioned.
- Partitioning schemes cannot be altered after a table is created. To change partitioning, you have to re-create the table.

Best Practices for Partitioning

This section suggests some general guidelines for partitioning tables. Whether a table benefits from partitioning is a function of several factors, including how the table is queried, how it is partitioned, and how much data is actually stored in each partition. Read the following sections to understand how these factors affect your partitioning choices.

Remember that partitioning is an optional feature. The extent to which you use it may depend on specific performance expectations.

Which columns should be the partition columns?

Choose partition columns only after evaluating the following criteria. Partitioning on a table column is useful in three specific contexts:

- When the data in the column can be sorted and is frequently accessed by range using comparison or equality operators such as `<`, `>`, `>=`, or `=`.
- When the column is commonly used alone in a `GROUP BY` clause.
- When the column is frequently used in equi-joins.

In all of these cases, partitioning decreases the total memory and runtime required by applicable queries.

For example:

- If you have time-series data and your queries group or join on date-time values, partition by the appropriate date or timestamp column. You can create one partition of data per some interval, such as day, week, or month.
- If you join fact tables with other fact tables (large "fact-to-fact joins" or "data merges"), partition both fact tables in the same way on the primary and foreign key columns.
- If you have a common logical unit that you report on in your fact tables – for example, the offices or stores that transactions took place in, partition by that unit as well.
- If you have a very large dimension table that is often joined to a fact table, partition on the dimension surrogate key.
- If you have a column in the fact table that is often used for filtering, partition on that column (for example, currency codes or country codes).

In general, all fact tables should be partitioned. Small dimension tables do not need to be partitioned, and tables that are less than 10GB in size may not benefit from partitioning at all.

When to use RANGE partitioning

Consider the following points:

- Is RANGE partitioning supported for the column, given its data type? See the table in Partitioning Options.
- Do you know reasonable upper and lower limits for the values in the column? That is, do you know the *domain* of the column?
- Do you often select from the column by using comparison operators, such as `<` or `>`?

If RANGE partitioning is available for the column type, and you know the domain of the column, it is always preferable to select the RANGE partition type. For example, you can partition on very specific ranges of values, such as dates or numbers.

Using a RANGE instead of a HASH partition can reduce the amount of data read when rows from the table are selected using comparison operators in a WHERE clause on the partitioned column. Using a HASH partition on the same column prevents this optimization.

Note: If you know the range of most of the data, but are unsure of the exact limits, be sure to include the `OUTSIDE RANGE` option. If the column may contain `NULL` values, be sure to specify `IS NULL`, or you will be unable to load any rows that contain `NULL`. See Partitioning Options.

When to use HASH partitioning

In all cases where RANGE partitioning does not apply, use HASH partitioning. For example, you can hash-partition columns that contain character strings, telephone numbers, and IP addresses. For HASH partitioning, you do not need to specify how the partitions are defined, just how many partitions to create.

Choosing an optimal number of partitions

You also need to think about how many partitions your table will contain. A partitioned table is like a cube with individual partition columns as its dimensions. When you select a partition column, you define, implicitly or explicitly, a set of resulting partitions for that column. In turn, that number of partitions is multiplied by the number of partitions for other columns on the same table. The product of this calculation (the total number of "sub-partitions") is an important constraint on effective partitioning.

Consider the following points:

- Although you can use up to four columns in the same table as partition columns, you should use as few as possible to produce the optimal number of partitions.
- Partitioning schemes are always bounded by two limits:
  - The number of partitions per column (about 32,000)
  - The product of all partitioned columns per table (about 250,000).
- The amount of data you expect to store in the table is directly related to the number of optimal partitions. For optimal performance, the table should be big enough and the product of the number of partitions small enough that each sub-partition is either completely empty or contains at least 2-10GB of row data.
- Expected increases in the range of values in a column (future dates, for example), or the amount of data in a table may change your requirements over time. Partitioning cannot be updated at runtime; therefore, you need to create sufficient partitions ahead of time for business needs for the next few years. For example, if you are partitioning on a range of dates, the end of the range should extend well into the future.

Notes on bulk loads

Where possible, try to bulk load data one partition at a time. This means that your source data needs to be prepared in files that contain one partition of data each. For example, if the data is partitioned by range on dates with an interval of one day per partition, each data file should contain a day's worth of data.

If you cannot set up your loads in this way, query performance may be affected.

Some Example Calculations

The following examples demonstrate how to calculate an effective number of partitions on a single table:

Example 1

Say you intend to use three partition columns (c1, c2, c3) on a table (t1) that contains 1TB of data:

- If the three columns evaluate to 5, 5, and 4 partitions each, the product of this partitioning scheme is **100** sub-partitions (5*5*4).
- 1TB of data/100 sub-partitions = **10GB** of data per partition, which is a good number. (It is fine if some of these partitions are empty.)

Example 2

Say you intend to use two partition columns (c1, c2) on a table (t2) that contains 10TB of data:

- If the two columns evaluate to 100 and 1000 partitions each, the product of this partitioning scheme is **100,000** sub-partitions (100*1000).
- 10TB of data/100000 sub-partitions = **100MB** of data per partition, which is much too small.
- You would need to think about reducing the number of partitions by using only one column or a different range, for example.

Example 3

Say you intend to use four partition columns (c1, c2, c3, c4) on a table (t3) that contains 20TB of data:

- If the four columns evaluate to 100, 5, 8, and 200 partitions each, the product of this partitioning scheme is **800000** sub-partitions (10*5*8*200).
- This number of partitions exceeds the system limit, and you will not be able to create the table at all. The CREATE TABLE statement will return an error.
- If you partition on the first three columns only, the product will be 4000 (100*5*8).
- 20TB of data/4000 sub-partitions = **5GB** of data per partition, which is a reasonable number but on the low side.
- It might be optimal to partition this table on only two columns. Pick the two that best fit the criteria described earlier in this section for selecting partition columns.

System Columns in Tables

Yellowbrick Documentation > How-to Guides > Create Tables > System Columns in Tables

Platforms: All platforms

Parent topic: Create Tables

Every table in a Yellowbrick database contains four system columns that contain metadata:

- `ROWTXID` : the transaction ID that created a row (BIGINT)
- `ROWUPDATED` : a Boolean value that indicates whether a row has been modified
- `ROWUNIQUE` : a monotonically increasing 64-bit number that uniquely identifies a row in a table (BIGINT)
- `ROWID` : numeric values that identify rows, *but may not always be unique*

The first three columns are used internally to support backup and restore operations and database replication. The values in the `ROWUNIQUE` column are immutable; this column is a reliable means of determining unique rows in tables.

Important: `ROWID` values are not immutable and may change over time. Therefore, they are useful only within the scope of a single SQL statement, and not within transaction blocks. It is possible for a given row to have a different row ID in between different statements within the same transaction. Also, there is no direct correlation between row IDs and the order in which rows were added to the table; recently inserted rows may have IDs that are lower than IDs for previously inserted rows.

Values in all four system columns cannot be modified; however, they can be selected (and inserted into other tables if needed).

Examples with ROWUNIQUE, ROWTXID, and ROWUPDATED

Select the unique value for each row in the `awayteam` table:

```
premdb=# select rowunique, * from awayteam order by 1;
 rowunique | atid |          name
-----------+------+------------------------
         1 |   90 | Swindon Town
         2 |   91 | Tottenham Hotspur
         3 |   92 | Watford
         4 |   93 | West Bromwich Albion
         5 |   94 | West Ham United
         6 |   95 | Wigan Athletic
         7 |   96 | Wimbledon
         8 |   97 | Wolverhampton Wanderers
         9 |   98 | [NULL]
        10 |   99 | [NULL]
        11 |  100 | [NULL]
        12 |   51 | Arsenal
        13 |   52 | Aston Villa
        14 |   53 | Barnsley
...
```

Run queries that select `ROWUNIQUE` , `ROWTXID` , and `ROWUPDATED` from the `newmatchstats` table:

```
premdb=# select rowunique, rowtxid, rowupdated from newmatchstats order by 1 limit 5;
 rowunique | rowtxid | rowupdated
-----------+---------+------------
         1 |   38656 | t
         2 |   38656 | t
```

```
        3 |   38656 | t
        4 |   38656 | t
        5 |   38656 | t
(5 rows)

premdb=# select count(*) from newmatchstats where rowupdated='f';
 count
--------
 765934
(1 row)

premdb=# select count(distinct(rowtxid)) from newmatchstats;
 count
-------
    90
(1 row)

premdb=# create table newmatchstats_rowunique as select rowunique as rownumber from newmatchstats;
SELECT 774540
```

Examples with ROWID Functions

Row IDs can be decomposed into information that is useful for monitoring database activity. For example, based on a row ID, you can find out which compute node processed a particular row or where it is physically stored. Rows that have not yet been flushed to the column store also have valid row IDs, but they are displayed as negative values.

The following system functions provide ways to query and extract information from row IDs:

- sys.worker_id_from_rowid(rowid): Return the worker ID for a row.
- sys.filesys_number_from_rowid(rowid): Return the file system number (SSD) for a row.
- sys.inode_number_from_rowid(rowid): Return the inode number (shard) for a row.
- sys.row_number_from_row_id(rowid): Return the row number (unique per worker, SSD, and shard).

These functions accept either the string `rowid` as their input or a specific `rowid` value. In most cases you will use the string `rowid` to return the specific row ID for each row in a set of rows. (If you specify an individual row ID as the input to the function, make sure it is valid; the system does not detect row IDs that are not valid or do not exist.)

For example, the following query returns specific row IDs and row numbers for a set of rows in the `match` table:

```
premdb=# select rowid, sys.row_number_from_rowid(rowid),* from match where seasonid=21 limit 10;
     rowid       | row_number_from_rowid | seasonid |      matchday       | htid | atid | ftscore | htscore
-----------------+-----------------------+----------+---------------------+------+------+---------+---------
 1126011584388774 |                  7846 |       21 | 2013-02-23 00:00:00 |    2 |   52 | 2-1     | 1-0
 1126011584388775 |                  7847 |       21 | 2012-09-29 00:00:00 |    2 |   63 | 1-2     | 1-1
 1126011584388776 |                  7848 |       21 | 2013-04-16 00:00:00 |    2 |   67 | 0-0     | 0-0
 1126011584388777 |                  7849 |       21 | 2012-11-10 00:00:00 |    2 |   68 | 3-3     | 2-2
 1126011584388778 |                  7850 |       21 | 2013-01-30 00:00:00 |    2 |   73 | 2-2     | 0-1
 1126011584388779 |                  7851 |       21 | 2013-01-13 00:00:00 |    2 |   74 | 0-2     | 0-2
 1126011584388780 |                  7852 |       21 | 2013-04-28 00:00:00 |    2 |   75 | 1-1     | 1-1
 1126011584388781 |                  7853 |       21 | 2012-12-29 00:00:00 |    2 |   77 | 7-3     | 1-1
 1126011584388782 |                  7854 |       21 | 2013-04-13 00:00:00 |    2 |   78 | 3-1     | 0-0
 1126011584388783 |                  7855 |       21 | 2012-10-27 00:00:00 |    2 |   82 | 1-0     | 0-0
(10 rows)
```

The following example finds out where (on which compute node) the rows in the `matchstats` table are stored:

```
yellowbrick_test=# select sys.worker_id_from_rowid(rowid), count(*)
from matchstats group by sys.worker_id_from_rowid(rowid) order by 1;
 worker_id_from_rowid | count
----------------------+-------
                   0 | 24437
```

```
                  1 | 24244
                  2 | 24761
                  3 | 24361
                  4 | 24000
                  5 | 24348
                  6 | 24497
                  7 | 24270
                  8 | 24600
                  9 | 24665
                 10 | 24409
                 11 | 24359
                 12 | 24528
                 13 | 24176
                 14 | 24590
(15 rows)
```

To return the UUID for each compute node instead of its logical ID, use the `sys.worker_uuid` function:

```
yellowbrick_test=# select sys.worker_uuid(sys.worker_id_from_rowid(rowid)), count(*)
from crdm_giftcard group by sys.worker_id_from_rowid(rowid) order by sys.worker_id_from_rowid(rowid);
           worker_uuid              | count
------------------------------------+-------
 00000000-0000-0000-0000-38b8ebd00154 | 24437
 00000000-0000-0000-0000-38b8ebd000dc | 24244
 00000000-0000-0000-0000-38b8ebd002c6 | 24761
 00000000-0000-0000-0000-38b8ebd00069 | 24361
 00000000-0000-0000-0000-38b8ebd00041 | 24000
 00000000-0000-0000-0000-38b8ebd00091 | 24348
 00000000-0000-0000-0000-38b8ebd000aa | 24497
 00000000-0000-0000-0000-38b8ebd003c5 | 24270
 00000000-0000-0000-0000-38b8ebd0009b | 24600
 00000000-0000-0000-0000-38b8ebd00997 | 24665
 00000000-0000-0000-0000-38b8ebd002bc | 24409
 00000000-0000-0000-0000-38b8ebd003a2 | 24359
 00000000-0000-0000-0000-38b8ebd00028 | 24528
 00000000-0000-0000-0000-38b8ebd00212 | 24176
 00000000-0000-0000-0000-38b8ebd0025d | 24590
(15 rows)
```

Loading Tables

Yellowbrick Documentation > How-to Guides > Load Tables

Platforms: All platforms

Parent topic: How-to Guides

In this section:

This section covers how to bulk load data into Yellowbrick tables by using the `ybload` client.

Alternatively, on cloud platforms see SQL-Based Loads from External Storage and the LOAD TABLE command.

Bulk Loading Tables

In this section:

The Yellowbrick bulk loader (`ybload`) is a Java-based bulk data loader that you start from a client system. You can load very large data files from remote systems by running this utility. Yellowbrick recommends using the bulk loader to load all of your database tables.

The loader distributes the data in parallel directly to compute nodes, based on the distribution key in the `CREATE TABLE` statement. The utility loads a single destination table that you specify, by default *appending* the loaded rows to any existing rows in the table (including any duplicate rows). You can also use `ybload` to update, delete, or "upsert" rows. An upsert updates existing rows or inserts new rows in a table as part of a single `ybload` operation. For updates and upserts, you can manage how duplicate rows are processed.

Yellowbrick recommends that you upgrade to the latest version of the `ybtools` clients when you upgrade the cluster so that your client and server versions correspond. The client tools are backward-compatible but not always forward-compatible.

Bulk Load Examples

Platforms: All platforms

Parent topic: Bulk Loading Tables

In this section:

The examples in this section show how to use various `ybload` options in conjunction with different types of loads: inserts, deletes, updates, and upserts.

Note: The output includes a printout of the command-line options that you used; however, do not try to copy and reuse this command information as printed. Quotes may be removed, variables substituted, wildcard characters expanded, and so on.

The first insert example shows the `ABOUT CLIENT` message block, which is always returned to the console unless the `--quiet` option and associated logging options are used. Subsequent examples omit this output.

See also Parquet Load Examples, S3 Object Storage Examples, and Azure Blob Storage Examples.

Bulk Load Insert Examples

Platforms: All platforms

Parent topic: Bulk Load Examples

The following examples show how to use various `ybload` options in the context of a default `INSERT` load. See also Parquet Load Examples and Deletes, Updates, and Upserts.

Load the match table with the -d option from a single CSV file

```
$ ybload -d premdb --username bobr -t match -W --format csv /home/bobr/match.csv
Password for user bobr:
18:18:32.650 [ INFO] ABOUT CLIENT:
   app.cli_args         = -d premdb --username bobr -t match -W --format csv /home/bobr/match.csv
   app.name_and_version = ybload version 2.0.0-10711
   java.home            = /usr/lib/jvm/java-8-oracle/jre
   java.version         = 1.8.0_101
   jvm.memory           = 981.50 MB (max=14.22 GB)
   jvm.name_and_version = Java HotSpot(TM) 64-Bit Server VM (build 25.101-b13, mixed mode)
   jvm.options          = -Xmx16g, -Xms1g, -ea, -Dapp.name=ybload, -Dapp.pid=19168, -Dapp.repo=/usr/lib/ybtools/lib, -Dapp.home=/u
   jvm.vendor           = Oracle Corporation
   os.name_and_version  = Linux 4.4.0-31-generic (amd64)

18:18:32.697 [ INFO] Gathering metadata on input files
18:18:32.914 [ INFO] Assuming source encoding matches database server encoding: LATIN9
18:18:32.946 [ INFO] Starting 1 source PreReaders
18:18:32.967 [ INFO] Auto-detected line separator = '\n'
18:18:32.977 [ INFO] Configuration (record/field separation):
   --format                          : CSV
   --delimiter                       : ,
   --linesep                         : \n
   --quote-char                      : "
   --escape-char                     : "
   --trim-white                      : false
   --skip-blank-lines                : true
   --on-missing-field                : ERROR
   --on-extra-field                  : ERROR
   --on-unescaped-embedded-quote     : ERROR
   Internal scanner                  : RecordScannerQuote_1_1
18:18:33.004 [ INFO] Using database locale: C
18:18:33.846 [ INFO] Bad rows will be written to /home/bobr/match.csv.20190129181833.bad
18:18:33.854 [ INFO] Starting import of 1 files
18:18:34.127 [ INFO] Starting 4 segment readers
18:18:34.128 [ INFO] Opening transaction #1...
18:18:34.183 [ INFO] Opened transaction #1
18:18:34.234 [ INFO] Flushing last of 8606 rows for transaction #1
18:18:34.238 [ INFO] Committing 8606 rows into transaction #1...
18:18:34.256 [ INFO] Committed transaction #1 after a total of 292604 bytes and 8606 good rows
18:18:34.275 [ INFO] READ:305.8KB(193.0KB/s). ROWS G/B: 8606/0( 5.30K/s). WRITE:285.7KB(180.3KB/s).  TIME E/R:   0:00:01/ --:--:--
18:18:34.276 [ INFO] SUCCESSFUL BULK LOAD: Loaded 8606 good rows in   0:00:01 (READ: 193.0KB/s WRITE: 180.3KB/s)
```

Load the same table from a CSV file with a single header line

```
$ ybload -d premdb --username bobr -t match -W --format csv --num-header-lines 1 /home/bobr/match_with_header.csv
Password for user bobr:
...
18:39:24.888 [ INFO] Gathering metadata on input files
18:39:25.163 [ INFO] Assuming source encoding matches database server encoding: LATIN9
18:39:25.233 [ INFO] Starting 1 source PreReaders
18:39:25.256 [ INFO] Auto-detected line separator = '\n'
18:39:25.263 [ INFO] Configuration (record/field separation):
   --format                          : CSV
   --delimiter                       : ,
   --linesep                         : \n
   --quote-char                      : "
   --escape-char                     : "
   --trim-white                      : false
   --skip-blank-lines                : true
   --on-missing-field                : ERROR
   --on-extra-field                  : ERROR
   --on-unescaped-embedded-quote     : ERROR
   Internal scanner                  : RecordScannerQuote_1_1
18:39:25.290 [ INFO] Using database locale: C
18:39:26.047 [ INFO] Bad rows will be written to /home/bobr/match_with_header.csv.20190129183926.bad
18:39:26.055 [ INFO] Starting import of 1 files
18:39:26.326 [ INFO] Starting 4 segment readers
18:39:26.327 [ INFO] Opening transaction #1...
18:39:26.374 [ INFO] Opened transaction #1
18:39:26.411 [ INFO] Flushing last of 8606 rows for transaction #1
18:39:26.414 [ INFO] Committing 8606 rows into transaction #1...
18:39:26.433 [ INFO] Committed transaction #1 after a total of 292604 bytes and 8606 good rows
18:39:26.452 [ INFO] READ:305.8KB(194.7KB/s). ROWS G/B: 8606/0( 5.35K/s). WRITE:285.7KB(181.9KB/s).  TIME E/R:   0:00:01/ --:--:--
18:39:26.453 [ INFO] SUCCESSFUL BULK LOAD: Loaded 8606 good rows in   0:00:01 (READ: 194.7KB/s WRITE: 181.9KB/s)
```

Load the same table from multiple CSV files

```
$ ybload -d premdb --username bobr -t match -W --format csv /home/bobr/premdata/matchdata/*
Password for user bobr:
...
18:50:39.278 [ INFO] Auto-detected line separator = '\n'
18:50:39.288 [ INFO] Configuration (record/field separation):
   --format                          : CSV
   --delimiter                       : ,
   --linesep                         : \n
   --quote-char                      : "
   --escape-char                     : "
   --trim-white                      : false
   --skip-blank-lines                : true
   --on-missing-field                : ERROR
   --on-extra-field                  : ERROR
   --on-unescaped-embedded-quote     : ERROR
   Internal scanner                  : RecordScannerQuote_1_1
18:50:39.320 [ INFO] Using database locale: C
18:50:40.119 [ INFO] Bad rows will be written to /home/bobr/ybload.20190129185040.bad
18:50:40.127 [ INFO] Starting import of 3 files
18:50:40.416 [ INFO] Starting 4 segment readers
18:50:40.417 [ INFO] Opening transaction #1...
18:50:40.461 [ INFO] Opened transaction #1
18:50:40.557 [ INFO] Flushing last of 25818 rows for transaction #1
18:50:40.561 [ INFO] Committing 25818 rows into transaction #1...
18:50:40.586 [ INFO] Committed transaction #1 after a total of 877812 bytes and 25818 good rows
18:50:40.607 [ INFO] READ:917.4KB(570.9KB/s). ROWS G/B: 25818/0(15.69K/s). WRITE:857.2KB(533.5KB/s).  TIME E/R:   0:00:01/ --:--:-
18:50:40.607 [ INFO] SUCCESSFUL BULK LOAD: Loaded 25818 good rows in   0:00:01 (READ: 570.9KB/s WRITE: 533.5KB/s)
```

Load a table with console logging turned off

This example turns off console logging, sends the output to a file, and sets the log level for the output that goes to that file.

```
$ ybload -d premdb --username bobr -t match -W --format csv --quiet --logfile load_match_log --logfile-log-level INFO /home/bobr/m
Password for user bobr:

$ more load_match_log
2019-01-30 16:00:21.830 [ INFO] <main>  ABOUT CLIENT:
   app.cli_args         = -d premdb --username bobr -t match -W --format csv --quiet --logfile load_match_log --logfile-log-level
   app.name_and_version = ybload version 2.0.0-10711
   java.home            = /usr/lib/jvm/java-8-oracle/jre
   java.version         = 1.8.0_101
   jvm.memory           = 981.50 MB (max=14.22 GB)
   jvm.name_and_version = Java HotSpot(TM) 64-Bit Server VM (build 25.101-b13, mixed mode)
   jvm.options          = -Xmx16g, -Xms1g, -ea, -Dapp.name=ybload, -Dapp.pid=12595, -Dapp.repo=/usr/lib/ybtools/lib, -Dapp.home=/u
   jvm.vendor           = Oracle Corporation
   os.name_and_version  = Linux 4.4.0-31-generic (amd64)

2019-01-30 16:00:21.831 [ INFO] <main>  Logfile written to load_match_log
...
2019-01-30 16:00:23.580 [ INFO] <main>  SUCCESSFUL BULK LOAD: Loaded 8606 good rows in   0:00:01 (READ: 179.2KB/s WRITE: 167.5KB/s
```

Stop and roll back a load if >3 bad rows are found and log the errors

```
$ ybload -d premdb --username bobr -t team --max-bad-rows 3 -W --quiet --logfile load_team_log --logfile-log-level error /home/bob
Password for user bobr:

brumsby@brumsby:~$ more load_team_log
2019-01-30 16:10:46.559 [ERROR] <main>  Unable to separate row into fields for parsing: Too few fields (8 < 9)
Bad row = "4,53,Barnsley,Tykes,Barnsley,Oakwell Stadium,23009,0"
2019-01-30 16:10:46.561 [ERROR] <main>  Unable to separate row into fields for parsing: Too few fields (8 < 9)
Bad row = "18,67,Everton,Toffees,Liverpool,Goodison Park,40221,38.124"
2019-01-30 16:10:46.561 [ERROR] <main>  Unable to separate row into fields for parsing: Too few fields (8 < 9)
Bad row = "31,80,Oldham Athletic,Latics,Oldham,Boundary Park,13309,0"
2019-01-30 16:10:46.562 [ERROR] <main>  Unable to separate row into fields for parsing: Too few fields (8 < 9)
Bad row = "32,81,Portsmouth,Pompey,Portsmouth,Fratton Park,21100,0"
2019-01-30 16:10:46.562 [FATAL] <main>  Number of bad rows exceeded maximum (3)
2019-01-30 16:10:46.620 [FATAL] <main>  FAILED BULK LOAD: Nothing committed
```

Load a table from a text file with a special escape character

```
$ ybload -d premdb --username bobr -W -t matchstats --format text --escape-char '/' /home/bobr/premdata/matchstats.txt
Password for user bobr:
...
```

Load a table from a pipe-delimited file with literal "null" values

```
$ ybload -d premdb --username bobr -W -t team --format text --delimiter '|' --nullmarker 'null' /home/bobr/premdata/team2019.txt
Password for user bobr:
...
```

Load a table from a CSV file that has an extra field

This example loads a table from a source file that has an extra field at the end of each line, which is removed (not loaded). To achieve this result, the `ybload` command specifies the `--parse-header-line` and `--on-extra-field remove` options.

The `season` table has four columns, but the source file in this example has five values per line (comma-separated):

```
$ more season_five_cols.csv
SEASONID,SEASON_NAME,NUMTEAMS,WINNERS
25,2016-2017,20,Chelsea,380
26,2017-2018,20,Manchester City,380
27,2018-2019,20,Manchester City,380
28,2019-2020,20,Liverpool,380
29,2020-2021,20,Manchester City,380
30,2021-2022,20,Manchester City,380
```

The following load detects the mismatch in the number of columns compared to fields, but succeeds in loading the table by removing the last field for each row:

```
$ ybload -d premdb -t season -U yellowbrick -W --delimiter ',' --format csv --parse-header-line --on-extra-field remove season_fiv
Password for user yellowbrick:
...
Configuration (record/field separation):
   format                  : CSV
   delimiter               : ,
   linesep                 : \n
   quoteChar               : "
   escapeChar              : "
   trimWhite               : false
   trimTrailingWhite       : false
   skipBlankLines          : true
   onMissingField          : ERROR
   onExtraField            : REMOVE
   onUnescapedEmbeddedQuote: ERROR
...
```

The resulting rows look like this:

```
premdb=# select * from season where seasonid between 25 and 30;
 seasonid | season_name | numteams |     winners
----------+-------------+----------+-----------------
       25 | 2016-2017   |       20 | Chelsea
       26 | 2017-2018   |       20 | Manchester City
       27 | 2018-2019   |       20 | Manchester City
       28 | 2019-2020   |       20 | Liverpool
       29 | 2020-2021   |       20 | Manchester City
       30 | 2021-2022   |       20 | Manchester City
(6 rows)
```

Load a table from a CSV file with a missing field

This example loads a table from a source file that has a missing field at the end of each line, which is filled in with `NULL` values. The `season` table has four columns, but the source file in this example has only three values (comma-separated). The `winners` column is defined in the header, but no `winners` values are present in each line:

```
$ more season_three_cols.csv
SEASONID,SEASON_NAME,NUMTEAMS,WINNERS
25,2016-2017,20
26,2017-2018,20
27,2018-2019,20
28,2019-2020,20
```

```
29,2020-2021,20
30,2021-2022,20
```

To load data from this file, the `ybload` command specifies the `--parse-header-line` and `--on-missing-field supplynull` options. The table is loaded with `NULL` values for the missing column.

```
$ ybload -d premdb -t season -U yellowbrick -W --delimiter ',' --format csv --parse-header-line --on-missing-field supplynull seas
Password for user yellowbrick:
...
Configuration (record/field separation):
   format                  : CSV
   delimiter               : ,
   linesep                 : \n
   quoteChar               : "
   escapeChar              : "
   trimWhite               : false
   trimTrailingWhite       : false
   skipBlankLines          : true
   onMissingField          : SUPPLYNULL
   onExtraField            : ERROR
   onUnescapedEmbeddedQuote: ERROR
...
```

The resulting rows look like this:

```
premdb=# select * from season where seasonid between 25 and 30;
 seasonid | season_name | numteams | winners
----------+-------------+----------+---------
      25 | 2016-2017   |       20 | [NULL]
      26 | 2017-2018   |       20 | [NULL]
      27 | 2018-2019   |       20 | [NULL]
      28 | 2019-2020   |       20 | [NULL]
      29 | 2020-2021   |       20 | [NULL]
      30 | 2021-2022   |       20 | [NULL]
(6 rows)
```

Load a calendar table with a specific date format

The `--date-style` option is set to `MDY`, which means that `ybload` can parse and load incoming date values such as `08-13-2018`.

```
$ ybload -d premdb --username bobr -t -W season_date --format text --delimiter '|' --date-style MDY /home/ybdata/season_dates.txt
Database login password:
...
```

Load a table with a case-sensitive name

Note the `-t` syntax required to load a table that was created with a quoted identifier:

```
$ ybload -d premdb --username bobr -W -t \"AwayTeam\" --delimiter ',' /home/premdata/awayteam.csv
...
12:19:02.492 [ INFO] Bulk Loading /home/premdata/awayteam.csv into TABLE **"public"."AwayTeam"** at 2018-10-27T12:19:01.757-07:00[
...
```

Delete, Update, and Upsert Examples

Platforms: All platforms

Parent topic: Bulk Load Examples

This section contains examples of loads that use the `--write-op delete`, `update`, and `upsert` modes.

UPDATE and UPSERT Example

For example, table `five_seasons` contains:

```
premdb=# select * from five_seasons;
 seasonid | season_name | numteams |      winners
----------+-------------+----------+-------------------
       1 | 1992-1993   |       22 | Manchester United
       2 | 1993-1994   |       22 | Manchester United
       3 | 1994-1995   |       22 | Blackburn Rovers
       4 | 1995-1996   |       20 | Manchester United
       5 | 1996-1997   |       20 | Manchester United
(5 rows)
```

Assume that the source file for the load contains ten rows in which the `winners` value always ends with `FC`:

```
1,1992-1993,22,Manchester United FC
2,1993-1994,22,Manchester United FC
3,1994-1995,22,Blackburn Rovers FC
4,1995-1996,20,Manchester United FC
5,1996-1997,20,Manchester United FC
6,1997-1998,20,Arsenal FC
7,1998-1999,20,Manchester United FC
8,1999-2000,20,Manchester United FC
9,2000-2001,20,Manchester United FC
10,2001-2002,20,Arsenal FC
```

If the `seasonid` column is the primary key of the table and is used to match rows in the table with rows in the source file, a `--write-op update` load would result in the modification of all five existing rows. Rows `6` through `10` would be read but not loaded.

Starting with the same five-row table and source file as the first `UPDATE` example, a `--write-op upsert` load would update five rows but also append the five new rows. The result would be a ten-row table.

```
premdb=# select * from five_seasons order by 1;
 seasonid | season_name | numteams |       winners
----------+-------------+----------+---------------------
       1 | 1992-1993   |       22 | Manchester United FC
       2 | 1993-1994   |       22 | Manchester United FC
       3 | 1994-1995   |       22 | Blackburn Rovers FC
       4 | 1995-1996   |       20 | Manchester United FC
       5 | 1996-1997   |       20 | Manchester United FC
       6 | 1997-1998   |       20 | Arsenal FC
       7 | 1998-1999   |       20 | Manchester United FC
       8 | 1999-2000   |       20 | Manchester United FC
       9 | 2000-2001   |       20 | Manchester United FC
```

```
      10 | 2001-2002   |       20 | Arsenal FC
(10 rows)
```

DELETE Example

For this example, assume that the `five_seasons` table has the following 12 rows in it:

```
premdb=# select * from five_seasons order by 1;
 seasonid | season_name | numteams |      winners
----------+-------------+----------+-------------------
       0 | 1991-1992   |       22 | [NULL]
       1 | 1992-1993   |       22 | Manchester United
       2 | 1993-1994   |       22 | Manchester United
       3 | 1994-1995   |       22 | Blackburn Rovers
       4 | 1995-1996   |       20 | Manchester United
       5 | 1996-1997   |       20 | Manchester United
       6 | 1997-1998   |       20 | Arsenal
       7 | 1998-1999   |       20 | Manchester United
       8 | 1999-2000   |       20 | Manchester United
       9 | 2000-2001   |       20 | Manchester United
      10 | 2001-2002   |       20 | Arsenal
      30 | 2020-2021   |       20 | [NULL]
(12 rows)
```

Assume that the source file contains 7 records as follows:

```
0,,,
3,1994-1995,22,Blackburn Rovers
6,1997-1998,20,Arsenal
10,2001-2002,20,Arsenal
11,,,
12,,,
30,2020-2021,20,
```

Using the `seasonid` key field, a `--write-op delete` load would delete 5 of these records (those with `seasonid` 0, 3, 6, 10, and 30). The resulting table would have 7 rows:

```
premdb=# select * from five_seasons order by 1;
 seasonid | season_name | numteams |      winners
----------+-------------+----------+-------------------
       1 | 1992-1993   |       22 | Manchester United
       2 | 1993-1994   |       22 | Manchester United
       4 | 1995-1996   |       20 | Manchester United
       5 | 1996-1997   |       20 | Manchester United
       7 | 1998-1999   |       20 | Manchester United
       8 | 1999-2000   |       20 | Manchester United
       9 | 2000-2001   |       20 | Manchester United
(7 rows)
```

UPSERT Example with ybload Output and System View Queries

This example shows the `ybload` command and output for a very small `upsert` load. Then it shows how to query `sys.log_load` and `sys.log_query` to get more information about the load operation.

A table called `minimatch` contains 15 rows:

```
premdb=# select * from minimatch order by seasonid,htid,atid;
 seasonid |     matchday        | htid | atid | ftscore | htscore
----------+---------------------+------+------+---------+---------
       22 | 2014-03-29 15:00:00 |    2 |   74 | 1-1     | 0-1
       22 | 2014-02-12 00:00:00 |    2 |   75 | 0-0     | 0-0
       22 | 2014-04-28 15:00:00 |    2 |   77 | 3-0     | 2-0
       22 | 2013-10-19 00:00:00 |    2 |   78 | 4-1     | 1-0
       22 | 2013-11-23 15:00:00 |    2 |   86 | 2-0     | 1-0
       22 | 2013-09-22 00:00:00 |    2 |   87 | 3-1     | 2-1
       22 | 2014-02-22 15:00:00 |    2 |   88 | 4-1     | 3-0
       22 | 2014-03-25 00:00:00 |    2 |   89 | 2-2     | 0-1
       22 | 2013-09-01 15:00:00 |    2 |   91 | 1-0     | 1-0
       22 | 2014-05-04 00:00:00 |    2 |   93 | 1-0     | 1-0
       25 | 2020-10-24 13:00:00 |    3 |   75 | 1-1     | 1-0
       25 | 2020-10-24 12:30:00 |    4 |   76 | 6-2     | 2-2
       25 | 2020-10-24 15:00:00 |    5 |   87 | 1-4     | 1-2
       25 | 2020-10-25 11:00:00 |    6 |   89 | 0-3     | 0-1
       25 | 2020-10-25 13:00:00 |    7 |   91 | 3-0     | 1-0
(15 rows)
```

The source file for an upsert load looks like this:

```
$ more minimatch_upsert.csv
22,2014-02-12 15:00:00,2,75,0-0,0-0
22,2013-10-19 15:00:00,2,78,4-1,1-0
22,2013-09-22 15:00:00,2,87,3-1,2-1
22,2014-03-25 15:00:00,2,89,2-2,0-1
22,2014-05-04 15:00:00,2,93,1-0,1-0
22,2014-03-29 15:00:00,2,74,1-1,0-1
22,2014-04-28 15:00:00,2,77,3-0,2-0
22,2013-11-23 15:00:00,2,86,2-0,1-0
22,2014-02-22 15:00:00,2,88,4-1,3-0
22,2013-09-01 15:00:00,2,91,1-0,1-0
25,2020-10-24 13:00:00,3,75,1-1,1-0
25,2020-10-24 12:30:00,4,76,6-2,2-2
25,2020-10-24 15:00:00,5,87,1-4,1-2
25,2020-10-25 11:00:00,6,89,0-3,0-1
25,2020-10-25 13:00:00,7,91,3-0,1-0
30,2020-10-26 13:00:00,8,88,4-2,2-2
```

The file contains 16 rows, including a new row with `seasonid 30`. 10 rows in the file match the rows that are already in the table. 5 other rows match, except for a change in the timestamp for the `matchday` column.

An `upsert` load reads the `16` rows from the source file, using three key fields:

```
$ ybload -d premdb --username bobr -t minimatch -W --format csv --write-op upsert --key-field-names seasonid,htid,atid /home/brums
...
20:34:57.159 [ INFO] Flushing last 16 rows (of 16 total) in transaction #1 for minimatch
READ:576.0 B(333.6 B/s). ROWS G/B: 16/0( 9.27 /s). WRITE:544.0 B(315.1 B/s).  TIME E/R:   0:00:01/ --:--:--20:34:57.174 [ INFO] Co
20:34:57.514 [ INFO] Committed transaction #1 after a total of 544 bytes and 16 good rows for minimatch
20:34:57.531 [ INFO] READ:576.0 B(278.5 B/s). ROWS G/B: 16/0( 7.74 /s). WRITE:544.0 B(263.0 B/s).  TIME E/R:   0:00:02/ --:--:--
20:34:57.533 [ INFO] SUCCESSFUL BULK LOAD: Read 16 good rows from the source file(s) in   0:00:02 (READ: 278.5 B/s WRITE: 263.0 B/
```

To find out more about the results of this load, query `sys.log_load`:

```
premdb=# select session_key, table_name, username, end_time, transaction_first, inserted_rows
from sys.log_load where username='bobr' and table_name='minimatch' and error_string is null
order by end_time desc;
                         session_key                          | table_name | username |          end_time          | transactio
--------------------------------------------------------------+------------+----------+----------------------------+----------
```

```
 BrJx6HcnNQkNOGxorNSPD76G-br5LQwqtgcM5HPDCclCej4OOLAaqwzcd9wF1dnK | minimatch  | bobr     | 2020-10-29 20:34:57.552-07 |
 ...
```

Given the transaction ID for the load in question, you can query the `sys.log_query` view for more details and statistics. (Alternatively, do this in one step by joining the `sys.log_load` view to the `sys.log_query` view on `transaction_first=transaction_id`.)

```
premdb=# select substr(query_text,1,25), rows_inserted
from sys.log_query where transaction_id=92925 order by done_time asc;
         substr           | rows_inserted
--------------------------+---------------
 YBULKLOAD INTO "UPSERT_1_ |            16
 UPDATE "premdb"."public". |            15
 INSERT INTO "premdb"."pub |             1
 ANALYZE HLL public.minima |             0
 DROP TABLE ID pg_temp_10. |             0
(5 rows)
```

Note that an `upsert` load is processed in five separate steps:

- `YBULKLOAD` into a temporary table.
- `UPDATE` the target table.
- `INSERT INTO` the target table.
- `ANALYZE` the target table.
- `DROP` the temporary table.

Based on the results of this query, you know that `16` rows were read from the source file, `15` rows were updated, and `1` completely new row was inserted.

Note that 15 rows are updated because 15 rows matched on the key fields: `seasonid,htid,atid`. Only 5 of these rows were actually modified (their `matchday` values were changed), but all of them were written to the database. That is, 10 existing rows were written to the database with the exact same information that was stored before, and 5 existing rows were written with a change to one column.

The updated table now contains the following 16 rows:

```
premdb=# select * from minimatch order by seasonid,htid,atid;
 seasonid |      matchday       | htid | atid | ftscore | htscore
----------+---------------------+------+------+---------+---------
       22 | 2014-03-29 15:00:00 |    2 |   74 | 1-1     | 0-1
       22 | 2014-02-12 15:00:00 |    2 |   75 | 0-0     | 0-0
       22 | 2014-04-28 15:00:00 |    2 |   77 | 3-0     | 2-0
       22 | 2013-10-19 15:00:00 |    2 |   78 | 4-1     | 1-0
       22 | 2013-11-23 15:00:00 |    2 |   86 | 2-0     | 1-0
       22 | 2013-09-22 15:00:00 |    2 |   87 | 3-1     | 2-1
       22 | 2014-02-22 15:00:00 |    2 |   88 | 4-1     | 3-0
       22 | 2014-03-25 15:00:00 |    2 |   89 | 2-2     | 0-1
       22 | 2013-09-01 15:00:00 |    2 |   91 | 1-0     | 1-0
       22 | 2014-05-04 15:00:00 |    2 |   93 | 1-0     | 1-0
       25 | 2020-10-24 13:00:00 |    3 |   75 | 1-1     | 1-0
       25 | 2020-10-24 12:30:00 |    4 |   76 | 6-2     | 2-2
       25 | 2020-10-24 15:00:00 |    5 |   87 | 1-4     | 1-2
       25 | 2020-10-25 11:00:00 |    6 |   89 | 0-3     | 0-1
       25 | 2020-10-25 13:00:00 |    7 |   91 | 3-0     | 1-0
       30 | 2020-10-26 13:00:00 |    8 |   88 | 4-2     | 2-2
(16 rows)
```

Examples with Duplicate Rows

The following examples demonstrate the behavior of different types of `ybload` operations when the `--duplicate-handler` option is used.

UPSERT Example

For example, table `five_seasons` contains:

```
premdb=# select * from five_seasons;
 seasonid | season_name | numteams |      winners
----------+-------------+----------+-------------------
       1 | 1992-1993   |       22 | Manchester United
       2 | 1993-1994   |       22 | Manchester United
       3 | 1994-1995   |       22 | Blackburn Rovers
       4 | 1995-1996   |       20 | Manchester United
       5 | 1996-1997   |       20 | Manchester United
(5 rows)
```

Assume that a source file contains the following 15 records. Note that there are duplicate records for `seasonid 3` , `6` , and `10` . The `winners` column values are different in these duplicate rows.

```
0,1991-1992,22,
1,1992-1993,22,Manchester United
2,1993-1994,22,Manchester United
3,1994-1995,22,Blackburn Rovers
3,1994-1995,22,Blackburn Rovers FC
4,1995-1996,20,Manchester United
5,1996-1997,20,Manchester United
6,1997-1998,20,Arsenal
6,1997-1998,20,Arsenal FC
7,1998-1999,20,Manchester United
8,1999-2000,20,Manchester United
9,2000-2001,20,Manchester United
10,2001-2002,20,Arsenal
10,2001-2002,20,Arsenal FC
30,2020-2021,20,
```

If it does not matter which of the duplicates is used for key matching, you can set `--duplicate-handler` to `none` or `random` . If you want a deterministic approach to matching duplicates, use an `ORDER BY` clause in the `ybload` command. For example:

```
--write-op upsert --duplicate-handler "order by winners asc"
```

For the 15 records shown here, `order by winners asc` will produce different results from `order by winners desc` . Ascending order means that `Arsenal` sorts above `Arsenal FC` , and `Blackburn Rovers` sorts above `Blackburn Rovers FC` .

If `order by winners asc` is used, the rows where `winners` values do not end in `FC` will be the chosen duplicate rows. The results of an `UPSERT` load will be:

```
premdb=# select * from five_seasons order by 1 asc;
 seasonid | season_name | numteams |      winners
----------+-------------+----------+-------------------
       0 | 1991-1992   |       22 | [NULL]
       1 | 1992-1993   |       22 | Manchester United
       2 | 1993-1994   |       22 | Manchester United
       3 | 1994-1995   |       22 | Blackburn Rovers
       4 | 1995-1996   |       20 | Manchester United
       5 | 1996-1997   |       20 | Manchester United
       6 | 1997-1998   |       20 | Arsenal
       7 | 1998-1999   |       20 | Manchester United
       8 | 1999-2000   |       20 | Manchester United
       9 | 2000-2001   |       20 | Manchester United
      10 | 2001-2002   |       20 | Arsenal
      30 | 2020-2021   |       20 | [NULL]
(12 rows)
```

If `order by winners desc` is used instead for the same `upsert` load, the `winners` values that end in `FC` will be chosen for `seasonid 3`, `6`, and `10`.

Note that you cannot use a declared key column as an `ORDER BY` column in the `ybload` command.

See also ybload Options.

INSERT Example

For example, consider the following source file, which contains 16 records, including three rows with a `seasonid` of `6`:

```
0,1991-1992,22,
1,1992-1993,22,Manchester United
2,1993-1994,22,Manchester United
3,1994-1995,22,Blackburn Rovers
3,1994-1995,22,Blackburn Rovers FC
4,1995-1996,20,Manchester United
5,1996-1997,20,Manchester United
6,1997-1998,20,Arsenal
6,1997-1998,20,Arsenal FC
6,1997-1998,20,Arsenal FC
7,1998-1999,20,Manchester United
8,1999-2000,20,Manchester United
9,2000-2001,20,Manchester United
10,2001-2002,20,Arsenal
10,2001-2002,20,Arsenal FC
30,2020-2021,20,
```

Before the `insert` load, table `five_seasons` contains:

```
premdb=# select * from five_seasons;
 seasonid | season_name | numteams |      winners
----------+-------------+----------+-------------------
       1 | 1992-1993   |       22 | Manchester United
       2 | 1993-1994   |       22 | Manchester United
       3 | 1994-1995   |       22 | Blackburn Rovers
       4 | 1995-1996   |       20 | Manchester United
       5 | 1996-1997   |       20 | Manchester United
(5 rows)
```

An `insert` load without duplicate handling results in 21 rows:

```
premdb=# select * from five_seasons order by 1;
 seasonid | season_name | numteams |       winners
----------+-------------+----------+--------------------
        0 | 1991-1992   |       22 | [NULL]
        1 | 1992-1993   |       22 | Manchester United
        1 | 1992-1993   |       22 | Manchester United
        2 | 1993-1994   |       22 | Manchester United
        2 | 1993-1994   |       22 | Manchester United
        3 | 1994-1995   |       22 | Blackburn Rovers
        3 | 1994-1995   |       22 | Blackburn Rovers
        3 | 1994-1995   |       22 | Blackburn Rovers FC
        4 | 1995-1996   |       20 | Manchester United
        4 | 1995-1996   |       20 | Manchester United
        5 | 1996-1997   |       20 | Manchester United
        5 | 1996-1997   |       20 | Manchester United
        6 | 1997-1998   |       20 | Arsenal
        6 | 1997-1998   |       20 | Arsenal FC
        6 | 1997-1998   |       20 | Arsenal FC
        7 | 1998-1999   |       20 | Manchester United
        8 | 1999-2000   |       20 | Manchester United
        9 | 2000-2001   |       20 | Manchester United
       10 | 2001-2002   |       20 | Arsenal
       10 | 2001-2002   |       20 | Arsenal FC
       30 | 2020-2021   |       20 | [NULL]
(21 rows)
```

The same load with `--duplicate-handler "order by winners asc"` results in 17 rows:

```
premdb=# select * from five_seasons order by 1;
 seasonid | season_name | numteams |      winners
----------+-------------+----------+-------------------
        0 | 1991-1992   |       22 | [NULL]
        1 | 1992-1993   |       22 | Manchester United
        1 | 1992-1993   |       22 | Manchester United
        2 | 1993-1994   |       22 | Manchester United
        2 | 1993-1994   |       22 | Manchester United
        3 | 1994-1995   |       22 | Blackburn Rovers
        3 | 1994-1995   |       22 | Blackburn Rovers
        4 | 1995-1996   |       20 | Manchester United
        4 | 1995-1996   |       20 | Manchester United
        5 | 1996-1997   |       20 | Manchester United
        5 | 1996-1997   |       20 | Manchester United
        6 | 1997-1998   |       20 | Arsenal
        7 | 1998-1999   |       20 | Manchester United
        8 | 1999-2000   |       20 | Manchester United
        9 | 2000-2001   |       20 | Manchester United
       10 | 2001-2002   |       20 | Arsenal
       30 | 2020-2021   |       20 | [NULL]
(17 rows)
```

The only duplicates that are discarded are the rows with FC in the winners column. The duplicates for `seasonid` values 1 through 5 are loaded. They were already in the table and are not affected by the duplicate handling of new source rows.

Running a Bulk Load

Platforms: All platforms

Parent topic: Bulk Loading Tables

In this section:

This section summarizes the steps required to run a bulk load into a Yellowbrick table. For more details, see ybload Command and ybload Options.

To run a bulk load:

1. Make sure the target Yellowbrick database is up and running and the `ybload` client is installed (as part of the `ybtools` package).
2. Create the table in the database before running the load operation. Use the CREATE TABLE statement (or CREATE TABLE AS or SELECT INTO).
3. Determine the exact location and format of the source file (or set of source files) for the load. The following source types are supported:

- Files local to the client system where you are running `ybload`
- `stdin`
- Named pipes
- Amazon Simple Storage Service (Amazon S3) files. See Loading from Amazon S3.
- Azure Blob object stores. See Loading from Azure Blob Storage. **Note:** Try to export the data from the source system in the character set that you used when you created the Yellowbrick database. For example, load exported UTF8 data into a UTF8 database. Any required character set translation is likely to have an impact on import performance during the load.

4. From a client system where the `ybtools` clients are installed, run the `ybload` command, *as a user account that is not a superuser*. Superuser accounts may not have sufficient memory to run bulk load operations.

You must define a few essential pieces of information. On an as-needed basis, you can also specify a range of other processing, logging, and parsing options.

```
ybload [options] -t [SCHEMA.]TABLENAME SOURCE [SOURCE]...
```

- `[options]`: You will need to set some database connection values if they are not already set with environment variables. Other options depend on the nature of the data, the type of load operation you want to run, and related formatting requirements. Consider using the `--dryrun` option for a new load operation; this will help you determine the set of options you need in order to load the table successfully. If you want to delete, update, or upsert rows, specify the `--write-op` option. For more details, see ybload Options or the online help text.

  Important: Setting the `--format` option is strongly recommended for all load operations.

- `TABLENAME` : the target table, including its schema name (optional but recommended). If you do not specify the schema name, the table is assumed to be in the `public` schema. The schema of the target table is not based on the user's `search_path` , regardless of how it is set.

  Important: The user who runs the load must have `BULK LOAD` permissions on the database and INSERT permissions on the table (but not ownership of the table).

- `SOURCEFILE` : one or more source files and their path (locally or on a supported object storage site). Each source file should be a flat file with a consistent field delimiter (comma-separated or tab-delimited, for example) or a `parquet` file. Use spaces to separate multiple file references. You can also use wildcard characters (such as `*`) to refer to multiple files or a whole directory of files. Single quotes around source file names are optional.

  For flat files, compressed `gzip` , `bzip2` , `xz` , `pack200` , and `lz4` formats may be detected based on their file extensions:

  - `.gz` = `gzip`
  - `.bzip2` or `.bz2` = `bzip`
  - `.xz` = `xz`
  - `.pack200` = `pack200`
  - `.lz4` = `lz4` `ybload` can also detect compression headers within regular compressed files, but the file extension mapping overrides this header detection. The `--source-compression` option, if specified, overrides both of these detection methods.

  Note: `lz4` compression is not as compact as `gzip` but takes less time to both compress and decompress.

  For `parquet` files, compression is detected automatically based on metadata in the files.

  At a minimum, each source file should contain fields for every non-nullable column in the table. Files do not have to match the target table exactly, one to one, in terms of fields mapping to columns and the order of the fields. Options are available to work around these mismatches. If the fields in your source file do not align with the columns in the target table, use either `--field-names` or `--parse-header-line` to map fields to columns.

  Note: To load columns with `DEFAULT` values, use the `--field-names` option and omit those columns from the list. See ybload Options.

  Empty source files do not prevent bulk loads from proceeding. If `ybload` encounters an empty source file, the load operation skips to the next file, if any.

- To load from `stdin` (standard input, an unnamed pipe), enter the single-dash character (-) at the end of the `ybload` command. This must be the last character on the command line, and it must be prefixed with -- to indicate that option parsing is complete. For example: `./ybload -d premdb --username bobr -t match``--` -

- If your workflow involves multiple processes communicating through named pipes, `ybload` can consume directly from your producer processes. Simply specify the named pipes on the `ybload` command line in the same way that you would name regular files. Mixing pipe and non-pipe data sources for a load is not recommended; this could cause a deadlock with the program that the pipes communicate with.

Loading Tables from Parquet Files

Platforms: All platforms

Parent topic: Running a Bulk Load

In this section:

This section explains how to load a table from Apache Parquet source files (a structured, columnar storage format). Certain load options and parameters that work for flat files are not supported for `parquet` format loads, and a few options are specific to `parquet` loads.

Parquet Schema Support

Before attempting to load `parquet` data into a Yellowbrick table:

- Download `parquet-tools` (Apache Parquet command-line tools and utilities) so that you have a convenient way to inspect the schema of specific files, including data types, input column names, and the structure of individual fields.

These tools are not provided by Yellowbrick; you can download them from various sites. For example, for macOS clients, go to: https://formulae.brew.sh/formula/parquet-tools

- Make sure that the target column names in the table DDL match the names in the `parquet` schema. Source fields (input columns) and target table columns (DDL column names) are matched by *name*.
- Check the data types used in the source files and the overall structure of the data. Native `parquet` data types are automatically mapped (and cast) to the standard set of Yellowbrick data types; you do not need to specify any mapping. See Parquet Schema Mapping and Type Casting for details.

However, if your source files contain unsupported types or a schema that `ybload` does not recognize, by default `ybload` returns expected errors. Certain primitive types, logical annotations, and nested structures are not supported:

- `INTERVAL` logical type
- Nested fields (nested data is not loaded by default, but can be serialized and loaded). For example:

```
message schema {
  optional group employee {
   optional int64 age;
  }
}
```

- Nested `LIST` and `MAP` logical types
- Fields with a repetition level of `repeated` that occurs *outside* the context of a `LIST` logical type. For example, the following schema is not supported:

  ```
  message schema {
      repeated int64 int64_list;
  }
  ```

  However, the following schema is supported:

```
message schema {
  optional group int64_list (LIST) {
    repeated group list {
      optional int64 item;
    }
  }
}
```

In addition, the Apache Parquet specification is constantly evolving, and various implementations exist in the field, some of which do not follow the specification strictly. In turn, `ybload` may not support certain source files.

The `--ignore-unsupported-schema` option is a means of bypassing errors for unsupported types and schemas. This option takes effect when the Parquet metadata is read, and removes the unsupported fields from the source file schema *before* any mapping is done. You need to be aware of the implications of this behavior. For example, if a field is ignored in the `parquet` file, but required (declared `NOT NULL`) in the table, `ybload` returns an error.

Although nested data is not supported by default, the `--serialize-nested-as-json` option serializes nested `parquet` data in JSON format so that it can be loaded. Serialized JSON strings can be loaded into `VARCHAR` columns.

- Always specify the `--format parquet` option in the `ybload` command. No other special `parquet` format parameters are required.

Parquet Date and Time Conversion

Platforms: All platforms

Parent topic: Loading Tables from Parquet Files

This section shows how date and time logical types are converted and formatted as `VARCHAR` columns when `parquet` source files are loaded into Yellowbrick tables. These conversions conform to the ISO-8601 standard.

The `TIME` and `TIMESTAMP` conversions produce the shortest string that contains the full time value, with any omitted parts implied to be zero.

Parquet Type	VARCHAR Column Format	Examples
`DATE`	`uuuu-MM-dd`	`2022-03-10`

`TIME`	`HH:mm`
	`HH:mm:ss`
	`HH:mm:ss.SSS`
	`HH:mm:ss.SSSSSS`
	`HH:mm:ss.SSSSSSSSS`
		`17:48:17.481`

Parquet Type	VARCHAR Column Format	Examples

`TIMESTAMP`	`uuuu-MM-dd'T'HH:mmXXXXX`
	`uuuu-MM-dd'T'HH:mm:ssXXXXX`
	`uuuu-MM-dd'T'HH:mm:ss.SSSXXXXX`
	`uuuu-MM-dd'T'HH:mm:ss.SSSSSSXXXXX`
	`uuuu-MM-dd'T'HH:mm:ss.SSSSSSSSSXXXXX`
		`2022-03-10T17:52:26.219Z`
		`2022-02-17T23:29:57.713215Z`

`TIMESTAMP`	`uuuu-MM-dd'T'HH:mm`
	`uuuu-MM-dd'T'HH:mm:ss`
	`uuuu-MM-dd'T'HH:mm:ss.SSS`
	`uuuu-MM-dd'T'HH:mm:ss.SSSSSS`
	`uuuu-MM-dd'T'HH:mm:ss.SSSSSSSSS`
		`2022-03-10T17:52:26.260`

Parquet Load Examples

Platforms: All platforms

Parent topic: Loading Tables from Parquet Files

This section shows a few examples of `ybload` commands that load from `parquet` source files.

Simple parquet load from the local file system

The following command loads the `match` table from a local file called `match.snappy.parquet`:

```
$ ybload -d premdb --username yellowbrick -t match -W --format parquet
--logfile /tmp/match.log --logfile-log-level info /tmp/match.snappy.parquet
Password for user yellowbrick:
...
10:15:40.089 [ INFO] Logfile written to /tmp/match.log
10:15:40.142 [ INFO] Gathering metadata on input files
10:15:40.658 [ INFO] Bad rows will be written to /tmp/match.snappy.parquet.20211215101540.bad
...
10:15:41.151 [ INFO] Flushing last 8606 rows (of 8606 total) in transaction #1 for match
10:15:41.159 [ INFO] Committing 8606 rows into transaction #1 for match ...
10:15:41.186 [ INFO] Committed transaction #1 after a total of 292604 bytes and 8606 good rows for match
10:15:41.201 [ INFO] READ:41.07KB(17.49KB/s). ROWS G/B: 8606/0( 3.58K/s). WRITE:285.7KB(121.7KB/s).  TIME E/R:   0:00:02/ --:--:--
10:15:41.202 [ INFO] SUCCESSFUL BULK LOAD: Loaded 8606 good rows from 1 source(s) in   0:00:02 (READ: 17.49KB/s WRITE: 121.7KB/s)
```

Loading a table when source and target column names do not align

This example demonstrates a potential problem (and a solution) when the `parquet` schema and the DDL for the target table do not have identical column names. Here is the schema of a table called `newmatch`:

```
premdb=# \d newmatch
              Table "public.newmatch"
   Column    |            Type             | Modifiers
-------------+-----------------------------+-----------
 seasonid    | smallint                    |
 newmatchday | timestamp without time zone |
 htid        | smallint                    |
 atid        | smallint                    |
 ftscore     | character(3)                |
 htscore     | character(3)                |

Distribution: Hash (seasonid)
```

Here is the schema of the `match.snappy.parquet` file:

```
% parquet-tools schema match.snappy.parquet
message spark_schema {
  optional int32 seasonid;
  optional binary matchday (STRING);
  optional int32 htid;
```

```
  optional int32 atid;
  optional binary ftscore (STRING);
  optional binary htscore (STRING);
}
```

Note that the second column `newmatchday` in the table is called `matchday` in the `parquet` file. Consequently, when you load `newmatch` from this file, the load completes but puts `NULL` values into the `newmatchday` column:

```
premdb=# select * from newmatch where seasonid=12 order by 1 limit 5;
 seasonid | newmatchday | htid | atid | ftscore | htscore
----------+-------------+------+------+---------+---------
       12 | [NULL]      |   42 |   75 | 1-2     | 0-2
       12 | [NULL]      |   42 |   76 | 0-0     | 0-0
       12 | [NULL]      |   42 |   77 | 1-0     | 0-0
       12 | [NULL]      |   42 |   81 | 4-3     | 2-1
       12 | [NULL]      |   42 |   86 | 1-3     | 0-2
(5 rows)
```

The solution in this case is to use the `--field-defs` option to override the default field names specified by the source file. You must specify all of the source fields, but you can rename `matchday` as `newmatchday` so that the target table accepts it.

```
$ ybload -d premdb --username bobr -W -t newmatch_matchday_not_null -W --format parquet
--field-defs seasonid,newmatchday,htid,atid,ftscore,htscore
/home/brumsby/premdb-snappy-parquet/match.snappy.parquet
Password for user bobr:
...
```

Parquet load with nested data inserted as a JSON string

The following example loads a source file with the following schema to demonstrate how nested data structures in `parquet` data can be serialized into a single column in the target table:

```
% parquet-tools schema person_data_1.parquet
message schema {
  optional binary name (STRING);
  optional int64 age;
  optional group languages {
   optional binary mother_language (STRING);
   optional group other_languages (LIST) {
     repeated group list {
       optional binary item (STRING);
     }
   }
  }
}
```

The target table for the load, `person_lang_not_null_default`, has the following DDL:

```
premdb=# \d person_lang_not_null_default;
          Table "public.person_lang_not_null_default"
  Column   |          Type          |         Modifiers
-----------+------------------------+---------------------------
 name      | character varying(25)  |
 age       | smallint               |
 languages | character varying(100) | not null default 'English'
```

Note that the third column, `languages`, is declared `NOT NULL` and has a default value. This column is intended to map to the nested `languages` structure in the `parquet` file, which consists of `languages`, `mother_language`, and `other_languages` values.

The following `ybload` command uses the `--serialize-nested-as-json` option to load the nested data as a JSON string into a single `VARCHAR` column:

```
$ ./ybload -d premdb -t person_lang_not_null_default -W --username bobr
--format parquet --serialize-nested-as-json /home/bobr/pqdata/person_data_1.parquet
Password for user bobr:
...
15:46:02.550 [ INFO] Committed transaction #1 after a total of 260 bytes and 3 good rows for person_lang_not_null_default
15:46:02.562 [ INFO] READ: 1.80KB( 1.63KB/s). ROWS G/B: 3/0( 2.73 /s). WRITE:260.0 B(236.2 B/s).  TIME E/R:   0:00:01/ --:--:--
15:46:02.562 [ INFO] SUCCESSFUL BULK LOAD: Loaded 3 good rows from 1 source(s) in   0:00:01 (READ:  1.63KB/s WRITE: 236.2 B/s)
```

After the load, the table contains these three rows:

```
premdb=# select * from person_lang_not_null_default;
 name | age |                                languages
------+-----+-------------------------------------------------------------------------------
 Tom  |  35 | {"mother_language":"Spanish","other_languages":["English","German","French"]}
 Bob  |  25 | {"mother_language":"English","other_languages":["German","French"]}
 Ruby |  30 | {"mother_language":"English","other_languages":["German","French"]}
(3 rows)
```

The same `ybload` command *without* the `--serialize-nested-as-json` option would cause the load to fail:

```
$ ./ybload -d premdb -t person_lang_not_null_default -W --username bobr
--format parquet /home/bobr/pqdata/person_data_1.parquet
Password for user bobr:
...
15:54:54.851 [ INFO] Gathering metadata on input files
15:54:55.235 [FATAL] Parquet Error: Nested structure not supported, at field:
optional group languages {
  optional binary mother_language (STRING);
  optional group other_languages (LIST) {
   repeated group list {
     optional binary item (STRING);
   }
  }
}
```

See the following example for details about working around this issue by ignoring the unsupported nested schema.

Parquet load with nested data, ignoring unsupported data types

This example uses the same source file and target table as the previous example. In this case, the `--ignore-unsupported-schema` is used to ignore the nested data and instead load the third column of the table with its declared `DEFAULT` value.

```
$ ./ybload -d premdb -t person_lang_not_null_default -W --username bobr
--format parquet --ignore-unsupported-schema /home/bobr/pqdata/person_data_1.parquet
Password for user bobr:
...
16:32:19.326 [ INFO] SUCCESSFUL BULK LOAD: Loaded 3 good rows from 1 source(s) in   0:00:01 (READ:  1.59KB/s WRITE: 40.65 B/s)
```

The following rows are loaded:

```
premdb=# select * from person_lang_not_null_default;
 name | age | languages
------+-----+-----------
 Bob  |  25 | English
 Ruby |  30 | English
 Tom  |  35 | English
(3 rows)
```

If the `languages` column does not have a `DEFAULT` value *and* cannot be `NULL`, the same load fails:

```
premdb=# \d person_lang_not_null
     Table "public.person_lang_not_null"
  Column   |          Type          | Modifiers
-----------+------------------------+-----------
 name      | character varying(25)  |
 age       | smallint               |
 languages | character varying(100) | not null

...
$ ./ybload -d premdb -t person_lang_not_null -W --username bobr --format parquet --ignore-unsupported-schema /home/bobr/pqdata/per
Password for user bobr:
...
16:43:27.146 [ INFO] Gathering metadata on input files
16:43:27.539 [FATAL] No source field found for the following destination columns:
   languages
Source Fields:
   name
   age
Destination Columns:
   name
   age
   languages
```

If the `languages` column *is nullable*, the same load succeeds, and `NULL` values are loaded into the `languages` column:

```
premdb=# \d person_lang
          Table "public.person_lang"
  Column   |          Type          | Modifiers
-----------+------------------------+-----------
 name      | character varying(25)  |
 age       | smallint               |
 languages | character varying(100) |

...
$ ./ybload -d premdb -t person_lang -W --username bobr
--format parquet --ignore-unsupported-schema /home/bobr/pqdata/person_data_1.parquet
Password for user bobr:
...
16:50:24.463 [ INFO] SUCCESSFUL BULK LOAD: Loaded 3 good rows from 1 source(s) in   0:00:01 (READ:  1.64KB/s WRITE: 41.95 B/s)
...
premdb=# select * from person_lang;
 name | age | languages
------+-----+-----------
 Tom  |  35 | [NULL]
 Bob  |  25 | [NULL]
 Ruby |  30 | [NULL]
(3 rows
```

Parquet Schema Mapping and Type Casting

This section lays out the mapping and casting support for `parquet` types to Yellowbrick data types (data types supported for storage in columns in Yellowbrick tables).

Mapping for Parquet Boolean Type

The `parquet boolean` type maps directly to the Yellowbrick `boolean` data type. No other mappings are supported.

Mappings for Parquet INT32 Types

The following table indicates which `parquet INT32` data types map to Yellowbrick data types, either directly or with casting.

The first row in the table refers to the `parquet` primitive type, and the subsequent rows to annotated logical types.

	CHAR, VARCHAR	BOOLEAN	SMALLINT	INT	BIGINT	REAL	DOUBLE	DECIMAL	DATE	TIM
INT32	Yes, with cast	No	Yes, with cast	Yes	Yes, with cast	Yes, with cast	Yes, with cast	Yes, with cast	No	No
INT/UINT (8/16/32, sign)	Yes, with cast	No	Yes: INT(8), UINT(8), INT(16)	Yes: UINT(16),INT(32)	Yes: UINT(32)	Yes, with cast	Yes, with cast	Yes, with cast	No	No
DECIMAL (1-9)	Yes, with cast	No	Yes, with cast	Yes, with cast	Yes, with cast	Yes, with cast	Yes, with cast	Yes	No	No
DATE	Yes, with cast	No	No	No	No	No	No	No	Yes	No
TIME (MILLIS)	Yes, with cast	No	No	No	No	No	No	No	No	Yes

Mappings for Parquet INT64 Types

The following table indicates which `parquet INT64` data types map to Yellowbrick data types, either directly or with casting.

The first row in the table refers to the `parquet` primitive type, and the subsequent rows to annotated logical types.

	CHAR, VARCHAR	BOOLEAN	SMALLINT	INT	BIGINT	REAL	DOUBLE	DECIMAL	DATE	TIME	TIM
INT64	Yes, with cast	No	Yes, with cast	Yes, with cast	Yes, with cast	Yes, with cast	Yes, with cast	Yes, with cast	No	No	No

	CHAR, VARCHAR	BOOLEAN	SMALLINT	INT	BIGINT	REAL	DOUBLE	DECIMAL	DATE	TIME	TIM
INT/UINT(64, sign)	Yes, with cast	No	Yes, with cast	Yes, with cast	Yes: INT(64)	Yes, with cast	Yes, with cast	Yes: UINT(64)	No	No	No
DECIMAL (1-18)	Yes, with cast	No	Yes, with cast	Yes, with cast	Yes, with cast	Yes, with cast	Yes, with cast	Yes	No	No	No
TIME (MICROS, NANOS)	Yes, with cast	No	No	No	No	No	No	No	No	Yes	No
TIMESTAMP (UTC, unit)	Yes, with cast	No	No	No	No	No	No	No	Yes, with cast	Yes, with cast	Yes

Mapping for Parquet FLOAT, DOUBLE, and INT96 Types

The following table indicates which `parquet FLOAT`, `DOUBLE`, and `INT96` data types map to Yellowbrick data types, either directly or with casting.

	CHAR, VARCHAR	BOOLEAN	SMALLINT	INT	BIGINT	REAL	DOUBLE	DECIMAL	DATE	TIME	TIMES
FLOAT	Yes, with cast	No	No	No	No	Yes	Yes, with cast	No	No	No	No
DOUBLE	Yes, with cast	No	No	No	No	Yes, with cast	Yes	No	No	No	No
INT96 (--int96-as-timestamp)	Yes, with cast	No	No	No	No	No	No	No	Yes, with cast	Yes, with cast	Yes
INT96 (--no-int96-as-timestamp)	Yes, with cast	No	Yes, with cast	Yes, with cast	Yes, with cast	Yes, with cast	Yes, with cast	Yes	No	No	No

Mapping for Parquet Byte Array Types

The following table indicates which `parquet` byte array data types map to Yellowbrick data types, either directly or with casting.

The first row in the table refers to the `parquet` primitive type, and the subsequent rows to annotated logical types.

	CHAR, VARCHAR	BOOLEAN	SMALLINT	INT	BIGINT	REAL	DOUBLE	DECIMAL	DATE	TIME	TIM
BYTE ARRAY	Yes	No	No	No	No	No	No	No	No	No	No
STRING/UTF-8	Yes	Yes, with cast	Yes, with cast	Yes, with cast	Yes, with cast	Yes, with cast	Yes, with cast	Yes, with cast	Yes, with cast	Yes, with cast	Yes
ENUM	Yes	No	No	No	No	No	No	No	No	No	No

	CHAR, VARCHAR	BOOLEAN	SMALLINT	INT	BIGINT	REAL	DOUBLE	DECIMAL	DATE	TIME	TIM
DECIMAL(N)	Yes, with cast	No	Yes, with cast	Yes, with cast	Yes, with cast	Yes, with cast	Yes, with cast	Yes	No	No	No
JSON	Yes	No	No	No	No	No	No	No	No	No	No
BSON	Yes	No	No	No	No	No	No	No	No	No	No

Note: When loading data from parquet bytea array to a bytea column or varchar column, leading and trailing whitespaces are preserved.

Mapping for Parquet Fixed-Length Byte-Array Types

The following table indicates which `parquet` fixed-length byte-array data types map to Yellowbrick data types, either directly or with casting.

The first row in the table refers to the `parquet` primitive type, and the subsequent rows to annotated logical types.

	CHAR, VARCHAR	BOOLEAN	SMALLINT	INT	BIGINT	REAL	DOUBLE	DECIMAL	DATE	TIME	TI
FIXED-LENGTH BYTE-ARRAY	Yes	No	No	No	No	No	No	No	No	No	N
16/UUID	Yes, with cast	No	No	No	No	No	No	No	No	No	N
N/DECIMAL(N)	Yes, with cast	No	Yes, with cast	Yes, with cast	Yes, with cast	Yes, with cast	Yes, with cast	Yes	No	No	N
12/Interval	No	No	No	No	No	No	No	No	No	No	N

Analyzing Rejected Rows

This section describes some common cases in which rows are rejected and explains how to interpret and fix the errors. Common error conditions include:

- Incorrect field delimiter specified in the `ybload` command
- Incorrect line separator specified in the `ybload` command
- Mismatch between number of columns in table and number of fields in input data
- Mismatch between data types in table and values in input data fields
- Blank lines in source files (or unexpected header/footer information)
- Out-of-range values in source files (for numeric data)
- Incorrect format for dates and timestamps
- Incorrect format for `NULL` values

Incorrect Number of Fields

The following example is a simple case where the number of fields in the source file (6) is greater than the number of columns in the target table (5).

```
$ more match.csv.20160707112858.bad
# Bulk Loading /home/premdata/match.csv into TABLE matchstats at 2016-07-07T11:28:58.607-07:00[America/Los_Angeles]

#error: lineByteRange(0-35)
#reason: Too many fields (6 > 5)
1,1992-08-01 00:00:00,2,52,0-1,-

#error: lineByteRange(35-70)
#reason: Too many fields (6 > 5)
1,1992-08-01 00:00:00,2,55,0-1,-

#error: lineByteRange(70-105)
#reason: Too many fields (6 > 5)
1,1992-08-01 00:00:00,2,63,2-1,-

...
```

To fix this load, you can modify the target table, modify the source file, or use the `--csv-allow-too-many-fields` option in the `ybload` command.

Incorrect Field Delimiter

Note the location in the row (`...<---`) where the error is occurring. The arrow points to a sequence of dots (`...`), which represent the extent of the field that generated the error.

In this case, the problem is the field delimiter for the rows, not the date format of the first column. The load was attempted with a default comma delimiter, but the source file uses a pipe character as its delimiter. The bulk loader ignores the pipe characters, attempts to read all five columns as a single date column, and fails (predictably) on the date format.

The `fieldinfo` line in the output contains the following information:

- `destName`: the name of the destination table column that the field in the source file is being parsed into.
- `destType`: the data type of the destination table column.

```
#error: lineByteRange(810-890):fieldByteRange(1-75)
#reason: Doesn't match any of the allowed date formats: YMD,DMONY,MONDY
#fieldinfo: srcIndex:1 destName:matchday destType:DATE  options:{"formats":[{"style":"YMD","delim":"auto"},{"style":"DMONY","delim
"delim":"auto"}]}
#.....................................................................<--------- Doesn't match any of the allowed date format
 1993-08-14 | Liverpool              | Sheffield Wednesday     | 2-0     |

#error: lineByteRange(890-970):fieldByteRange(1-75)
#reason: Doesn't match any of the allowed date formats: YMD,DMONY,MONDY
#fieldinfo: srcIndex:1 destName:matchday destType:DATE  options:{"formats":[{"style":"YMD","delim":"auto"},{"style":"DMONY","delim
"delim":"auto"}]}
#.....................................................................<--------- Doesn't match any of the allowed date format
 1993-08-14 | Southampton            | Everton                 | 0-2     |
```

To fix this load, add the delimiter option to the `ybload` command: `--csv-delimiter '|'`

Incorrect Date Format

In this example, the incoming dates do not match any of the default formats. See ybload Date Formats.

```
#error: lineByteRange(0-58):fieldByteRange(38-56)
#reason: Doesn't match any of the allowed date formats: YMD,DMONY,MONDY
#fieldinfo: srcIndex:10 destName:saletime destType:TIMESTAMP  options:{"dateformats":[{"style":"YMD","delim":"auto"},{"style":"DMO
#                                     .................<--------- Doesn't match any of the allowed date formats: YMD,DMONY,MONDY
1|1|36861|21191|7872|1875|4|728|109.2|2/18/2008 02:36:48

#error: lineByteRange(571-628):fieldByteRange(38-55)
#reason: Doesn't match any of the allowed date formats: YMD,DMONY,MONDY
#fieldinfo: srcIndex:10 destName:saletime destType:TIMESTAMP  options:{"dateformats":[{"style":"YMD","delim":"auto"},{"style":"DMO
#                                     .................<--------- Doesn't match any of the allowed date formats: YMD,DMONY,MONDY
11|12|45635|8435|4769|2042|2|130|19.5|8/4/2008 03:06:36
```

Incorrect Format for NULL Values

In this example, the parser does not recognize that the `\N` string is intended to be the null string. The parser expects an integer value for the column.

```
#error: lineByteRange(7035-7069):fieldByteRange(30-32)
#reason: Non-digit character
#fieldinfo: srcIndex:5 destName:venueseats destType:INTEGER  options:{}
#                             ..<--------- Non-digit character
250|Sahara Hotel|Las Vegas|NV|\N

#error: lineByteRange(7136-7170):fieldByteRange(30-32)
#reason: Non-digit character
#fieldinfo: srcIndex:5 destName:venueseats destType:INTEGER  options:{}
#                             ..<--------- Non-digit character
253|Mirage Hotel|Las Vegas|NV|\N
```

This example can be fixed with the `--nullmarker` option:

```
$ ybload --csv-delimiter '|' --nullmarker '\N' -t venue /home/data/venue.tbl
...
```

Fields Not Allowed To Be Null

In this example, the DDL for the table declared that some Boolean columns could not be null:

```
#error: lineByteRange(1324-1403):fieldByteRange(58-58)
#reason: Field not allowed to be null
#fieldinfo: srcIndex:9 destName:likesports destType:BOOLEAN NOT NULL  options:{}
#                                                    <--------- Field not allowed to be null
12|FVK28WAS|Bruce|Beck|Kona|OH|ac@velit.ca|(617) 527-9908|||FALSE|||FALSE||||

#error: lineByteRange(135-267):fieldByteRange(100-100)
#reason: Field not allowed to be null
#fieldinfo: srcIndex:9 destName:likesports destType:BOOLEAN NOT NULL  options:{}
#                                                                                                  <--------- Field not allowed t
2|PGL08LJI|Vladimir|Humphrey|Murfreesboro|SK|Suspendisse.tristique@nonnisiAenean.edu|(783) 492-1886|||TRUE|TRUE|||TRUE|FALSE|TRUE

#error: lineByteRange(1403-1532):fieldByteRange(102-102)
#reason: Field not allowed to be null
#fieldinfo: srcIndex:9 destName:likesports destType:BOOLEAN NOT NULL  options:{}
#                                                                                                    <--------- Field not allowed
13|QTF33MCG|Henry|Cochran|Bossier City|QC|Aliquam.vulputate.ullamcorper@amalesuada.org|(783) 105-0989||TRUE|||||TRUE|TRUE|TRUE|
```

Deletes, Updates, and Upserts

In addition to default append (or `insert`) loads, you can use `ybload` to delete, update, or "upsert" the rows in a table. You can set the `--write-op` option to one of the following settings:

- `insert`: Default "append" behavior occurs.
- `delete`: Rows in the table that match rows in source files are deleted.
- `update`: Rows in the table that match rows in source files are updated (column values are modified as needed).
- `upsert`: Rows in the table that match rows in source files are updated, and rows in the source file that do not already exist in the table are inserted as new rows (appended). This mode combines the ybload `insert` and `update` modes into one operation.

Note: When rows are updated, every qualifying row for the update is written to the database, whether the contents of the row changed or not. `ybload` does not distinguish between matching rows that contain changed column values and matching rows that are exactly the same as existing rows. All updated rows are written to the database. Therefore, it is important to use `update` and `upsert` loads only when you know the source data includes a significant number of updated rows (rows in which at least one column value changes).

Deletes, updates, and upserts all depend on key fields for matching target table rows with source file rows, as explained in the following section.

Key Fields

The source files for a load are read in order to find rows that should be deleted, updated, or updated *and* inserted (upserted). One or more designated key fields are used to match table rows to source file records. If the target table for a non-insert load does not have a primary key, the `ybload` operation must specify one or more key fields explicitly in the command line, using either the `--key-field-names` option or the `--field-defs` option. For example:

```
--key-field-names seasonid, season_name
```

or:

```
--field-defs seasonid int primary key
```

See also Delete, Update, and Upsert Examples and Examples with Duplicate Rows.

Handling Duplicate Rows

When the `--write-op` option is set to `insert`, `update`, or `upsert`, a decision has to be made about matching rows when duplicates are found in the source files. If duplicates exist, `ybload` matches on one of them in order to insert, update, or upsert the row in question.

Yellowbrick tables do not enforce primary key and uniqueness constraints; therefore, duplicate rows are generally allowed to be bulk-loaded or inserted. For all loads, duplicate handling applies to new source rows; existing duplicate rows in the table are not affected. For `update` and `upsert` loads, *rows in the source file* may be deduplicated, depending on the duplicate-handling method that you choose.

The `--duplicate-handler` option provides different ways to manage duplicates:

- `order by` (SQL clause): source rows are sorted, and the first matching row that is found is chosen to update each row in the target table.
- `none`: source rows are assumed to be unique; the `ybload` behavior is undefined if duplicate source rows exist. Sometimes the load will fail if duplicate rows are found in the source file. This option is non-deterministic and not recommended if you know the source file has duplicates. If you know that the source file cannot have duplicates, `none` is recommended because the load will run faster.
- `random`: non-deterministic; a random matching source row is used to update each row in the target table.

The following tables describe the behavior when different duplicate-handling methods are used for each type of load.

INSERT Loads

Number of matching rows in source file	Behavior
0	No rows inserted.
1	Single row inserted.
>1 and `--duplicate-handler none`	Each matching source row is inserted.
>1 and `--duplicate-handler order by` | `random`	Single row inserted (the source row to insert is selected based on the chosen `--duplicate-handler` option).

UPDATE Loads

Number of matching rows in source file	Number of matching rows in target table	Behavior
0	0, 1, or >1	No updates.
1	0	No updates.
1	1	The target row is updated with column values from the source row.
1	>1	Each matching target row is updated with column values from the source row.
>1 and `--duplicate-handler none`	0	No updates.

Number of matching rows in source file	Number of matching rows in target table	Behavior
>1 and `--duplicate-handler none`	1 or >1	Not recommended, non-deterministic. Possible outcomes: - Matching target rows get updated with column values from one of the source rows (but which source row is unknown). - `ybload` operation fails with an error such as "attempted to update a row multiple times."
>1 and `--duplicate-handler order by | random`	0	No updates.
>1 and `--duplicate-handler order by | random`	1	The target row is updated with column values from one source row (the source row to update from is defined by the `--duplicate-handler` option).
>1 and `--duplicate-handler order by | random`	>1	Each matching target row is updated with column values from one source row (the source row to update from is defined by the `--duplicate-handler` option).

UPSERT Loads

Number of matching rows in source file	Number of matching rows in target table	Behavior
0	0, 1, or >1	No updates or inserts.
1	0	1 target row inserted.
1	1	The target row is updated with column values from the source row.
1	>1	Each matching target row is updated with column values from the source row.
>1 and `--duplicate-handler none`	0	Multiple target rows are inserted (one for each matching source row).
>1 and `--duplicate-handler none`	1 or >1	Not recommended, non-deterministic. Possible outcomes: - Matching target rows get updated with column values from one of the source rows (but which source row is unknown). - `ybload` operation fails with an error such as "attempted to update a row multiple times."
>1 and `--duplicate-handler order by | random`	0	A single target row is inserted (the source row to insert is defined by the `--duplicate-handler` option.
>1 and `--duplicate-handler order by | random`	1	The target row is updated with column values from one source row (the source row to update from is defined by the `--duplicate-handler` option.
>1 and `--duplicate-handler order by | random`	>1	Each matching target row is updated with column values from one source row (the source row to update from is defined by the `--duplicate-handler` option.

DELETE Loads

The `--duplicate-handler` option is ignored for `delete` loads.

Number of matching rows in source file	Number of matching rows in target table	Behavior
0	0, 1, or >1	No deletes.
1	0	No deletes.
1 or >1	1 or >1	All matching target rows are deleted.
>1	0	No deletes.
>1	1 or >1	All matching target rows are deleted.

Loading Data Exported from SQL Server

This section covers best practices for loading data into a Yellowbrick database when the source of the data is a table in a Microsoft SQL Server database. There are specific recommendations for both exporting the data from SQL Server and loading the data with the `ybload` client.

Best Practices for Exporting Data from SQL Server

Yellowbrick strongly recommends that you prepare a clean `csv` file before trying to load it, especially if your character strings contain any of the following:

- The same character that is used as the field separator, such as commas or pipe symbols
- The same character that is used as the row separator, such as EOL markers
- Quotes

If you intend to use the SQL Server `bcp` utility to export data, use the `"query"` option to select the data out of the table and escape quotes and other characters that may cause problems when you load. By default, the `bcp` utility does not escape quotes and separators that are found in character strings. If you use a simple "copy table to file" approach and your data has the characteristics listed above, you will need to set some special options in the `ybload` command to parse the incoming data successfully. In general, it is more efficient to clean up the data on its way out of the source table than to work around data problems when you set up and run the `ybload` operation.

For example, you can use a `bcp` command that looks something like this:

```
bcp "query" queryout <filename> -c  -S <server> -t,
...
```

where `-t,` exports a comma-delimited file and `"query"` is a SELECT statement. If `column2` in `table1` has character strings that contain quotes, your query should look like this:

```
select column1, '\"' + REPLACE(column2,'\"','\\\"') + '\"' as column2 from table1 ...
```

The purpose of the `REPLACE` function is to replace double quotes within strings with escaped double quotes (`'\"'`). The entire character string will also be enclosed by double quotes.

Note: If you have a pre-2016 version of SQL Server, which does not export data in the UTF-8 encoding, use the `-w` option in the `bcp` command (instead of the `-c` option). The `-w` option runs the export using Unicode characters.

By using the `-w` option in the `bcp` command and the `--encoding utf16` option in the `ybload` command, you will be able to export and load your data. Also make sure that your Yellowbrick database uses the `utf8` encoding, not the default `latin9` encoding.

For details about `bcp` syntax and options, see the Microsoft SQL Server documentation.

---

bcp Example on Windows

The following example works on Windows platforms:

```
bcp DW.dbo.DimDate out dw.dbo_DimDate.csv -S ABCDE\PROD -T -c -t 0x1F -r 0x0A -a 32768
```

The following `ybload` command will work with this `bcp` command:

```
ybload -d dw -t dbo.dimdate --delimiter 0x1F --format bcp dw.dbo.dimdate.csv
```

---

bcp Example on Linux

A `bcp` for Linux utility is available for extracting data from SQL Server. Here is an example `bcp` Linux command:

```
bcp "USE DW; SELECT * FROM dbo.DimDate(NOLOCK) ;" queryout "/mnt/data/dw.dbo_DimDate.csv" –S dw.abcd.com -c -t "0x1F" -a 32768
```

The same `ybload` command that is shown in the Windows example will work in this case.

Workarounds for Loading Exported Data

If you are unable to produce clean exported source files for bulk loading purposes, the `ybload` program provides some options that can parse the data and work around problems:

Null Bytes

Null-byte strings (`0x00`) may be exported in your data, either representing complete empty fields or empty strings within fields. You can use the `--emptymarker \0` option to replace `0x00` with empty strings, and the `--ignorenullbytes` option to ignore `0x00` within strings.

Hexadecimal Values

Timestamps may be exported as 64-bit hexadecimal values. The `ybload` program can parse and load these values if you use the `inhex` option for the appropriate fields.

For details about bulk load syntax and options, see ybload Options.

Loading Generated Key Values

This section explains how to use a sequence as a generated key when bulk loading a table with the `ybload` command. Follow these steps:

1. Create a sequence. For example:

```
premdb=# create sequence matchid;
CREATE SEQUENCE
```

2. Create a table with a `BIGINT` column that has a `DEFAULT` column constraint. Define the default value as the result of the `NEXTVAL` function.

For example:

```
premdb=# create table matchstats(matchkey bigint default nextval('matchid'),
seasonid smallint, matchday date, htid smallint, atid smallint, moment varchar(5));
CREATE TABLE
premdb=# \d matchstats
                    Table "public.matchstats"
  Column  |         Type         |              Modifiers
----------+----------------------+-----------------------------------
 matchkey | bigint               | default nextval('matchid'::regclass)
 seasonid | smallint             |
 matchday | date                 |
 htid     | smallint             |
 atid     | smallint             |
 moment   | character varying(5) |

Distribution: Hash (matchkey)
```

3. Make sure the user who will run the load has `INSERT` privilege on the table and `UPDATE` privilege (or `ALL` privileges) on the sequence.

4. Bulk load the table with the `--field-defs` option, listing all columns except the first column in the table, which will be generated from the sequence. Make sure the source file contains data for the columns that you list. For example:

```
$ ybload -d premdb --username bobr -W -t matchstats --format csv
--field-defs seasonid,matchday,htid,atid,moment /home/ybdata/newmatchstats.csv
```

5. Check the results of the load. For example:

```
premdb=# select * from matchstats order by 1 desc limit 5;
 matchkey  | seasonid |  matchday  | htid | atid | moment
-----------+----------+------------+------+------+---------
 789709825 |        3 | 1995-02-22 |    3 |   72 | 00:51
 789708801 |        3 | 1995-01-02 |    3 |   71 | 00:51
 789707777 |        3 | 1994-09-10 |    3 |   70 | 00:51
 789706753 |        3 | 1994-12-10 |    3 |   67 | 00:51
 789705729 |        3 | 1994-08-27 |    3 |   65 | 00:51
(5 rows)
...
```

6. Verify that the generated `matchkey` values are unique. For example:

```
premdb=# select count(distinct matchkey), count(*) from matchstats;
  count  |  count
---------+---------
 1049932 | 1049932
(1 row)
```

Monitoring Load Operations

You can monitor active bulk load operations in Yellowbrick Manager. The Load Assistant reports progress periodically, with a brief summary when the load is complete. See Loading a Table via the Load Assistant.

Regardless of the method you choose for loading tables, you can filter the rows under **Query Activity** and switch between **Active** and **Historical** views. Filter on `ybulkload` to see information about bulk load operations. Note that, by default, rejected rows do not stop a load from running to completion. They are logged to a file (and its location is reported).

You can also run queries against the following system views:

- sys.load
- sys.log_load
- sys.query
- sys.log_query

For example, assume that user `bobr` runs the following load of 25 million rows into the `newmatchstats` table:

```
$ ybload -d premdb --username bobr -t newmatchstats -W --format csv /home/brumsby/newmatchstats25mil.csv
Password for user bobr:
...
13:30:44.841 [ INFO] Flushing last 6818022 rows (of 24785280 total) in transaction #1 for newmatchstats
READ:600.1MB(94.58MB/s). ROWS G/B: 24785280/0( 3.73M/s). WRITE:661.8MB(104.3MB/s).  TIME E/R:   0:00:06/  0:00:0013:30:45.153 [ IN
13:30:46.535 [ INFO] Committed transaction #1 after a total of 693987840 bytes and 24785280 good rows for newmatchstats
13:30:46.560 [ INFO] READ:600.1MB(77.65MB/s). ROWS G/B: 24785280/0( 3.06M/s). WRITE:661.8MB(85.64MB/s).  TIME E/R:   0:00:07/  0:0
13:30:46.561 [ INFO] SUCCESSFUL BULK LOAD: Loaded 24785280 good rows in   0:00:07 (READ: 77.65MB/s WRITE: 85.64MB/s)
```

Using the transaction ID (`transaction_first=slq.transaction_id`), the following join of `sys.log_load` and `sys.log_query` returns detailed information about that load:

```
premdb=# select sll.table_name, sll.username, sll.end_time, sll.transaction_first, sll.inserted_rows, substr(slq.query_text,1,50)
from sys.log_load sll join sys.log_query slq on sll.transaction_first=slq.transaction_id
where sll.username='bobr' and sll.table_name='newmatchstats'
order by 4 desc;
  table_name   | username |          end_time          | transaction_first | inserted_rows |                    command
---------------+----------+----------------------------+-------------------+---------------+-----------------------------------
 newmatchstats | bobr     | 2020-10-30 13:30:46.579-07 |             96045 |      24785280 | YBULKLOAD INTO "premdb"."public"."new
(1 row)
```

See also Delete, Update, and Upsert Examples.

Note that the `application_name` for a `ybload` operation, available in the `sys.query` and `sys.log_query` views, is `ybload <version>`. For example:

```
ybload 6.0.0-a8363f32.1792
```

Resuming a Partial Load

Platforms: All platforms

Parent topic: Running a Bulk Load

If a bulk load consists of multiple transactions and fails after some transactions have been committed, you can resume the load. Use the `--resume-partial-load-from-offset` option to resume at a specific byte offset, as reported by the load messages. For example:

```
...
2018-01-17 17:02:59.989 [FATAL] <main>  FAILED BULK LOAD: Last commit occurred after 300 good rows
2018-01-17 17:02:59.989 [ WARN] <main>  At the time of the last commit:
   300 good row(s) had been committed
   3 bad row(s) had been skipped
   1 source(s) had been partially loaded

2018-01-17 17:02:59.989 [ WARN] <main>
To resume loading from the last committed position, invoke ybload as follows:
   1) ybload <original options> --resume-partial-load-from-offset 150000 /data/tests/tmp1.csv

2018-01-17 17:02:59.989 [ WARN] <main>  BEWARE: Additional bad rows were written to the bad row file after the last commit
2018-01-17 17:02:59.989 [ WARN] <main>          When fixing rows in the bad row file, ignore any bad rows that follow this message
2018-01-17 17:02:59.990 [ WARN] <main>          "----- successful commit after 3 bad rows -----"
```

In the following example, the load can restart from the beginning of the fourth file. Therefore the `--resume-partial-load-from-offset` option is not necessary:

```
...
2018-01-17 17:03:06.949 [FATAL] <main>  FAILED BULK LOAD: Last commit occurred after 300 good rows
2018-01-17 17:03:06.949 [ WARN] <main>  At the time of the last commit:
   300 good row(s) had been committed
   3 bad row(s) had been skipped
   3 source(s) had been completely loaded
   6 source(s) had not started to load

2018-01-17 17:03:06.950 [ WARN] <main>
To resume loading from the last committed position, invoke ybload as follows:
   1) ybload <original options>  \
   /data/tests/tmp4.csv  \
   /data/tests/tmp5.csv  \
   /data/tests/tmp6.csv  \
   /data/tests/tmp7.csv  \
   /data/tests/tmp8.csv  \
   /data/tests/tmp9.csv

2018-01-17 17:03:06.950 [ WARN] <main>  BEWARE: Additional bad rows were written to the bad row file after the last commit
2018-01-17 17:03:06.950 [ WARN] <main>          When fixing rows in the bad row file, ignore any bad rows that follow this message
2018-01-17 17:03:06.950 [ WARN] <main>          "----- successful commit after 3 bad rows -----"
```

In the third example, two separate `ybload` operations need to be run to complete the load:

```
...
2018-01-17 17:03:12.118 [FATAL] <main>  FAILED BULK LOAD: Last commit occurred after 800 good rows
2018-01-17 17:03:12.118 [ WARN] <main>  At the time of the last commit:
   800 good row(s) had been committed
   8 bad row(s) had been skipped
```

```
   1 source(s) had been completely loaded
   1 source(s) had been partially loaded
   1 source(s) had not started to load

2018-01-17 17:03:12.119 [ WARN] <main>
To resume loading from the last committed position, invoke ybload as follows:
   1) ybload <original options> --resume-partial-load-from-offset 100000 /data/tests/big2.csv
   2) ybload <original options>  \
   /data/tests/big3.csv

2018-01-17 17:03:12.119 [ WARN] <main>  BEWARE: Additional bad rows were written to the bad row file after the last commit
2018-01-17 17:03:12.119 [ WARN] <main>          When fixing rows in the bad row file, ignore any bad rows that follow this message
2018-01-17 17:03:12.119 [ WARN] <main>          "----- successful commit after 8 bad rows -----"
```

Saving Load Options to a File

You can use the `@file` option to save a set of reusable format settings for a `ybload` operation. This feature facilitates reuse of options for parsing and processing source files.

You cannot use `@file` to save and re-run an entire `ybload` specification.

To save and use an options file:

1. Determine the set of formatting options that works to load a particular table.
2. Check that these options are appropriate by loading the table once or by using the `--dryrun` option.
3. Save the options to a text file in the same location as your source file (or a similar accessible location). For example:

```
$ more ybload_options_file
--date-style
MDY
--trim-white
--ignore-emptymarker-case
--y2base
2000
--format
TEXT
--delimiter
,
--linesep
\n
--escape-char
\
--skip-blank-lines
--on-missing-field
SUPPLYNULL
--on-extra-field
REMOVE
```

Note: The name of an option and its value must be on separate lines in the file. You can also include database connection options in the file.

4. Run the `ybload` command with the `@file` option. For example:

```
$ ybload -d premdb @/home/yb100/premdata/season_load_options.txt
-t season_date /home/yb100/premdata/season_dates.txt
```

You can set up different files to satisfy the requirements of different `ybload` operations for all of your tables. If needed, you can use more than one file in the same load operation:

```
ybload @file1 @file2 ...
```

Setting the Commit Interval

Bulk load operations commit rows to the database based on settings for the following two options. You can override these thresholds in the `ybload` command:

- `--rows-per-transaction` defaults to 263 – 1 rows (9223372036854775807), which is effectively unlimited.
- `--bytes-per-transaction` defaults to 1TB (1099511627776 bytes).

The threshold that is met first is applied. Given these defaults, a 1TB source file would be committed all at once at the end of a load. However, these settings are guidelines for `ybload`, not hard rules. When `ybload` detects that one of the thresholds has been reached, the load will commit soon after that. Do not expect individual commits (transactions) to process exactly the number of rows or bytes that the settings specify.

Although other databases may perform faster with a smaller commit interval and a larger number of commits, `ybload` performs best with a larger interval and a smaller number of commits. The only real purpose for setting a smaller commit interval is to improve "data visibility," not performance. For example, if you want incoming queries to see the latest data as it commits, and you are not concerned if the load has not completed when those queries are run, you could use a much smaller commit interval, such as 10GB or 100GB.

Specifying NULL Behavior for Different Columns

This section explains how you can use the `--nullmarker` option in different ways in the same bulk load. You may run into a situation where different columns in the same table must be loaded with `NULL` values, but the source file for the load defines `NULL` in different ways.

By default, `ybload` treats adjacent delimiters without text between them as `NULL` values and adjacent quotes within delimiters as empty strings. For example, assume that a line in a comma-delimited source file looks like this:

```
123,,456,"",789
```

This line represents 5 fields or columns:

- Column 1: 123
- Column 2: `NULL` (If this column is not nullable, `ybload` returns an error.)
- Column 3: 456
- Column 4: empty string (If this column cannot contain an empty string, `ybload` returns an error; only `CHAR` and `VARCHAR` columns can contain empty strings.)
- Column 5: 789

If the fields in your source files and their corresponding table columns conform with this pattern with respect to `NULL` values and empty strings, `ybload` will load the data correctly by default. However, if your data contains special values that are intended to represent `NULL`, read the rest of this section to understand how to load those values correctly (or discard them as appropriate). See also the description of the `--emptymarker` option under ybload Options.

Consider the DDL for the following table:

```
create table team_with_nulls(
teamid smallint not null,
htid smallint not null,
atid smallint not null,
name varchar(30),
nickname varchar(20),
city varchar(20),
stadium varchar(50),
debut date)
distribute replicate;
```

Note that the four `VARCHAR` columns are nullable, and the final `DATE` column is also nullable. Now consider the following 5 lines in the source file:

```
46,47,96,Wimbledon,Dons,London,Selhurst Park,1991/08/01
47,48,97,Wolverhampton Wanderers,Wolves,Wolverhampton,Molineux Stadium,2003/08/01
48,49,98,,null,,,----/--/--
49,50,99,,null,,,----/--/--
50,51,100,,null,,,----/--/--
```

- Columns 4, 6, and 7 have consecutive delimiters, implying `NULL` values.
- Column 5 contains an explicit `null` character string, but this string must be stored as `NULL`.

- Column 8 has `----/--/--` in some rows. This value also needs to be loaded as `NULL`.

Fortunately, `ybload` has options that work for all fields in the source file, for all fields with a given data type, or for specific named fields. For example, within the same `ybload` command, you could set the `--nullmarker` option globally, for all `VARCHAR` fields, and for one or more specific fields. The per-type and per-field options, if present, override the global settings.

In this case, the per-type option will not solve the issue with the `VARCHAR` columns because one of them has a different marker for `NULL`. However, the global default `--nullmarker` option takes care of the columns where adjacent delimiters indicate `NULL`, and you can specify a different `NULL` value for column 5. For column 8, the only `DATE` column in the table, you can use either a per-field option or a per-type option:

- Column 5: load `null` as `NULL`, using:

```
--per-field-options '{"5":{"nullmarker":"null"}}'
```

- Column 8: load `----/--/--` as `NULL`, using :

```
--date-field-options '{"nullmarker": "----/--/--"}'
```

or:

```
--per-field-options '{"8":{"nullmarker":"----/--/--"}}'
```

The complete `ybload` command looks like this:

```
$ ybload -d premdb -t team_with_nulls --username bobr -W /home/premdata/team_with_nulls.csv
--format csv --date-field-options '{"nullmarker": "----/--/--"}' --per-field-options '{"5":{"nullmarker":"null"}}'
```

The global `--nullmarker ''` option is not required because consecutive delimiters represent the default null marker.

The resulting rows in the table look like this:

```
$ ybsql premdb
Null display is "[NULL]"
...
premdb=# select * from team_with_nulls;
 teamid | htid | atid |          name           | nickname |     city      |     stadium      |   debut
--------+------+------+-------------------------+----------+---------------+------------------+------------
     46 |   47 |   96 | Wimbledon               | Dons     | London        | Selhurst Park    | 1991-08-01
     47 |   48 |   97 | Wolverhampton Wanderers | Wolves   | Wolverhampton | Molineux Stadium | 2003-08-01
     48 |   49 |   98 | [NULL]                  | [NULL]   | [NULL]        | [NULL]           | [NULL]
     49 |   50 |   99 | [NULL]                  | [NULL]   | [NULL]        | [NULL]           | [NULL]
     50 |   51 |  100 | [NULL]                  | [NULL]   | [NULL]        | [NULL]           | [NULL]
(5 rows)
```

Note that in this `ybsql` session `NULL` values are intentionally displayed as `[NULL]` for readability. (See ybsql Display Properties for information about the `\pset` command.)

Using the Dry Run Option

Use the `--dryrun` option to test the `ybload` options you are setting before doing the actual load. For example:

```
$ ybload --dryrun -d premdb -t match --format csv --username bobr -W /home/bobr/match.csv
Password for user bobr:
17:03:07.065 [ INFO] ABOUT CLIENT:
   app.cli_args         = --dryrun -d premdb -t match --format csv --username bobr -W /home/bobr/match.csv
   app.name_and_version = ybload version 2.0.0-10711
   java.home            = /usr/lib/jvm/java-8-oracle/jre
   java.version         = 1.8.0_101
   jvm.memory           = 981.50 MB (max=14.22 GB)
   jvm.name_and_version = Java HotSpot(TM) 64-Bit Server VM (build 25.101-b13, mixed mode)
   jvm.options          = -Xmx16g, -Xms1g, -ea, -Dapp.name=ybload, -Dapp.pid=2823, -Dapp.repo=/usr/lib/ybtools/lib, -Dapp.home=/us
   jvm.vendor           = Oracle Corporation
   os.name_and_version  = Linux 4.4.0-31-generic (amd64)

17:03:07.144 [ INFO] Gathering metadata on input files
17:03:07.487 [ INFO] Assuming source encoding matches database server encoding: LATIN9
17:03:07.759 [ INFO] Starting 1 source PreReaders
17:03:07.777 [ INFO] Auto-detected line separator = '\n'
17:03:07.784 [ INFO] Configuration (record/field separation):
   --format                          : CSV
   --delimiter                       : ,
   --linesep                         : \n
   --quote-char                      : "
   --escape-char                     : "
   --trim-white                      : false
   --skip-blank-lines                : true
   --on-missing-field                : ERROR
   --on-extra-field                  : ERROR
   --on-unescaped-embedded-quote     : ERROR
   Internal scanner                  : RecordScannerQuote_1_1
17:03:07.808 [ INFO] Using database locale: C
17:03:07.930 [ INFO] Bad rows will be written to /home/bobr/match.csv.20190204170307.bad
17:03:07.937 [ WARN] Running with --dryrun. Nothing will be written to database
17:03:07.938 [ INFO] Starting import of 1 files
17:03:08.392 [ INFO] Starting 4 segment readers
17:03:08.393 [ INFO] Opening transaction #1...
17:03:08.395 [ INFO] Opened transaction #1
17:03:08.471 [ INFO] Flushing last of 8606 rows for transaction #1
17:03:08.474 [ INFO] Committing 0 rows into transaction #1...
17:03:08.475 [ INFO] Committed transaction #1 after a total of 292604 bytes and 8606 good rows
17:03:08.477 [ INFO] READ:305.8KB(224.0KB/s). ROWS G/B: 8606/0( 6.16K/s). WRITE:285.7KB(209.3KB/s).  TIME E/R:   0:00:01/ --:--:--
17:03:08.477 [ INFO] SUCCESSFUL BULK LOAD (--dryrun): Loaded 8606 good rows in   0:00:01 (READ: 224.0KB/s WRITE: 209.3KB/s)
```

Based on this information, you know that you are ready to run the load without the `--dryrun` option and actually commit rows to the match table:

```
$ $ ybload -d premdb -t match --format csv --username bobr -W /home/bobr/match.csv
Password for user bobr:
...
```

If you use the `--dryrun` option and your load is not set up correctly, you will see messages that help you identify the problem. For example, in this case, the wrong table name (`newmatchstats`) is specified, and none of the rows from the source file (`match.csv`) can be loaded:

```
$ ybload --dryrun -d premdb -t newmatchstats --username bobr -W /home/bobr/match.csv
Password for user bobr:
...
17:35:19.139 [ERROR] Unable to separate row into fields for parsing: Too many fields (6 > 5)
Bad row = «1,1992-08-01 00:00:00,2,52,0-1,-  »
...
```

If necessary, you can go to the specified bad rows file to understand why these rows do not qualify for loading into this table. In this example, the source file contains one more column than the specified table:

```
$ more match.csv.20180521173518.bad
...
#error: lineByteRange(0-35)
#reason: Too many fields (6 > 5)
1,1992-08-01 00:00:00,2,52,0-1,-

#error: lineByteRange(35-70)
#reason: Too many fields (6 > 5)
1,1992-08-01 00:00:00,2,55,0-1,-
...
```

ybload Command

In this section:

Each `ybload` command defines:

- The target table, which must exist in the database
- An absolute or relative path to an input file or multiple input files (local or S3)
- Database connection options
- Options for processing the load, returning output, and parsing the input data

Note: The `ybload` user must have `INSERT` privileges on the target table but does not have to own the table.

Basic Syntax

```
ybload [options] -t [SCHEMA.]TABLENAME SOURCE [SOURCE]...
```

The `-t` option supports either `TABLENAME` or `SCHEMA.TABLENAME`. If you do not specify the schema name, the table is assumed to be in the `public` schema. The schema of the target table is not based on the user's `search_path`, regardless of how it is set.

`DATABASE.SCHEMA.TABLENAME` is not supported. If you want to specify the database name for a table, use the `-d` option.

Note: If you used a quoted case-sensitive identifier to create the target table, you must quote the table name and escape the quotes in the `ybload` command line. For example, if your table is named `PremLeagueStats`:

```
-t \"PremLeagueStats\"
```

The `ybload` client tool assumes that the character encoding of the source file (LATIN9 or UTF-8, for example) matches the encoding of the target database. If this not the case, see the --encoding option.

Calling the ybload Utility

The `ybload` command connects to the database via JDBC as a Java client application. After you have downloaded and installed the bulk loader client (as part of the `ybtools` package), you can run the Linux and Windows executable programs:

- `ybload` on Linux
- `ybload.exe` on Windows

Getting Online Help

Use the following `ybload` options to return online help text:

- `-?` or `--help` for a usage summary and coverage of the basic options.
- `--help-advanced` for a usage summary and a list of advanced options *only*; to see all options, specify both `--help` and `--help-advanced`:

```
$ ./ybload --help --help-advanced
```

The following command runs the bulk loader and returns basic help text for load options.

```
$ ./ybload --help
ybload version 5.4.0-20220314113018

ybload loads files into an existing table in a Yellowbrick Database.

All files must have the same field layout.
...
```

Bulk Load Manager and Data Transfer Ports

In addition to the `YBPORT` setting for the database connection, the bulk loader uses the following default ports:

- Bulk Load Manager port: `11111` for managing load transactions and progress.
- Data transfer ports: `31000-41000`. On a cluster with 15 active compute nodes, 30 ports in this range are used to load the data across the cluster (2 per active node).

Setting Other Options

The bulk loader includes a large number of options that define ways to process the load, log results, and parse incoming data correctly. See ybload Options and Common Options in ybtools for details. Frequently used settings include null markers and expected formats for dates and times. You can define some of the more advanced parsing options for specific data types and fields very flexibly by using JSON objects.

Note: By default, `ybload` appends rows to tables (also referred to as inserting rows). However, you can use the `--write-op` option to set up a load that deletes, updates, or "upserts" rows.

Character Escape Sequences

Several `ybload` and `ybunload` options support the following escape sequences as option values:

- `\t` : tab
- `\b` : backspace
- `\n` : newline
- `\r` : carriage return
- `\f` : formfeed
- `\'` : single quotation mark
- `\"` : double quotation mark
- `\\` : backslash
- `\0` : null byte
- `\us` : ASCII unit separator
- `\rs` : ASCII record separator

You can specify these escape sequences as shown in this list, or you can use the equivalent hexadecimal or Unicode escape sequences.

Hexadecimal escape sequences must start with `0x` (lowercase `x`). Any number of hex digits may follow. Unicode escape sequences start with `u+` or `U+`. The capitalization and number of hex digits are both flexible: `u+1f`, `U+01F`, and `u+001F` are all equivalent.

The following table shows two examples. The three escape sequence columns show how you would specify the values in a `ybload` or `ybunload` command. The ASCII decimal values are shown for reference; you cannot specify these values in the command.

ASCII Decimal Value	Escape Sequence	Hexadecimal Escape Sequence	Unicode Escape Sequence
`31`	`\us`	`0x1F`	`U+001F`
`9`	`\t`	`0x09`	`U+0009`

See also Escape Sequences for Non-Printable Characters.

Escaping Quotes in Windows Clients

This section explains how to escape quotes when you are running `ybload` and `ybunload` tools on the Windows command line (`cmd.exe`) or in Windows PowerShell.

For example, consider the following option setting:

```
{"nullmarker":"00/00/0000"}
```

To specify this kind of setting in UNIX and Linux shells, such as `sh` and `bash` , you can use standard JSON wrapped in a pair of single quotes. For example, to specify the `nullmarker` setting for all date fields:

```
--date-field-options '{"nullmarker":"00/00/0000"}'
```

However, Windows platforms do not support single quotes on the command line, so the option must be wrapped in double quotes. Also, because the syntax already contains some double quotes, they must be escaped. Note that the rules for escaping double quotes are different for `cmd.exe` / `.BAT` files and Windows PowerShell. See the following examples.

If you are running interactively via `cmd.exe` or from a `.BAT` file:

```
```

--date-field-options "{"nullmarker":"00/00/0000"}"

```
   The JSON in this example has a backslash character in front of all the double quote characters, and the whole string is wrapped

If you are running interactively via PowerShell:
:   ```
--date-field-options "{\""nullmarker\"":\""00/00/0000\""}"
```

For PowerShell, you have to place a backslash *and* a double quotation mark in front of all the double quotation mark characters.

NULL and Empty Markers

Any column that is declared nullable in a `CREATE TABLE` statement can store a `NULL` value. A *null marker* is a character string that defines the presence of a `NULL` value. Typical null markers are the 4-character string `NULL` and the 2-character string `\N`.

Only `CHAR` and `VARCHAR` columns can store an empty value (a zero-length string). An *empty marker* is a string that defines the presence of an empty string between adjacent fields.

Depending on the format of the exported source data that you are going to load, you may have to use `ybload` options that define how `NULL` values or empty strings are marked. These options are called `--nullmarker` and `--emptymarker`. In some cases, you may need to specify one of these options so that `ybload` can distinguish intended `NULL` values from intended empty strings. In practice, you should not have to specify both of these options in any `ybload` command; most exported data is formatted such that either one type of marker is needed or no markers are needed.

The need for these options depends in part on the `--format` setting you use:

- `CSV`: standard CSV files do not need markers to be identified, but custom CSV files may require them. A standard CSV file identifies:
- Empty strings by placing two quotes inside two field delimiters: `<delimiter>''<delimiter>`. For example, if the field delimiter is a comma: `,"",`
- `NULL` values by placing field delimiters immediately next to each other: `<delimiter><delimiter>`. For example, if the field delimiter is a comma: `,,` A sample CSV data row with five fields might look like this:

```
"2018","","Mini",,"Cooper S"
```

In this example, the second field contains an empty string, and the fourth field contains a `NULL` value. If your source files look like this, you do not need to specify `--nullmarker` or `--emptymarker`.

- `TEXT`: you may need to specify a null marker or an empty marker. The exported data may specify empty strings with adjacent delimiters and `NULL` values with either `\N` or the string `NULL` (for example). The `ybload` null marker default is `\N`.

Check the format of the exported `TEXT` data and make sure you know how each marker is represented. Some flat file formats do not natively distinguish between `NULL`s and empty strings; therefore, you need to discover the format of the exported data and specify the appropriate `ybload` options.

- `BCP`: If you are using `--format BCP`, you should not specify a null marker or an empty marker in the `ybload` command. The default empty marker is `\0`, which matches the BCP exported data. Adjacent delimiters define `NULL`s in BCP format.

---

ybload Behavior If Both Markers Are Specified

The following tables demonstrate the `ybload` processing rules that apply if you specify both a null marker and an empty marker in a `ybload` command. In general, specifying both markers is not necessary, and the command output will contain a warning message to that effect.

Note: The examples in the first table apply to the `CSV` format only.

Source Data for Three Adjacent Fields	Data Type of Second Field	ybload Marker Settings	Second Field Parsed As?	Notes

Source Data for Three Adjacent Fields	Data Type of Second Field	ybload Marker Settings	Second Field Parsed As?	Notes
`abc,"",def`	`CHAR/VARCHAR`	N/A	Empty string	`--nullmarker` and `--emptymarker` settings are ignored.
`abc,"",def`	Non-string	`N/A`	NULL	`--nullmarker` and `--emptymarker` settings are ignored.
`abc,"FOO",def`	`CHAR/VARCHAR`	`--nullmarker 'FOO'` and `--emptymarker 'FOO'`	The string `FOO`	The quotes "protect" the string `FOO` from being interpreted as a marker.

Note: The examples in the second table apply to all three formats.

Source Data for Three Adjacent Fields	Data Type of Second Field	ybload Marker Settings	Second Field Parsed As?	Notes
`abc,,def`	Non-string	`N/A`	NULL	`--nullmarker` and `--emptymarker` settings are ignored.
`abc,,def`	`CHAR/VARCHAR`	`Not defined`	NULL	
`abc,,def`	`CHAR/VARCHAR`	`--nullmarker` is set	Empty string	
`abc,,def`	`CHAR/VARCHAR`	`--emptymarker` is set	`NULL`	
`abc,,def`	`CHAR/VARCHAR`	`--nullmarker` and `--emptymarker` both set	`NULL`	
`abc,FOO,def`	`CHAR/VARCHAR`	`--nullmarker 'FOO'`	`NULL`	
`abc,FOO,def`	`CHAR/VARCHAR`	`--emptymarker 'FOO'`	Empty string	
`abc,foo,def`	`CHAR/VARCHAR`	`--nullmarker 'FOO'` and `--ignore-nullmarker-case` is set	`NULL`	
`abc,foo,def`	`CHAR/VARCHAR`	`--nullmarker 'FOO'` and `--ignore-nullmarker-case` is not set	The string `foo`	
`abc,foo,def`	`CHAR/VARCHAR`	`--emptymarker 'FOO'` and `--ignore-emptymarker-case` is set	Empty string	
`abc,foo,def`	`CHAR/VARCHAR`	`--emptymarker 'FOO'` and `--ignore-emptymarker-case` is not set	The string `foo`	

Setting --format for Flat Files

Platforms: All platforms

Parent topic: ybload Command

The `--format` option is a mechanism for specifying the expected format of `ybload` source files, given some assumptions about the standard behavior of the export program that produced those files. Specifying the `--format` option is always recommended for flat files, but the setting you choose must be based on knowledge of the data and the export program that produced the data.

Note: This section does not apply to structured Parquet files; to load these files, always specify `--format parquet` in the `ybload` command. Auto-detection of Parquet files is not supported.

In general, `ybload` tries to import raw data intelligently, given some assumptions about which characters need to be read in exactly as they are and which characters need to be interpreted as having a special function. These special characters often have to be removed from the source rather than loaded, in a sense undoing what the exporter had done to prepare the data.

In particular, the `--format` option defines the expected layout of flat files with respect to line separators and field delimiters. Line separators mark the ends of rows, and field delimiters mark the separators between columns. If `ybload` does not handle these separators and delimiters correctly, the load may fail or reject rows. One of the main problems that choosing the correct format can solve is the handling of embedded delimiter characters, where the delimiter character appears inside actual data values and must be loaded, not removed. A typical example is a comma that appears, on purpose, within character strings in a CSV file.

The `--format` option has three flat-file settings:

- `CSV`: comma-separated values file. This format assumes that embedded delimiters are protected by the use of quotes to wrap entire fields. When a `CSV` file is loaded, quotes that surround field values will be removed, not loaded.

For example, the string `"Citizens, or Blues"` can be loaded as `Citizens, or Blues` into the `nickname` column of the `team` table, despite the fact that the field delimiter is a comma.

- `TEXT`: flat text file. This format assumes that delimiters in field values were protected by being preceded with a backslash (`\`). When a `TEXT` file is loaded, backslash characters that precede delimiter characters will be removed, not loaded.

For example, if the field delimiter is a comma, the string `Citizens\, or Blues` can be loaded into the `nickname` column of the `team` table as `Citizens, or Blues`.

- `BCP`: as exported from the Microsoft SQL Server `bcp` utility. This format assumes that delimiters in fields were not protected; therefore, the incoming data must not have embedded delimiters. The incoming data is read exactly as it is.

Defaults for Other Options When --format Is Set

When the `--format` option is set, other `ybload` options have specific default values, as shown in the following table. Regardless of the `--format` setting, and even if `--format` is not specified at all, `ybload` tries to automatically detect the `--delimiter` and `--linesep` characters. If the attempt at auto-detection fails, the load continues but uses fallback characters, which may or may not suit the incoming data.

Option	CSV	TEXT	BCP
`--delimiter`	`,`	`\t`	`,`
`--linesep`	`LF`	`LF`	`CRLF`
`--quote-char`	`"`	N/A	N/A

Option	CSV	TEXT	BCP
`--escape-char`	`"`	`\`	N/A
`--nullmarker`	empty/none	`\N` (two characters)	empty/none
`--emptymarker`	empty/none	empty/none	`\0` (1 character)
`--convert-c-escape`	`false`	`true`	`false`

See also ybload Options.

Notes on --format TEXT Escape Sequences

When you use the `TEXT` option, `ybload` does not recognize all escape sequences. For example, the string `abc\wdef` is loaded literally as `abc\wdef`, not respecting `\w` as an escape sequence. (Note that the equivalent `ybsql \copy from` command with the `format text` option *does* recognize this escape sequence, and others like it.)

The `--format TEXT` option recognizes only the following escape sequences:

- `escape_char escape_char = escape_char`

For example: `\\`

- `escape_char delimiter = delimiter`

For example: `\|`

- `escape_char linesep = linesep`

For example: `\\n`

The `ybload` escape character defaults to the backslash character (`\`), but you can change it by setting the `--escape-char` option.

When you use the `TEXT` option, `ybload` defaults to using the `--convert-c-escape` option. However, this option recognizes and converts only the following escape sequences:

- `\\` : backslash
- `\b` : backspace
- `\f` : formfeed
- `\n` : newline
- `\r` : carriage return
- `\t` : tab
- `\v` : vertical tab

All other escape sequences are not converted.

Note: If you change the escape character (`--escape-char`) and use `--format TEXT`, the `--convert-c-escape` option still only looks for backslash `\` sequences.

ybload Advanced Processing Options

The options described in this section are advanced data processing options that you can use to optimize bulk load performance. You can run a specific load operation with default settings and capture statistics for comparison. You may need to try several different configurations before finding the optimal combination of option settings. Before using these options, you need a basic understanding of the phases of a bulk load operation.

Phases of a Bulk Load Operation

A `ybload` operation consists of five main data-processing phases that all occur on the client machine:

1. **Read** (or "slurp"): read raw bytes, separate into rows and fields
2. **Parse**: interpret field strings and convert them into a machine-readable binary representation. *The parse phase is more resource-intensive than the read phase and takes longer to do; it is beneficial to add parallelism at this stage.*
3. **Encode**: combine multiple parsed rows into a binary packet that can be sent to the database
4. **Compress** (optional): compress binary packets before sending
5. **Send**: send packets over the network

The main performance goal is to maximize the write rate in the final send stage (as reported at the end of every successful `ybload` operation).

Factors that can limit the write rate include:

- Speed of network card or other network hardware:
- A 1GB network card will never be able to write more than ~120MB/sec
- Sharing the network with other network-intensive tasks reduces the `ybload` capacity
- Complexity of data types in source files: for example, a file with many TIMESTAMP fields will parse slower than a file with many integers.
- Complexity of CSV options required to properly load the source file: for example, `--allowcontrolascii` requires extra processing.
- Speed of disk where the source files are stored: network disk (such as NFS) is slower than HDD, and HDD is slower than SSD.

To compensate for these factors, `ybload` operations can be tuned to exercise more cores during the read and parse phases. All of the available cores should be used to their saturation point. The following section explains how to set some options for parallel reading and parsing.

Parallel Reading of Source Files

Reading files in parallel maximizes throughput during a load. A `ybload` operation can assign parallel reader processes to a set of source files in two different ways:

- Readers can be spread *within* source files. This method is known as "striding." Multiple readers start reading the same file from different points in the file, then "leap-frog" each other through the file until the whole file is read. Source files are processed *sequentially*.
- Readers can be spread *across* files, such that multiple readers read individual files separately, from beginning to end (without striding). In this case, the source files are read *concurrently*, with different readers starting work on different files at the same time.

In some cases, especially when running on very large computers, `ybload` will use both of these methods to obtain even faster load speeds. The `--read-sources-concurrently` option can be used to achieve this behavior.

The `--read-sources-concurrently` option defines a policy for parallel reading of source files, which is a way to control the amount of load concurrency (resulting in potentially faster load speed). However, faster loads may have to be balanced against the need to preserve data locality (resulting in potentially faster query speed).

For example, to maximize query performance very large tables often require tuning (in the form of effective distribution and/or partitioning choices). You might partition a table by `event_date` so that queries that filter on that column run faster. If you are doing this kind of tuning, you would also want to load data in sorted order (for example, one file per day with records inside each file sorted by time). If `ybload` can preserve the sorted order of the data while bulk-inserting rows, the data will cluster well in the new shards on the storage system, in turn preserving fast query performance. However, if `ybload` were to insert a handful of rows from one date, followed by rows from a different date, and then go back to the first date, the data in the new shards would be mixed and query performance would suffer.

Default ALLOW Setting

When you set the `--read-sources-concurrently` option, you need to consider whether data locality is an issue; it may be relevant to some workloads or some tables, but not all. Given that `ybload` cannot know whether data locality or load speed is more important for a given table, the default setting (`ALLOW`) supports a hybrid approach to concurrency that is based on the following assumptions:

- Separate files contain data from separate periods; therefore, loading them one at a time will best preserve data locality.
- Preserving data locality (by reading one source file at a time) is more important provided that load performance does not suffer too much. How much performance will suffer is a function of the ratio of “slow” files to “fast” files. If more than 50% of the sources are “slow”, they will be read concurrently. In this context, "slow" files are:
- **Compressed files**: slower to read than uncompressed files because of the CPU cost of decompression
- **Remote files** loaded via NFS or from object storage (`nfs://` , `s3://` , `azure://`): often slower to read than local files because of file server load and network overhead.
- **Streamed files** (`STDIN` or named pipes): often slower to read because the data is being generated dynamically by another program.

`ALLOW` is a good choice when you are not sure about the data you are loading and you want `ybload` to make a decision for you. The `ybload` operation will attempt to keep the client system busy by using a combination of concurrent reading across files and parallel reading within files. It will favor sequential file reading in order to preserve data locality, but will allow concurrent file reading in order to keep the client system busy.

Note that the size and number of source files are not factors that you should consider when choosing between concurrent and sequential reading.

Alternative Settings for --read-sources-concurrently

The default `ALLOW` setting is appropriate for most bulk loads, but the following choices are also available:

- `ALWAYS` : always read a set of files concurrently, without striding within files. This is required when `ybload` is reading from multiple named pipes. In rare cases it can also be used to increase `ybload` throughput when `ALLOW` is not aggressive enough in the decisions it makes. This option maximizes load speed (at the cost of data locality and future query speed). `ybload` will automatically choose the actual number of sources to read concurrently (in an attempt to maximize load speed). For example, even if you have 1,000 source files, `ybload` may choose to concurrently read from only 3 of them at a time.
- `<NUMBER>` : similar behavior to `ALWAYS` , but the user specifies how many of the source files to read concurrently.

Note: This option is an advanced feature that should only be used to fine-tune load speed. The proper number to specify should be determined from test runs with different numbers. More is not always better, even when the goal is to maximize load speed.

If `ybload` is sharing a computer with other CPU-intensive or disk-intensive programs, you might even use this mechanism to reduce the read concurrency used by `ybload` and leave some disk/CPU resources for other programs to consume.

- `NEVER` : never read a set of files concurrently; use striding to finish reading each file in sequence. This option is a good choice when temporal locality of the data is important. If you have source files that are pre-sorted by a time-based column and the table you are loading has a `SORT` constraint defined on that column, reading files sequentially one at a time (while allocating parallel readers to work within each file) will maintain the order of the data and provide better load performance. Data locality and future read speed are maximized at the cost of load speed.

Number of Readers and Parsers per Reader

In the context of a `ybload` operation, the number of readers (`--num-readers`) defines the degree of parallel reading on the client system. In the multi-striding case, this means the number of parallel readers per file; in the non-striding case, it means the number of readers available to process a set of files concurrently.

You can increase parallelism during the parse phase by setting the number of parsers per reader (`--num-parsers-per-reader`). The number of readers and number of parsers per reader both affect the *shape of the pipeline*: how the data is consumed from the source and processed before being sent over the network:

- If both options are set to 1 (**--num-readers 1 --num-parsers-per-reader 1**), the shape looks like this:

```
READ -> PARSE -> ENCODE -> COMPRESS -> SEND
```

Parallel reading is disabled in this case.

- If you increase the number of parsers (**--num-readers 1 --num-parsers-per-reader 2**), the shape changes as follows:

```
      /-> PARSE -> ENCODE -> COMPRESS -\
READ -                                  -> SEND
      \-> PARSE -> ENCODE -> COMPRESS -/
```

The parse phase runs in parallel.

- If you also increase the number of readers (**--num-readers 2 --num-parsers-per-reader 2**), the shape changes as follows:

```
      /-> PARSE -> ENCODE -> COMPRESS -\
READ -                                 --\
      \-> PARSE -> ENCODE -> COMPRESS -/   \
                                            \
                                             --> SEND
                                            /
      /-> PARSE -> ENCODE -> COMPRESS -\   /
READ -                                 --/
      \-> PARSE -> ENCODE -> COMPRESS -/
```

The read and parsing phases both run in parallel.

Best Practices for Optimizing Parallel Reading

By default, `--num-readers` is set to the number of cores divided by 2. For example, on a client system with 8 cores, the default number of readers is 4, and on a system with 16 cores, the default is 8. (To disable parallel reading, set `--num-readers` to 1.) By default, `--num-parsers-per-reader` is set to 2.

You need to determine the best settings for these options *per table* that you load, given the variation in size (of the table and its source files) and the data types of the columns. If multiple clients are being used, you also need to consider any variations in the client systems themselves. Additionally, you need to revisit the load configuration whenever a client system changes: for example, after upgrading a client machine from 8 cores with a 1Gb network card to 18 cores with a 10Gb network.

Follow these steps to find the optimal strategy for setting the parallel reading options:

1. Run a load of a moderately sized file with the following options:

```
--num-readers 1 --num-parsers-per-reader 2
```

Run the load through to completion for a minimum of 3 to 5 minutes to make sure you capture reasonably accurate timings. Watch (and record) the write rate as reported by `ybload`.

2. Re-run the same load with the same file, incrementing `--num-parsers-per-reader`. Check the write rate again.
3. Repeat the same load with larger (or smaller) `--num-parsers-per-reader` settings until you identify the value that returns the highest write rate.

4. Repeat the same load with an incremented value for `--num-readers`. Calculate this incremented value as follows:

```
Number of cores / chosen number of parsers per reader
```

Round down the result. For example:

```
16 cores / 4 parsers per reader = 4
18 cores / 5 parsers per reader = 3
```

Record the write rate.

5. If you are already at maximum network capacity, reduce the number of readers to its lowest number while maintaining that maximum speed. If you have not reached maximum network capacity, increase the number of readers until you reach the maximum or do not see any improvement. Choose the largest `--num-readers` value that showed improvement over the next smaller one.
6. Optionally, run the load two more times, using:

- Number of readers from the previous step and parsers per reader `+1`
- Number of readers from the previous step and parsers per reader `-1` Pick the combination that returns the best write rate. Sometimes the best parsers per reader number will be slightly different when readers=1 compared to when readers=the value chosen in this procedure.

Setting a Compression Policy

The `--compression-policy` option defines how the data buffers are formatted on the client before being sent over the network to the compute nodes. This option has three settings:

- `AUTO`: the default; if sockets are busy on the target system, compress the data. This is the optimal setting for most bulk loads.
- `ALWAYS`: always compress the data before sending it. This setting is sometimes a good choice when other applications require network capacity (as well as `ybload`), and potentially slowing down bulk loads is not an issue.

Note: Do not use the `ALWAYS` setting for data protection. Some packets may still be sent without compression (for example, in the rare case that the compressed packet is larger than the uncompressed packet). Also, LZ4 encryption, which is used for compression, is very easy to reverse and cannot be used to protect the data. To protect the data, run `ybload` with the `--secured` option.

- `NEVER`: never compress the data before sending. This option is rarely used but may be applicable if compression does not reduce network bandwidth when loading a particular table.

Reset commit timeout for bulk loading

The `--reset-commit-timeout-on-update-status` option allows the commit / rollback timeout to be reset after each `updateStatus` RPC. The timeout is 60 seconds and ensures that the server waits for a commit message to arrive from the bulk client before rolling back a transaction.

In some scenarios, bulk loading may complete successfully, but the commit message is delayed. To handle such situations and reduce the likelihood of a premature rollback, you can use this option to extend the commit / rollback timeout.

ybload Date Formats

Platforms: All platforms

Parent topic: ybload Command

This section describes the date formats supported by `ybload` operations. A `ybload` date format consists of two parts:

- A date *style*, which describes how a date value is expressed. For example, `YMD` style means that a date value is expected to have year, month, and day pieces, in that order.
- A date *delimiter*, which is the character that separates the date parts, such as a hyphen (`-`) or a backslash (`/`).

You can define acceptable date formats within `--date-field-options`, using the following syntax. In this example, `MONDY2` is the date style and `-` is the delimiter:

```
"formats":[{"style":"MDY","delim":"-"}]
```

You can also define date formats within `--per-field-options`, when the field in question is being loaded into a `DATE` column.

As a shortcut, you can define date styles with the `--date-style` option, but you cannot set the delimiter with that option.

Supported Date Styles

The following table lists the legal date styles that you can use within `--date-field-options`, `--per-field-options`, and `--date-style`.

Date Mask	Examples
YMD	2016-08-13
DMY	13-08-2016
MDY	08/13/2016
MONDY	Aug 13 2016
DMONY	13 Aug 2016
Y2MD	16-08-13
DMY2	13-08-16
MDY2	08/13/16
MONDY2	Aug 13 16
DMONY2	13 Aug 16

Note: If you do not specify date styles explicitly, the default styles that are accepted are `YMD`, `DMONY`, and `MONDY`.

Note the following requirements for defining date styles:

- If you are using a date style with a two-digit year (`Y2`), you need to set the `--y2base` option.
- The bulk load parser recognizes `YMD`, `DMONY`, and `MONDY` automatically, but the other styles need to be set explicitly.
- The `DMY` and `MDY` date styles are ambiguous and cannot be specified together in one `ybload` command.

Supported Date Delimiters

The `delim` setting defines the delimiter between the day, month, and year parts of a date. You can use default supported delimiters within the date string or set a different delimiter.

The following delimiters are supported:

- `auto` (the default), which means any one of the following: " ", ".", "-", "/", "|", ","
- Empty string (`""`). Note that `auto` also allows the empty string (`""`) when `style` is set to `YMD` .
- Any other explicitly specified character sequence. You may have fields to load that have special delimiter strings.

Examples

The following `--date-field-options` setting allows dates with two different date styles to be loaded from the same file:

```
--date-field-options '{"formats":[{"style":"YMD"},{"style":"MDY"}]}'
```

In this case, dates such as `2016-10-05` and `10-05-2016` can be loaded.

The following example extends the first by specifying that the delimiter for both date styles must be a tab character (`\t`):

```
--date-field-options '{"formats":[{"style":"YMD","delim":"\t"},{"style":"MDY","delim":"\t"}]}'
```

In this case, dates such as `2016 10 05` and `10 05 2016` , where the white space represents a tab, can be loaded.

The following example sets the `y2base` year to 1968 and the date style to `MDY2` :

```
--date-field-options {"y2base":1968,"formats":["MDY2"]}
```

In this case, dates such as `10-11-10` are understood to refer to the year 2010.

ybload Field Options

The `ybload` command supports advanced options for parsing fields with different data types. This section covers these options, which you specify in JSON format.

There are three ways to specify field options:

- On a per-field basis, naming specific fields either by ordinal position in the file or by destination column name in the table.
- On a per-type basis, defining behavior for all of the fields that have a given data type.
- Using default settings, when no per-field or per-type settings are defined.

Per-field options, if set, override per-type options. Per-field and per-type options, if set, override default field options.

Default Field Options (--default-field-options)

Use `--default-field-options '{"nullmarker": STRING}'` to specify field options for fields that do not have their own per-field options or per-type options.

Default: `'{}'`

Field Options per Data Type

For every data type that Yellowbrick supports, you can define parsing behavior for all fields of that type. These options are used in the absence of option settings for specific fields.

The integer types (`SMALLINT` , `INTEGER` , `BIGINT`) all support an `inhex` option. When you use this option for one or more fields, `ybload` parses the incoming data in those fields as hexadecimal numbers (instead of decimal values).

- `BIGINT` : `--bigint-field-options '{"nullmarker": STRING, "inhex": "TRUE" | "FALSE"}'`

Default: `'{"inhex":false}'`

- `BOOLEAN` : `--boolean-field-options '{"nullmarker": STRING}'`

Default: `'{}'`

- `BYTEA` : `--bytea-field-options '{"convertCEscape": TRUE | FALSE}'`

Default: `'{"convertCEscape":true}'`

- `CHAR` and `VARCHAR` : `--char-field-options '{"nullmarker": STRING, "emptymarker": STRING, "onZeroChar": REMOVE | ERROR, "convertAsciiControl": BOOLEAN, "convertCEscape": BOOLEAN, "castToVarchar": HEX | ESCAPE, "bytea_output": HEX | ESCAPE"}'`

Default: `'{}'`

- `JSON` : `--json-field-options '{"validateJson": "TRUE" | "FALSE"}'`

Default: `'{}'`

- `JSONB` : `--jsonb-field-options '{"validateJson": "TRUE" | "FALSE"}'`

Default: `'{"validateJson": true}'`

- `DATE` : `--date-field-options '{"nullmarker": STRING, "y2base": NUMBER, "formats": [<DateFormat>...]}'`

Default: `'{"formats":[{"style":"YMD","delim":"auto"},{"style":"DMONY","delim":"auto"},{"style":"MONDY","delim":"auto"}]}'`

See ybload Date Formats.

- `DECIMAL` : `--decimal-field-options '{"nullmarker": STRING}'`

Default: `'{}'`

- `DOUBLE` : `--double-field-options '{"nullmarker": STRING}'`

Default: `'{}'`

- `GEOGRAPHY` : `--geography-field-options '{"nullmarker": STRING}'`

Default: `'{}'`

- `INTEGER` : `--integer-field-options '{"nullmarker": STRING, "inhex": "TRUE" | "FALSE"}'`

Default: `'{"inhex":false}'`

- `IPV4` : `--ipv4-field-options '{"nullmarker": STRING}'`

Default: `'{}'`

- `IPV6` : `--ipv6-field-options '{"nullmarker": STRING}'`

Default: `'{}'`

- `MACADDR` : `--macaddr-field-options '{"nullmarker": STRING}'`

Default: `'{}'`

- `MACADDR8` : `--macaddr8-field-options '{"nullmarker": STRING}'`

Default: `'{}'`

- `REAL` : `--real-field-options '{"nullmarker": STRING}'`

Default: `'{}'`

- `SMALLINT` : `--smallint-field-options '{"nullmarker": STRING, "inhex": "TRUE" | "FALSE"}'`

Default: `'{"inhex":false}'`

- `TIME` : `--time-field-options '{"nullmarker": STRING, "timeformats": [<TimeFormat>...]}'`

Default: `'{"timeformats":[{"style":"HMSs","delim":"auto"},{"style":"HMS","delim":"auto"},{"style":"HM","delim":"auto"}]}'`

See ybload Time Formats.

- `TIMESTAMP WITHOUT TIME ZONE` : `'{"nullmarker": STRING, "y2base": NUMBER, "dateformats": [<DateFormat>...], "timeformats": [<TimeFormat>...]}'`

Unspecified `<TimestampFieldOptions>` fields are populated from the matching fields in `--date-field-options`.

Default: `'{"timeformats":[{"style":"HMSs","delim":"auto"},{"style":"HMS","delim":"auto"},{"style":"HM","delim":"auto"},{"style":"None","delim":"auto"}]}'`

See ybload Timestamp Formats and ybload Time Formats.

- `TIMESTAMPTZ` : `'{"nullmarker": STRING, "y2base": NUMBER, "dateformats": [<DateFormat>...], "timeformats": [<TimeFormat>...], "timezone": STRING}'`

Unspecified `<TimestampTzFieldOptions>` fields are populated from the matching fields in `--timestamp-field-options` .

Default: `'{}'`

See ybload Timestamp Formats and ybload Time Formats.

- `UUID` : `--uuid-field-options '{"nullmarker": STRING}'`

Default: `'{}'`

Options per Field (--per-field-options)

You can specify `--per-field-options` multiple times in a single `ybload` command. The JSON object keys are either source field numbers (1-based) or destination column names. The value for each key must be a `<...FieldOptions>` object that is appropriate for the data type of the field.

The following example sets `DATE` field options specifically for the `salesdate` field:

```
--per-field-options {"salesdate":{"nullmarker":"--NODATE--","y2base":2000,"formats":[{"style":"MONDY2","delim":"_"}]}}
```

The following example accounts for field `5` in the source file:

```
--per-field-options {"5":{"nullmarker":"?"}}
```

The following example applies to field `1` , an integer field, and specifies that `ybload` should parse the field values as hexadecimal numbers.

```
--per-field-options {"1": {"inhex": true}}
```

ybload Options

This section contains detailed descriptions of the bulk load options. Note the following points about the format of these options and their values:

- Options are listed in alphabetical order for quick reference.
- Option names are shown in lowercase; *they are case-sensitive*.
- Specific valid option values (such as `true` and `false`) are shown in lowercase. Variables for option values, such as `STRING`, are shown in uppercase. Option values *are not case-sensitive*.
- The requirements for quoting option strings vary by client platform. Values are shown without quotes, but quotes are sometimes required. For example, if you specify the `#` character in a Linux shell, it must be enclosed by single or double quotes. If you are using a Windows client, see also Escaping Quotes in Windows Clients.
- See also Common Options in ybtools.
- (stdin) : Load from `stdin` (standard input). To load from `stdin` instead of named source files, enter the single-dash character (-) at the end of the `ybload` command. This must be the last character on the command line, and it must be prefixed with `--` to indicate that option parsing is complete. For example: `./ybload -d premdb --username bobr -t match -- -`

@file

Specify a file that includes a set of options and values to use for the operation. See Saving Load Options to a File and Saving Unload Options to a File.

You can use this option and the `--select-file` option in the same `ybunload` command.

--bad-row-file STRING

Define the location and name of a file where rejected rows will be logged. If you do not specify this option, the file defaults to the name `SOURCE_FILENAME.TIMESTAMP.bad` and is written to a location that is reported early in the console or log file output. If the file already exists, it is truncated.

Note:

1. When object storage is used for loading data, bad rows must be written to the local file system. Specifying a bad row file in a
2. Bad row files are not supported for the `GEOGRAPHY` data type. If a `GEOGRAPHY` column contains malformed or unsupported input

--bigint-field-options

See ybload Field Options.

--boolean-field-options

See ybload Field Options.

--bytes-per-transaction

Set the number of bytes to load per commit. The default is 1TB (1099511627776 bytes). You can set this option to modify the frequency of commits when bulk loads are running. This option works in conjunction with `--rows-per-transaction`. The threshold that is met first is applied.

--char-field-options

See ybload Field Options.

--comment-char ASCII_CHARACTER

Define the comment character that is used in source files. The default value is the pound sign (#). The value must be a single ASCII character or a valid escape sequence. If `--skip-comment-lines` is set, commented rows in the source file are *skipped*, not rejected, and do not appear in the bad rows file.

When the `--format` option is set, the comment character may be a single-byte or multi-byte character.

--compression-policy

Define the compression policy for data buffers before they are sent from the client to the compute nodes. See ybload Advanced Processing Options.

--convert-ascii-control, --no-convert-ascii-control

Allow the caret control ASCII character `^@` to be parsed as a single-byte representation of `null` within a character string. The default is `--no-convert-ascii-control`. See also ybload Field Options.

--convert-c-escape, --no-convert-c-escape

Convert (or do not convert) C-style escape sequences when they appear in `CHAR` and `VARCHAR` fields. For example, the two-character sequence `\t` can be converted into a single tab character (`0x09`) or it can be loaded unchanged.

This option does not apply to fields with data types other than `CHAR` and `VARCHAR`. For example, in an `INTEGER` field, the value `4\x35`, where `\x3` is the C-style escape sequence for the number 5, returns an error: `'\' is invalid digit`. (However, the ybsql \copy command will convert C-style escape sequences in all fields.)

This option is only supported when the `--format` option is specified. If `--format text` is used, `--convert-c-escape` is the default behavior. If `--format csv` or `--format bcp` is used, `--no-convert-c-escape` is the default.

--date-field-options

See ybload Date Formats.

--date-style YMD | DMY | MDY | MONDY | DMONY | Y2MD | DMY2 | MDY2 | MONDY2 | DMONY2

Define the date format in terms of the order and style of the date parts, to avoid any ambiguity in parsing dates. For example, `--date-style MDY` means accept date values such as `08-13-2016`.

If you specify one of the `Y2` values, such as `DMY2`, you must also specify a `--y2base` value.

You can specify the `--date-style` option more than once in a single `ybload` command. For example:

```
--y2base 1990 --date-style MDY --date-style MONDY --date-style DMY2
```

Note that you can use JSON-style formatting to abbreviate this syntax.

If `--date-style` is not specified, the defaults derive from `--date-field-options`.

--decimal-field-options

See ybload Field Options.

Note that `ybload` accepts both the dot character (.) and the comma character (,) as the separator for decimal values.

--default-field-options {JSON formatted <FieldOptions>}

Specify the field options to use for fields that do not have their own per-field options or per-type options. See ybload Field Options. The default value is `{}`.

--delimiter SPECIAL_CHARACTER

Define the special character that the source file uses as its field delimiter. All of the following are supported:

- A single Unicode character
- A hex value that corresponds to any ASCII control code (such as `0x1f`)
- A valid escape sequence

When the `--format` option is set, the delimiter may be a multi-byte character.

If you do not specify a field delimiter, `ybload` auto-detects it from among the following characters:

- `,`
- `|`
- `\t`
- `\us`
- `\uFFFA`

See also Setting --format for Flat Files.

--double-field-options

See ybload Field Options.

--dryrun

Do a dry run of the load operation without committing any data to the table. See Using the Dry Run Option.

--duplicate-handler NONE | RANDOM | ORDER BY <sql clause>

Specify how to handle duplicate source rows when the `--write-op` option is set to `insert`, `update`, or `upsert`. (`--duplicate-handler` is ignored for `--write-op delete` loads.)

- `NONE` (the default): source rows are assumed to be unique. The `ybload` behavior is undefined if duplicate source rows exist; `UPDATE` and `UPSERT` operations will sometimes fail.
- `RANDOM`: a random matching source row is used to update each row in the target table.
- `"ORDER BY <sql clause>"`: source rows are sorted, and the first matching row that is found is used to update each row in the target table. For example: `"ORDER BY order_date DESC, order_time DESC"`. Enclose the `ORDER BY` clause in double quotes.

  See ORDER BY Clause for the complete syntax that is supported, with the exception that the `--duplicate-handler` option does not support ordinal numbers to identify columns.

  You cannot use a declared key column in the `--duplicate-handler order by` clause.

  See also Handling Duplicate Rows and Examples with Duplicate Rows.

--emptymarker STRING_WITH_ESCAPES

Define a marker that matches the character used to represent an empty string in your source file (an empty `CHAR` or `VARCHAR` field). The value may be a string, a character, or a valid escape sequence.

You may need to quote the `emptymarker` character. For example, you can use `--emptymarker '\t'` on the command line, but not `--emptymarker \t`.

Note: To use `"` as the `emptymarker` character, you have to quote strings with a different character (`"` is the default). For example:

```
--emptymarker '\"'  --quote-char '|'
```

If null bytes exist within strings, use the `--on-zero-char REMOVE` option.

See also NULL and Empty Markers.

--encoding ENCODING_NAME | -E ENCODING_NAME

Specify the source file's encoding (character set: `UTF8`, `LATIN9`, `ISO-8859-1`, `UTF16`). If no encoding is specified, the encoding of the source file is assumed to match the database encoding. If the encoding of the source file does not match the encoding of the destination database, `ybload` will use Java transcoding to "translate" the source data to the encoding of the destination database.

Check the documentation for the version of Java you are using on the `ybload` client system. For example, if you are using the Oracle JVM, check this list of supported character sets. Names from either of the first two columns can be used with the `--encoding` option. See also the discussion of encodings in Creating Databases.

For the fastest load performance under `ybload`, export the data from the source system in the character set of the destination database. For example, load exported `UTF8` data into a `UTF8` database. Any required transcoding is likely to have an impact on load performance.

Note: To load UTF-16 data, create a `UTF8` database and set the `--encoding` option to `UTF16`.

--escape-char SPECIAL_CHARACTER

Specify an escape character: any single Unicode character. The behavior of this option depends on the `--format` choice:

- `--format CSV`: used to escape embedded quote characters inside quoted fields. The default escape character is `"`.
- `--format TEXT`: used to escape embedded field delimiters and line separators. The default escape character is `\`.
- `--format BCP`: disallowed with `BCP` format, which does not use escape characters.

When the `--format` option is set, the escape character may be a multi-byte character.

--field-defs SQL_NAME | SQL_COLUMN_DEF,...

Define a comma-separated list of source field names or field definitions. A field name simply represents the name of the corresponding column in the table. A field definition also includes the data type and any constraints on the column. Column definitions are used to create temporary table columns for source-only fields, as needed by the `--duplicate-handler` option for `UPDATE` and `UPSERT` loads.

You do not need to specify a list if the source fields match the order, number, and data type of the destination table columns. The `--parse-header-line` option, which only supports field names, overrides the `--field-defs` option.

Field names are case-insensitive unless they are quoted (same behavior as standard SQL).

For example:

```
--field-defs id,name,"Balance",date
```

```
--field-defs id BIGINT PRIMARY KEY,update_order INT NOT NULL,balance,date
```

If the target table has default values assigned for certain columns, you can specify a partial column list with this option and the default values will be loaded for the other columns. For example, if a table `t1` has three columns, `c1`, `c2`, and `c3`, and `c1` was created with a `DEFAULT` constraint, the `--field-defs` list could be `c2, c3`.

See also Loading Generated Key Values.

--format CSV | TEXT | BCP | PARQUET

Specify the formatting style of the incoming data (how the source files were formatted by the export or unload tool that produced them).

For flat files, this option refers to how field delimiters are protected in the data:

- `CSV` : Delimiters in field values were protected by wrapping the field values in quotes. For example: `"2012, Mini Cooper S, ALL4"`
- `TEXT` : Delimiters in field values were protected by preceding them with a backslash escape character. For example: `2012\, Mini Cooper S\, ALL4`
- `BCP` : Delimiters in fields were not protected (for compatibility with the Microsoft SQL Server `bcp` tool). For example: `2012, Mini Cooper S, ALL4`

See also Setting --format for Flat Files and Loading Tables from Parquet Files.

--ignore-emptymarker-case, --no-ignore-emptymarker-case

The first option supports case-insensitive empty-marker comparisons. If you use `--ignore-emptymarker-case`, values of `EMPTY`, `Empty`, and `empty` are all recognized as empty values when `--emptymarker` is set to `NONE`. These options apply globally; you cannot specify them per data type or per field. They are applied globally regardless of how the `--emptymarker` option was specified.

--ignore-nullmarker-case, --no-ignore-nullmarker-case

The first option supports case-insensitive null-marker comparisons. If you use `--ignore-nullmarker-case`, values of `NULL`, `Null`, and `null` are all recognized as null values when `--nullmarker` is set to `NULL`. These options apply globally; you cannot specify them per data type or per field. They are applied globally regardless of how the `--nullmarker` option was specified.

--ignore-unsupported-schema, --no-ignore-unsupported-schema

If `--ignore-unsupported-schema` is specified, `ybload` proceeds with a `parquet` load when source fields contain unsupported data types or a nested structure. See Parquet Load Examples. Note that the load may still fail when a target column for unsupported data is declared `NOT NULL` or has no `DEFAULT` value. The failure occurs because `ybload` proceeds as if the unsupported fields are not in the file.

When `--no-ignore-unsupported-schema` is specified (or neither option is specified, which is the default behavior), `ybload` returns an error when source fields contain unsupported data types or a nested structure. See also Loading Tables from Parquet Files and Parquet Load Examples.

These options do not apply to loads of flat files.

--int96-as-timestamp, --no-int96-as-timestamp

Interpret INT96 data (without a logical type) as a timestamp to provide compatibility with third-party systems, such as Impala, Hive and Spark.

--integer-field-options

See ybload Field Options.

--ip-field-options

See ybload Field Options.

--key-field-names SQL_NAME,...

Comma separated list of source field names to be used as key fields for `--write-op` operations. Key field names must be declared as `NOT NULL` in the `CREATE TABLE` statement.

If this option is not specified, primary keys either specified with the `--field-defs` option or declared by the target table are used. If no key fields are specified and no primary key exists, an attempt to load a table with `--write-op` `delete`, `update`, or `upsert` will result in an error.

Field names are case-insensitive unless they are quoted (same behavior as standard SQL). For example:

```
--key-field-names user_name,"deptId"
```

The `--parse-header-line` option cannot be used in conjunction with the `--key-field-names` option.

--linesep LINE_SEPARATOR

Define the line separator (or row separator) that is used in source files: any single Unicode character or the `\r\n` escape sequence.

If you do not specify a line separator, `ybload` auto-detects it from among the following characters:

- `\n`
- `\r\n`
- `\rs`
- `\uFFFB`

When the `--format` option is set, the line separator may be a multi-byte character.

--locale LOCALE_NAME | -L LOCALE_NAME

The name of the locale to use for parsing dates, timestamps, and so on. If the locale is not specified, the database locale is assumed to be `C`.

Locale names must be of the following form:

```
<language code>[_<country code>[_<variant code>]]
```

For example:

```
--locale en
--locale en_US
--locale zh_CN
```

Variant codes are rarely used; for details, see the Java documentation.

--mac-field-options

Specify field options for `MACADDR` and `MACADDR8` fields. See ybload Field Options.

--max-bad-rows NUMBER

Set the maximum number of rejected rows that `ybload` will tolerate before aborting and *starting* to roll back the transaction. (Additional bad rows may be reported before the transaction has finished aborting.) The default is `-1`, which means do not cancel and roll back, regardless of the number of bad rows.

Note: `--max-bad-rows 32` means 32 bad rows are allowed; the load will fail on the 33rd bad row.

Rejected rows are written to the location specified with the `--bad-row-file` option.

--nullmarker STRING

Define a string that matches the string used to represent `null` in your source file. If this option is unspecified or set to an empty string, adjacent delimiters without text between them are parsed as `NULL` values. This option supports valid escape sequences.

You cannot load data in which more than one value in the same column is intended to be parsed as `NULL`. However, you can load data in which different columns have different values for `NULL`. See Specifying NULL Behavior for Different Columns and NULL and Empty Markers.

--num-cores NUMBER_MIN_1

Set the number of CPU cores `ybload` will attempt to use to `1` or greater. By default, `ybload` tries to saturate all of the CPU cores on the host computer. For example, the default values for `--num-readers` and the number of actual concurrent readers used by `--read-sources-concurrently ALWAYS` are both based on the number of cores on the host computer. The primary purpose of this setting is to restrict `ybload` to the use of fewer resources when it shares the host computer with other programs.

--num-header-lines [NUMBER]

Ignore one or more header lines at the top of the source file. (The first or only header line in a file is typically a list of field names.) The default is `0`, or `1` if `--parse-header-line` is specified. The maximum number is `5`. If you specify multiple source files, the first line is skipped in all of them.

--num-readers

Define the behavior for reading input files in parallel. See ybload Advanced Processing Options.

Note: When you are loading from named pipes, setting this option equal to the number of pipes is recommended. Not loading from pipes concurrently may cause a deadlock with the program that the pipes communicate with.

--num-parsers-per-reader

Define multiple parsers per reader. See ybload Advanced Processing Options.

--object-storage-*

See Object Storage Options. These options apply to loads and unloads from and to Azure, AWS S3, and S3-compatible systems.

--on-extra-field [REMOVE | ERROR]

Specify `REMOVE` to allow rows to be loaded when the source file (`CSV`, `TEXT`, or `BCP`) contains either more fields than the columns defined in the header, as detected when the `--parse-header-line` option is used, or more fields than the columns defined with the `--field-defs` option.

If neither `--parse-header-line` nor `--field-defs` is specified, `--on-extra-field` has no effect; `ybload` expects the source file to have the same number of fields as the number of columns in the target table. The `--parse-header-line` option, if specified, overrides the `--field-defs` option.

This option takes effect only when the extra fields are at the end of the line in the source file. For example:

```
colA, colB, colC           <- header
data1, data2, data3, data4  <- on-extra-field
```

The default behavior (`ERROR`) is to reject rows with extra fields at the end of the line.

--on-invalid-char [REPLACE | ERROR]

Specify the action to take when the source file contains characters that cannot be represented in the database (or characters that are invalid in the source file itself):

- The replacement character for a LATIN9 database is the question mark: `0x3F` (`?`).
- The replacement character for a UTF8 database is the Unicode replacement character: `U+FFFD` (a question mark in a diamond).

The default is `REPLACE`.

--on-missing-field [SUPPLYNULL | ERROR]

Specify `SUPPLYNULL` to allow rows to be loaded when the source file (`CSV`, `TEXT`, or `BCP`) contains either fewer fields than the columns defined in the header, as detected when the `--parse-header-line` option is used, or fewer fields than the columns defined with the `--field-defs` option.

If neither `--parse-header-line` nor `--field-defs` is specified, `--on-missing-field` has no effect; `ybload` expects the source file to have the same number of fields as the number of columns in the target table. The `--parse-header-line` option, if specified, overrides the `--field-defs` option.

This option takes effect only when the missing fields are at the end of the line in the source file. For example:

```
colA, colB, colC    <- header
data1, data2        <- on-missing-field
```

The default behavior (`ERROR`) is to reject rows with missing fields at the end of the line.

--on-string-too-long TRUNCATE | ERROR

Truncate character strings that are longer than the specified column width (or return an error). The default is `ERROR`.

--on-unescaped-embedded-quote [PRESERVE | ERROR]

Preserve or return an error when unescaped quotes are found inside quoted strings. The default is `ERROR`.

--on-zero-char [REMOVE | ERROR]

Remove null bytes (`0x00` characters) that appear within strings in `CHAR` and `VARCHAR` fields (or return an error). The default is `ERROR`.

--parse-header-line

Use the header line at the top of the source file (a list of field names) to determine the column names in the target table. This option overrides the `--field-defs` option.

--per-field-options {JSON Object}

Specify parsing options for individual fields. See ybload Field Options.

--pre-parse-buf-size NUMBER_MIN_1

Specify the size of the pre-parse buffers, which should be larger than the size of any single row in the source data. The default value is `192744` bytes. (The maximum width of a row in a table is 64231 bytes.)

--quote-char SPECIAL_CHARACTER

Define the character that is used in source files to quote field values that contain embedded delimiters. Specify any single Unicode character. The default is `"`. This option only applies when you are using `--format CSV`.

When `--format CSV` is set, the delimiter may be a multi-byte character.

--read-sources-concurrently ALWAYS | NEVER | ALLOW | <NUMBER>

Define the behavior for reading source files in parallel: `ALWAYS`, `NEVER`, `ALLOW` (the default), or a specific number of source files. See ybload Advanced Processing Options.

Note: If you are loading from multiple pipes and this option is not set to `ALWAYS`, `ybload` sets it to `ALWAYS` and returns an `INFO` message stating that change. (Not reading from pipes concurrently could cause a deadlock with the program that they communicate with.)

--real-field-options

See ybload Field Options.

--resume-partial-load-from-offset NUMBER

Skip the specified number of bytes in the source file in order to resume a failed bulk load. The default value is `0`. See Resuming a Partial Load.

--rows-per-transaction

Set the number of rows to load per commit. The default is set to the maximum number of rows that can be loaded (263 – 1). You can reduce this number to increase the frequency of commits when bulk loads are running. This option works in conjunction with `--bytes-per-transaction`. The threshold that is met first is applied.

--serialize-nested-as-json, --no-serialize-nested-as-json

If `--serialize-nested-as-json` is specified, `ybload` processes nested fields in complex `parquet` files as JSON. If `--no-serialize-nested-as-json` is specified (or neither option is specified), `ybload` returns an error when attempting to map complex nested fields to the columns in the target table. These options do not apply to loads of flat files. See Loading Tables from Parquet Files and Parquet Load Examples.

--skip-blank-lines, --no-skip-blank-lines

Skip blank lines in the source file (`true`) or detect blank rows as bad rows (`false`). The default value is `true` .

--skip-comment-lines, --no-skip-comment-lines

Skip lines that are commented out in the source file or detect them as bad rows. The default value is `false` (do not skip).

Note: If you use `--skip-comment-lines` , make sure the data does not contain any lines that begin with the comment character (which defaults to `#` but may bet set to a different character with the `--comment-char` option).

--skip-missing-files, --no-skip-missing-files

If a file is removed after `ybload` has expanded the path to the source files into a list, skip the file and continue the load operation. This option is useful when millions of source files (or serveral very large files) are being loaded over an extended period of time. As the load progresses, it is possible that an administrator may delete one or more files from the source location. You may or may not want to stop the load in this situation. The default behavior is to not skip missing files.

When a load proceeds because a missing file is skipped, you will see a message like this:

```
Skipped [filepath] because the file is removed after listing
```

--skip-open-failed-files, --no-skip-open-failed-files

If a file fails to open after `ybload` has expanded the path to the source files into a list, skip the file and continue the load operation. It is possible that the original file listing contains one or more files that have been modified and cannot be opened (for example, if permissions on the file were changed after the load started). You may or may not want to stop the load in this situation. The default behavior is to not skip files that fail to open.

When a load proceeds because a file that fails to open is skipped, you will see a message like this:

```
Skipped [filepath] because the file can't be opened
```

This option only works when no bytes have already been read from the file in question. Files read from object storage may be loaded in parts, but a load will fail rather than skip a file that has been partially read.

--smallint-field-options

See ybload Field Options.

--source-compression NONE | AUTO | GZ | BZIP2 | XZ | PACK200 | LZ4

This option explicitly defines the type of compression used by source data and is primarily intended for data sources that do not have file names (such as `STDIN`).

The default value is `AUTO` . When specified, this option overrides other compression detection methods. If this option is not specified, for flat files `ybload` auto-detects the compression type based on their file name extension.

For `parquet` files, `ybload` auto-detects the compression type based on metadata in the files and supports the following compression types: `SNAPPY` , `ZSTD` , and `LZO` . You do not need to specify the `--source-compression` option for `parquet` files.

--table (or -t)

Name the target table to load and optionally its schema: `schema_name.table_name` . If you do not specify the schema name, the table is assumed to be in the `public` schema. The schema of the target table is not based on the user's `search_path` , regardless of how it is set. The table must exist in the database that you are connecting to for the load. (Do not try to specify the database name as part of the table name. Use the `YBDATABASE` environment variable, `-d` , or `--dbname` .)

Note: If you used a quoted case-sensitive identifier to create the target table, you must quote the table name and escape the quotes in the `ybload` command line. For example, if your table is named `PremLeagueStats` :

```
-t \"PremLeagueStats\"
```

--time-field-options

See ybload Field Options.

--timestamp-field-options

See ybload Field Options.

--timestamptz-field-options

See ybload Field Options.

--trim-white, --no-trim-white

Trim or retain leading and trailing whitespace characters in each field. With `--format csv`, whitespace characters inside of quotes are always preserved. The default value is `false` (preserved).

--trim-trailing-white, --no-trim-trailing-white

Trim or retain trailing whitespace characters in each field. With `--format csv`, whitespace characters inside of quotes are always preserved. The default value is `false` (preserved).

--truncate-before-insert, --no-truncate-before-insert

Truncate the target table before inserting new rows. Use this option if you want to ensure that the load runs against an empty table. The `TRUNCATE` statement is executed in the first transaction of the bulk load session. If the bulk load fails before that transaction is committed, the `TRUNCATE` is rolled back.

To use this option, you must have `DELETE`, `TRUNCATE`, and `INSERT` privileges on the target table (in addition to `BULK LOAD` privilege on the database).

--uuid-field-options

See ybload Field Options.

--write-op INSERT | UPDATE | DELETE | UPSERT

How the source rows should be written to the target table.

- `INSERT` (the default): insert source rows as new rows (append them to the table).
- `UPDATE`: update rows that match source rows.
- `DELETE`: delete rows that match source rows.
- `UPSERT`: update rows that match source rows, and insert new rows.

Updates, deletes, and upserts require primary keys or declared key fields to match against the target table. See `--key-field-names`.

By default, source rows are inserted (appended to the table) and duplicate rows are not discarded. Incoming rows are assumed to be unique. Updates and upserts may require duplicate handling. See the `--duplicate-handler` option.

Note: Make sure the user running `ybload` has `BULK LOAD` privileges on the database and appropriate additional privileges on the target table:

- `INSERT` for default loads
- `UPDATE` and `SELECT` for `--write-op update` and `--write-op upsert` loads
- `DELETE` and `SELECT` for `--write-op delete` loads

See also Deletes, Updates, and Upserts and Delete, Update, and Upsert Examples.

--y2base YEAR

Define the pivot year (such as `1970`) for two-digit year values (such as `97` and `16`). For example, if `--y2base` is set to `1970`, two-digit years of `70` and later are assumed to be in the 1900s. Values of `69` and earlier are assumed to be in the 2000s.

If you want to specify one of the `Y2` values for `--date-style`, such as `DMY2`, you must also specify a `--y2base` value.

ybload Time Formats

This section describes the `time` formats supported by `ybload` operations.

A valid time value expresses the time of day without a date or a time zone. The value consists of hours, minutes, and seconds of the form `HH:MM:SS.ssssss`, where `ssssss` refers to a maximum of six digits for fractional seconds. Time values must fall in the range of `00:00:00` to `24:00:00`.

Time values may also include the `am` or `pm` designation. For example:

```
02:02:45.6789 am
```

The space character before the `am` or `pm` label is optional.

In `--time-field-options`, you can specify `timeformats` with one of the following `style` entries. (The same style choices can be applied with `--timestamp-field-options` and `--timestamptz-field-options`.)

Time Format	Examples
`HMSs`	`00:05:30.000001` `23:59:59.999`
`HMS`	`00:05:30` `23:59:59`
`HM`	`00:05` `23:59`

For example, your `ybload` command could specify the following syntax:

```
--time-field-options '{"timeformats":[{"style":"HMSs"}]}'
```

This example specifies that all `TIME` columns in the table you are loading will accept time values only if they contain fractional seconds. Therefore, a row that contains `00:05:30.000001` in a `TIME` column would be loaded, but a row that contains `00:05:30` would be rejected.

ybload Timestamp Formats

Platforms: All platforms

Parent topic: ybload Command

This section describes the `timestamp` and `timestamptz` formats supported by `ybload` operations. These formats consist of the following parts:

- A date *style*, which describes how a date value is expressed. See date formats.
- A date *delimiter*, which is the character that separates the date parts. See date formats.
- An optional time component, which expresses the time of day. If no time is expressed in the loaded value, the default time value that `ybsql` returns is `00:00:00` for a `timestamp` column or `00:00:00-07` for a `timestamptz` column (where `07` represents the time zone of the current session).
- An optional time zone string (`timestamptz` only), expressed as a UTC offset value with the following format: `+xx:xx` or `-xx:xx`. Yellowbrick recommends that you export `timestamptz` data in this format before attempting to load it.

If no time zone is expressed in the loaded value, the default value that `ybsql` returns is exactly the timestamp you loaded with no offset. If a time zone is expressed, the value that `ybsql` returns is an offset of the timestamp you loaded based on the specified time zone. See the examples in the following table. See also Differences in Time Zone Parsing for Bulk Loads and INSERTs.

The space in between the time component and the time zone string (UTC offset) is optional. For example, all of the following values are valid:

```
2017-01-01 01:01:01 +05:00
2017-01-01 01:01:01+05:00
2017-01-01 01:01:01 -05:00
2017-01-01 01:01:01-05:00
```

Note: The `ybload` utility supports IANA time zone names, such as `US/Pacific` and `Europe/Dublin`, but does not support other typical time zone abbreviations such as `PST` or `PDT`.

For example, the following value is valid:

```
2017-01-01 01:01:01 US/Pacific
```

The following value is not valid:

```
2017-01-01 01:01:01 PDT
```

The following examples show several supported `timestamptz` values as they appear in the source file for a bulk load and as they are returned in query results:

Timestamp Value in ybload Source File	Value as Returned by ybsql (US/Pacific time zone for the session)

Timestamp Value in ybload Source File	Value as Returned by ybsql (US/Pacific time zone for the session)

`2017-05-06 15:00:00`
`2017-05-06 15:00:00 -05:00`
`2017-05-06 15:00:00 +05:00`
`2017-05-06 15:00:00-05:00`
`2017-05-06 15:00:00+05:00`
`2017-05-06 15:00:00 Europe/Dublin`
	`2017-05-06 15:00:00-07`
	`2017-05-06 13:00:00-07`
	`2017-05-06 03:00:00-07`
	`2017-05-06 13:00:00-07`
	`2017-05-06 03:00:00-07`
	`2017-05-06 07:00:00-07`

You can define a mask for acceptable `timestamp` and `timestamptz` formats within `--date-field-options`, using the following syntax:

```
--timestamptz-field-options '{"nullmarker": STRING, "emptymarker": STRING, "y2base": NUMBER, "dateformats": [<DateFormat>...], "ti
```

For example, the following load sets the time zone for all `timestamptz` fields:

```
./ybload -d premdb --username bobr --password -t matchtz
--timestamptz-field-options '{"timezone": "+05:00"}' /home/ybdata/matchtz.csv
```

You can also define masks for timestamp formats within `--per-field-options`, when the field in question is being loaded into a `TIMESTAMP` or `TIMESTAMPTZ` column.

Differences in Time Zone Parsing for Bulk Loads and INSERTs

`ybload` and `INSERT` operations interpret some time zone strings in timestamps differently, resulting in different values being stored and returned for what may appear to be the same source values:

- In accordance with the ISO 8601 standard, the `ybload` client interprets time zone strings such as `GMT+05`, `UTC+05`, or `+05` as offsets that are *east of Greenwich Mean Time*.
- When you run `INSERT` statements, the parser uses the POSIX standard for time zones. Therefore, strings such as `GMT+05` and `UTC+05` are interpreted as *west of Greenwich*. The abbreviated form, `+05`, is an exception to this behavior; it is interpreted as *east of Greenwich*, in agreement with `ybload` parsing.

Because of these differences, you may need to take special care when you are either loading tables with `timestamptz` columns or inserting values into those columns.

The following table shows some examples of equivalent time zone strings, as specified via `ybload` and `INSERT` commands.

Time Zone String in ybload	Equivalent Time Zone String in INSERT
`GMT+05`	`GMT-05`
`UTC+01`	`UTC-01`
`GMT-10`	`GMT+10`
`UTC-07`	`UTC+07`
`+05`	`+05` (same as `ybload`)
`-07`	`-07` (same as `ybload`)

To put these differences in context, assume that the following three rows are loaded via `ybload` into a table `loadtz`:

```
1,'2017-05-06 15:00:00 GMT+05'
2,'2017-05-06 15:00:00 UTC+05'
3,'2017-05-06 15:00:00 +05'
```

The results will be three identical timestamps:

```
premdb=# select * from loadtz order by 1;
 c1 |          c2
----+-----------------------
  1 | 2017-05-06 03:00:00-07
  2 | 2017-05-06 03:00:00-07
  3 | 2017-05-06 03:00:00-07
(3 rows)
```

If you use `INSERT` statements to append three more rows to the table, using constant values for the same timestamps, the results will be as follows:

```
premdb=# insert into loadtz values(4,'2017-05-06 15:00:00 GMT+05');
INSERT 0 1
premdb=# insert into loadtz values(5,'2017-05-06 15:00:00 UTC+05');
INSERT 0 1
premdb=# insert into loadtz values(6,'2017-05-06 15:00:00 +05');
INSERT 0 1

premdb=# select * from loadtz order by 1;
 c1 |          c2
----+-----------------------
  1 | 2017-05-06 03:00:00-07
  2 | 2017-05-06 03:00:00-07
  3 | 2017-05-06 03:00:00-07
  4 | 2017-05-06 13:00:00-07
  5 | 2017-05-06 13:00:00-07
  6 | 2017-05-06 03:00:00-07
(6 rows)
```

Note that only row `6` matches the loaded rows. Rows `4` and `5` match each other.

SQL-Based Loads from External Storage

Platforms: EE: All cloud platforms

Parent topic: Load Tables

In this section:

This section covers commands that support SQL-based data loads from source files stored outside the database in object stores. These SQL commands provide "data movement" functionality that is accessible directly from database applications. Users can connect to object storage via SQL objects and load tables from data that is stored and updated externally.

Important: For efficient data loading from object storage, your target tables must be regular user tables, as defined with standard `CREATE TABLE` statements.

For details about loading tables with the `ybload` client, see Bulk Loading Tables.

LOAD TABLE Steps

Platforms: EE: All cloud platforms

Parent topic: Load Data with SQL

Follow these steps to set up a load from an object storage bucket into a regular table by using a series of related SQL commands. This is a generally more flexible approach than using the **Load Assisant** in Yellowbrick Manager because the SQL can be scripted and reused.

1. Make sure you have an active cloud storage account and know your endpoint URL, security credentials, bucket or storage account name, and so on.
2. Upload the source files that you want to load from into the bucket or container if they are not already there. Make note of their file type (`CSV` or `gzip` , for example).
3. If the target table for the load does not already exist, create a *regular user table* that you can load from the source files. Use a `CREATE TABLE` or `CTAS` statement. Make sure that the columns and data types align with the source data.
4. Create the following external objects. If you are creating them for the first time, create them in this order to avoid dependency issues:
5. An *external storage* object that defines your object storage endpoint, region, and account credentials. See CREATE EXTERNAL STORAGE.
6. An *external format* object that defines the file type and the load options that describe the format of the source files, such as the delimiter and how to handle different field types. (These options have a different naming convention but they correspond in behavior to a subset of the options that are used by the `ybload` client.) See CREATE EXTERNAL FORMAT.
7. An *external location* object that specifies the bucket or container for the source files and provides references to the external storage and format objects that you already created. See CREATE EXTERNAL LOCATION.
8. Load the target table, using a LOAD TABLE statement. Specify the set of files you want to load, the external location, and the external format. You can also specify some load-processing options; again, these options correspond in behavior to a subset of the options that are used by the `ybload` client.
9. Monitor and check the results of the load by running system view queries (`sys.load` , `sys.query`). You can also use the activity reports in the Yellowbrick Manager.

Tip: For more details about creating target tables and using `ybload` options, see Loading Tables.

Loading a Table via the Load Assistant

Platforms: EE: All cloud platforms

Parent topic: Load Data with SQL

You can use the Yellowbrick Manager **Load Assistant** to automate the process of loading a table from files in object storage. In the background, SQL commands are generated and run to complete this procedure, culminating with a step that loads the table from qualifying source files in the specified bucket or container.

You can see the syntax of the SQL commands as you go along. To get more information about these commands, or if you prefer to run them manually, see SQL Commands.

Note: Normally you would need your account information and credentials ready at hand before loading tables. When you are loading anonymously from a public bucket or container, as in this tutorial, no credentials are required. This example loads from AWS S3 data, but the steps are the same when you load from Azure object storage.

1. Create the `premdb` sample database and run both setup scripts, as described in step 3 of the walkthrough.
2. Navigate to the *Load Assistant* (⇫ icon in the leftmost pane)
3. Select your instance, the `yellowbrick_trial` database, and the `small-load` cluster,
4. Select the external location called `premdbs3data`.
5. Clear the **Suffix** field (defaults to `/`).
6. Select **Specific Files**, then enter the string `/premdb/newmatch` in the **File Prefix** field.

Important: If you do not filter by **Specific Files** or **Pattern**, the system will attempt to load all the files in the specified location. In turn, the load will only proceed if files are found that have the correct layout of source fields and data types for the target table.

Select this file and click **Next** to proceed.

See also Notes on the Manage External Resources Option.

7. On the **Format Data** page, accept the default format and click **Next**.

In the context of this tutorial, using the default format (`premdbs3format`) that is associated with the external location (`premdbs3data`) is recommended. Alternatively, you can click **Customize Format** and explicitly set the format options you want to use.

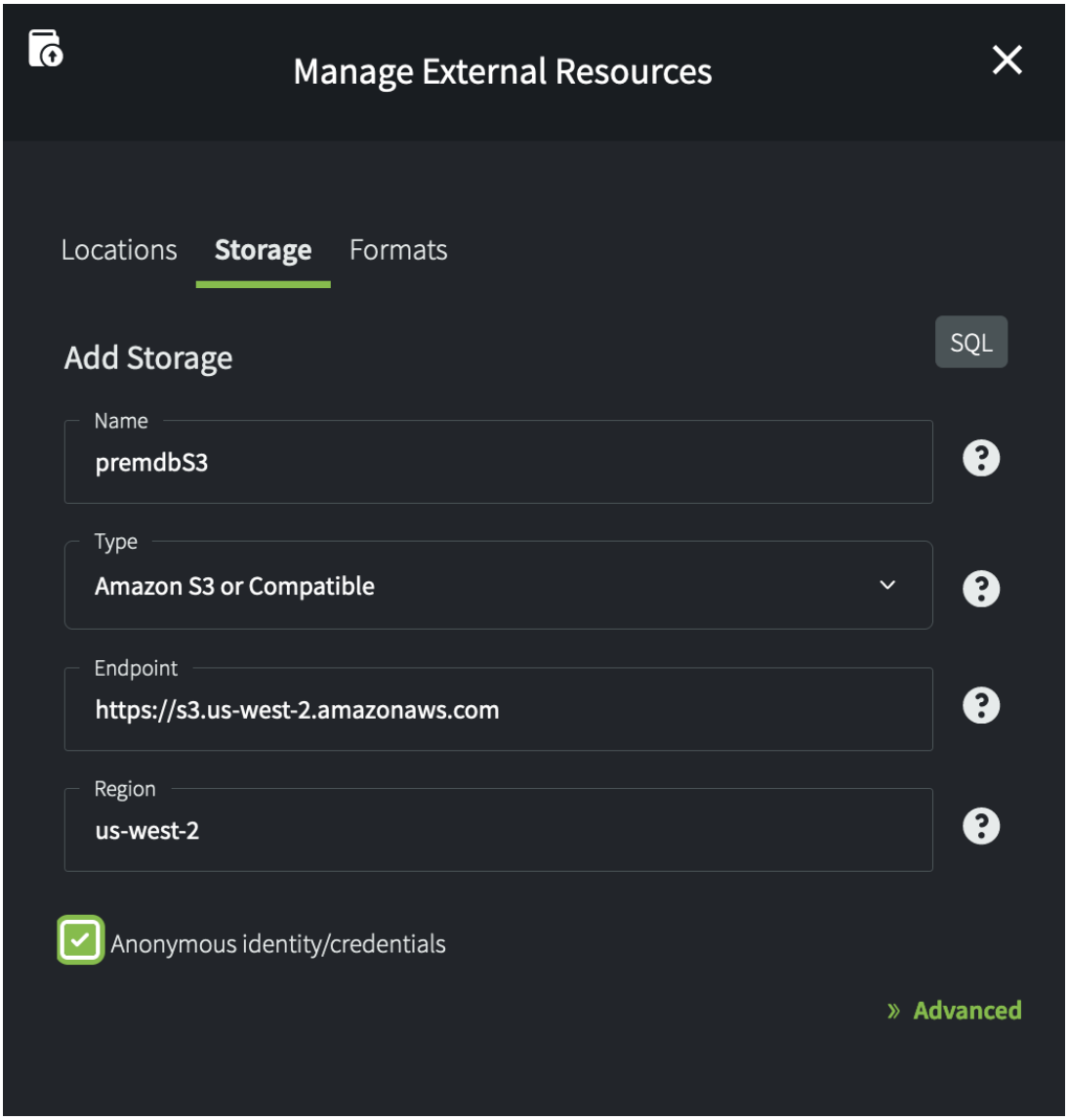
8. Select the target schema (`premdb`) and table (`newmatchstats`), accept the default values for error handling and load options, and click **Load**.
9. Run the load and watch its progress. When it is complete, you will see a load summary indicating that the load is complete, with approximately 24.7M additional rows. Click the **Log** button if you want to download and inspect the log file for the load.
10. Finally, in the Query Editor, select the table to check its updated contents.

Notes on the Manage External Resources Option

In this tutorial you used external resources (objects) that were already created for you in the script you ran to create the `premdb` tables. Alternatively, in the Load Assistant, you could click **Manage External Resources** and work through the setup of three different objects:

- An external storage object
- An optional external format object that describes the format of the source files in the bucket or container
- An external location that defines the bucket or container name

Tip: When you are creating these objects for the first time in Yellowbrick Manager, start by creating an external storage object. For example:



In the format section, click **Next** to use the default format (the format you just created). Alternatively, you could select another format from the drop-down list, or an alternative customized format (**Customize Format**).

Loading Data from Object Storage

In this section:

This section describes how to use `ybload` to bulk load tables from supported object storage systems.

See also Object Storage Options and SQL-Based Loads from External Storage.

Note: When object storage is used for loading data, bad rows must be written to the local file system. Specifying a bad row file in an object storage location, such as an S3 bucket, is not supported.

Loading from Amazon S3

In this section:

These instructions assume that you already have an Amazon Simple Storage Service (Amazon S3) account, know the S3 location of the files you want to load, and have your S3 credentials ready at hand. If not, follow the instructions on the AWS site before proceeding.

To bulk load a table from files in an Amazon S3 bucket:

1. Obtain your S3 security credentials (in this example, an access key ID, a secret access key, and the region). See S3 Authentication Methods for information about alternative ways to authenticate.

2. Run `aws configure` or use `set` commands to set the environment variables for the S3 access keys and the region.

For example, on Linux:

```
$ aws configure
AWS Access Key ID [***************SWUA]: XXXXXXXXXXXXXXXXXXX
AWS Secret Access Key [***************n2L+]: ************************************
Default region name [None]: us-east-2
Default output format [None]:
```

For example, on Windows:

```
set AWS_ACCESS_KEY_ID=XXXXXXXXXXXXXXXXXXX
set AWS_SECRET_ACCESS_KEY=****************************************
set AWS_REGION=us-east-2
```

You can also specify `ybload` command-line options to authenticate with access keys.

3. Record the S3 location of the file you want to load. For example:

```
s3://ybbob/premdb/match01.csv
```

where `ybbob` is the S3 bucket name and `premdb` is a folder in that bucket.

You can only access one object store per `ybload` operation. For example, you cannot specify connection details for both an S3 client and an Azure client in the same command, and you cannot load data from multiple S3 endpoints or storage accounts.

For more details, see S3 Object Storage URIs.

4. Run the `ybload` command in the usual way, providing the name of a bad row file on the local client system and the S3 path to the source file. For example:

```
$ ybload -d premdb --username bobr -W -t match --format csv --delimiter ','
--bad-row-file /home/brumsby/s3bad s3://ybbob/premdb/match01.csv
```

To load multiple source files from the `premdb` folder with the prefix `match`, you could use:

```
s3://ybbob/premdb/match
```

Alternatively, list each S3 file separately. For example:

```
s3://ybbob/premdb/matchst01.csv s3://ybbob/premdb/match02.csv s3://ybbob/premdb/matchst03.csv
```

If you specify a log file, it must also be on the local file system (like the bad row file).

S3 Authentication Methods

Platforms: All platforms

Parent topic: Loading from Amazon S3

There are several different ways to authenticate when you are using `ybload` to load data from AWS S3 or S3-compatible object storage. You can authenticate implicitly by using supported S3-specific mechanisms, or explicitly by using `ybload` command-line options. Your organization's S3 administrator should provide instructions for the approach you should use. See Best Practices for Managing AWS Access Keys for further recommendations.

S3 credentials must be provided in a manner supported by `ybload` and the AWS Java SDK:

- Secure methods (integrated into your organization's login/identity mechanism):
- EC2 roles (when running on Amazon EC2 instances)
- SAML 2.0-compatible identity provider
- Custom identity provider bridge to AWS
- Other methods:
- Object Storage Options
- Environment variables: `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`. Other AWS environment variables are also supported, including `AWS_SESSION_TOKEN`, `AWS_PROFILE`, `AWS_REGION`, and `AWS_CREDENTIAL_PROFILES_FILE`.
- URI query parameters: `aws_access_key_id` and `aws_secret_access_key`
- A credential file, typically located at `~/.aws/credentials` (location may vary by platform)
- Required Permissions
- Note that `ybload` requires access to read both the object and object metadata. Metadata access is a grantable permission separate from read access. Attempting to load a file from a public bucket with the option to download the file using an HTTP URL will cause `ybload` to fail and throw the following error:

```
Forbidden (Service: Amazon S3; Status Code: 403; Error Code: 403 Forbidden; Request ID: xxxxxxxxxxxxxxx)
```

An installation of the AWS CLI is not required, but it does provide the `aws configure` command, which is useful for setting credentials. For details, see the AWS Command Line Interface documentation.

Order of Precedence for Authentication Methods

You may have multiple credential settings available to you, based on your AWS account setup and how you script the `ybload` command. Therefore it is important to know which authentication mechanism takes precedence when the `ybload` command is run. Note that credentials set explicitly with `ybload` object storage options, if set (either on the command line or in a properties file), always take precedence over implicit credentials set via `aws configure` or other methods.

The order of precedence is as follows:

1. Access key and access key ID specified with the `--object-store-identity` and `--object-store-credential` command-line options

2. Access key and access key ID specified *with* the `yb.file.` prefix in one of the following:
3. URI parameter
4. Properties file named with the `--object-store-provider-config` command-line option
5. Access key and access key ID *without* the `yb.file.` prefix in one of the following:
6. URI parameter
7. Properties file named with the `--object-store-provider-config` command-line option
8. Implicit authentication via environment variables and supported AWS configuration files: `~/.aws/*` files: (`~/.aws/credentials` or `~/.aws/config`)

S3 Object Storage Examples

Platforms: All platforms

Parent topic: Loading from Amazon S3

The following examples show different ways to load a table from an S3 bucket:

Load a table from an AWS S3 bucket, using aws configure for authentication:

This example uses the `aws configure` command to set the access key, identity, and region:

```
$ aws configure
AWS Access Key ID [****************SWUA]: XXXXXXXXXXXXXXXXXXX
AWS Secret Access Key [****************n2L+]: ************************************
Default region name [None]: us-east-2
Default output format [None]:
```

```
$ ybload -d premdb --username bobr -W -t match --format csv --delimiter ',' --bad-row-file /home/brumsby/s3bad
s3://ybbob/premdb/match.csv
Password for user bobr:
14:01:43.158 [ INFO] ABOUT CLIENT:
...
14:01:43.342 [ INFO] Assuming source encoding matches database server encoding: LATIN9
14:01:44.923 [ INFO] Starting source s3://ybbob/premdb/match.csv
14:01:45.053 [ INFO] Using database locale: C
14:01:45.056 [ INFO] Auto-detected line separator = '\n'
14:01:45.189 [ INFO]
...
14:01:47.665 [ INFO] SUCCESSFUL BULK LOAD: Loaded 8606 good rows in   0:00:04 (READ: 69.12KB/s WRITE: 64.59KB/s)
```

Load the table again, using ybload --object-store options for authentication:

This example loads the same table again but uses `ybload` options to set the identity, credential, and region values:

```
$ ybload -d premdb --username bobr -W -t match --format csv --delimiter ',' --bad-row-file /home/brumsby/s3bad
--object-store-identity 'XXXXXXXXXXXXXXXXXXX'
--object-store-credential '************************************'
--object-store-region 'us-east-2'
s3://ybbob/premdb/match.csv
Password for user bobr:
14:23:20.736 [ INFO] ABOUT CLIENT:
...
14:23:20.807 [ INFO] Gathering metadata on input files
14:23:20.818 [ INFO] Loaded file handler for s3://
14:23:20.857 [ INFO]
Configuration (S3 client):
   endpoint           : null
   region             : us-east-2
   identity/credential: ********
14:23:20.924 [ INFO] Assuming source encoding matches database server encoding: LATIN9
14:23:22.744 [ INFO] Starting source s3://ybbob/premdb/match.csv
...
```

Load the table again, using a properties file for authentication:

This example loads the same table again but uses a Java properties file to set the identity, credential, and region values. First create the properties file, then name the file with the `--object-store-provider-config` option:

```
$ more ybbobrS3.properties

yb.file.region = us-east-2
yb.file.identity = XXXXXXXXXXXXXXXXXXX
yb.file.credential = ************************************

$ ybload -d premdb --username bobr -W -t match --format csv --delimiter ',' --bad-row-file /home/brumsby/s3bad
--object-store-provider-config ybbobrS3.properties
s3://ybbob/premdb/match.csv
Password for user bobr:
16:58:02.579 [ INFO] ABOUT CLIENT:
...
16:58:02.658 [ INFO] Gathering metadata on input files
16:58:02.673 [ INFO] Loaded file handler for s3://
16:58:02.713 [ INFO]
Configuration (S3 client):
   endpoint           : null
   region             : us-east-2
   identity/credential: ********
...
```

S3 Object Storage URIs

Platforms: All platforms

Parent topic: Loading from Amazon S3

The `ybload` client supports several patterns for Amazon S3 and S3-compatible URIs. All of these patterns identify the location of the source data for a bulk load operation.

URI Pattern	Loads	Example
`s3://BUCKET`	All keys (files) in the bucket	`s3://premdb`
`s3://BUCKET/KEY`	Current version of the named key	`s3://premdb/team01.csv`
`s3://BUCKET/KEY?versionId=VERSION`	Specific version of the named key	`s3://premdb/team01.csv?versionId=w_B5qT8s5MkiT09.IRHay0lW.PycsHTS`
`s3://BUCKET/KEY_PREFIX`	All keys that match the prefix	`s3://premdb/team`

You can specify multiple URIs on the `ybload` command line if you want to identify multiple source files explicitly. However, the sources must belong to the same S3 endpoint.

A valid URI must not contain a space character unless it is encoded with the `%20` escape sequence. For example, if you upload a `season.csv` source file into `ybbobr/premdb/new files`, the following URI will work:

```
s3://ybbobr/premdb/new%20files/season.csv
```

Note: The following "path-based" URI patterns are not supported:

```
http://SUB.s3.DOMAIN/BUCKET/KEY (or https)
http://s3.DOMAIN/BUCKET/KEY (or https)
```

URI Parameters

As an alternative to specifying `ybload --object-store` options, you can specify the S3 configuration in the form of URI parameters. To use a URI parameter, drop the `--object-store-` prefix from the `ybload` option name. For example, the `--object-store-endpoint` option is specified in the following URI as `endpoint=`:

```
s3://my_bucket/my_key?endpoint=http://nycloadsrv1.nyc.yellowbrick.io:9000
```

Note: If you provide multiple source URIs in a single `ybload` command, only the first URI will be checked for configuration options.

Loading from Azure Blob Storage

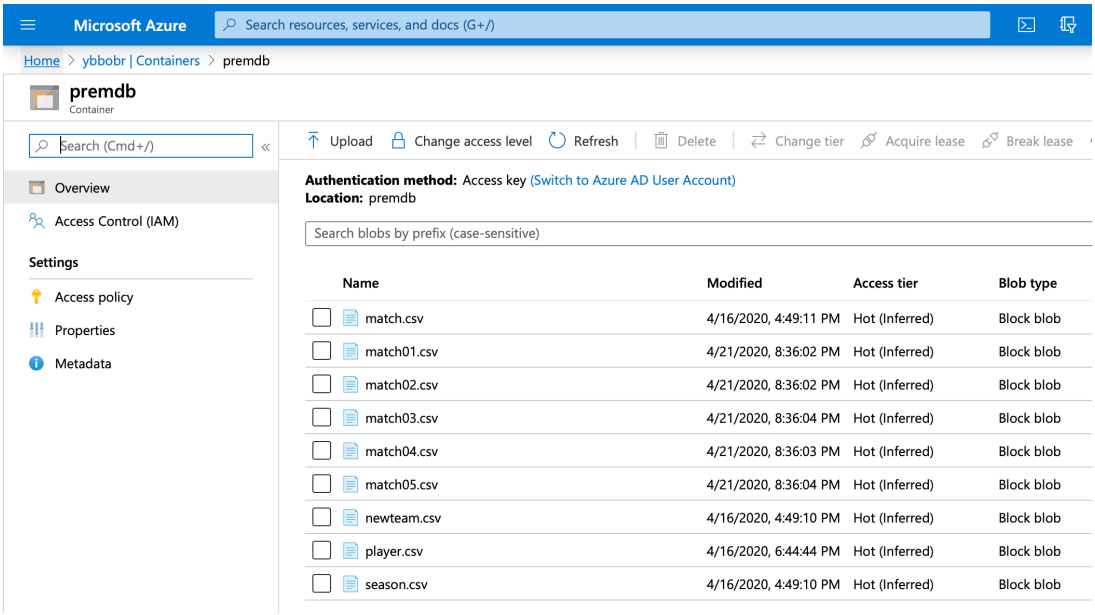
Platforms: All platforms

Parent topic: Loading Data from Object Storage

In this section:

This section explains how to use the `ybload` client to load tables from an Azure Blob object store.

These instructions assume that you already have access to an Azure Blob storage account that contains the data to be loaded. For example:



If you do not have access to a storage account, follow the instructions in the Microsoft documentation before proceeding with the information in this section.

To bulk load a table from data in an Azure Blob container:

1. Use one of the documented Azure Authentication Methods.

For example, to authenticate via the `ybload` command-line options, you will need:

- A storage account name (your "identity"). Specifying the Azure endpoint (`https://<storage account>.blob.core.windows.net`) in addition to the account name is optional.
- An access key or a generated SAS token (your "credential") You can only access one object store per `ybload` operation. For example, you cannot specify connection details for both an Azure client and an S3 client in the same command, and you cannot load data from multiple Azure endpoints or storage accounts.

2. Identify the location in Azure of the data you want to load (`container/blob`).

At the end of the `ybload` command line, you will specify the source data in a URI, which is an abbreviated Azure path. For example:

```
azure://premdb/match.csv
```

You can load multiple files by specifying a prefix for the blob instead of a complete file name. For example, the following command will find all files in the `premdb` container that begin with `match0`:

```
azure://premdb/match0
```

You can load data from multiple containers within the same `ybload` command if they belong to the same storage account.

3. Run the `ybload` command in the usual way. Include the following details in the command line:

- Standard `ybload` options such as `--format` and `--delimiter`
- A local path on the client system for the bad row file. (If you specify a log file, it must also be on the local file system.)
- Azure identity (storage account name) and credential (access key or SAS token)

  These options are not needed if you are using the Azure CLI (`az login`) means of authentication. However, you will need to specify the `--object-store-endpoint` option.

- Location(s) of the source data (specified last on the command line, without an option name) For example, the following command would load a single blob (`match.csv`) from the `premdb` container:

```
$ ybload -d premdb --username bobr -W -t match --format csv --delimiter ',
' --bad-row-file '/home/brumsby/newazurebad'
--object-store-credential '************************************'
--object-store-identity 'ybbobr'
azure://premdb/match.csv
```

Alternatively, you can save the identity and credential values to a Java properties file and name the file in the `ybload` command. See Object Storage Options and Azure Blob Storage URIs.

Azure Authentication Methods

Platforms: All platforms

Parent topic: Loading from Azure Blob Storage

There are several different ways to authenticate when you are using `ybload` to load data from Azure Blob storage. You can authenticate implicitly by using supported Azure-identity methods, such as service principals, or you can authenticate explicitly by using `ybload` command-line options. You can use the Azure CLI (`az login`) command or the Azure web portal to create and manage user accounts, service principals, and so on.

Your organization's Azure administrator should provide instructions for the mechanism you should use. For more details about creating and managing Azure accounts, see the Microsoft Azure documentation.

Order of Precedence for Authentication Methods

You may have multiple credential settings available to you, based on your Azure account setup and how you script the `ybload` command. Therefore it is important to know which authentication mechanism takes precedence when the `ybload` command is run. Note that credentials set with `ybload` object storage options, if set (on the command line or in a properties file), always take precedence over implicit Azure account credentials.

The order of precedence is as follows:

1. Access key or SAS token specified with the `ybload --object-store-credential` command-line option
2. Access key or SAS token specified *with* the `yb.file.` prefix in one of the following:
3. URI parameter
4. Properties file named with the `--object-store-provider-config` command-line option
5. Access key or SAS token specified *without* the `yb.file.` prefix in one of the following:
6. URI parameter
7. Properties file named with the `--object-store-provider-config` command-line option
8. Implicit authentication, using one of the following Azure-identity methods:

- Service principal environment variables
- Managed identity credential (for use on Azure VMs)
- Shared Token Cache Credential (provided by authentication to Azure with Visual Studio 2019)

  Note: This mechanism used to be disabled by default for `ybload` before version `6.7` . For version `6.7` or above, to disable this type of credential, unset any `AZURE_USERNAME` environment variable. For version `6.6` or below, to enable this type of credential, specify the name of a Java properties file with the `--object-store-provider-config` option. In the file, include the following entry:

  ```
  allowSharedTokenCacheCredential = true
  ```

- Azure CLI credentials (authentication via `az login`)

Azure Role Assignments

An Azure account that you intend to use for authentication when loading data from an Azure blob must have the following role assignments:

- Reader
- Storage Blob Data Reader

These are the minimum role assignments required by `ybload` operations. Your Azure user accounts may be configured with assignments that exceed these requirements.

Azure Environment Variables

`ybload` operations from Azure Blob storage support the following environment variables. You can set these variables and reference them when you are using a service principal account for authentication:

- `AZURE_CLIENT_ID`: the application ID for a service principal
- `AZURE_CLIENT_SECRET`: the secret key for a service principal
- `AZURE_TENANT_ID`: the Azure Active Directory (AD) tenant for a service principal

Azure Blob Storage Examples

Platforms: All platforms

Parent topic: Loading from Azure Blob Storage

The following `ybload` examples show how to load Yellowbrick tables from Azure Blob storage.

Load a table from a single CSV file in an Azure container:

Here is an example of a load from Azure Blob storage. The Azure storage account is `ybbobr` and the source file `match.csv` was uploaded to a container called `premdb`.

The `--object-store-identity` and `--object-store-credential` options must be specified, but the `--object-store-endpoint` option is optional.

```
$ ybload -d premdb --username bobr -W -t match --format csv --delimiter ',' --bad-row-file '/home/brumsby/newazurebad'
--object-store-identity 'ybbobr'
--object-store-credential '************************************'
--object-store-endpoint "https://ybbobr.blob.core.windows.net/"
azure://premdb/match.csv
Password for user bobr:
18:49:56.744 [ INFO] ABOUT CLIENT:
   app.cli_args         = "-d" "premdb" "--username" "<USERNAME>" "-W" "-t" "match" "--format" "csv" "--delimiter" "," "--bad-row-
   app.name_and_version = "ybload version 4.0.1-22337"
   java.version         = "1.8.0_101"
   jvm.memory           = "981.50 MB (max=14.22 GB)"
   jvm.name_and_version = "Java HotSpot(TM) 64-Bit Server VM (build 25.101-b13, mixed mode)"
   jvm.options          = "-Xmx16g, -Xms1g, -Dapp.name=ybload, -Dapp.pid=23460, -Dapp.repo=/opt/ybtools/lib, -Dapp.home=/opt/ybtoo
   jvm.vendor           = "Oracle Corporation"
   os.name_and_version  = "Linux 4.4.0-31-generic (amd64)"

18:49:56.821 [ INFO] Gathering metadata on input files
18:49:56.831 [ INFO] Loaded file handler for azure://
18:49:56.863 [ INFO]
Configuration (Azure client):
   endpoint           : https://ybbobr.blob.core.windows.net/
   identity/credential: ********
18:49:56.918 [ INFO] Assuming source encoding matches database server encoding: LATIN9
18:49:58.878 [ INFO] Starting source azure://premdb/match.csv
18:49:59.026 [ INFO] Using database locale: C
18:49:59.032 [ INFO] Auto-detected line separator = '\n'
18:49:59.154 [ INFO]
Configuration (pipeline):
   numSources         : 1
   numReaders         : 4
   numParsersPerReader: 2
Configuration (record/field separation):
   --format                    : CSV
   --delimiter                 : ,
   --linesep                   : \n
   --quote-char                : "
   --escape-char               : "
   --no-trim-white
   --skip-blank-lines
   --on-missing-field          : ERROR
   --on-extra-field            : ERROR
   --on-unescaped-embedded-quote: ERROR
Configuration (pre-parsing):
```

```
   --on-zero-char           : ERROR
   --on-string-too-long     : ERROR
   --on-invalid-char        : REMOVE
   --no-convert-ascii-control
   --no-convert-c-escape
Configuration (session):
   tableName       : "premdb"."public"."match"
   keepAliveSeconds: 60
   maxBadRows      : Unlimited
   sessionKey      : DKkIwVzD4vaaAyCKu43oNuucwSpn-avzIRYo9xfdY6hmWUNdCIkiZ0awIOzj5S__
Configuration (transaction):
   transactionType    : BySize
   rowsPerTransaction : Unlimited
   bytesPerTransaction: 1.0TB
18:49:59.618 [ INFO] Bad rows will be written to /home/brumsby/newazurebad
18:50:00.138 [ INFO] Opening transaction #1 for match ...
18:50:00.223 [ INFO] Opened transaction #1 for match
18:50:00.231 [ INFO] Flushing last 8606 rows (of 8606 total) in transaction #1 for match
READ:305.8KB(88.83KB/s). ROWS G/B: 8606/0( 2.44K/s). WRITE:285.7KB(83.01KB/s).  TIME E/R:   0:00:03/ --:--:--18:50:00.256 [ INFO]
18:50:01.446 [ INFO] Committed transaction #1 after a total of 292604 bytes and 8606 good rows for match
18:50:01.464 [ INFO] READ:305.8KB(66.00KB/s). ROWS G/B: 8606/0( 1.81K/s). WRITE:285.7KB(61.67KB/s).  TIME E/R:   0:00:04/ --:--:--
18:50:01.465 [ INFO] SUCCESSFUL BULK LOAD: Loaded 8606 good rows in   0:00:04 (READ: 66.00KB/s WRITE: 61.67KB/s)
```

Load a table from multiple CSV files in an Azure container:

In this example, the URI for the load is `azure://premdb/match0`. Five files are found in the `premdb` container with the prefix `match0`:

```
$ ybload -d premdb --username bobr -W -t match --format csv --delimiter ',' --bad-row-file '/home/brumsby/newazurebad'
--object-store-credential '************************************'
--object-store-identity 'ybbobr' azure://premdb/match0
Password for user bobr:
20:40:03.409 [ INFO] ABOUT CLIENT:
...
20:40:03.479 [ INFO] Gathering metadata on input files
20:40:03.489 [ INFO] Loaded file handler for azure://
20:40:03.522 [ INFO]
Configuration (Azure client):
   endpoint           : null
   identity/credential: ********
20:40:03.572 [ INFO] Assuming source encoding matches database server encoding: LATIN9
20:40:05.474 [ INFO] Expanded azure://premdb/match0 into 5 sources
20:40:05.486 [ INFO] Choosing to read sources concurrently because most sources are likely to be slow
20:40:05.829 [ INFO] Starting source azure://premdb/match04.csv
20:40:05.946 [ INFO] Starting source azure://premdb/match03.csv
20:40:05.947 [ INFO] Starting source azure://premdb/match05.csv
20:40:05.949 [ INFO] Starting source azure://premdb/match01.csv
20:40:05.950 [ INFO] Starting source azure://premdb/match02.csv
20:40:05.999 [ INFO] Using database locale: C
20:40:06.004 [ INFO] Auto-detected line separator = '\n'
20:40:06.153 [ INFO]
Configuration (pipeline):
   numSources             : 5
   readSourcesConcurrently: 5
   numReaders             : 4
   numParsersPerReader    : 2
Configuration (record/field separation):
...
```

The same load could be run by specifying the five sources explicitly at the end of the command:

```
azure://premdb/match01.csv
azure://premdb/match02.csv
```

```
azure://premdb/match03.csv
azure://premdb/match04.csv
azure://premdb/match05.csv
```

Load a table from files in separate Azure containers:

The following command loads a table from two files, one in the `premdb` container and one in the `premdbnew` container:

```
$ ybload -d premdb --username bobr -W -t match --format csv --delimiter ',' --bad-row-file '/home/brumsby/newazurebad'
--object-store-credential '************************************'
--object-store-identity 'ybbobr'
azure://premdb/match05.csv
azure://premdbnew/match01.csv
```

Use a properties file to load a table from an Azure container:

First create a properties file on the client system. For example:

```
$ more ybbobr.properties

yb.file.endpoint = https://ybbobr.blob.core.windows.net
yb.file.identity = ybbobr
yb.file.credential = ************************************
```

Now run the `ybload` command and name the `ybbobr.properties` file in the `--object-store-provider-config` option:

```
$ ybload -d premdb --username bobr -W -t match --format csv --delimiter ',' --bad-row-file '/home/brumsby/newazurebad'
--object-store-provider-config ybbobr.properties
azure://premdb/match05.csv
Password for user bobr:
21:51:15.868 [ INFO] ABOUT CLIENT:
   app.cli_args         = "-d" "premdb" "--username" "<USERNAME>" "-W" "-t" "match" "--format" "csv" "--delimiter" "," "--bad-row-
   app.name_and_version = "ybload version 4.1.0-22656"
   java.version         = "1.8.0_101"
   jvm.memory           = "981.50 MB (max=14.22 GB)"
   jvm.name_and_version = "Java HotSpot(TM) 64-Bit Server VM (build 25.101-b13, mixed mode)"
   jvm.options          = "-Xmx16g, -Xms1g, -Dapp.name=ybload, -Dapp.pid=6255, -Dapp.repo=/opt/ybtools/lib, -Dapp.home=/opt/ybtool
   jvm.vendor           = "Oracle Corporation"
   os.name_and_version  = "Linux 4.4.0-31-generic (amd64)"

21:51:15.950 [ INFO] Gathering metadata on input files
21:51:15.971 [ INFO] Loaded file handler for azure://
21:51:16.009 [ INFO]
Configuration (Azure client):
   endpoint           : https://ybbobr.blob.core.windows.net
   identity/credential: ********
...
```

Load a compressed file from an Azure container:

This example simply shows that `ybload` can load compressed `.gz` files from an object store. No compression options need to be specified. The file is recognized by its extension.

```
$ ybload -d premdb --username bobr -W -t match --format csv --delimiter ',' --bad-row-file '/home/brumsby/newazurebad'
--object-store-provider-config ybbobr.properties
azure://premdbnew/match01.gz
...
```

Use az login with an authentication code:

The `az login` command in this example prompts for authentication with a code (OIDC token) on the Microsoft web portal. After this is done, the `ybload` command can be run *without* the `--object-store-identity` and `--object-store-credential` options.

```
$ az login
To sign in, use a web browser to open the page https://microsoft.com/devicelogin and enter the code FJCDS23HP to authenticate.
[
  {
   "cloudName": "AzureCloud",
   "homeTenantId": "************************************",
   "id": "************************************",
   "isDefault": true,
   "managedByTenants": [],
   "name": "JBond Engineering",
   "state": "Enabled",
   "tenantId": "************************************",
   "user": {
     "name": "james.bond@agent007.com",
     "type": "user"
   }
  }
]
$ ybload -d premdb --username jbond -W -t match --format csv --delimiter ',' --bad-row-file '/home/jbond/newazurebad'
--object-store-endpoint "https://ybjbond.blob.core.windows.net/"
azure://premdb/match.csv
Password for user jbond:
...
```

Use the Azure CLI to create a service principal for authentication:

This example shows how to:

- Create a service principal with the `reader` role (this can be done via the CLI or the web portal)
- Set the following Azure environment variables:
- `AZURE_CLIENT_ID`
- `AZURE_CLIENT_SECRET`
- `AZURE_TENANT_ID`
- Assign the Storage Blob Data Reader role to `AZURE_CLIENT_ID` (again this can be done via the CLI or the web portal)
- Use the service principal (and associated environment variables) to authenticate with the `az login` command

```
$ az ad sp create-for-rbac -n “https://ybbobr” --role reader --scope /subscriptions/b0093a34-8f38-4752-8f4b-5b9e924ccc16/resourceG
$ export AZURE_CLIENT_ID=“************************************"
$ export AZURE_CLIENT_SECRET=“************************************”
$ export AZURE_TENANT_ID=“************************************”
$ az role assignment create --role “Storage Blob Data Reader” --assignee $AZURE_CLIENT_ID --scope /subscriptions/b0093a34-8f38-475
$ az login --service-principal --username $AZURE_CLIENT_ID --password $AZURE_CLIENT_SECRET --tenant $AZURE_TENANT_ID
...
```

At this point, you can run a `ybload` command using implicit authentication via the established service principal identity and credential.

Azure Blob Storage URIs

Platforms: All platforms

Parent topic: Loading from Azure Blob Storage

The `ybload` client supports several patterns for Azure Blob storage URIs. All of these patterns identify the location of the source data for a bulk load operation. An Azure container (like an S3 bucket) contains one or more "blobs" or "keys." From the point of view of a `ybload` operation, a blob is effectively a source file. Blobs are stored in "containers."

URI Pattern	Loads	Example
`azure://CONTAINER`	All blobs in the container	`azure://premdb`
`azure://CONTAINER/BLOB`	Current version of the named blob	`azure://premdb/team01.csv`
`azure://CONTAINER/BLOB?snapshot=SNAPSHOT`	Specific snapshot (version) of the named blob	`azure://premdb/team01.csv?snapshot=2020-04-21T01:59:24.7012660Z`
`azure://CONTAINER/BLOB_PREFIX`	All blobs that match the prefix	`azure://premdb/team`

A valid URI must not contain a space character unless it is encoded with the `%20` escape sequence. For example, if you upload a `season.csv` source file into a folder named `premdb/new files`, the following URI will work:

```
azure://premdb/new%20files/season.csv
```

You can specify multiple URIs on the `ybload` command line if you want to identify multiple source files explicitly. However, the sources must belong to the same storage account.

URI Parameters

As an alternative to specifying `ybload --object-store` options, you can specify the Azure configuration in the form of URI parameters. To use a URI parameter, drop the `--object-store-` prefix from the `ybload` option name. For example, the `--object-store-endpoint` option is specified in the following URI as `endpoint=`:

```
azure://my_container/my_blob?endpoint=https://mystorageaccount.blob.core.windows.net
```

Note: If you provide multiple source URIs in a single `ybload` command, only the first URI will be checked for configuration options.

Loading from S3-Compatible Object Storage

In general, follow the instructions in Loading from Amazon S3, using the S3 Object Storage URIs and S3 Authentication Methods. Before attempting to run `ybload`, you must already have an account with the S3-compatible object storage provider, know the S3 location of the files you want to load, and have your S3 credentials ready at hand. Your storage provider may have other requirements; please refer to the provider's documentation for details.

The `ybload` client supports S3 Java SDK v2, which requires that a region be supplied and configured even if your S3-compatible object store does not in itself require or support regions. If a region is not provided, connections to the object store will fail. To work around this requirement, you can specify a "dummy" region.

For example, if you are working exclusively with S3-compatible object stores, you can use an AWS configuration file to provide the dummy value:

```
~/.aws/config
[default]
region = dummy
```

Note that the `yb.file.` prefix is not supported in this context.

You can only access one object store per `ybload` operation. For example, you cannot load data from multiple S3 endpoints or storage accounts in a single command.

Loading Tables with Spark

Platforms: All platforms

Parent topic: Load Tables

In this section:

This section describes how to bulk load data from source files that `ybload` cannot read directly, such as Avro and ORC files. Regardless of its source location (HDFS, NFS, S3, and so on), you can load data in different formats by running Apache Spark jobs that use the Yellowbrick `ybrelay` service to call `ybload`. Exported data flows from the Spark application platform to a `ybrelay` server running the `ybrelay` service, then is loaded into Yellowbrick database tables using `ybload` operations.

Follow these steps to bulk load Yellowbrick tables via Spark and `ybrelay`. Subsequent sections explain these steps in detail and provide examples.

1. Install and set up Apache Spark and the `ybrelay` service.
2. Define the parameters for a `spark-submit` command:

- Native Spark options
- Spark application options:
  - Yellowbrick database connectivity
  - `ybrelay` connectivity
  - General options
  - `ybload` options, if needed

3. Run the `spark-submit` command.
4. Monitor the resulting `ybload` operation.

Setting up and Running a Spark Job

Platforms: All platforms

Parent topic: Loading Tables with Spark

In this section:

This section describes the setup required to run a Spark job that loads a Yellowbrick table.

Spark Application for Yellowbrick

When you install `ybtools`, the installation includes the following directory:

```
/opt/ybtools/integrations/spark
```

This `spark` directory contains the application `.jar` file that you use to run Spark jobs via `ybrelay`.

For installation and operational information, see the Apache Spark documentation.

About Spark Jobs

Spark jobs can be submitted in various ways (standalone, cluster mode, using YARN as a resource manager, and so on). For details, see the Spark documentation.

In general, a Spark job consists of:

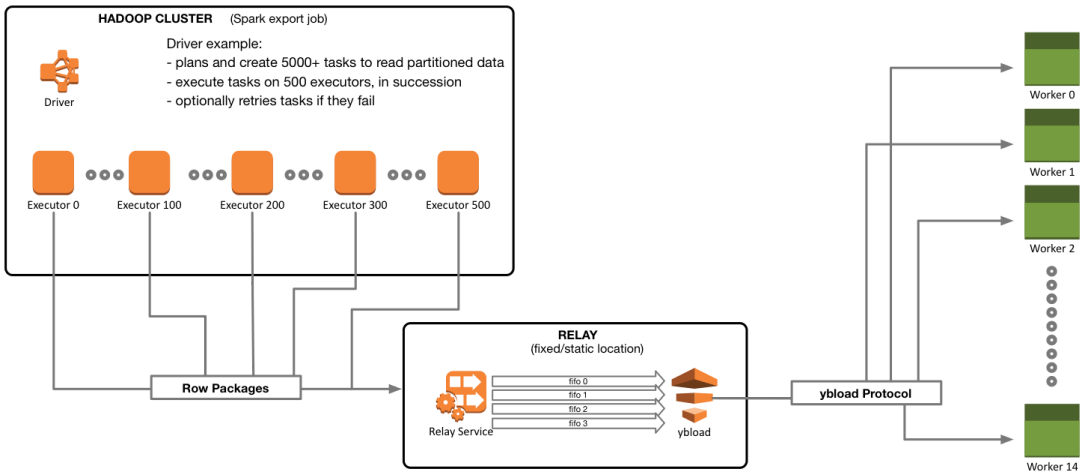
```
N + 1 compute (N executors, 1 driver)
```

The driver executes first:

- *Driver* can run inline with the CLI submission or on-grid.
- *Driver* plans and executes tasks on *executors*.

Executors run tasks, and each task is given a slice of the data. Executors only need 2 cores and 1GB of memory.

A Spark job for Yellowbrick submits an application that uses `ybrelay` to stream high volumes of data into the table in parallel. Any file format that the Spark application can read is supported; `ybrelay` runs as a service that accepts row packages from Spark, then sends the data in parallel streams to `ybload` for fast bulk loading, as shown in the following diagram.



Summary of the spark-submit Command

A Spark job runs with a set of native Spark options and Spark application options. The source directory for the data must be specified, along with its format. Connectivity to both the Yellowbrick system and the `ybrelay` client must also be specified. Many other optional settings may be provided, including column information for the table, pre- or post-processing SQL to run, and options that are passed to `ybload`.

Note that `ybrelay` runs with its own set of options; these are not set in the Spark job. See ybrelay Options.

The `spark-submit` or `spark2-submit` command consists of multiple sets of parameters:

- Native Spark options, which vary by use case:

```
./bin/spark-submit \
  --class <main-class> \
  --master <master-url> \
  --deploy-mode <deploy-mode> \
  ...
```

The `--class option` must be set to:

```
--class io.yellowbrick.spark.cli.SparkExport \
```

- The Yellowbrick Spark application jar:

```
/opt/ybtools/*version*/integrations/spark/relay-integration-spark*shaded*.jar \
```

- Yellowbrick Spark application options that define the source directory and format of the data, connectivity to the Yellowbrick database, connectivity to the `ybrelay` service, and other optional settings:

```
--export-directory hdfs:///user/warehouse/hive/${schema}.${table} \
--format parquet \
--yb-host $YT_HOST \
--yb-port $YT_PORT \
--yb-user $YT_DEFAULT_USER \
--yb-password $YT_DEFAULT_PASSWORD \
--yb-database $YT_DEFAULT_DATABASE \
```

```
--yb-table "${schema}.${table}" \
--yb-relay-host $YT_RELAY_HOST \
--yb-relay-port 21212 \
--log-level DEBUG \
...
```

- `ybload` options that can be passed through the `spark-submit` command if needed:

```
--load-option "--max-bad-rows 0" \
--load-option "--on-missing-field SUPPLYNULL" \
--load-option "--on-extra-field REMOVE"
...
```

Setting SPARK_MAJOR_VERSION

If multiple versions of Spark are installed in the environment where the Spark job will run, you need to set the Spark version that you want to use for your job. You can do this by setting the `SPARK_MAJOR_VERSION` environment variable. For example, if Spark 1.6.2 and Spark 2.0 are both installed on a node, and you want to run your job with Spark 2.0 (`spark2-submit`), set `SPARK_MAJOR_VERSION` to `2` :

```
export SPARK_MAJOR_VERSION=2
```

Error Tolerance

By default, Spark does not tolerate any errors for distributed Spark executors (tasks). Therefore, any error encountered by an executor will cause the entire Spark job to fail, rolling back the data that was exported to the point of failure.

You can use the `--task-failures` option to modify this behavior; however, failed and restarted tasks are likely to cause data duplication during the load process. You can manage data duplication for failed tasks in the following ways.

If an application key is present in each row of data (such as a `primary_key_id`), you can deduplicate the data by running a simple `DELETE` command:

```
delete
from @{config.ybTableName} t
where t.load_transaction_id = '@{transactionId}'
and rowid not in (
select max(rowid)
from @{config.ybTableName}
where load_transaction_id = '@{transactionId}'
group by primary_key_id
);
```

If a primary key is not available, you can use another technique, which requires three steps:

1. Add 3 new columns to the target table: `partition_id`, `attempt_number`, and `load_transaction_id`. Define these three columns as `int`, `int`, and `varchar(64)`:

```
ALTER TABLE ${table} ADD COLUMN partition_id INT;
ALTER TABLE ${table} ADD COLUMN attempt_number INT;
ALTER TABLE ${table} ADD COLUMN load_transaction_id VARCHAR(64);
```

2. Provide a `computed-columns.properties` file and reference it using the `--computed-columns` argument. This file should contain entries like these:

```
partition_id=taskContext.partitionId()
attempt_number=taskContext.attemptNumber()
load_transaction_id=transactionId
```

Each row of data will be populated with the load transaction ID, the partition ID, and the attempt number. If an executor/task fails and is restarted by the Spark container, the task will run again with a higher attempt number.

3. Run a `DELETE` statement to remove duplicates. For example:

```
delete
from @{config.ybTableName} t
using (
select partition_id, max(attempt_number) as final_attempt
from @{config.ybTableName}
where load_transaction_id = '@{transactionId}'
group by 1
) final_attempts
```

```
where t.load_transaction_id = '@{transactionId}'
and t.partition_id = final_attempts.partition_id
and t.attempt_number != final_attempts.final_attempt;
```

Note: You can specify this exact SQL statement as input to the `--post-sql` option.

Examples of Spark Jobs

Platforms: All platforms

Parent topic: Setting up and Running a Spark Job

These example job submissions are for a single standalone spark cluster, reading from a local file system using the `file://` URI prefix. Other well-known protocols (such as S3, NFS, HDFS) can be used.

This example submits a job that exports data from a CSV file with a header:

```
/opt/spark/spark-2.4.3-bin-hadoop2.7/bin/spark-submit \
--class io.yellowbrick.spark.cli.SparkExport \
--master local \
--executor-memory 2G \
--num-executors 1 \
--executor-cores 1 \
/opt/ybtools/integrations/spark/relay-integration-spark-*-shaded.jar \
--format csv \
--export-directory file:///opt/s3/data/data/People.csv \
--read-options /opt/spark/spark-2.4.3-bin-hadoop2.7/read-options-csv.properties \
--logs /opt/s3/data/logs \
--yb-relay-host localhost \
--yb-relay-port 21212 \
--yb-host mydemo.yellowbrick.io \
--yb-port 5432 \
--yb-user username \
--yb-password password \
--yb-database ybspark_demo \
--yb-table people2
```

For spark to read the header line successfully, the referenced `read-options-csv.properties` file must contain:

```
header=true
```

The following example exports data from a JSON file:

```
/opt/spark/spark-2.4.3-bin-hadoop2.7/bin/spark-submit \
--class io.yellowbrick.spark.cli.SparkExport \
--master local \
--executor-memory 2G \
--num-executors 1 \
--executor-cores 1 \
/opt/ybtools/integrations/spark/relay-integration-spark-*-shaded.jar \
--format json \
--export-directory file:///opt/s3/data/data/people_json.json \
--read-options /opt/spark/spark-2.4.3-bin-hadoop2.7/read-options-json.properties \
--logs /opt/s3/data/logs \
--yb-relay-host localhost \
--yb-relay-port 21212 \
--yb-host mydemo.yellowbrick.io \
--yb-port 5432 \
--yb-user username \
--yb-password password \
```

```
--yb-database ybspark_demo \
--yb-table people2
```

Depending on how well-formed the JSON document is, some manipulation might be needed. You might need to create a `read-options-json.properties` file that contains:

```
multiLine=true
```

The incoming JSON data in this case looks like this:

```
[{
  "playerID" : "aardsda01",
  "birthYear" : "1981",
  ...
  "finalGame" : "2015-08-23"
}, {
  "playerID" : "aaronha01",
  "birthYear" : "1934",
  ...
  "finalGame" : "1976-10-03"
}]
```

Output for a Spark Job

This section presents an example of the output that is logged when you run a Spark job that uses `ybrelay` and `ybload` to load data into a Yellowbrick table.

```
2019-07-31 11:33:56.394 [ INFO] <main>  ABOUT CLIENT:
   app.cli_args =
"-h" "eco1-ha.yellowbrick.io"
"-U" "agoade"
"-d" "agoade_demo"
"--read-sources-concurrently" "ALWAYS"
"--delimiter" "0x7C"
"--quote-char" "0x22"
"--escape-char" "0x22"
"--linesep" "LF"
"--num-readers" "4"
"--bad-row-file" "/var/log/ybrelay/logs/20190731113355-local-1564598030512.bad"
"--logfile" "/var/log/ybrelay/logs/20190731113355-local-1564598030512.log"
"--remote-control-logfile" "/var/log/ybrelay/logs/20190731113355-local-1564598030512.control.log"
"-t" "people2"
"--field-names" "playerid,birthyear,birthmonth,birthday,birthcountry,birthstate,birthcity,deathyear,deathmonth,deathday,deathcount
"--no-trim-white" "--log-level" "WARN"
"--logfile-log-level" "DEBUG"
"/var/lib/ybrelay/work/local-1564598030512.fifos/task.0"
"/var/lib/ybrelay/work/local-1564598030512.fifos/task.1"
"/var/lib/ybrelay/work/local-1564598030512.fifos/task.2"
"/var/lib/ybrelay/work/local-1564598030512.fifos/task.3"
   app.name_and_version = "Yellowbrick Data Relay version 3.0.0-13610"
   java.home            = "/usr/lib/jvm/java11/jre"
   java.version         = "11"
   jvm.memory           = "981.50 MB (max=14.22 GB)"
   jvm.name_and_version = "OpenJDK 64-Bit Server VM (build 25.212-b04, mixed mode)"
   jvm.options          = "-Xmx16g, -Xms1g"
   jvm.vendor           = "Oracle Corporation"
   os.name_and_version  = "Linux 3.10.0-957.10.1.el7.x86_64 (amd64)"

2019-07-31 11:33:56.395 [ INFO] <main>  Logfile written to /var/log/ybrelay/logs/20190731113355-local-1564598030512.log
2019-07-31 11:33:56.437 [DEBUG] <pool-1-thread-1>  BgTask "ybload db init" started
2019-07-31 11:33:56.459 [DEBUG] <pool-1-thread-2>  BgTask "LFSourceContext" started
2019-07-31 11:33:56.460 [DEBUG] <pool-1-thread-3>  BgTask "LFConcurrencyContext" started
2019-07-31 11:33:56.460 [ INFO] <LFSourceContext>  Gathering metadata on input files
2019-07-31 11:33:56.460 [DEBUG] <pool-1-thread-4>  BgTask "Choosing apx source encoding" started
2019-07-31 11:33:56.461 [DEBUG] <pool-1-thread-5>  BgTask "Determining actual source encoding" started
2019-07-31 11:33:56.468 [DEBUG] <pool-1-thread-6>  BgTask "TranscodeContext" started
2019-07-31 11:33:56.469 [DEBUG] <pool-1-thread-7>  BgTask "Encoding rSep" started
2019-07-31 11:33:56.469 [DEBUG] <pool-1-thread-8>  BgTask "Encoding fSep" started
2019-07-31 11:33:56.470 [DEBUG] <pool-1-thread-9>  BgTask "LFScanContext" started
2019-07-31 11:33:56.474 [DEBUG] <main>  Wait for source information
2019-07-31 11:33:56.474 [DEBUG] <pool-1-thread-10>  BgTask "BulkSession init" started
2019-07-31 11:33:56.484 [DEBUG] <LFSourceContext>  BgTask "LFSourceContext" exited
2019-07-31 11:33:56.484 [DEBUG] <main>  Received source information
2019-07-31 11:33:56.485 [DEBUG] <LFConcurrencyContext>  BgTask "LFConcurrencyContext" exited
2019-07-31 11:33:56.488 [DEBUG] <main>  Setting noMoreData on sourceInfoQueue
2019-07-31 11:33:56.488 [DEBUG] <main>  Wait for concurrency information
2019-07-31 11:33:56.488 [DEBUG] <main>  Creating 13 segment permits
2019-07-31 11:33:56.490 [DEBUG] <main>  Pre-allocating 13 source buffers (total RAM 650.0MB)
2019-07-31 11:33:56.726 [DEBUG] <ybload db init>  getTableSchema(input): "agoade_demo"."public"."people2"
```

```
2019-07-31 11:33:56.726 [ INFO] <Choosing apx source encoding>  Assuming source encoding matches database server encoding: LATIN9
2019-07-31 11:33:56.726 [DEBUG] <Choosing apx source encoding>  BgTask "Choosing apx source encoding" exited
2019-07-31 11:33:56.748 [DEBUG] <ybload db init>  getTableSchema(normalized): "agoade_demo"."public"."people2"
2019-07-31 11:33:56.755 [DEBUG] <ybload db init>  BgTask "ybload db init" exited
2019-07-31 11:33:56.919 [ INFO] <main>  Starting 4 source PreReaders
2019-07-31 11:33:56.922 [DEBUG] <BgTask idle thread>  BgTask "PreReader #0" started
2019-07-31 11:33:56.923 [DEBUG] <BgTask idle thread>  BgTask "Sequencer #0" started
2019-07-31 11:33:56.923 [DEBUG] <BgTask idle thread>  BgTask "PreReader #1" started
2019-07-31 11:33:56.923 [DEBUG] <BgTask idle thread>  BgTask "Sequencer #1" started
2019-07-31 11:33:56.923 [DEBUG] <pool-1-thread-11>  BgTask "PreReader #2" started
2019-07-31 11:33:56.924 [DEBUG] <pool-1-thread-12>  BgTask "Sequencer #2" started
2019-07-31 11:33:56.924 [DEBUG] <pool-1-thread-13>  BgTask "PreReader #3" started
2019-07-31 11:33:56.924 [DEBUG] <pool-1-thread-14>  BgTask "Sequencer #3" started
2019-07-31 11:33:56.925 [DEBUG] <pool-1-thread-15>  BgTask "Wait for PreReaders" started
2019-07-31 11:33:56.926 [DEBUG] <pool-1-thread-16>  BgTask "Wait for Sequencers" started
2019-07-31 11:33:58.753 [DEBUG] <PreReader #2>  Starting pre-reading of /var/lib/ybrelay/work/local-1564598030512.fifos/task.3
2019-07-31 11:33:58.754 [DEBUG] <PreReader #1>  Starting pre-reading of /var/lib/ybrelay/work/local-1564598030512.fifos/task.0
2019-07-31 11:33:58.754 [DEBUG] <PreReader #0>  Starting pre-reading of /var/lib/ybrelay/work/local-1564598030512.fifos/task.1
2019-07-31 11:33:58.754 [DEBUG] <PreReader #3>  Starting pre-reading of /var/lib/ybrelay/work/local-1564598030512.fifos/task.2
2019-07-31 11:34:00.591 [DEBUG] <Determining actual source encoding>  BgTask "Determining actual source encoding" exited
2019-07-31 11:34:00.591 [DEBUG] <TranscodeContext>  BgTask "TranscodeContext" exited
2019-07-31 11:34:00.593 [ INFO] <main>  Using database locale: C
2019-07-31 11:34:00.594 [DEBUG] <Encoding fSep>  BgTask "Encoding fSep" exited
2019-07-31 11:34:00.594 [DEBUG] <Encoding rSep>  BgTask "Encoding rSep" exited
2019-07-31 11:34:00.604 [ INFO] <LFScanContext>  Configuration (record/field separation):
   --format                          : CSV
   Internal-mode                     : QUOTE
   --delimiter                       : |
   --linesep                         : \n
   --quote-char                      : "
   --escape-char                     : "
   --trim-white                      : false
   --skip-blank-lines                : true
   --on-missing-field                : ERROR
   --on-extra-field                  : ERROR
   --on-unescaped-embedded-quote     : ERROR
   Internal scanner                  : RecordScannerQuote_1_1
2019-07-31 11:34:00.604 [DEBUG] <LFScanContext>  BgTask "LFScanContext" exited
2019-07-31 11:34:00.694 [DEBUG] <main>  Destination Schema = TABLE: "agoade_demo"."public"."people2"
   "playerid" type=VARCHAR(256)
   "birthyear" type=BIGINT
   "birthmonth" type=BIGINT
   "birthday" type=BIGINT
   "birthcountry" type=VARCHAR(256)
   "birthstate" type=VARCHAR(256)
   "birthcity" type=VARCHAR(256)
   "deathyear" type=BIGINT
   "deathmonth" type=BIGINT
   "deathday" type=BIGINT
   "deathcountry" type=VARCHAR(256)
   "deathstate" type=VARCHAR(256)
   "deathcity" type=VARCHAR(256)
   "namefirst" type=VARCHAR(256)
   "namelast" type=VARCHAR(256)
   "namegiven" type=VARCHAR(256)
   "weight" type=BIGINT
   "height" type=BIGINT
   "bats" type=VARCHAR(256)
   "throws" type=VARCHAR(256)
   "debut" type=VARCHAR(256)
   "finalgame" type=VARCHAR(256)
   "retroid" type=VARCHAR(256)
   "bbrefid" type=VARCHAR(256)

2019-07-31 11:34:00.737 [ INFO] <main>  Configuration (pre-parsing):
   --on-zero-char                    : ERROR
   --on-string-too-long              : ERROR
   --on-invalid-char                 : REMOVE
```

```
   --no-convert-ascii-control
   --no-convert-c-escape
2019-07-31 11:34:00.749 [ INFO] <main>  Configuration (session):
   tableName       : "agoade_demo"."public"."people2"
   keepAliveSeconds: 60
   maxBadRows      : -1
2019-07-31 11:34:01.325 [ INFO] <BulkSession init>  Session Key = CWL72YSdnTskgHuL2qzs8urbok9kQmG5TXj3EiZL7kCtds54Bd-kmUHZp2zfpgtn
2019-07-31 11:34:01.331 [DEBUG] <BulkSession init>  Sending BulkService.init(#columns=24, columns="playerid", "birthyear", "birthm
2019-07-31 11:34:01.596 [DEBUG] <BulkSession init>  Received BulkService.init() response for people2
2019-07-31 11:34:01.597 [DEBUG] <BulkSession init>  BgTask "BulkSession init" exited
2019-07-31 11:34:01.604 [ INFO] <main>  Bad rows will be written to /var/log/ybrelay/logs/20190731113355-local-1564598030512.bad
2019-07-31 11:34:01.617 [DEBUG] <main>  Running with 1.2GB of memory. Expandable to 14.2GB
2019-07-31 11:34:01.619 [ INFO] <main>  Starting import of 4 files
2019-07-31 11:34:02.043 [ INFO] <main>  Starting 4 segment readers
2019-07-31 11:34:02.044 [DEBUG] <BgTask idle thread>  BgTask "segment reader #1" started
2019-07-31 11:34:02.044 [DEBUG] <BgTask idle thread>  BgTask "segment reader #0" started
2019-07-31 11:34:02.044 [DEBUG] <BgTask idle thread>  BgTask "segment reader #2" started
2019-07-31 11:34:02.044 [DEBUG] <BgTask idle thread>  BgTask "segment reader #3" started
2019-07-31 11:34:02.047 [ INFO] <main>  Opening transaction #1 for people2 ...
2019-07-31 11:34:02.943 [ INFO] <main>  Opened transaction #1 for people2
2019-07-31 11:34:02.952 [DEBUG] <BgTask idle thread>  BgTask "sender 1.00" started
2019-07-31 11:34:02.952 [DEBUG] <BgTask idle thread>  BgTask "sender 1.01" started
2019-07-31 11:34:02.953 [DEBUG] <pool-1-thread-17>  BgTask "sender 1.02" started
2019-07-31 11:34:02.953 [DEBUG] <pool-1-thread-18>  BgTask "sender 1.03" started
2019-07-31 11:34:02.954 [DEBUG] <pool-1-thread-19>  BgTask "sender 1.04" started
2019-07-31 11:34:02.954 [DEBUG] <sender 1.02>  Worker(/10.34.205.10:31272) -- Opening
2019-07-31 11:34:02.955 [DEBUG] <sender 1.03>  Worker(/10.34.205.10:31273) -- Opening
2019-07-31 11:34:02.955 [DEBUG] <sender 1.04>  Worker(/10.34.205.10:31274) -- Opening
2019-07-31 11:34:02.955 [DEBUG] <pool-1-thread-20>  BgTask "sender 1.05" started
2019-07-31 11:34:02.954 [DEBUG] <sender 1.01>  Worker(/10.34.205.10:31271) -- Opening
2019-07-31 11:34:02.955 [DEBUG] <sender 1.00>  Worker(/10.34.205.10:31270) -- Opening
2019-07-31 11:34:02.957 [DEBUG] <pool-1-thread-26>  BgTask "sender 1.11" started
2019-07-31 11:34:02.957 [DEBUG] <pool-1-thread-25>  BgTask "sender 1.10" started
2019-07-31 11:34:02.957 [DEBUG] <pool-1-thread-24>  BgTask "sender 1.09" started
2019-07-31 11:34:02.957 [DEBUG] <pool-1-thread-23>  BgTask "sender 1.08" started
2019-07-31 11:34:02.957 [DEBUG] <pool-1-thread-22>  BgTask "sender 1.07" started
2019-07-31 11:34:02.957 [DEBUG] <sender 1.05>  Worker(/10.34.205.10:31275) -- Opening
2019-07-31 11:34:02.955 [DEBUG] <pool-1-thread-21>  BgTask "sender 1.06" started
2019-07-31 11:34:02.962 [DEBUG] <sender 1.07>  Worker(/10.34.205.10:31277) -- Opening
2019-07-31 11:34:02.961 [DEBUG] <sender 1.08>  Worker(/10.34.205.10:31278) -- Opening
2019-07-31 11:34:02.960 [DEBUG] <sender 1.09>  Worker(/10.34.205.10:31279) -- Opening
2019-07-31 11:34:02.960 [DEBUG] <pool-1-thread-29>  BgTask "sender 1.14" started
2019-07-31 11:34:02.960 [DEBUG] <pool-1-thread-28>  BgTask "sender 1.13" started
2019-07-31 11:34:02.960 [DEBUG] <sender 1.10>  Worker(/10.34.205.10:31280) -- Opening
2019-07-31 11:34:02.959 [DEBUG] <sender 1.11>  Worker(/10.34.205.10:31281) -- Opening
2019-07-31 11:34:02.958 [DEBUG] <pool-1-thread-27>  BgTask "sender 1.12" started
2019-07-31 11:34:02.965 [DEBUG] <sender 1.13>  Worker(/10.34.205.10:31283) -- Opening
2019-07-31 11:34:02.963 [DEBUG] <sender 1.14>  Worker(/10.34.205.10:31284) -- Opening
2019-07-31 11:34:02.963 [DEBUG] <sender 1.06>  Worker(/10.34.205.10:31276) -- Opening
2019-07-31 11:34:02.965 [DEBUG] <sender 1.12>  Worker(/10.34.205.10:31282) -- Opening
2019-07-31 11:34:02.969 [DEBUG] <sender 1.04>  Worker(/10.34.205.10:31274) -- Opened from LocalSocket(/10.34.205.61:49906)
2019-07-31 11:34:02.969 [DEBUG] <sender 1.01>  Worker(/10.34.205.10:31271) -- Opened from LocalSocket(/10.34.205.61:36678)
2019-07-31 11:34:02.969 [DEBUG] <sender 1.09>  Worker(/10.34.205.10:31279) -- Opened from LocalSocket(/10.34.205.61:46766)
2019-07-31 11:34:02.969 [DEBUG] <sender 1.08>  Worker(/10.34.205.10:31278) -- Opened from LocalSocket(/10.34.205.61:46930)
2019-07-31 11:34:02.969 [DEBUG] <sender 1.10>  Worker(/10.34.205.10:31280) -- Opened from LocalSocket(/10.34.205.61:36994)
2019-07-31 11:34:02.969 [DEBUG] <sender 1.02>  Worker(/10.34.205.10:31272) -- Opened from LocalSocket(/10.34.205.61:49262)
2019-07-31 11:34:02.969 [DEBUG] <sender 1.03>  Worker(/10.34.205.10:31273) -- Opened from LocalSocket(/10.34.205.61:37606)
2019-07-31 11:34:02.969 [DEBUG] <sender 1.06>  Worker(/10.34.205.10:31276) -- Opened from LocalSocket(/10.34.205.61:59056)
2019-07-31 11:34:02.969 [DEBUG] <sender 1.14>  Worker(/10.34.205.10:31284) -- Opened from LocalSocket(/10.34.205.61:33288)
2019-07-31 11:34:02.969 [DEBUG] <sender 1.12>  Worker(/10.34.205.10:31282) -- Opened from LocalSocket(/10.34.205.61:37322)
2019-07-31 11:34:02.969 [DEBUG] <sender 1.05>  Worker(/10.34.205.10:31275) -- Opened from LocalSocket(/10.34.205.61:35468)
2019-07-31 11:34:02.969 [DEBUG] <sender 1.00>  Worker(/10.34.205.10:31270) -- Opened from LocalSocket(/10.34.205.61:44866)
2019-07-31 11:34:02.969 [DEBUG] <sender 1.11>  Worker(/10.34.205.10:31281) -- Opened from LocalSocket(/10.34.205.61:60542)
2019-07-31 11:34:02.969 [DEBUG] <sender 1.07>  Worker(/10.34.205.10:31277) -- Opened from LocalSocket(/10.34.205.61:58296)
2019-07-31 11:34:02.969 [DEBUG] <sender 1.13>  Worker(/10.34.205.10:31283) -- Opened from LocalSocket(/10.34.205.61:54852)
2019-07-31 11:34:02.972 [DEBUG] <sender 1.03>  Worker(/10.34.205.10:31273) -- Sent HELLO
2019-07-31 11:34:02.972 [DEBUG] <sender 1.09>  Worker(/10.34.205.10:31279) -- Sent HELLO
2019-07-31 11:34:02.972 [DEBUG] <sender 1.00>  Worker(/10.34.205.10:31270) -- Sent HELLO
```

```
2019-07-31 11:34:02.972 [DEBUG] <sender 1.12>  Worker(/10.34.205.10:31282) -- Sent HELLO
2019-07-31 11:34:02.972 [DEBUG] <sender 1.07>  Worker(/10.34.205.10:31277) -- Sent HELLO
2019-07-31 11:34:02.972 [DEBUG] <sender 1.06>  Worker(/10.34.205.10:31276) -- Sent HELLO
2019-07-31 11:34:02.972 [DEBUG] <sender 1.01>  Worker(/10.34.205.10:31271) -- Sent HELLO
2019-07-31 11:34:02.972 [DEBUG] <sender 1.11>  Worker(/10.34.205.10:31281) -- Sent HELLO
2019-07-31 11:34:02.972 [DEBUG] <sender 1.02>  Worker(/10.34.205.10:31272) -- Sent HELLO
2019-07-31 11:34:02.972 [DEBUG] <sender 1.04>  Worker(/10.34.205.10:31274) -- Sent HELLO
2019-07-31 11:34:02.972 [DEBUG] <sender 1.14>  Worker(/10.34.205.10:31284) -- Sent HELLO
2019-07-31 11:34:02.972 [DEBUG] <sender 1.13>  Worker(/10.34.205.10:31283) -- Sent HELLO
2019-07-31 11:34:02.972 [DEBUG] <sender 1.05>  Worker(/10.34.205.10:31275) -- Sent HELLO
2019-07-31 11:34:02.972 [DEBUG] <sender 1.08>  Worker(/10.34.205.10:31278) -- Sent HELLO
2019-07-31 11:34:02.972 [DEBUG] <sender 1.10>  Worker(/10.34.205.10:31280) -- Sent HELLO
2019-07-31 11:34:03.896 [ INFO] <PreReader #3>  Skipping empty source '/var/lib/ybrelay/work/local-1564598030512.fifos/task.2'
2019-07-31 11:34:03.896 [ INFO] <PreReader #0>  Skipping empty source '/var/lib/ybrelay/work/local-1564598030512.fifos/task.1'
2019-07-31 11:34:03.902 [DEBUG] <PreReader #3>  Read 0 segments for source#2
2019-07-31 11:34:03.903 [DEBUG] <PreReader #3>  Marking source#2 as 'completed'
2019-07-31 11:34:03.903 [DEBUG] <PreReader #0>  Read 0 segments for source#1
2019-07-31 11:34:03.903 [DEBUG] <PreReader #3>  Setting noMoreData on sequencingQueue #3
2019-07-31 11:34:03.903 [DEBUG] <PreReader #0>  Marking source#1 as 'completed'
2019-07-31 11:34:03.903 [DEBUG] <PreReader #3>  BgTask "PreReader #3" exited
2019-07-31 11:34:04.222 [DEBUG] <Sequencer #3>  BgTask "Sequencer #3" exited
2019-07-31 11:34:04.222 [DEBUG] <PreReader #0>  Setting noMoreData on sequencingQueue #0
2019-07-31 11:34:04.223 [DEBUG] <PreReader #0>  BgTask "PreReader #0" exited
2019-07-31 11:34:04.223 [DEBUG] <Sequencer #0>  BgTask "Sequencer #0" exited
2019-07-31 11:34:04.236 [DEBUG] <PreReader #2>  Read 1 segments for source#3
2019-07-31 11:34:04.236 [DEBUG] <PreReader #1>  Read 1 segments for source#0
2019-07-31 11:34:04.237 [DEBUG] <segment reader #1>  Segment0.0: starting 2 bundle lanes
2019-07-31 11:34:04.237 [DEBUG] <PreReader #1>  Setting noMoreData on sequencingQueue #1
2019-07-31 11:34:04.237 [DEBUG] <segment reader #0>  Segment3.0: starting 2 bundle lanes
2019-07-31 11:34:04.237 [DEBUG] <PreReader #2>  Setting noMoreData on sequencingQueue #2
2019-07-31 11:34:04.237 [DEBUG] <PreReader #1>  BgTask "PreReader #1" exited
2019-07-31 11:34:04.238 [DEBUG] <PreReader #2>  BgTask "PreReader #2" exited
2019-07-31 11:34:04.238 [DEBUG] <Wait for PreReaders>  Last PreReader finished. Calling postReadQueue.setNoMoreData()
2019-07-31 11:34:04.238 [DEBUG] <Wait for PreReaders>  Setting noMoreData on postReadQueue
2019-07-31 11:34:04.239 [DEBUG] <Wait for PreReaders>  BgTask "Wait for PreReaders" exited
2019-07-31 11:34:04.239 [DEBUG] <BgTask idle thread>  BgTask "segment parser #1.0" started
2019-07-31 11:34:04.239 [DEBUG] <segment reader #2>  BgTask "segment reader #2" exited
2019-07-31 11:34:04.239 [DEBUG] <BgTask idle thread>  BgTask "segment parser #1.1" started
2019-07-31 11:34:04.239 [DEBUG] <BgTask idle thread>  BgTask "segment parser #0.1" started
2019-07-31 11:34:04.239 [DEBUG] <segment reader #3>  BgTask "segment reader #3" exited
2019-07-31 11:34:04.239 [DEBUG] <BgTask idle thread>  BgTask "segment parser #0.0" started
2019-07-31 11:34:04.311 [DEBUG] <segment reader #0>  Segment3.0: finalizing audit w/ eofChannelPos=477866
2019-07-31 11:34:04.311 [DEBUG] <segment reader #0>  Setting noMoreData on slurpedBundleQueue #0
2019-07-31 11:34:04.376 [DEBUG] <segment parser #0.1>  BgTask "segment parser #0.1" exited
2019-07-31 11:34:04.391 [DEBUG] <segment parser #0.0>  BgTask "segment parser #0.0" exited
2019-07-31 11:34:04.391 [DEBUG] <segment reader #0>  Segment3.0: all bundle lanes exited
2019-07-31 11:34:04.391 [DEBUG] <segment reader #0>  Segment3.0: mustAdoptNext=null; curFailed=false
2019-07-31 11:34:04.391 [DEBUG] <segment reader #0>  BgTask "segment reader #0" exited
2019-07-31 11:34:04.391 [DEBUG] <Sequencer #2>  Segment3.0: sequenced. Submitting to finalizer...
2019-07-31 11:34:04.392 [DEBUG] <Sequencer #2>  BgTask "Sequencer #2" exited
2019-07-31 11:34:04.398 [DEBUG] <sender 1.09>  Marking source#3 as 'completed'
2019-07-31 11:34:04.435 [DEBUG] <segment reader #1>  Segment0.0: finalizing audit w/ eofChannelPos=2097199
2019-07-31 11:34:04.435 [DEBUG] <segment reader #1>  Setting noMoreData on slurpedBundleQueue #1
2019-07-31 11:34:04.451 [DEBUG] <segment parser #1.0>  BgTask "segment parser #1.0" exited
2019-07-31 11:34:04.475 [DEBUG] <segment parser #1.1>  BgTask "segment parser #1.1" exited
2019-07-31 11:34:04.475 [DEBUG] <segment reader #1>  Segment0.0: all bundle lanes exited
2019-07-31 11:34:04.476 [DEBUG] <segment reader #1>  Segment0.0: mustAdoptNext=null; curFailed=false
2019-07-31 11:34:04.476 [DEBUG] <Sequencer #1>  Segment0.0: sequenced. Submitting to finalizer...
2019-07-31 11:34:04.476 [DEBUG] <segment reader #1>  BgTask "segment reader #1" exited
2019-07-31 11:34:04.476 [DEBUG] <Sequencer #1>  BgTask "Sequencer #1" exited
2019-07-31 11:34:04.476 [DEBUG] <Wait for Sequencers>  Last Sequencer finished. Calling preFinalizeQueue.setNoMoreData()
2019-07-31 11:34:04.477 [DEBUG] <Wait for Sequencers>  Setting noMoreData on preFinalizeQueue
2019-07-31 11:34:04.477 [DEBUG] <Wait for Sequencers>  BgTask "Wait for Sequencers" exited
2019-07-31 11:34:04.477 [ INFO] <main>  Flushing last 15981 rows (of 19617 total) for transaction 1
2019-07-31 11:34:04.477 [DEBUG] <main>  Setting noMoreData on preSendQueue
2019-07-31 11:34:04.479 [DEBUG] <sender 1.12>  Marking source#0 as 'completed'
2019-07-31 11:34:04.480 [DEBUG] <sender 1.12>  Validating overall success before closing worker sockets
2019-07-31 11:34:04.481 [DEBUG] <sender 1.12>  Declaring success (maybe partial) because we sent some rows
```

```
2019-07-31 11:34:04.484 [DEBUG] <sender 1.06>  Worker(/10.34.205.10:31276) -- Sent EOF after 3 buffers, 8979 rows, and 0 idle msgs
2019-07-31 11:34:04.484 [DEBUG] <sender 1.03>  Worker(/10.34.205.10:31273) -- Sent EOF after 1 buffers, 2990 rows, and 0 idle msgs
2019-07-31 11:34:04.484 [DEBUG] <sender 1.13>  Worker(/10.34.205.10:31283) -- Sent EOF after 0 buffers, 0 rows, and 0 idle msgs
2019-07-31 11:34:04.484 [DEBUG] <sender 1.04>  Worker(/10.34.205.10:31274) -- Sent EOF after 0 buffers, 0 rows, and 0 idle msgs
2019-07-31 11:34:04.485 [DEBUG] <sender 1.04>  Worker(/10.34.205.10:31274) -- Flushing and Closing
2019-07-31 11:34:04.484 [DEBUG] <sender 1.05>  Worker(/10.34.205.10:31275) -- Sent EOF after 0 buffers, 0 rows, and 0 idle msgs
2019-07-31 11:34:04.484 [DEBUG] <sender 1.10>  Worker(/10.34.205.10:31280) -- Sent EOF after 0 buffers, 0 rows, and 0 idle msgs
2019-07-31 11:34:04.484 [DEBUG] <sender 1.08>  Worker(/10.34.205.10:31278) -- Sent EOF after 0 buffers, 0 rows, and 0 idle msgs
2019-07-31 11:34:04.485 [DEBUG] <sender 1.08>  Worker(/10.34.205.10:31278) -- Flushing and Closing
2019-07-31 11:34:04.484 [DEBUG] <sender 1.09>  Worker(/10.34.205.10:31279) -- Sent EOF after 2 buffers, 3636 rows, and 0 idle msgs
2019-07-31 11:34:04.484 [DEBUG] <sender 1.14>  Worker(/10.34.205.10:31284) -- Sent EOF after 0 buffers, 0 rows, and 0 idle msgs
2019-07-31 11:34:04.484 [DEBUG] <sender 1.01>  Worker(/10.34.205.10:31271) -- Sent EOF after 1 buffers, 1201 rows, and 0 idle msgs
2019-07-31 11:34:04.486 [DEBUG] <sender 1.01>  Worker(/10.34.205.10:31271) -- Flushing and Closing
2019-07-31 11:34:04.484 [DEBUG] <sender 1.07>  Worker(/10.34.205.10:31277) -- Sent EOF after 0 buffers, 0 rows, and 0 idle msgs
2019-07-31 11:34:04.484 [DEBUG] <sender 1.12>  Worker(/10.34.205.10:31282) -- Sent EOF after 1 buffers, 2811 rows, and 0 idle msgs
2019-07-31 11:34:04.484 [DEBUG] <sender 1.11>  Worker(/10.34.205.10:31281) -- Sent EOF after 0 buffers, 0 rows, and 0 idle msgs
2019-07-31 11:34:04.484 [DEBUG] <sender 1.02>  Worker(/10.34.205.10:31272) -- Sent EOF after 0 buffers, 0 rows, and 0 idle msgs
2019-07-31 11:34:04.486 [DEBUG] <sender 1.02>  Worker(/10.34.205.10:31272) -- Flushing and Closing
2019-07-31 11:34:04.484 [DEBUG] <sender 1.00>  Worker(/10.34.205.10:31270) -- Sent EOF after 0 buffers, 0 rows, and 0 idle msgs
2019-07-31 11:34:04.486 [DEBUG] <sender 1.11>  Worker(/10.34.205.10:31281) -- Flushing and Closing
2019-07-31 11:34:04.486 [DEBUG] <sender 1.12>  Worker(/10.34.205.10:31282) -- Flushing and Closing
2019-07-31 11:34:04.486 [DEBUG] <sender 1.07>  Worker(/10.34.205.10:31277) -- Flushing and Closing
2019-07-31 11:34:04.485 [DEBUG] <sender 1.14>  Worker(/10.34.205.10:31284) -- Flushing and Closing
2019-07-31 11:34:04.485 [DEBUG] <sender 1.09>  Worker(/10.34.205.10:31279) -- Flushing and Closing
2019-07-31 11:34:04.485 [DEBUG] <sender 1.10>  Worker(/10.34.205.10:31280) -- Flushing and Closing
2019-07-31 11:34:04.485 [DEBUG] <sender 1.05>  Worker(/10.34.205.10:31275) -- Flushing and Closing
2019-07-31 11:34:04.484 [DEBUG] <sender 1.13>  Worker(/10.34.205.10:31283) -- Flushing and Closing
2019-07-31 11:34:04.484 [DEBUG] <sender 1.03>  Worker(/10.34.205.10:31273) -- Flushing and Closing
2019-07-31 11:34:04.484 [DEBUG] <sender 1.06>  Worker(/10.34.205.10:31276) -- Flushing and Closing
2019-07-31 11:34:04.486 [DEBUG] <sender 1.00>  Worker(/10.34.205.10:31270) -- Flushing and Closing
2019-07-31 11:34:04.489 [DEBUG] <sender 1.01>  Worker(/10.34.205.10:31271) -- Flushed and Closed
2019-07-31 11:34:04.489 [DEBUG] <sender 1.05>  Worker(/10.34.205.10:31275) -- Flushed and Closed
2019-07-31 11:34:04.489 [DEBUG] <sender 1.03>  Worker(/10.34.205.10:31273) -- Flushed and Closed
2019-07-31 11:34:04.489 [DEBUG] <sender 1.14>  Worker(/10.34.205.10:31284) -- Flushed and Closed
2019-07-31 11:34:04.489 [DEBUG] <sender 1.13>  Worker(/10.34.205.10:31283) -- Flushed and Closed
2019-07-31 11:34:04.489 [DEBUG] <sender 1.14>  BgTask "sender 1.14" exited
2019-07-31 11:34:04.489 [DEBUG] <sender 1.03>  BgTask "sender 1.03" exited
2019-07-31 11:34:04.489 [DEBUG] <sender 1.06>  Worker(/10.34.205.10:31276) -- Flushed and Closed
2019-07-31 11:34:04.489 [DEBUG] <sender 1.05>  BgTask "sender 1.05" exited
2019-07-31 11:34:04.489 [DEBUG] <sender 1.11>  Worker(/10.34.205.10:31281) -- Flushed and Closed
2019-07-31 11:34:04.489 [DEBUG] <sender 1.01>  BgTask "sender 1.01" exited
2019-07-31 11:34:04.489 [DEBUG] <sender 1.10>  Worker(/10.34.205.10:31280) -- Flushed and Closed
2019-07-31 11:34:04.489 [DEBUG] <sender 1.04>  Worker(/10.34.205.10:31274) -- Flushed and Closed
2019-07-31 11:34:04.490 [DEBUG] <sender 1.00>  Worker(/10.34.205.10:31270) -- Flushed and Closed
2019-07-31 11:34:04.490 [DEBUG] <sender 1.10>  BgTask "sender 1.10" exited
2019-07-31 11:34:04.489 [DEBUG] <sender 1.02>  Worker(/10.34.205.10:31272) -- Flushed and Closed
2019-07-31 11:34:04.489 [DEBUG] <sender 1.11>  BgTask "sender 1.11" exited
2019-07-31 11:34:04.489 [DEBUG] <sender 1.12>  Worker(/10.34.205.10:31282) -- Flushed and Closed
2019-07-31 11:34:04.489 [DEBUG] <sender 1.08>  Worker(/10.34.205.10:31278) -- Flushed and Closed
2019-07-31 11:34:04.489 [DEBUG] <sender 1.09>  Worker(/10.34.205.10:31279) -- Flushed and Closed
2019-07-31 11:34:04.491 [DEBUG] <sender 1.09>  BgTask "sender 1.09" exited
2019-07-31 11:34:04.489 [DEBUG] <sender 1.07>  Worker(/10.34.205.10:31277) -- Flushed and Closed
2019-07-31 11:34:04.491 [DEBUG] <sender 1.07>  BgTask "sender 1.07" exited
2019-07-31 11:34:04.489 [DEBUG] <sender 1.06>  BgTask "sender 1.06" exited
2019-07-31 11:34:04.489 [DEBUG] <sender 1.13>  BgTask "sender 1.13" exited
2019-07-31 11:34:04.491 [DEBUG] <sender 1.08>  BgTask "sender 1.08" exited
2019-07-31 11:34:04.491 [DEBUG] <sender 1.12>  BgTask "sender 1.12" exited
2019-07-31 11:34:04.491 [DEBUG] <sender 1.02>  BgTask "sender 1.02" exited
2019-07-31 11:34:04.490 [DEBUG] <sender 1.00>  BgTask "sender 1.00" exited
2019-07-31 11:34:04.490 [DEBUG] <sender 1.04>  BgTask "sender 1.04" exited
2019-07-31 11:34:04.496 [ INFO] <main>  Committing 19617 rows into transaction #1 for people2 ...
2019-07-31 11:34:06.640 [DEBUG] <main>  Received BulkService.commitTransaction() response for people2
2019-07-31 11:34:06.641 [ INFO] <main>  Committed transaction #1 after a total of 3432647 bytes and 19617 good rows
2019-07-31 11:34:06.641 [DEBUG] <main>  SessionNotifier: stopping
2019-07-31 11:34:06.689 [ INFO] <main>  READ: 2.46MB(246.8KB/s). ROWS G/B: 19617/0( 1.88K/s). WRITE: 3.27MB(329.0KB/s).  TIME E/R:
2019-07-31 11:34:06.691 [ INFO] <main>  SUCCESSFUL BULK LOAD: Loaded 19617 good rows in   0:00:10 (READ: 246.8KB/s WRITE: 329.0KB/
2019-07-31 11:34:06.691 [DEBUG] <main>  SessionNotifier.close() calling ntsStopAndReport(false)
2019-07-31 11:34:06.697 [DEBUG] <main>  Sending BulkService.done(success=true, clientMsg=null) for people2
```

```
2019-07-31 11:34:06.705 [DEBUG] <main>  Received BulkService.done() response for people2
2019-07-31 11:34:06.709 [DEBUG] <main>  All top-level threads have stopped
```

Spark Application Options

Platforms: All platforms

Parent topic: Setting up and Running a Spark Job

The options listed in this section are Spark application settings that you can use when you submit a Spark job that exports data to a Yellowbrick database. Some of these settings are required.

Required Settings

Every Spark job that exports data to Yellowbrick must define the format and location of the incoming data and connectivity settings for both the Yellowbrick database and the `ybrelay` service:

--export-directory STRING

A single directory, a single file reference, a comma-delimited list of directories/files, or a file listing reference if the argument begins with `list:`. For example:

```
--export-directory file:///opt/s3/data/data/People.csv

--export-directory list:/home/ybtests/ybrelay/spark/conf/export_file
```

--format

The default is `parquet`. `--format` is pluggable; for example, `avro` and `xml` can be loaded by providing an external package. For example: `--format com.databricks.spark.avro`

The supported formats are:

- `json`
- `csv`
- `text`
- `orc`
- `xml` (via plugin; see https://github.com/databricks/spark-xml))
- `avro` (via plugin; see https://github.com/databricks/spark-avro)
- `jdbc` (`--export-directory` not required). Specifying this format triggers the requirement to set the `--jdbc-*` options.

Other Options

--bad-column-names

A subset of columns for recording in the `.badrows` file that the ybload operation produces. For example:

```
--bad-column-names event_id,YBLOAD_ERROR_COLUMN,YBLOAD_ERROR_REASON
```

The pseudo-columns `YBLOAD_ERROR_COLUMN` and `YBLOAD_ERROR_REASON` can be supplied to record the column containing an error and the reason, respectively.

--buffer-size

Buffer size for network packets to relay, in bytes (default: `2097152`).

--cacert STRING

Customize trust with secured communication; use this option in combination with the `--secured` option. Enter the file name of a custom PEM-encoded certificate or the file name and password for a Java KeyStore (JKS).

For PEM format, the file must be named with a `.pem`, `.cert`, `.cer`, `.crt`, or `.key` extension. For example:

```
--cacert cacert.pem
```

For JKS format, files are always password-protected. Use the following format:

```
--cacert yellowbrick.jks:changeit
```

where the `:` character separates the file name and the password.

--column-names

Optionally, a list of the column names in the destination table. (If not specified, they are discovered when the Spark job connects to the target database.)

--computed-columns FILE | STRING

Specify one or more columns in expression form or use a properties file that contains expressions. The expression language is MVEL. This option is useful for including job-specific or task-specific context for the target table.

The following variables are supplied to each invocation of a computed column value:

- `transactionId`: the identifier of the row in the load job
- Spark row struct. See https://spark.apache.org/docs/latest/api/java/org/apache/spark/sql/Row.html
- taskContext: Spark task context. See https://spark.apache.org/docs/latest/api/java/org/apache/spark/TaskContext.html

For example:

```
two_columns=row.get(row.fieldIndex('one')) + '|' + row.get(row.fieldIndex('two'))
task_partition_id=taskContext.partitionId()
task_stage_id=taskContext.stageId()
```

--connection-timeout-seconds SECONDS

Communications timeout for connections, specified in seconds. Default: `120`.

--dbname, -d

Name of the destination database (equivalent to the `YBDATABASE` environment variable). Default: `yellowbrick`

--disable-trust, -k

Disable SSL/TLS trust when using secured communications. Trust is enabled by default. See also Verifying SSL/TLS Encryption.

Important: This option is not supported for use on production systems and is only recommended for testing purposes. It may be useful to disable trust during testing, then enable it when a formal signed certificate is installed on the system.

--filter EXPRESSION

A valid Spark SQL filter expression to constrain the data. A filter expression is like a SQL `WHERE` clause but does not use the `WHERE` keyword.

For example, filter a source `ORC` file on `_col1`:

```
--filter " _col1 between 18 and 20 "
--filter
```

--help

Show usage/help. Default: `false`.

--host, -h

Destination server host name. Default: `localhost`

--jdbc-url STRING

JDBC URL to connect to.

--jdbc-properties STRING

JDBC properties file, containing configuration parameters for the external database.

--jdbc-table STRING

JDBC table to load.

--load-option

An option that can be passed to the `ybload` process. See ybload Options for possible options and values. You can specify `--load-option` multiple times in the command, once per load option. The following `ybload` options cannot be specified: `--csv-delimiter`, `--csv-quote`, `--csv-quote-escape`, `--linesep`, `--nullmarker`.

For example:

```
--load-option "--on-missing-field SUPPLYNULL"
```

--log-level STRING

Specify the export log level. Valid values: `ERROR`, `WARN`, `INFO`, `DEBUG`, `TRACE`.

--logs STRING

Specify a directory where `ybrelay` logs will be exported, including a summary of the end of the `ybload` job.

--map FILE

Specify a column mapping file that contains simple column mappings (destination column=source column). For example, specify `--map column-mappings.txt`, where `column-mappings.txt` is a Java properties-style file with:

```
destination_column_one=source_column_one
```

In this case, a column called `destination_column_one` would be given input data from `source_column_one`.

--password, -W

Interactive prompt for the database user's password (equivalent to the `YBPASSWORD` environment variable). No default.

--port

Destination server port number (equivalent to the `YBPORT` environment variable). Default: `5432`

--pre-sql, --post-sql

Specify a SQL statement that is invoked by the Spark driver before or after the load job. Alternatively, you can use a file reference by prefixing the file name with `file:`. For example, `--pre-sql file:prepare-job.sql` assumes that the file `prepare-job.sql` contains a SQL template to be executed before the execution of the job itself.

Template values can be inserted using the MVEL template syntax. The following template variables are provided:

- `transactionId`: the identifier of this load job
- `sparkSession`: see Spark session.
- `sparkContext`: see Spark context.
- `config`: The supplied and parsed configuration for the export. Included values are `config.ybTableName`, `config.ybHost`, and so on.

For example:

```
--pre-sql "insert into job_log(start_time, finish_time, transaction_id) values(current_timestamp, null, '@{transactionId}')"
--post-sql "update job_log set finish_time = current_timestamp where transaction_id = '@{transactionId}'"
```

--queue-depth NUMBER

Queue depth for outbound network packets to relay. Default: `16`.

--read-options STRING

Specify read options (such as `basePath=`) in an external properties file and provide the path to the file.

Some of the formats for export require such options to optimize the read process or change its behavior. For example, you can manage the export of a single partition directory structure for all formats by using `basePath`. For example:

```
--export-directory /user/hive/warehouse/my_table/part1=foo/part2=bar
...
--read-options read-options.properties

read-options.properties:
basePath=/user/hive/warehouse/my_table
```

If the `basePath` property is not specified, Spark will export the data in the given export directory but will not export the partition keys. For read options that require a unicode or escape code (for example, when specifying an exotic delimiter setting), use the unicode escaping syntax as follows:

```
read-options.properties: separator=\u0001
```

--read-schema FILE

Specify a schema, in a Spark-specific format, that feeds the read process for certain files that do not contain schema information, such as header-less CSV and ORC files. Specify the name of a file that contains the schema in the correct format. For example:

```
--read-schema schema_for_table.json
```

See Spark Format for Reading Schemas.

--read-timeout-seconds SECONDS

Communications timeout for reads, specified in seconds. Default: `0`.

--secured

Use SSL/TLS to secure all communications. The default is not secured. See also Verifying SSL/TLS Encryption.

--spark-log-level STRING

Specify the Apache Spark log level setting. Valid values: `ERROR`, `WARN`, `INFO`, `DEBUG`, `TRACE`.

--table, -t

Name of the target table to load.

--task-failures NUMBER

Specify the maximum number of tolerated errors for any Spark task. Default: `1`. For details, see Error Tolerance. (See also the `spark.task.maxFailures` property.)

--truncate-prefix STRING, --truncate-suffix STRING

Specify a string that will be prepended or appended to data that exceeds the declared length of target columns.

--username, -U

Database login username (equivalent to the YBUSER environment variable). No default.

--write-timeout-seconds SECONDS

Communications timeout for writes, specified in seconds. Default: `0`.

--yb-relay-host STRING

Host name of the system running the `ybrelay` service. Default: `localhost`.

--yb-relay-port NUMBER

`ybrelay` port number. Default: `21212`.

Legacy Options

These options are deprecated but available for backward compatibility.

--yb-database STRING
Yellowbrick database name. The `--dbname` option is preferred.

--yb-host STRING
Yellowbrick database host name. The `--host` option is preferred.

--yb-password STRING
Yellowbrick user's password. The `--password` option is preferred.

--yb-port NUMBER
Yellowbrick database port number. Default: `5432`. The `--port` option is preferred.

--yb-table STRING
Name of the Yellowbrick table being loaded. The `--table` option is preferred.

--yb-user STRING
Yellowbrick user name. The `--user` option is preferred.

Spark Format for Reading Schemas

Platforms: All platforms

Parent topic: Setting up and Running a Spark Job

You can use the `--read-schema` Spark application option to define the read process for files that do not contain schema information, such as header-less CSV and ORC files.

For example, here is the partial schema description for a specific table:

```
{
  "type": "struct",
  "fields": [
   { "name": "ws_sold_date_sk","type": "integer", "nullable": true },
   { "name": "ws_sold_time_sk","type": "integer", "nullable": true },
   { "name": "ws_ship_date_sk","type": "integer", "nullable": true },
   { "name": "ws_item_sk","type": "integer", "nullable": false },
   { "name": "ws_wholesale_cost","type": "decimal(7,2)", "nullable": true },
   ...
  ]
}
```

In this format, you have to map each column from the input schema that is being *read*, using one of the following types. (The configuration for *writes* is determined dynamically by `ybload` and the Spark export process.) The typical mappings for Yellowbrick data types are shown in parentheses:

- `decimal` (`decimal`)
- `double` (`double precision`)
- `float` (`real`)
- `long` (`bigint`)
- `integer` (`int` / `integer`)
- `short` (`smallint`)
- `byte` (`tinyint`)
- `string` (`varchar`)
- `boolean` (`bool` / `boolean`)
- `date` (`date`)
- `timestamp` (`timestamp`)

Note: Yellowbrick does not support all the data types that Spark supports. For example, the `array` type is not supported.

Setting Up the ybrelay Service

Platforms: All platforms

Parent topic: Loading Tables with Spark

In this section:

This section explains how to set up `ybrelay`, which is part of the standard Yellowbrick `ybtools` package. Before installing `ybtools`, you must remove prior versions of `ybtools` from your system. Then you can install the current version of `ybtools` on the system where you want to run `ybrelay`. For installation instructions, see Installing ybtools.

Important: `ybrelay` is not supported on Windows clients.

A dedicated `ybrelay` instance must have network connectivity to the Yellowbrick database and must be running on a host that can be reached by the nodes where the Spark application is running. Optionally, you can set up `ybrelay` to run with TLS enabled for secure network communication between Spark, the `ybrelay` server, and the Data Warehouse.

Creating a Java Keystore

Platforms: All platforms

Parent topic: Setting Up the ybrelay Service

To enable TLS support, security information for the `ybrelay` service must be passed to the server via a Java keystore file, which typically contains:

- The private key for the server to secure identity
- The public certificate reflecting the identity of the server
- The public certificate to confirm the acceptable identity of all Data Warehouse systems it will be relaying to

You can create a keystore with a self-signed private key and certificate by running a `keytool -genkeypair` command:

```
$ keytool -genkeypair -noprompt -trustcacerts \
-keyalg RSA \
-alias <ENTRY_ALIAS> \
-keystore <OUTPUT_FILE> \
-storepass <STORE_PASSWORD> \
-dname CN=<HOSTNAME> \
-storetype jks
```

ENTRY_ALIAS
: Arbitrary alias that refers to the keypair in the truststore.

OUTPUT_FILE
: File name for the keystore.

STORE_PASSWORD
: Password used to encrypt the keystore.

HOSTNAME
: For public key infrastructure (PKI) trust, the host name specified must exactly match the host name of the `ybrelay` server to which the clients will be connecting.

You can add a certificate to an existing keystore. For example, you can add a Data Warehouse public certificate or CA certificate for trust authority by using the `keytool -importcert` command:

```
$ keytool -importcert -file <CERTIFICATE> -keystore <KEYSTORE_FILE> -v
```

CERTIFICATE
: PEM-encoded certificate file.

KEYSTORE_FILE
: File path of the existing Java keystore.

If you need to retrieve the certificate for an existing system, you can run the following `openssl` command:

```
openssl s_client -showcerts -servername yb100.nyc.yellowbrick.io -connect yb100.nyc.yellowbrick.io:443 </dev/null
```

The certificate will be encoded between the `-----BEGIN CERTIFICATE-----` and `-----END CERTIFICATE-----` markers.

Note: The host name of the certificate, which is typically listed as the common name (`CN`) portion of the certificate, must exactly match the host name being used by the application client connecting to `ybrelay`, or the `ybrelay` service connecting to the Data Warehouse.

Separate Keystores for the ybrelay Service and the Spark Clients

As a best practice, Yellowbrick recommends that you create two separate keystores: a `ybrelay` service keystore and a client keystore, as directed in the following procedure. The service keystore that contains the *private key* for the `ybrelay` service (which should be secured and not circulated among the clients) will belong to the `ybrelay` service host only. The client keystore, which will be distributed across many Spark hosts, will contain only *public key* material.

1. Use the `keytool` command to create a `ybrelay` service keystore, such as `service.jks`.
2. Export the service's public certificate from the service keystore:

```
keytool -export -keystore service.jks -alias service -file service.crt
```

3. Create a new client keystore by importing the service public certificate:

```
keytool -import -alias service -file service.crt -keystore client.jks
```

4. Import the Data Warehouse public certificate into the client keystore:

```
keytool -import -alias cluster -file cluster.crt -keystore client.jks
```

The result of this procedure is two keystores:

- `service.jks`, which includes:
- The public certificate for the `ybrelay` service
- The private key for the `ybrelay` service
- The public certificate for the Yellowbrick cluster
- `client.jks`, which includes:
- The public certificate for the `ybrelay` service
- The public certificate for the Yellowbrick cluster

Running the ybrelay-init Script

Platforms: All platforms

Parent topic: Setting Up the ybrelay Service

After installing `ybtools`, you can run the `ybrelay-init` script to create and start the `ybrelay systemd` service.

Note: As a prerequisite to running the script, the `JAVA_HOME` environment variable must be set.

1. On CentOS/RHEL 7 and Ubuntu 16.04/18.04, initialize the `ybrelay` service by running the following:

```
ybrelay-init
INFO: Creating System Group: ybrelay with GID: 923
INFO: Creating System User: ybrelay with UID: 923
INFO: Creating Systemd service
INFO: JAVA_HOME environment variable detected
INFO: Adding JAVA_HOME configuration to ybrelay.config
Created symlink from /etc/systemd/system/multi-user.target.wants/ybrelay.service to /etc/systemd/system/ybrelay.service.
INFO: Starting Systemd service
```

2. Verify that `ybrelay.service` was started properly by running the following:

```
systemctl status ybrelay.service
  ybrelay.service - Yellowbrick Data Load Relay Service
   Loaded: loaded (/etc/systemd/system/ybrelay.service; enabled; vendor preset: disabled)
   Active: active (running) since Thu 2019-11-14 18:31:23 UTC; 57s ago
Main PID: 3320 (java)
   CGroup: /system.slice/ybrelay.service
           └─3320 /opt/java/jre11/bin/java -Dvisualvm.display.name=YB:Relay -Xms512m -Xmx8g -ea -classpath
/opt/ybtools/etc:/opt/ybtools/lib/relay-service-3.2.0-SNAPSHOT.jar:/opt/ybtools/lib/client-com...

Nov 14 18:31:23 rhel7.localdomain systemd[1]: Started Yellowbrick Data Load Relay Service.
Nov 14 18:31:23 rhel7.localdomain ybrelay[3320]: Warning; java is not in your PATH. Using JAVA_HOME fallback:
/opt/java/jre11/bin
```

Running ybrelay-init with the --keystore option

You can run the `ybrelay-init` script to create and start the `ybrelay systemd` service. To enable a TLS configuration for network communication between the `ybrelay` server and the Data Warehouse, use the `--keystore` option when you run the script. (To disable TLS trust for testing purposes, use the `--disable-trust` option.) See also ybrelay-init Options.

Note: The `JAVA_HOME` environment variable must be set before the `ybrelay-init` script can be run.

1. On CentOS/RHEL 7 and Ubuntu 16.04/18.04, initialize the `ybrelay` service by running the following command:

```
$ sudo ybrelay-init --keystore /tmp/keystore.jks --prompt-password
Enter keystore password:
INFO: Creating system group: ybrelay with GID: 923
INFO: Creating system user: ybrelay with UID: 923
INFO: Copying keystore /tmp/keystore.jks to /opt/ybtools/config/keystore.jks
INFO: Creating keystore option: /opt/ybtools/config/keystore.jks:<PASSWORD>
INFO: Creating systemd service
```

```
INFO: JAVA_HOME environment variable detected
INFO: Adding JAVA_HOME configuration to ybrelay.config
Created symlink /etc/systemd/system/multi-user.target.wants/ybrelay.service → /etc/systemd/system/ybrelay.service.
INFO: Starting systemd service
```

2. Verify that `ybrelay.service` was started properly by running the following command:

```
systemctl status ybrelay.service
  ybrelay.service - Yellowbrick Data Load Relay Service
   Loaded: loaded (/etc/systemd/system/ybrelay.service; enabled; vendor preset: disabled)
   Active: active (running) since Thu 2020-12-10 18:31:23 UTC; 57s ago
...
```

The `ybrelay` service is now running, and it is configured to listen with TLS enabled for all Spark jobs. You do not need to start the service again before submitting jobs. However, when you run Spark jobs, you will need to specify some required TLS options that are specific to each job submission.

Starting the ybrelay Service Manually

If the environment variable `JAVA_HOME` is not set, `ybrelay-init` returns the following warnings:

```
ybrelay-init
INFO: Creating System Group: ybrelay with GID: 923
INFO: Creating System User: ybrelay with UID: 923
INFO: Creating Systemd service
WARN: JAVA_HOME environment variable not set
WARN: Please edit the ybrelay.config
WARN: Service will be not enabled
```

The following procedure explains how to set `JAVA_HOME` in the `ybrelay.config` file, then start the service manually. `ybrelay` cannot be started with `ybrelay-init` after the `ybrelay.config` file has been edited.

1. Edit the `/opt/ybtools/config/ybrelay.config` file and set the `JAVA_HOME` path:

```
JAVA_HOME=“/opt/java/jre11”
```

2. After editing the configuration, enable and start the service:

```
systemctl enable ybrelay.service
Created symlink from /etc/systemd/system/multi-user.target.wants/ybrelay.service to /etc/systemd/system/ybrelay.service
```

```
systemctl start ybrelay.service
```

3. Verify that the `ybrelay.service` was started properly:

```
systemctl status ybrelay.service
  ybrelay.service - Yellowbrick Data Load Relay Service
   Loaded: loaded (/etc/systemd/system/ybrelay.service; enabled; vendor preset: disabled)
   Active: active (running) since Thu 2019-11-14 18:40:33 UTC; 4s ago
Main PID: 3196 (java)
   CGroup: /system.slice/ybrelay.service
           └─3196 /opt/java/jre11/bin/java -Dvisualvm.display.name=YB:Relay -Xms512m -Xmx8g -ea -classpath
/opt/ybtools/etc:/opt/ybtools/lib/relay-service-3.2.0-SNAPSHOT.jar:/opt/ybtools/lib/client-com...

Nov 14 18:40:33 rhel7.localdomain systemd[1]: Started Yellowbrick Data Load Relay Service.
Nov 14 18:40:33 rhel7.localdomain ybrelay[3196]: Warning; java is not in your PATH. Using JAVA_HOME fallback:
/opt/java/jre11/bin.
```

TLS Support for ybrelay and Spark

Three separate hosts are involved in a `ybrelay` setup that runs a Spark job:

- Spark application platform (data source)
- `ybrelay` server (where the `ybrelay` tool is installed, with connectivity in both directions)
- Yellowbrick Data Warehouse (location of target database table)

The supported TLS configuration requires trust to be established on the entire path between the three hosts. This means that the `ybrelay` server is listening for requests on TLS, and the Yellowbrick Data Warehouse is listening for requests on TLS. Also, if a `ybrelay` service endpoint is configured for TLS, it is TLS-only; it cannot listen non-TLS.

The TLS configuration relies on keystores, which contain the required certificates and public/private key-pairs that the hosts use to verify trust when they receive connection requests. A keystore includes the key material for the relay service itself, and it will share the identity of the Data Warehouse with `ybload`. See Creating a Java Keystore.

The `ybrelay` service is typically started by `systemd`, so the `systemd` configuration needs to pass the keystore on service startup. See Running the ybrelay-init Script

---

TLS Requirements for Spark

Specify `--cacert` as a Spark application option to identify:

- The certificate used between Spark and the `ybrelay` server.
- The certificate used between Spark and the Data Warehouse.

Spark requires both certificates, except when the `--disable-trust` option is in use for testing. You can specify a keystore file as the value for the `--cacert` option.

See also Spark Application Options.

TLS Requirements for the ybrelay Server

Specify `--keystore` as a `ybrelay` service option to identify:

- The private key for the `ybrelay` service (its own private key).
- The certificate used between the `ybrelay` service and the Data Warehouse (to start the `ybload` session).

See Running the ybrelay-init Script and ybrelay Service Options.

Spark and ybrelay Glossary

Apache Spark
: Open-source platform for grid computing; a framework for solving analytics problems at large scale.

Avro
: Row-based storage format with its data definition in JSON, and the data itself in binary format, making it compact and efficient.

FIFOs
: Named pipes, as produced by the Linux `mkfifo` command.

HDFS
: Open-source Apache Hadoop distributed file system; manages very large data sets running on commodity hardware.

keystore
: A Java keystore (JKS) file that contains certificate and public/private key information required to run Spark jobs with TLS enabled.

Parquet
: Apache Parquet, an open-source column-oriented data storage format commonly used in Hadoop projects.

Spark application
: An application that generically consumes any data Spark feeds it (in row form).

Spark job
: A job that is submitted to Spark to handle large-scale data export or import.

TLS
: Transport Layer Security, a communications protocol for authenticated and encrypted connections over a network. The TLS and SSL terms tend to be used interchangeably. SSL/TLS is also used.

ybrelay
: Yellowbrick "relay" client tool that accepts incoming data in various formats from any external file system and calls `ybload` to bulk load it into tables.

ybload
: Yellowbrick bulk load client tool.

ybrelay Options

The `ybrelay` service can run without options; however, many options are available, as described in this section. This section also covers options available for the `ybrelay-init` script.

A simple start command would identify the work directory (`-w` or `--workdir` option):

```
$ ybrelay -w ./work
14:23:34.123 [ INFO] ABOUT CLIENT:
   app.cli_args         = "-w" "./work"
   app.name_and_version = "Yellowbrick Data Relay version 3.1.0-616"
   java.home            = "/usr/lib/jvm/java-11-oracle/jre"
   java.version         = "11"
   jvm.memory           = "491.00 MB (max=7.11 GB)"
   jvm.name_and_version = "Java HotSpot(TM) 64-Bit Server VM (build 25.101-b13, mixed mode)"
   jvm.options          = "-Xms512m, -Xmx8g, -ea, -Dapp.name=ybrelay, -Dapp.pid=11403, -Dapp.repo=/opt/ybtools/3.1.0.616/lib, -Dap
   jvm.vendor           = "Oracle Corporation"
   os.name_and_version  = "Linux 4.4.0-31-generic (amd64)"

14:23:34.125 [ INFO] Yellowbrick log level: DEBUG
...
```

Startup options may be specified in the `ybrelay.options` file.

Contents of the ybrelay.options File

The default `ybrelay.options` file, found in the `/opt/ybtools/<version>/config` directory, contains the following entries:

```
--workdir
/var/lib/ybrelay/work
--logdir
/var/log/ybrelay/logs
--log-level
WARN
--logfile
/var/log/ybrelay/ybrelay.log
--logfile-log-level
DEBUG
--logfile-backups
10
--logfile-size
10MB
```

The options in the file are defined in the following sections.

Common Options

The following `ybrelay` options are common to several `ybtools` clients.

@file

Specify a file that includes a set of `ybrelay` options and values to use. Options in the file must be specified in the following form (using separate lines for each option name and value):

```
option1
  value for option1
  option2
  value for option2
  ...
```

--cacert STRING

Customize trust with secured communication; use this option in combination with the `--secured` option. Enter the file name of a custom PEM-encoded certificate or the file name and password for a Java KeyStore (JKS).

For PEM format, the file must be named with a `.pem`, `.cert`, `.cer`, `.crt`, or `.key` extension. For example:

```
--cacert cacert.pem
```

For JKS format, files are always password-protected. Use the following format:

```
--cacert yellowbrick.jks:changeit
```

where the `:` character separates the file name from the password of the keystore.

See also Secure Connections for Java-based ybtools.

--disable-trust, -k

Disable SSL/TLS trust when using secured communications. Trust is enabled by default. See also Verifying SSL/TLS Encryption.

Important: This option is not supported for use on production systems and is only recommended for testing purposes. It may be useful to disable trust during testing, then enable it when a formal signed certificate is installed on the system.

-?, --help, --help-advanced

Return basic (or advanced) usage information for the `ybrelay` command and its options.

--java-version

Return the Java version that is running on the client system. The client tools require the 64-bit version of Java 8 (also known as Java 1.8). Java 9 and 10 are not supported.

--log-level

Specify the logging level for the default console output. The default level is `WARN`. (Use the `--logfile-log-level` option to specify the logging level for a named log file.)

Using the default `WARN` setting reduces the logs that `ybrelay` writes to `stderr`.

--logfile

Specify the name and location of a log file for `ybrelay` operations. If the specified file already exists, it will be truncated. If this option is not specified, no log file is written. When you specify this `--logfile` option, also specify a `--logfile-log-level` value other than `OFF`.

--logfile-log-level

Specify the logging level for a given log file (as defined with the `--logfile` option). If the level is not specified, it defaults to the `--log-level` value. You must specify a `--logfile-log-level` value other than `OFF` when you specify the `--logfile` option.

-q, --quiet

Do not write any output to the console. If `--quiet` is specified, you must also specify `--logfile`.

--secured

Use SSL/TLS to secure all communications. The default is not secured. See also Verifying SSL/TLS Encryption.

--version

Display the version of `ybrelay` you are running (as part of `ybtools`). This option is not intended to be combined with other options. For example:

```
$ ybrelay --version
Yellowbrick Data Relay version 5.1.0-33316
```

---

ybrelay Service Options

The following options are specific to the `ybrelay` service.

--address STRING, -a STRING

Listen address. Default: `0.0.0.0`.

--connection-timeout-seconds NUMBER

Connection timeout (in seconds). Default: `120`.

--idle-timeout NUMBER

Detect idle timeout for aborted loads (in minutes). Default: `5`

--keystore FILENAME[:PASSWORD]

Specify the path to the Java keystore that includes the TLS private key and certificate. For example:

```
--keystore mykeystore.jks:******
```

--log-retention-days NUMBER

Set the number of days to retain stored log files (in `/var/log/ybrelay/logs` or the specified `logdir`). Default: `14`. This directory can grow large if a large number of `ybload` logs are generated. For example, if ybrelay is running a distinct `ybload` operation every minute, at least 2880 files are generated daily.

--logdir STRING, -l STRING

Specify a path where log data is created and managed. Default: `/var/lib/ybrelay/work`.

--logfile-backups NUMBER

Set the maximum number of log file backups. Default: `10`. The `ybrelay.log` file is managed with the `--logfile-backups` and `--logfile-size` values, which set the rotation policies for this log file.

--logfile-size STRING

Set the maximum log file size. Default: `100MB`.

--port NUMBER, -p NUMBER

Listen port. Default: `21212`.

--queuesizebytes NUMBER, -s NUMBER

Set the queue size, in bytes (per load job). Default: `536870912`.

--read-timeout-seconds NUMBER

Set the read timeout (in seconds). Default: `0`.

--tasks NUMBER, -t NUMBER

Set the task count (for parallelism). Default: `4`.

--workdir STRING, -w STRING

Set the path to create and manage temporary data. Default: `/var/lib/ybrelay/work`.

--write-timeout-seconds NUMBER

Set the write timeout (in seconds). Default: `0`.

--ybload STRING, -y STRING

Set the path to the `ybload` client.

--ybload-args STRING

Set the default arguments to `ybload`. Default:

```
--no-trim-white --log-level WARN --logfile-log-level DEBUG
```

--ybload-vmargs STRING

Set VM arguments to `ybload` (if `--ybload` path is not used).

---

ybrelay-init Options

You can specify the following options when you run `ybrelay-init`:

--disable-trust

Use the `ybrelay` service `--disable-trust` option for the `systemd` service.

--force

Confirm that you want to overwrite an existing configuration of the `ybrelay` service. For example, you must use this option if you want to change a previous keystore configuration.

--help

Display help text for the `ybrelay-init` options.

--keystore

Enable use of the `ybrelay` service `--keystore` option for the `systemd` service. If you intend to use `ybrelay` in TLS mode, you must enable the `--keystore` option here, then use the `--keystore` option when you run the Spark job.

--prompt-password

Prompt the user for a password: for use in conjunction with the `--keystore` option. For example:

```
ybrelay-init --keystore ybkeystore.jks --prompt-password
```

Trickle Loading Data via JDBC

In this section:

Unlike traditional OLAP systems, a Yellowbrick database stores data in both column-oriented and row-oriented storage to support both high-performance large-scale analytics and real-time trickle loads. Column-oriented storage is common in OLAP systems because it is the most efficient storage for large-scale analytics; however, columnar storage requires data to be loaded in large batch sizes of ten million rows or more. Because of this restriction, most analytic databases only support large bulk data loads, which both delay the time until data is available for processing and complicate data transfer.

To support “real-time” data processing requires an approach that allows insertion of individual rows into the database. Therefore, Yellowbrick implemented a row-oriented storage engine in addition to columnar storage. This approach allows efficient loading of individual rows with SQL INSERT statements. Data is always visible in real time because queries automatically process data from both the row storage and the columnar storage. Additionally, Yellowbrick automatically manages data transfer between the row and column stores in the background to maximize performance without user intervention.

This section describes the best practices required to achieve up to 250,000 rows per second using SQL INSERT statements and the JDBC driver. For higher ingest rates, up to and exceeding 10,000,000 rows per second, you can use the Yellowbrick bulk loader (`ybload`).

Example of a Trickle Load Java Program

The following Java program tests different batch sizes and concurrent connections. You can modify and run the program to test and tune your workload, or you can contact your Yellowbrick Sales Engineer for assistance.

This example demonstrates several coding best practices that are necessary to optimize performance:

- Ensure no thread contention exists between concurrent loaders.
- Use `java.util.concurrent.Atomic<Class>`, `java.util.concurrent.<Class>Adder`, and so on, to perform accumulation of statistics among threads and to avoid thread lock contention, which decreases concurrency.
- Design for batch size and use statistics to find the optimal batch size for a given workload. Do this iteratively and for each table.
- Do not hand-write INSERT statements; instead use parameterized inserts using a `PreparedStatement` per thread.
- Use `addBatch/executeBatch` to achieve the batching.

Java

```java
package io.yellowbrick;

import java.sql.Connection;
import java.sql.Driver;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.Timestamp;
import java.util.Base64;
import java.util.HashMap;
import java.util.Map;
import java.util.Properties;
import java.util.Random;
import java.util.concurrent.CompletionService;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.ExecutorCompletionService;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.atomic.AtomicInteger;
import java.util.concurrent.atomic.LongAdder;

public class Load {
   private final String mUrl;
   private final Driver mDriver;
   private final Properties mProperties;
   private final Options mOptions;
   private final AtomicInteger mLoader = new AtomicInteger();
   private final LongAdder mInserted = new LongAdder();
   private long mStart;

   public Load(String url, Options options, Properties properties) throws SQLException {
       this.mUrl = url;
       this.mOptions = options;
       this.mProperties = properties;
       this.mDriver = DriverManager.getDriver(mUrl);
   }
```

```
private static final class Options {
    int mLoaders = 5;
    int mBatchSize = 500;
    int mTransactionSize = 500;
    int mInsertions = 1 * 1000 * 1000;
    String mDriver = "postgresql";
    String mTable = "load_test";
    boolean mTablePerLoad = false;
    boolean mDropTable = false;
    boolean mUnlogged = false;
}

public static void main(String[] args) throws Throwable {

    // Do arguments: required host port database user password
    Options options = new Options();
    if (args.length < 5) {
        System.err.println("Usage: Load host port database user password [option1=value1] [option2=value2]");
        System.err.println("Where: options can be:");
        System.err.printf("       [loaders=NN] # of loaders (default: %d)\n", options.mLoaders);
        System.err.printf("       [batchsize=NN] size of batches (default: %d)\n", options.mBatchSize);
        System.err.printf("       [transactionsize=NN] size of transactions (default: %d)\n", options.mTransactionSize);
        System.err.printf("       [insertions=NN] # of insertions (default: %d rows)\n", options.mInsertions);
        System.err.printf("       [driver=postgresql|pgsql] driver (default: %s)\n", options.mDriver);
        System.err.printf("       [table=tablename] driver (default: %s)\n", options.mTable);
        System.err.printf("       [table-per-load=true|false] unique table per load (default: %s)\n", options.mTablePerLoad);
        System.err.printf("       [drop-table=true|false] drop table before load (default: %s)\n", options.mDropTable);
        System.err.printf("       [unlogged=true|false] create table UNLOGGED (default: %s)\n", options.mUnlogged);
        System.err.println("  or any jdbc property for the jdbc driver");
        System.err.println();
        System.err.println("Example: Load yb11 5432 yellowbrick yellowbrick yellowbrick insertions=100000");
        System.exit(1);
    }

    // Specify required and optional properties.
    Properties properties = new Properties();
    properties.put("user", args[3]);
    properties.put("password", args[4]);
    for (int i = 5; i < args.length; ++i) {
        String[] parts = args[i].split("=");

        // Discover our app's internal options.  Others are passed to JDBC connection.
        if ("loaders".equals(parts[0])) {
            options.mLoaders = Integer.parseInt(parts[1]);
        } else if ("batchsize".equals(parts[0])) {
            options.mBatchSize = Integer.parseInt(parts[1]);
        } else if ("transactionsize".equals(parts[0])) {
            options.mTransactionSize = Integer.parseInt(parts[1]);
        } else if ("insertions".equals(parts[0])) {
            options.mInsertions = Integer.parseInt(parts[1]);
        } else if ("driver".equals(parts[0])) {
            options.mDriver = parts[1];
        } else if ("table".equals(parts[0])) {
            options.mTable = parts[1];
        } else if ("drop-table".equals(parts[0])) {
            options.mDropTable = "true".equals(parts[1]);
        } else if ("table-per-load".equals(parts[0])) {
            options.mTablePerLoad = "true".equals(parts[1]);
        } else if ("unlogged".equals(parts[0])) {
            options.mUnlogged = "true".equals(parts[1]);

        // A jdbc option.
        } else {
            properties.put(parts[0], parts[1]);
        }
    }
```

```
        // Make the loader, and run it.
        new Load(String.format("jdbc:%s://%s:%s/%s", options.mDriver, args[0], args[1], args[2]), options, properties).run();
    }

    private void run() throws Throwable {
        System.out.printf("Connecting to database %s with parameters %s\n", mUrl, mProperties);
        System.out.printf("Using %d loader threads with batch size %d, transaction size %d, to load %d rows\n", mOptions.mLoaders,

        // Create initial connection.
        try (Connection connection = connect()) {

            // Create N loader threads, and submit each to executor.
            ExecutorService executor = Executors.newFixedThreadPool(mOptions.mLoaders);
            CompletionService<Map<String, Object>> completionService = new ExecutorCompletionService<>(executor);
            try {

                // Unique table for the loader?
                if (!mOptions.mTablePerLoad) {
                    prepareTable(connection, mOptions.mTable);
                }

                // Record our start time.
                mStart = System.nanoTime();

                // Submit N loaders.
                for (int i = mOptions.mLoaders; i > 0; --i) {
                    completionService.submit(this::load);
                    System.out.printf("Launched loader %d\n",(mOptions.mLoaders - i) + 1);
                }

                // Now, poll through the completed work, waiting for each to resolve.
                for (int i = mOptions.mLoaders; i > 0; --i) {
                    try {
                        // Wait 1s, if we are not done, report and keep trying.
                        while (true) {
                            Future<?> task = completionService.poll(1, TimeUnit.SECONDS);
                            if (task == null) {
                                report();
                            } else {
                                Object loader = task.get();
                                System.out.printf("\nReceived completion for loader %s", loader);
                                break;
                            }
                        }

                    } catch (ExecutionException e) {
                        throw e.getCause();
                    } catch (InterruptedException e) {
                        Thread.currentThread().interrupt();
                        break;
                    }
                }

                // Final report.
                System.out.println();
                report();
                System.out.println();

                // Do final flush.
                if (!mOptions.mTablePerLoad) {
                    System.out.println("Doing final flush to " + mOptions.mTable);
                    long flushStart = System.nanoTime();
                    try (Statement statement = connection.createStatement()) {
                        statement.execute("YFLUSH " + mOptions.mTable);
                    }
                    System.out.printf("Flush took %d ms\n", TimeUnit.NANOSECONDS.toMillis(System.nanoTime() - flushStart));
                }
```

```
            } finally {
                executor.shutdown();
            }

            System.out.println("Done");
        }
    }

    private void prepareTable(Connection connection, String tableName) throws SQLException {

        // Drop table if required.
        if (mOptions.mDropTable) {
            try (Statement statement = connection.createStatement()) {
                statement.execute(String.format("DROP TABLE IF EXISTS %s", tableName));
            }
        }

        // Create table.
        try (Statement statement = connection.createStatement()) {
            statement.execute(String.format("CREATE %s TABLE IF NOT EXISTS %s(a VARCHAR(32), b VARCHAR(64), c BIGINT, d NUMERIC(10,
        }
    }

    private Map<String, Object> load() throws SQLException {
        Map<String, Object> result = new HashMap<>();
        int id = mLoader.incrementAndGet();
        result.put("id", id);

        // Initialize our randomizer; doesn't need to be cryptographically secure.
        Random random = new Random();

        // We use two buffers for a/b columns to randomize
        byte[] a = new byte[16];
        byte[] b = new byte[32];

        // Create initial connection.
        try (Connection connection = connect()) {
            connection.setAutoCommit(false);

            // Construct table name.
            String tableName = mOptions.mTablePerLoad ? String.format("%s__%d", mOptions.mTable, id) : mOptions.mTable;
            result.put("table", tableName);

            // Unique table for the loader?
            if (mOptions.mTablePerLoad) {
                prepareTable(connection, tableName);
            }

            // Create insertion statement.
            try (PreparedStatement preparedStatement = connection.prepareStatement(String.format("INSERT INTO %s VALUES(?, ?, ?, ?,

                // Loop until we've got the # of insertions we want.
                int inserted = 0, uncommitted = 0, committed = 0;
                while (mInserted.longValue() < mOptions.mInsertions) {

                    // Create a batch.
                    for (int i = mOptions.mBatchSize; i > 0; --i) {
                        preparedStatement.setString(1, randomString(random, a));
                        preparedStatement.setString(2, randomString(random, b));
                        preparedStatement.setLong(3, random.nextLong());
                        preparedStatement.setDouble(4, random.nextDouble());
                        preparedStatement.setTimestamp(5, new Timestamp(System.currentTimeMillis()));
                        preparedStatement.setBoolean(6, random.nextBoolean());
                        preparedStatement.addBatch();
                    }

                    // Execute the batch.
                    preparedStatement.executeBatch();
```

```
                    uncommitted += mOptions.mBatchSize;
                    if (uncommitted >= mOptions.mTransactionSize) {
                        connection.commit();
                        committed++;
                        uncommitted = 0;
                    }

                    // Increment what we did.
                    mInserted.add(mOptions.mBatchSize);
                    inserted += mOptions.mBatchSize;
                }
                connection.commit();
                result.put("inserted", inserted);
                result.put("committed", committed);
            }

            // Do final flush.
            if (mOptions.mTablePerLoad) {
                long flushStart = System.nanoTime();
                try (Statement statement = connection.createStatement()) {
                    statement.execute("YFLUSH " + tableName);
                }
                result.put("flush_ms", TimeUnit.NANOSECONDS.toMillis(System.nanoTime() - flushStart));
            }
        }

        return result;
    }

    private String randomString(Random random, byte[] buffer) {
        random.nextBytes(buffer);
        return Base64.getEncoder().encodeToString(buffer);
    }

    private void report() {
        long nanos = System.nanoTime() - mStart;
        long inserted = mInserted.longValue();
        double seconds = TimeUnit.NANOSECONDS.toSeconds(nanos);
        double rate = inserted / seconds;
        System.out.printf("\rInsertion rate: %.2f rows/second, [%d total, %d seconds]", rate, inserted, (long) seconds);
    }

    private Connection connect() throws SQLException {
        return this.mDriver.connect(mUrl, mProperties);
    }
}
```

Performance Factors

Trickle load performance is primarily affected by two key factors: batch size and concurrency.

Batch size refers to the number of rows within a single database transaction. Each transaction requires additional overhead when it commits data, including writing to the transaction log and replicating data between nodes for high availability in the case of hardware failure. Because of this overhead, you need to include multiple rows in a single transaction to maximize performance.

For example, this JDBC operation contains a single transaction with a batch size of two:

```
BEGIN TRANSACTION;
INSERT INTO foo VALUES (1,2,3,'hello');
INSERT INTO foo VALUES (2,3,4,'world');
COMMIT;
```

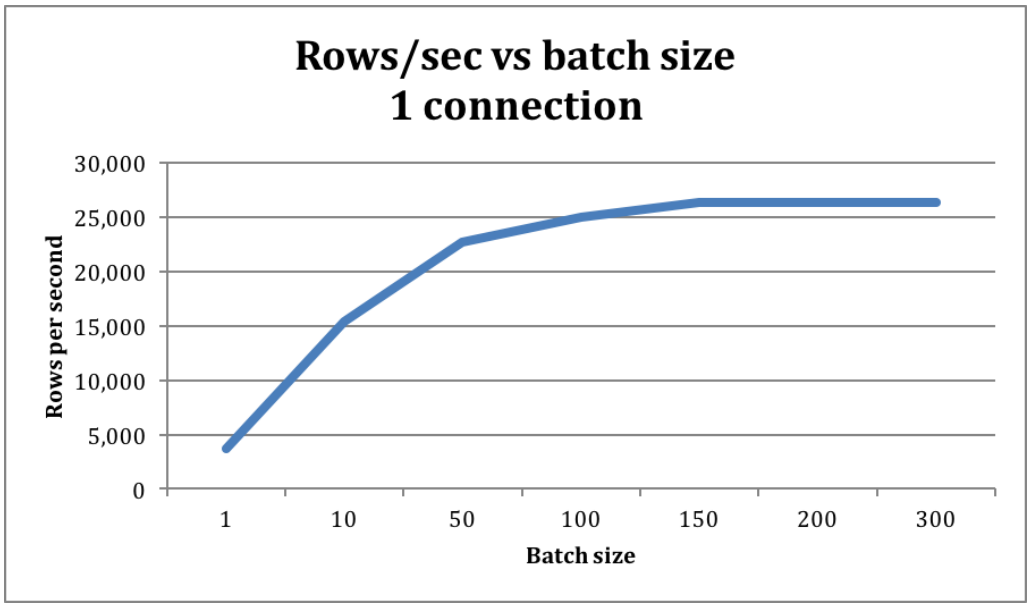
Concurrency refers to the number of simultaneous connections to the database, each performing separate data transfer operations in separate transactions. Because the process of loading data from the client to the server is latency bound (including network latency, JDBC driver overhead, disk latency, replication latency, and so on), to increase transfer rates, you need to increase parallelism. As shown in the example code, the best practice in Java is to create multiple connections, each one using a separate thread.

Performance Tuning

This section shows how batch size and concurrency affect performance and how to use these measurements to maximize throughput.

Effect of Batch Size on Performance

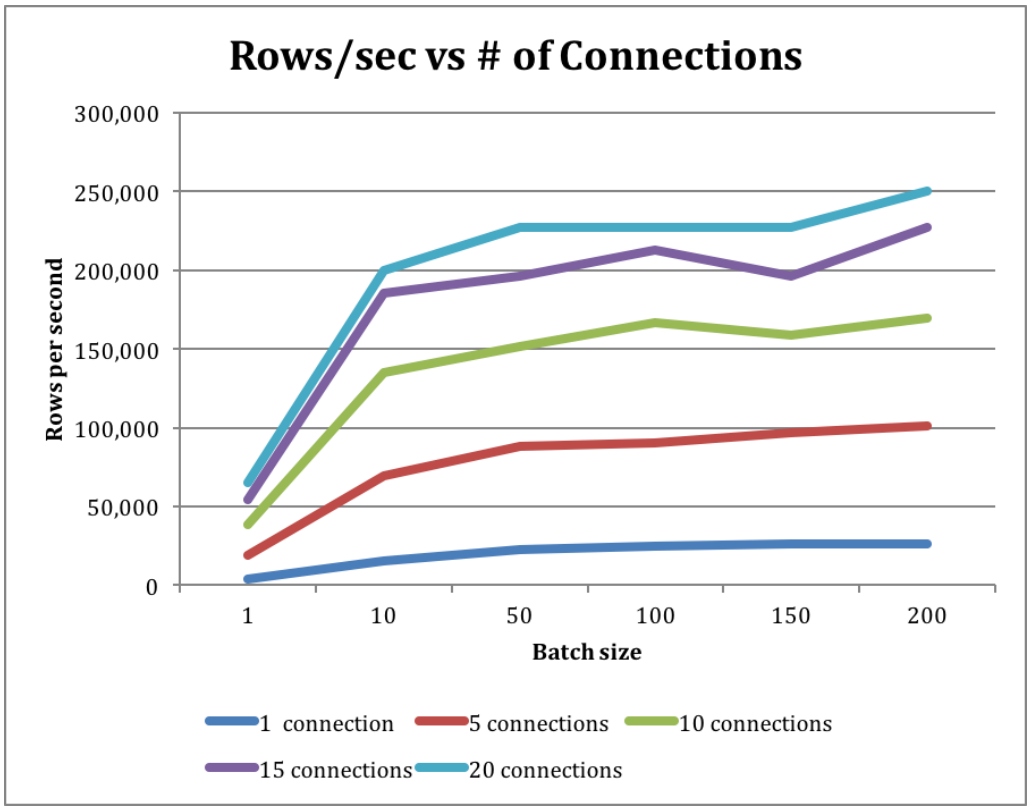
The following graph shows how batch size affects performance when data is inserted using a single connection:



With a single connection, a maximum throughput of ~25K rows per second requires a batch size of 100-150 INSERT statements per transaction. This value can vary depending on the row width for the current table. Therefore, Yellowbrick recommends testing different batch sizes per transaction to identify which batch size is optimal for a given table.

Effect of Concurrency on Performance

The following graph shows how performance scales as the number of connections and batch size increase:



You can achieve a maximum rate of ~200K rows per second with 15 concurrent connections and a batch size of 50 INSERT operations per transaction. You can also achieve this rate with 20 concurrent connections and a batch size of 10 INSERT operations per transaction. If fewer than 15 connections or batch sizes less than 10 are used, throughput begins to significantly decrease.

Prepared Statements and Command Batching

When you are running multiple INSERT operations, always use prepared statements for better performance, as well as command batching, as shown in the following code example.

Choice of JDBC Driver

For query workloads, the pgjdbc-ng JDBC driver is recommended because it offers significantly higher performance and lower latency over large queries: up to 10x higher throughput.

However, the pgjdbc-ng driver does not support batching of multiple statements efficiently. For inserting data, the regular PostgreSQL JDBC driver performs better and is recommended.

Running Queries

You can query your data in Yellowbrick in three ways, depend on whether you are using the appliance or are running Yellowbrick on public cloud platforms. In all instances, the native command line client tool `ybsql` can be used to execute SQL queries against Yellowbrick. Alternatively, you can use your preferred third-party ODBC/JDBC-compatible query tool.

If you are using Yellowbrick in public clouds, you can also query your data with the Query Editor in Yellowbrick Manager.

SQL Query Commands

Yellowbrick supports ANSI SQL queries, with support for a large set of SQL functions, including window functions, and pattern matching with POSIX regular expressions. For details, see SQL Functions and Operators. You can also use prepared statements.

Yellowbrick supports the following commands for submitting queries:

- SELECT
- SELECT INTO
- TABLE
- CREATE TABLE AS (CTAS)
- PREPARE, EXECUTE, DEALLOCATE (prepared statements)

For detailed syntax, see SQL Commands.

Transaction Blocks

You can run queries and other SQL commands within transaction blocks by using standard BEGIN and END statements. You can also create savepoints within transactions and roll back to these savepoints.

By default, when BEGIN and END statements are not used, Yellowbrick executes transactions in "autocommit" mode; each statement is executed in its own transaction and an implicit commit occurs at the end of the statement (assuming that execution was successful).

The only transaction isolation level that Yellowbrick supports is `read committed`.

Restrictions

Note the following restrictions on Yellowbrick queries:

- You can run read queries across physical databases, but you cannot write to remote databases. See Cross-Database Queries.
- You cannot run queries against a mixture of user-defined tables and system tables or views. One exception to this rule is the sys.const table, which can be queried in conjunction with regular tables.
- You cannot create a table by selecting from PostgreSQL system catalog tables.

Unloading Tables

Yellowbrick Documentation > How-to Guides > Unload Tables

Platforms: All platforms

Parent topic: How-to Guides

In this section:

This section explains how to unload tables and query results, using the `ybunload` client tool. The `ybunload` client is a high-performance parallel export tool that unloads data to files on the client system. The client connects to the manager node, then distributes the work of reading and writing the data to the compute blades. Data can be returned to the client in compressed (GZIP) or uncompressed files. The unloaded data can be streamed out to multiple output files or combined into a single file, depending on your requirements.

Alternatively, on cloud platforms see UNLOAD TABLE.

Download and install the unload client (as part of the ybtools package) from Yellowbrick Manager or the Customer Support download site. Then run the Linux and Windows executable programs:

- `ybunload` on Linux
- `ybunload.exe` on Windows

Note: Yellowbrick recommends that you upgrade to the latest version of the `ybtools` clients when you upgrade the cluster so that your client and server versions correspond. The client tools are backward-compatible but not always forward-compatible.

The following table shows which formats and storage targets are supported for `ybunload` operations:

Supported Formats	Supported Storage Targets	Notes
`csv`	Local file system	`csv` is the default unload format.
	S3 or S3-compatible object storage	
`text`	Local file system	
	S3 or S3-compatible object storage	
`parquet`	Local file system	
	S3 or S3-compatible object storage	
	Azure Blob storage, Azure Data Lake Storage Gen2	Data unloaded to Azure object storage must be unloaded in `parquet` format.

Unloading Data to Object Storage

Yellowbrick Documentation > How-to Guides > Unload Tables > Unloading Data to Object Storage

Platforms: All platforms

Parent topic: Unload Tables

In this section:

- S3 Authentication Methods
- Unloading Data to an S3 Bucket
- Unloading Data to Azure Storage
- Unloading Data to Google Storage

This section covers the details for unloading data from Yellowbrick tables to object storage systems:

- AWS S3
- S3-compatible storage
- Azure Blob storage
- Azure Data Lake Gen 2
- Google Storage

S3 Authentication Methods

Platforms: All platforms

Parent topic: Unloading Data to Object Storage

There are several different ways to authenticate when you are using `ybunload` to unload data from AWS S3 or S3-compatible object storage. You can authenticate implicitly by using supported S3-specific mechanisms, or explicitly by using `ybunload` command-line options. Your organization's S3 administrator should provide instructions for the approach you should use. See Best Practices for Managing AWS Access Keys for further recommendations.

S3 credentials must be provided in a manner supported by `ybunload` and the AWS Java SDK:

- Secure methods (integrated into your organization's login/identity mechanism):
  - EC2 roles (when running on Amazon EC2 instances)
  - SAML 2.0-compatible identity provider
  - Custom identity provider bridge to AWS
- Other methods:
  - Object Storage Options
  - Environment variables: `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`. Other AWS environment variables are also supported, including `AWS_SESSION_TOKEN`, `AWS_PROFILE`, `AWS_REGION`, and `AWS_CREDENTIAL_PROFILES_FILE`.
  - URI query parameters: `aws_access_key_id` and `aws_secret_access_key`
  - A credential file, typically located at `~/.aws/credentials` (location may vary by platform)
- Required Permissions
  - Note that `ybunload` requires access to write the object.

An installation of the AWS CLI is not required, but it does provide the `aws configure` command, which is useful for setting credentials. For details, see the AWS Command Line Interface documentation.

Order of Precedence for Authentication Methods

You may have multiple credential settings available to you, based on your AWS account setup and how you script the `ybunload` command. Therefore, it is important to know which authentication mechanism takes precedence when the `ybunload` command is run. Note that credentials set explicitly with `ybunload` object storage options, if set (either on the command line or in a properties file), always take precedence over implicit credentials set via `aws configure` or other methods.

The order of precedence is as follows:

1. Access key and access key ID specified with the `--object-store-identity` and `--object-store-credential` command-line options
2. Access key and access key ID specified *with* the `yb.file.` prefix in one of the following:
   1. URI parameter
   2. Properties file named with the `--object-store-provider-config` command-line option
3. Access key and access key ID *without* the `yb.file.` prefix in one of the following:
   1. URI parameter
   2. Properties file named with the `--object-store-provider-config` command-line option
4. Implicit authentication via environment variables and supported AWS configuration files: `~/.aws/*` files: (`~/.aws/credentials` or `~/.aws/config`)

Unloading Data to an S3 Bucket

> ⚠ Deprecated Behavior:
>
> The method of unloading to S3 buckets shown below is deprecated, unless legacy mode is enabled.
>
> To continue using this approach, enable legacy mode with `--use-legacy-output-backend`. See ybunload options for details.
>
> The new method of unloading from S3 buckets uses `--object-storage-<option>`. For usage instructions and authentication options, refer to S3 Authentication Methods.

These instructions assume that you already have an Amazon Simple Storage Service (Amazon S3) account, know the S3 location of the files you want to load, and have your access keys ready at hand. If not, follow the instructions on the AWS site before proceeding with the information here. Specific prerequisites include:

- An Amazon S3 account with valid credentials. S3 credentials must be provided in a manner supported by the Amazon AWS Java SDK:
- Secure methods (integrated into your organization's login/identity mechanism):
  - EC2 roles (when running on Amazon EC2 instances)
  - SAML 2.0-compatible identity provider
  - Custom identity provider bridge to Amazon AWS
- Other methods:
  - Environment variables: `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`
  - URI query parameters: `aws_access_key_id` and `aws_secret_access_key`
  - A credential file, typically located at `~/.aws/credentials` (location may vary per platform)
- Required Permissions
  - Note that `ybload` requires access to read both the object and object metadata. Metadata access is a grantable permission separate from read access. Attempting to load a file from a public bucket with the option to download the file using an HTTP URL will cause `ybload` to fail and throw the following error:

    ```
    Forbidden (Service: Amazon S3; Status Code: 403; Error Code: 403 Forbidden; Request ID: xxxxxxxxxxxxxxx)
    ```

Your organization's AWS administrator should provide instructions for the mechanism you use. See Best Practices for Managing AWS Access Keys for further recommendations.

- An installation of the AWS CLI (required on Linux platforms and optional on Windows platforms). This CLI provides the `aws configure` command for setting credentials. For details, see the AWS Command Line Interface documentation.

Assuming that these prerequisites are in place, you can unload data from a table or a query to a folder in an S3 bucket.

To unload a table or query results to an Amazon S3 bucket:

1. Record your AWS security credentials (access key ID and secret access key).
2. Run `aws configure` or use `set` commands to set the environment variables for the AWS access keys.

For example, on Linux:

```
$ aws configure
AWS Access Key ID [****************SWUA]:
AWS Secret Access Key [****************n2L+]:
Default region name [None]:
Default output format [None]:
```

For example, on Windows:

```
set AWS_ACCESS_KEY_ID=<key value>
set AWS_SECRET_ACCESS_KEY=<key value>
```

3. Record the S3 location of the folder you want to use as the destination for the unload operation. Two formats are supported in the `ybunload` syntax:

- The complete HTTPS link. For example:

  ```
  https://s3-us-west-2.amazonaws.com/yb-tmp/premdb/premdb_unloads
  ```

- The abbreviated S3 path. For example:

  ```
  s3://yb-tmp/premdb/premdb_unloads
  ```

In this example, `yb-tmp` is the S3 bucket name, with folders below it.

4. Run the `ybunload` command in the usual way, providing the correct S3 path to the source file. For example:

```
$ ybunload -d premdb -t match --username bobr -W -o s3://yb-tmp/premdb/premdb_unloads
```

Here is the equivalent command with the longer-form HTTPS link to the S3 file:

```
$ ybunload -d premdb -t match --username bobr -W -o https://s3-us-west-2.amazonaws.com/yb-tmp/premdb/premdb_unloads
```

If you specify a log file, it must be on the local file system. Specifying a log file in an object storage location, such as an S3 bucket, is not supported.

If an S3 unload fails and is left in a hung state, you can use the `ybunload` command with the `--cancel-hung-uploads` option to cancel the upload. For example:

```
$ ybunload -d premdb -t newmatchstats --username bobr -W -o s3://yb-tmp/premdb/premdb_unloads
Database login password:
18:10:15.922 [ INFO] ybunload version 1.2.2-5686
18:10:15.924 [ INFO] COMMAND LINE = -d premdb -t newmatchstats --username bobr -W -o s3://yb-tmp/premdb/premdb_unloads
18:10:15.930 [ INFO] Creating S3 client with default credential search path
18:10:16.429 [ INFO] Verifying unload statement...
18:10:16.501 [ INFO] Unload statement verified
18:10:16.501 [ INFO] Beginning unload to s3://yb-tmp/premdb/premdb_unloads
18:10:18.890 [ INFO] Key Name: unload_1_1_.csv Upload ID = Z9FDA9SSSrBY_1wLK8RKYPDggkmOa.RGhmuY3.BY0kfCdfR._bWBMzK_7D.y33sQ5DYD1Xe
18:10:19.070 [ INFO] state: RUNNING - Network BW: 11.56 MB/s Disk BW: 0.00 KB/s
18:10:20.069 [ INFO] state: RUNNING - Network BW: 3.08 MB/s Disk BW: 5.69 MB/s
...
^C18:10:43.385 [ERROR] Caught ^C Forcing abort
18:10:43.386 [ WARN] Aborting multi part upload: Z9FDA9SSSrBY_1wLK8RKYPDggkmOa.RGhmuY3.BY0kfCdfR._bWBMzK_7D.y33sQ5DYD1Xe8I.Q3.JSW0
...
[1]+  Stopped                 ybunload -d premdb -t newmatchstats --username bobr -W -o s3://yb-tmp/premdb/premdb_unloads
$ ybunload -d premdb --username bobr -W -o s3://yb-tmp/premdb/premdb_unloads --cancel-hung-uploads
Database login password:
18:13:02.479 [ INFO] ybunload version 1.2.2-5686
18:13:02.481 [ INFO] COMMAND LINE = -d premdb --username bobr -W -o s3://yb-tmp/premdb/premdb_unloads --cancel-hung-uploads
```

```
18:13:02.487 [ INFO] Creating S3 client with default credential search path
18:13:02.998 [ INFO] Looking for uploads with prefix unload
18:13:03.633 [ INFO] Found 1 multi part uploads.
18:13:03.634 [ INFO] Cancelling unload_1_1_.csv +ID: Z9FDA9SSSrBY_1wLK8RKYPDggkmOa.RGhmuY3.BY0kfCdfR._bWBMzK_7D.y33sQ5DYD1Xe8I.Q3.
$
```

Unloading Data to Azure Storage

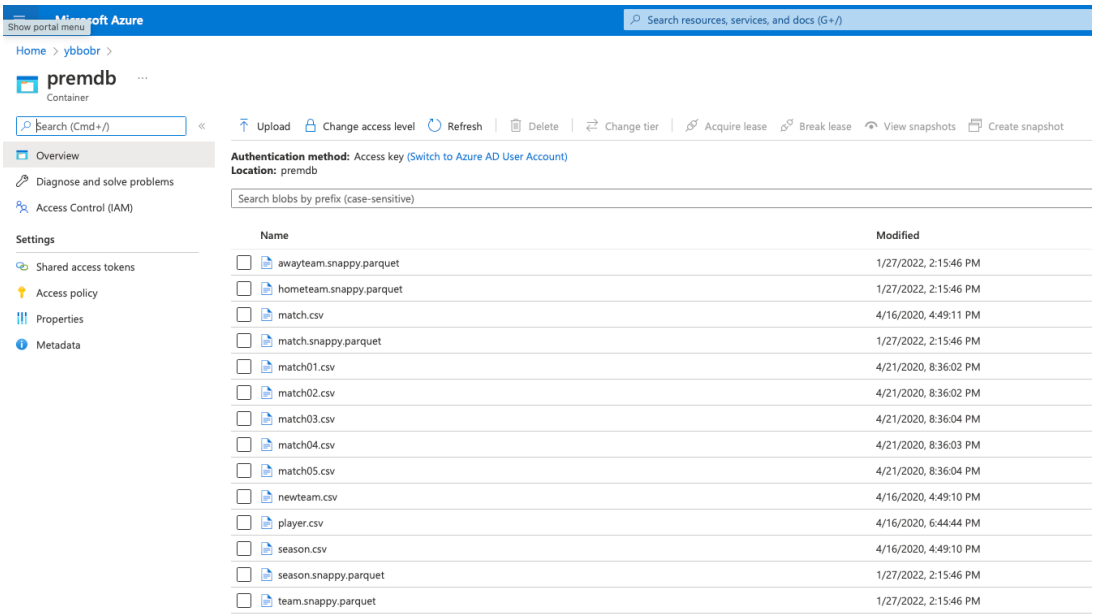
Platforms: All platforms

Parent topic: Unloading Data to Object Storage

These instructions assume that you already have a Microsoft Azure account, know the Azure location of the files you want to load, and have the appropriate credentials ready at hand. If not, follow the instructions on the Azure portal before proceeding with the information here.

These instructions apply to both Azure Blob Storage and Azure Data Lake Storage Gen 2.

For example:



Assuming that these prerequisites are in place, you can unload data from a table or a query to files stored in an Azure container.

To unload a table or the results of a query to an Azure container:

1. Use one of the documented Azure Authentication Methods.

For example, to authenticate via the `ybload` command-line options, you will need:

- A storage account name (your "identity"). Specifying the Azure endpoint (`https://<storage account>.blob.core.windows.net`) in addition to the account name is optional.
- An access key or a generated SAS token (your "credential") You can only access one object store per `ybunload` operation. For example, you cannot specify connection details for both an Azure client and an S3 client in the same command, and you cannot unload data from multiple Azure endpoints or storage accounts.

2. Identify the location in Azure where you want the unloaded files to be stored (`container/blob`).

At the end of the `ybunload` command line, you will specify the destination with the `-o` option and a URI, which is an abbreviated Azure path. For example:

```
-o azure://premdb/
```

3. Run the `ybunload` command in the usual way, but be sure to include:

- Azure identity (storage account name) and credential (access key or SAS token)

  These options are not needed if you are using the Azure CLI (`az login`) means of authentication. However, you will need to specify the `--object-store-endpoint` option.

  See Object Storage Options.

- `-o` option (destination directory)

  If you want the unloaded files to be named in a specific, recognizable way, use the `--prefix` option.

For example, the following command would unload the `match` table from the `premdb` database and store the unloaded files in parquet format in an Azure container:

```
$ ybunload -d premdb --username bobr -W -t match --format parquet
--object-store-credential '************************************'
--object-store-identity 'ybbobr'
-o azure://premdb/
```

Alternatively, you can save the identity and credential values to a Java properties file and name the file in the `ybunload` command. See Object Storage Options and Azure Blob Storage URIs.

Unloading Data to Google Storage

These instructions assume that you already have a Google Storage account, know the storage location of the files you want to load, and have the appropriate credentials ready at hand. If not, follow the instructions on the Google Storage portal before proceeding with the information here. Specific prerequisites include:

- A Google Storage account with valid credentials and appropriate permissions:
  - Credentials: You need to enable interoperability access in your Google Storage bucket and provide HMAC keys as `--object-storage-credential` and `--object-storage-identity`.
  - Permissions: The Google Storage account used for unloading to the bucket should have write permissions to the bucket.

Assuming that these prerequisites are in place, you can unload data from a table or a query to files stored in a Google Storage bucket.

To unload a table or the results of a query to a Google Storage bucket:

1. Use HMAC keys for authentication.

   You can only access one object store per `ybunload` operation. For example, you cannot specify connection details for both a Google Storage client and an S3 client in the same command, and you cannot unload data from multiple Google Storage endpoints or storage accounts.

2. Identify the bucket location in Google Storage where you want the unloaded files to be stored.

   At the end of the `ybunload` command line, you will specify the destination with the `-o` option and a URI, which is an abbreviated Google Storage path. For example:

   ```
   -o gs://premdb/
   ```

3. Run the `ybunload` command in the usual way, providing the correct bucket location.

   For example, the following command would unload the `match` table from the `premdb` database and store the unloaded files in parquet format in a Google Storage bucket:

   ```
   $ ybunload -d premdb --username bobr -W -t match --format parquet
   --object-store-credential '************************************'
   --object-store-identity 'ybbobr'
   --object-store-endpoint "https://storage.googleapis.com"
   -o gs://premdb/
   ```

   Alternatively, you can save the identity and credential values to a Java properties file and name the file in the `ybunload` command. See Object Storage Options.

   If you specify a log file, it must be on the local file system. Specifying a log file in an object storage location, such as an GS bucket, is not supported.

Unloading Data to Parquet Files

In this section:

This section explains how to unload data from Yellowbrick tables in Apache Parquet format (a binary, structured columnar storage format). Apache Parquet format is supported in `ybload` and `ybunload` operations. Certain options and parameters that work for flat files are not supported for `parquet` unloads, and a few options are specific to `parquet` unloads.

The standard set of Yellowbrick data types are automatically mapped to native and logical `parquet` data types; you do not need to specify any mapping. However, if you want to load (or reload) `parquet` data into a Yellowbrick table, you must make sure that the target column names in the table DDL match the names in the `parquet` schema. You can use `parquet-tools` to check the schema of `parquet` files before loading.

The following `ybunload` options are specific to Parquet unloads:

- `--format parquet`: required for all unloads to `parquet` files. This format must be set in the `ybunload` command.
- Some optional Parquet Processing Options. You may need to set or modify some of these options, depending on the specific requirements of the data you are unloading.

Note: If you are unloading data to Azure Blob Storage or Azure Data Lake Storage Gen2, it must be unloaded in `parquet` format.

Data Type Mapping for Parquet Unloads

Platforms: All platforms

Parent topic: Unloading Data to Parquet Files

The following table shows which `parquet` types are used when data from Yellowbrick tables is unloaded in `parquet` format. For more details about each type, go to the linked topic.

Note: You cannot modify the mapping of Yellowbrick types to `parquet` types when data is unloaded with `ybunload`. Only the following default mappings are supported. For example, even if the range of values in a `BIGINT` table column would fit a smaller target `parquet` data type, the target type will always be an `INT64` type, as shown in the table.

Yellowbrick Data Type	Mapped Data Type in Unloaded Parquet Files
BOOLEAN	Boolean
SMALLINT	`INT32` (16 bits, signed)
INTEGER	`INT32` (32 bits, signed)
BIGINT	`INT64` (64 bits, signed)
DECIMAL	`FIXED_LEN_BYTE_ARRAY` (ranging from 4 to 16 bytes)
REAL	Float
DOUBLE PRECISION	Double
UUID	`FIXED_LEN_BYTE_ARRAY` (16 bytes)
VARCHAR	String (Binary)
CHAR	
DATE	`INT32` (Date)
TIME	`INT64` (Time), adjusted to UTC, up to microseconds precision
TIMESTAMP	`INT64` (Timestamp), adjusted to UTC, up to microseconds precision
TIMESTAMP WITH TIME ZONE	
IPV4	String (Binary)
IPV6	
MACADDR	
MACADDR8	

Parquet Processing Options

This section covers `ybunload` options that are specific to unloads in `parquet` format. These options do not apply to unloads in `csv` and `text` format.

--opt-tmp-buffer-size NUMBER

Set the buffer size (in bytes) to be used for string operations such as transcoding. The default size is `3145728`, which is large enough, in most cases, to hold the maximum width of an unloaded row, given that the size of the row is multiplied for transcoding purposes. In some cases, the default buffer size may not be big enough. If `ybunload` returns a prompt to adjust this buffer, you can increase it; otherwise, use the default value. This option is similar to the `--pre-parse-buf-size` option in `ybload`.

--parquet-disable-dictionary-compression, --no-parquet-disable-dictionary-compression

Disable dictionary compression. Dictionary compression is enabled by default for parquet unloads.

--parquet-rowgroup-size NUMBER

Set the size, in bytes, of a parquet row group:

- Default: `52428800`
- Maximum size: `1073741824`
- Minimum size: `1048576`

A parquet row group is a segment of a parquet file (similar to a shard of data stored for a Yellowbrick table). A row group contains a set of rows and associated statistics, such as minimum and maximum values. You may need to limit the row group size for the following reasons:

- The row group size directly affects unload performance. Memory usage on the `ybunload` client will be proportionally higher with an increase in row group size.

  Skipping during reloads of unloaded parquet files will be limited if the range in maximum and minimum values for a row group is too large. For example, if all the rows for an unloaded table are written to the same row group in a parquet file, a subsequent load from that file cannot benefit from skipping. Other query planning optimizations, such as predicate pushdown, may be inhibited as well.

- `ybunload` may buffer up to one row group per blade. Increasing the row group size has a direct impact on `ybunload` memory usage, and for large systems, may cause `ybunload` to run out of memory.

--parquet-write-buffer-size NUMBER

Parquet write buffer size (in bytes). Default: 8192

ybunload Command

In this section:

Each `ybunload` command defines:

- A target table or a `SELECT` statement. All referenced tables must exist in the database, and the `ybunload` client user must have `SELECT` privileges on those tables.
- An absolute or relative path to a destination directory (local, S3, Azure)
- Database connection options
- Options for processing the unload, formatting the output, and compressing the data

---

Basic Syntax

```
ybunload { -t name | -s "query" } -o destination [options]
```

You can unload either all of the rows in a single table (`-t`) or the results of a valid query (`-s`). Only `SELECT` statements are supported with the `-s` or `--select` option.

You cannot unload the results of other SQL statements, such as `INSERT` or `CREATE TABLE AS` (`CTAS`).

Use `-?` or `--help` to return a list and description of the basic options:

```
$ ybunload --help
```

Saving Unload Options to a File

You can use the `@file` option to save a set of reusable settings to plug into a `ybunload` operation. You cannot use `@file` to save and re-run an entire `ybunload` specification.

Note: To save an unload query to a file, use the `--select-file` option.

To save and use options files with `@file`:

1. Determine the set of options that works to unload a particular data set.
2. Check that these options are appropriate by unloading the data once in the usual way.
3. Save the options to a text file in a location that is accessible to the `ybunload` client. For example:

```
$ more myoptions.txt
-d
premdb
--username
bobr
--format
text
-o
/home/ybdata/premdb_unloads
--prefix
match
```

Note: The name of an option and its value must be on separate lines in the file.

4. Run the `ybunload` command with the `@file` option:

```
$ ybunload @/home/ybdata/myoptions.txt -W --truncate-existing
Database login password:
16:24:16.209 [ INFO] ybunload version 5.4.0-20220228155915
16:24:16.211 [ INFO] COMMAND LINE = @/home/ybdata/myoptions.txt -W --truncate-existing
16:24:16.211 [ INFO] Verifying unload statement...
16:24:16.292 [ INFO] Unload statement verified
16:24:16.296 [ INFO] Beginning unload to /home/ybdata/premdb_unloads
16:24:17.090 [ INFO] Network I/O Complete. Waiting on file I/O
16:24:17.094 [ INFO] Finalizing...
16:24:17.096 [ INFO] Transfer complete
16:24:17.098 [ INFO] Transferred: 302.19 KB Avg Network BW: 21.09 MB/s Avg Disk write rate: 15.53 MB/s
```

ybunload Examples

The following `ybunload` examples do not show the `ABOUT CLIENT` messages that are routinely shown at the top of the output. These messages are logged mainly for troubleshooting purposes. For example:

```
13:50:28.174 [ INFO] ABOUT CLIENT:
   app.cli_args         = -d premdb -t newmatchstats --username yb100 -W --format text -o /home/yb100/premdb_unloads/nms --prefix
   app.name_and_version = ybunload version 2.0.0-9942
   java.home            = /usr/lib/jvm/java-8-oracle/jre
   java.version         = 1.8.0_101
   jvm.memory           = 512.00 MB (max=6.00 GB)
   jvm.name_and_version = Java HotSpot(TM) 64-Bit Server VM (build 25.101-b13, mixed mode)
   jvm.options          = -Xms512m, -Xmx6g, -XX:+UseG1GC, -Dapp.name=ybunload, -Dapp.pid=6295, -Dapp.repo=/usr/lib/ybtools/lib, -D
   jvm.vendor           = Oracle Corporation
   os.name_and_version  = Linux 4.4.0-31-generic (amd64)
```

Unload a table to a folder in an S3 bucket

Unload the `match` table to a folder inside an AWS S3 bucket. The bucket name is `yb-tmp`. See also Unloading Data to an S3 Bucket.

```
$ **ybunload -d premdb -t match --username bobr -W -o s3://yb-tmp/premdb/premdb\_unloads**
Database login password:
...
12:56:37.599 [ INFO] Verifying unload statement...
12:56:37.673 [ INFO] Unload statement verified
12:56:37.674 [ INFO] Beginning unload to s3://yb-tmp/premdb/premdb_unloads
12:56:38.352 [ INFO] Network I/O Complete. Waiting on file I/O
12:56:38.955 [ INFO] Key Name: unload_1_1_.csv Upload ID = SpQLEmQYU.4rbHB3NL7ZLXtdGN4Xs3ZGeWlc_F214HtI7FzBnWLsPZPH2UhVgOhJXKLDkna
12:56:39.357 [ INFO] state: FINALIZING - (Open Sockets:   0 - Open Files   0 Other Output Streams   1)
12:56:39.488 [ INFO] Finalizing...
12:56:39.489 [ INFO] Transfer complete
12:56:39.490 [ INFO] Transferred: 304.00 KB Avg Network BW: 24.75 MB/s Avg Disk write rate: 264.58 KB/s
```

Unload the results of a query to a text file

This example shows the query itself on the command line. An alternative approach is to use the `--select-file` option to call a file that contains the SQL statement. The `--select-file` option is recommended for longer, more complex queries.

```
$ **ybunload -d premdb --username yb100 -W --format text -o /home/brumsby/premdb\_unloads --truncate-existing --select
"select \*, \(total\_goals/380.00\)::dec\(3,2\) as goals\_per\_match
from \(
select season\_name, numteams,
sum\(substr\(ftscore,1,1\)::int\)+sum\(substr\(ftscore,3,1\)::int\) total\_goals
from season, match
where season.seasonid=match.seasonid and season.seasonid&gt;=4
group by season\_name,numteams
\) t1
order by 1;"
```

```
**
Database login password:
...
18:31:06.549 [ INFO] Verifying unload statement...
18:31:06.711 [ INFO] Unload statement verified
18:31:06.714 [ INFO] Beginning unload to /home/yb100/premdb_unloads
18:31:10.105 [ INFO] Network I/O Complete. Waiting on file I/O
18:31:10.204 [ INFO] Finalizing...
18:31:10.206 [ INFO] Transfer complete
18:31:10.207 [ INFO] Transferred: 429.00 B Avg Network BW: 40.64 KB/s Avg Disk write rate: 3.71 KB/s
```

Unload query results in parquet format

In this example, the results of the same query from the previous example are unloaded in `parquet` format to a local directory. The query text is passed in with the `--select-file` option and a prefix is used for the unload file.

```
$ ybunload -d premdb --username bobr -W --format parquet -o /home/brumsby/premdb_unloads --prefix goals_per_match --truncate-exist
Password for user bobr:
...
15:49:06.357 [ INFO] Removing existing files that match prefix=goals_per_match and extension=.parquet
15:49:06.368 [ INFO]     removing goals_per_match_1_0_.parquet
15:49:06.379 [ INFO] Verifying unload statement...
15:49:06.569 [ INFO] Unload statement verified
15:49:06.570 [ INFO] Beginning unload to /home/brumsby/premdb_unloads
15:49:06.587 [ INFO] Session Key = DgNzJrAIz_RIMXw4zhiKDkZf7TVkYNlAmOQFPYwuWUF8dbgbRaXcuBBlh0EUtwI=
15:49:07.670 [ INFO] Network I/O Complete. Waiting on file I/O
15:49:08.405 [ INFO] state: FINALIZING - (Open Sockets:   0 - Open Files   0 Other Output Streams   1)
15:49:08.511 [ INFO] Finalizing...
15:49:08.511 [ INFO] Transfer complete
15:49:08.512 [ INFO] Transferred: 4.00 B Avg Network BW: 2.39 KB/s Avg Disk write rate: 0.00 KB/s
```

The schema of the resulting file looks like this:

```
 % parquet-tools schema goals_per_match_1_0_.parquet
message schema {
  optional binary season_name (STRING);
  optional int32 numteams (INTEGER(16,true));
  optional int64 total_goals (INTEGER(64,true));
  optional fixed_len_byte_array(4) goals_per_match (DECIMAL(3,2));
}
```

The data in the file looks like this:

```
% parquet-tools cat goals_per_match_1_0_.parquet
season_name = 1995-1996
numteams = 20
total_goals = 988
goals_per_match = 2.60

season_name = 1996-1997
numteams = 20
total_goals = 970
goals_per_match = 2.55

...
```

See also Unloading Data to Azure Storage for another example of an unload in `parquet` format.

Unload a table to multiple 1GB files

Unload a table and set a maximum file size of 1GB per output file:

```
$ **ybunload -d newdb -t newmatchstats -W -o /home/yb100/premdb\_unloads --truncate-existing --max-file-size 1GB**
Database login password:
...
17:10:41.612 [ INFO] Verifying unload statement...
17:10:41.690 [ INFO] Unload statement verified
17:10:41.693 [ INFO] Truncating existing files that start with: unload
17:10:41.696 [ INFO] Beginning unload to /home/yb100/premdb_unloads
17:10:44.166 [ INFO] state: RUNNING - Network BW: 35.11 MB/s Disk BW: 66.22 MB/s
17:10:45.151 [ INFO] state: RUNNING - Network BW: 34.57 MB/s Disk BW: 67.50 MB/s
17:10:46.149 [ INFO] state: RUNNING - Network BW: 35.16 MB/s Disk BW: 66.36 MB/s
...
17:11:21.161 [ INFO] state: RUNNING - Network BW: 34.56 MB/s Disk BW: 65.70 MB/s
17:11:21.478 [ INFO] Network I/O Complete. Waiting on file I/O
17:11:21.479 [ INFO] Finalizing...
17:11:21.949 [ INFO] Transfer complete
17:11:21.949 [ INFO] Transferred: 2.46 GB Avg Network BW: 34.16 MB/s Avg Disk write rate: 64.86 MB/s
```

The resulting files look like this:

```
-rw-r--r--  1 yb100 users 999805680 May  4 17:10 unload_1_1_.csv
-rw-r--r--  1 yb100 users 999956727 May  4 17:11 unload_1_2_.csv
-rw-r--r--  1 yb100 users  11296574 May  4 17:11 unload_1_3_.csv
```

Unload a table in parallel, in GZIP format, using a prefix for the output file names

```
**$ ybunload -d premdb -t newmatchstats --compress gzip --parallel --prefix May16 -o /home/yb100/premdb\_unloads --username bobr -
Database login password:
...
12:40:40.542 [ INFO] Verifying unload statement...
12:40:40.612 [ INFO] Unload statement verified
12:40:40.615 [ INFO] Beginning unload to /home/yb100/premdb_unloads
12:40:41.409 [ INFO] Network I/O Complete. Waiting on file I/O
12:40:41.415 [ INFO] Finalizing...
12:40:41.415 [ INFO] Transfer complete
12:40:41.415 [ INFO] Transferred: 4.33 MB Avg Network BW: 41.27 MB/s Avg Disk write rate: 39.39 MB/s
```

The resulting files look like this:

```
-rw-r--r--  1 yb100 users   1117723 May 16 12:40 May16_1_1_.gz
-rw-r--r--  1 yb100 users   1179454 May 16 12:40 May16_2_1_.gz
-rw-r--r--  1 yb100 users   1226480 May 16 12:40 May16_3_1_.gz
-rw-r--r--  1 yb100 users   1019678 May 16 12:40 May16_4_1_.gz
```

ybunload Options

Yellowbrick Documentation > How-to Guides > Unload Tables > ybunload Command > ybunload Options

Platforms: All platforms

Parent topic: ybunload Command

This section contains detailed descriptions of the options you can use in a `ybunload` command. Note the following points about the format of these options and their values:

- Options are listed in alphabetical order for quick reference.
- Option names are shown in lowercase; *they are case-sensitive*.
- Specific valid option values (such as `true` and `false`) are shown in lowercase. Variables for option values, such as `STRING`, are shown in uppercase. Option values *are not case-sensitive*.
- The requirements for quoting option strings vary by client platform. Values are shown without quotes, but quotes are sometimes required. For example, if you specify the `#` character in a Linux shell, it must be enclosed by single or double quotes. If you are using a Windows client, see also Escaping Quotes in Windows Clients.

See also Common Options in ybtools and Parquet Processing Options.

--cancel-hung-uploads :

> ⚠ Deprecated Behavior:
>
> This option is deprecated, unless legacy mode is enabled. See option `--use-legacy-output-backend` for more details.

Attempt to cancel any multi-part uploads to S3 that were left in a hung state because of a previous failure. You must use this option independently; you cannot unload data and cancel uploads in the same `ybunload` command. In this case, the `-o` option specifies the destination for the uploads that were failing.

For example:

```
$ ybunload -d premdb --username bobr -W -o s3://yb-tmp/premdb/premdb_unloads --cancel-hung-uploads
```

Warning: Any other uploads to the specified bucket may be canceled if the files have the same prefix.

Note: Files that are uploaded to S3 are first written to temporary storage. After an interrupted unload, these files remain in this temporary location, but the `--cancel-hung-uploads` operation clears them. This operation does not clean up any files in the final unload destination (the actual path to the S3 bucket and its folders).

See also Unloading Data to an S3 Bucket and --max-file-size.

--column-prefix STRING, --parquet-column-prefix STRING

Supply a column prefix that overrides the default prefix (`_COL_`) used to generate and disambiguate column names in the schema of unloaded `csv` / `text` files with headers or `parquet` files. Column names in a parquet schema must be unique. For example, if a join query used in a `ybunload` command produces multiple columns with the same name, a prefix must be used to make them unique. When duplicate column names are found, the *second* column name and any additional duplicates are renamed with the prefix and a number (for example, `_COL_2` , `_COL_3`).

For example, in this case a simple cross-join of tables `t1` , `t2` , and `t3` produces three output columns named `c1` :

```
premdb=# select * from t1, t2, t3;
 c1 | c1 | c1
----+----+-----
  1 | 10 | 100
(1 row)
```

An unload of this row in `parquet` format with the default column prefix produces the following results:

```
% parquet-tools schema unload_2_0_.parquet
message schema {
  optional int32 c1 (INTEGER(32,true));
  optional int32 _COL_2 (INTEGER(32,true));
  optional int32 _COL_3 (INTEGER(32,true));
}
% parquet-tools head unload_2_0_4.parquet
c1 = 1
_COL_2 = 10
_COL_3 = 100
```

If you set `--parquet-column-prefix 'duplicate'` in the `ybunload` command, the columns will be named as follows:

```
% parquet-tools schema unload_2_0_.parquet
message schema {
  optional int32 c1 (INTEGER(32,true));
  optional int32 duplicate2 (INTEGER(32,true));
  optional int32 duplicate3 (INTEGER(32,true));
}
% parquet-tools head unload_2_0_.parquet
c1 = 1
duplicate2 = 10
duplicate3 = 100
```

--compress, -c NONE | GZIP_FAST | GZIP | GZIP_MORE | GZIP_BEST | GZIP_STREAM_FAST | GZIP_STREAM | GZIP_STREAM_MORE | GZIP_STREAM_BEST | SNAPPY | LZO | ZSTD

The default is `NONE`. The `GZIP_` options write data as GZIP compressed files in two different compression modes: "block mode" and "stream mode." The `GZIP*` options run in block mode, and the `GZIP_STREAM*` run in stream mode. See ybunload Output Files.

`GZIP` and `GZIP_FAST` are synonyms. `GZIP_MORE` provides better compression, but slower unload performance, and `GZIP_BEST` provides the best compression but much slower performance.

`GZIP_STREAM` and `GZIP_STREAM_FAST` are synonyms. `GZIP_STREAM_MORE` provides better compression, but slower unload performance, and `GZIP_STREAM_BEST` provides the best compression but much slower performance. The `GZIP_STREAM_*` options are intended to be used *only* if your downstream workflow tools cannot handle `gzip` files containing multiple compression blocks. Additionally, the `GZIP_STREAM_*` options consume significantly more network connections than the `GZIP*` options, meaning many network routers won't be able to handle the increased number of connections reliably.

The `SNAPPY`, `LZO`, and `ZSTD` options are only for use with `parquet` format unloads.

--delimiter STRING | UNICODE_CHARACTER

Define the field delimiter that will separate columns of data in output files. The default is a tab character (`'\t'`) in `text` format, and a comma in `csv` format.

Valid delimiters include special characters such as `'|'`, Unicode characters, and other supported escape sequences.

Note: The delimiter cannot be the null byte (`0x00`).

--disable-column-aliases, --no-disable-column-aliases

Do not retain column aliases that were specified in a SQL statement for a `ybunload` operation. Column names are automatically generated and used in place of any aliases to form the schema in the unloaded `csv` / `text` files with headers or `parquet` files. The default behavior is to retain column aliases.

For example, assume that the `ybunload` command specifies this query:

```
-s "select seasonid sid, matchday mday, ftscore ft, htscore ht from match order by 1"
```

If the `--disable-column-aliases` option is also specified, the aliases (`sid` , `mday` , `ft` , `ht`) in the query are not retained in the output files:

```
% parquet-tools schema DisableSelectAliases_2_0_.parquet
message schema {
  optional int32 seasonid (INTEGER(16,true));
  optional int64 matchday (TIMESTAMP(MICROS,true));
  optional binary ftscore (STRING);
  optional binary htscore (STRING);
}
```

If `--disable-column-aliases` is not specified (or `--no-disable-column-aliases` is specified), the aliases are retained:

```
% parquet-tools schema NoDisableSelectAliases_1_0_.parquet
message schema {
  optional int32 sid (INTEGER(16,true));
  optional int64 mday (TIMESTAMP(MICROS,true));
  optional binary ft (STRING);
  optional binary ht (STRING);
}
```

--disable-column-names, --no-disable-column-names

Do not retain column names that were specified in the DDL for a table that is being unloaded. Column names are automatically generated and used in place of the DDL column names to form the header row in the unloaded `csv` / `text` files with headers or `parquet` files. The default behavior is to retain column names.

For example, an unload of the `match` table with `--disable-column-names` specified will produce the following parquet schema:

```
% parquet-tools schema match_table_disablecols_1_0_.parquet
message schema {
  optional int32 _COL_1 (INTEGER(16,true));
  optional int64 _COL_2 (TIMESTAMP(MICROS,true));
  optional int32 _COL_3 (INTEGER(16,true));
  optional int32 _COL_4 (INTEGER(16,true));
  optional binary _COL_5 (STRING);
  optional binary _COL_6 (STRING);
```

The `_COL_*` column names replace the DDL column names:

```
premdb=# \d match
              Table "public.match"
  Column  |            Type             | Modifiers
----------+-----------------------------+-----------
 seasonid | smallint                    |
 matchday | timestamp without time zone |
 htid     | smallint                    |
 atid     | smallint                    |
 ftscore  | character(3)                |
 htscore  | character(3)                |
```

If the `--parquet-column-prefix` option is also specified, the column names are replaced accordingly. For example, `--parquet-column-prefix 'MatchCol'` produces:

```
% parquet-tools schema match_table_prefix_disablecols_1_0_.parquet
message schema {
  optional int32 MatchCol1 (INTEGER(16,true));
  optional int64 MatchCol2 (TIMESTAMP(MICROS,true));
```

```
    optional int32 MatchCol3 (INTEGER(16,true));
    optional int32 MatchCol4 (INTEGER(16,true));
    optional binary MatchCol5 (STRING);
    optional binary MatchCol6 (STRING);
  }
```

--force-quote 'column,column,...' | '*'

Specify a list of columns to quote in the output files, or specify `'*'` for all columns. This option is allowed only when you are using `csv` format. `NULL` values are not quoted, regardless of this option.

--format CSV | TEXT | PARQUET

Select the output format to use: `text` (tab-delimited by default), `csv` (comma-delimited by default), or `parquet`. The default format is `csv` and creates files with a `.csv` extension. Text format creates files with a `.txt` extension.

--header

Include a header row with column names for each output file, only applies to `csv` and `text` format.

--linesep LF | CR | CRLF | RS

Specify a row separator. The default is `LF`.

`RS`: The ASCII Record Separator (ASCII code `30`, Unicode `INFORMATIONAL SEPARATOR TWO`, hex `0x001e`).

--max-file-size

Specify the maximum size of a file that `ybunload` can export. For example: `10GB` or `1TB`. Do not use fractional values. The default maximum is 50GB. The minimum file size is 1GB.

Note: AWS S3 uploads are made up of a series of "parts." Each part must be >5MB, and no multi-part upload may exceed 10,000 parts. The default part size is 6MB, which means that *each file* that is uploaded to S3 cannot exceed 60GB. This restriction does not limit the total amount of data that `ybunload` can upload to S3, only the size of individual files. Increasing the default part size is not recommended.

--nullmarker

String to use as a null marker in the unloaded files. For example:

```
--nullmarker '[NULL]'
```

For `--format text` loads, the default null marker is `\N`.

For other formats, no default null marker is used. In the output, `NULL` values will appear as adjacent delimiters with no text between them.

-o

Name a local file directory where the unload directories and files will be placed. If you do not specify a location, your current working directory will be used. The default prefix for unload files is `unload`, but you can modify it by setting the `--prefix` option.

Note: If an output file with a given prefix already exists in the location where you are unloading data, `ybunload` returns an error. For example:

```
18:56:51.166 [ERROR] Unable to create output file: ./unload_1_1_.csv File already exists.
Remove existing file or use --truncate-existing
```

You can work around this error by removing the file manually, by using the `--truncate-existing` option, or by setting the `--prefix` option.

--object-storage-*

See Object Storage Options. These options apply to loads and unloads from and to Azure, AWS S3, and S3-compatible systems.

--parallel, --no-parallel

Enable parallel processing on all workers for the final sort operation that occurs when an unload query contains an ORDER BY clause. By default, the final ORDER BY sort runs on a single compute node. If the unload query does not have an explicit ORDER BY clause, this option has no effect.

When you use the `--parallel` option, files are unloaded in streams from each compute node, and the rows are guaranteed to be sorted within each file and within each stream. However, the complete set of files from all of the workers will not be unloaded in order.

If you intend to reload data that was unloaded with the `--parallel` option, it is recommended that you create the target table with a `SORT ON` column. The presence of a `SORT ON` column causes `ybload` source files to be loaded in fully sorted order. If you need to stitch the unload files together for loading or for use in other applications, you may need to write a script that checks the first and last lines of each unloaded file.

See also --select, -s and ybunload Output Files.

--prefix

Specify a prefix to attach to each unload file name. The default is `unload`. For example, if you use the prefix `04-30-18`, the files are numbered consecutively in the following format:

```
04-30-18_1_1_.csv
04-30-18_1_2_.csv
04-30-18_1_3_.csv
04-30-18_1_4_.csv
04-30-18_1_5_.csv
...
```

The convention for naming files is as follows:

```
<prefix_><streamID_><partnumber_>.<extension>
```

- `prefix_`: As defined, or `unload` by default.
- `stream id_`: A number assigned to each data stream from the workers. The streams are not in any particular order relative to a specific compute node, and a single compute node may provide multiple streams.
- `partnumber_`: An incrementing number starting from `1` for each stream.
- `.extension`: file type, such as `.csv` or `.gz`.

--quote UNICODE_CHARACTER

Specify the character to use for quoting.

--quote-escape UNICODE_CHARACTER

Specify the character to use for escaping quotes.

--select, -s

Run a SQL `SELECT` statement to unload data; any valid query is allowed. You cannot unload the results of other SQL statements, such as `INSERT` or `CREATE TABLE AS` (`CTAS`).

If any column in the `SELECT` list produces an `INTERVAL` data type, the unload operation will fail.

Note: If you want to unload data to a single file, use the `--select` option and include an `ORDER BY` clause in the query. Sorting the unloaded data in this way requires the entire result set of the query to fit into memory (or memory plus spill space).

--select-file, -f

Specify a file that contains the query you want to run in the `ybunload` operation (instead of entering the statement directly on the command line with the `--select` option). You can use this option and the `@file` option in the same `ybunload` command.

--stdout

Unload data to `stdout` instead of a destination directory. When you use this option, also specify `-q`, `--logfile`, and a `--logfile-log-level` value other than `OFF`.

--table, -t

This option supports `TABLENAME`, `SCHEMA.TABLENAME`, or `DATABASE.SCHEMA.TABLENAME` formats. If you do not specify the schema name, the table is assumed to be in the `public` schema. You do not have to quote table names that have mixed case. For example, all of the following table entries are valid for a table named `AwayTeam`:

```
-t public."AwayTeam"
-t 'public."AwayTeam"'
-t "AwayTeam"
-t AwayTeam
```

--truncate-existing

Remove all the existing output files in the destination directory that have the same prefix as the files being unloaded. You cannot use this option when unloading data to AWS S3 buckets.

--use-legacy-output-backend

Enables the legacy file I/O behavior for `ybunload`, as used in versions before 7.2.0.

⚠ Enabling this mode disables support for `--object-store-<option>`, which means `ybunload` will not be able to handle `gs`, `azure`, or `azdl2` URIs.

ybunload Output Files

Platforms: All platforms

Parent topic: ybunload Command

This section describes the output files that `ybunload` exports to the client.

Naming of Output Files

Use the `--prefix` option to give unique names to `ybunload` output files. If you do not use this option, files are named with the default `unload` prefix. When an unload operation produces multiple files, they are numbered consecutively. For example:

```
unload_1_1_.csv
unload_1_2_.csv
unload_1_3_.csv
...
```

The convention for naming files is as follows:

```
<prefix_><streamID_><partnumber_>.<extension>
```

- `prefix_`: As defined, or `unload` by default.
- `stream id_`: A number assigned to each data stream from the workers. The streams are not in any particular order relative to a specific compute node, and a single compute node may provide multiple streams.
- `partnumber_`: An incrementing number starting from `1` for each stream.
- `.extension`: file type, such as `.csv` or `.gz`.

Note: By default, unloaded `GZIP` compressed files do not have a file extension (such as `.csv` or `.txt`) when you decompress them. If necessary, you can use the `gunzip` command with the `-c` option to decompress and rename each file. For example:

```
$ gunzip -c unload_1_1_.gz > unload_1_1_.txt
```

Number and Size of Output Files

The number and size of files generated to complete the unload depends on the following factors:

- `max_file_size` setting for the `ybunload` command. If a file reaches this limit (by default, 50GB for regular file systems and 60GB for S3), `ybunload` closes that file and starts writing to a new file. This occurs as many times as necessary to complete the unload. The `max_file_size` value is not the maximum size of the unload; it is the maximum size of a single file generated by an unload stream from a single compute node. Each compute node unloads a separate data stream.
- The plan that is generated for the unload query and how many compute nodes have data at the top of the plan. A `ybunload` plan is the same as a `SELECT` query plan except that the top of the plan tree has a "data output" node.
- The compression (`--compress`) option that is chosen for the `ybunload` command.

Compressed and Uncompressed Files

Unload files are compressed in either "block mode" or "stream mode."

The `GZIP_*` compression options operate in block mode, which consolidates output files as much as possible, such that the number of files exported back to the client is no greater than it would be without the use of compression. The size of compressed compared to uncompressed files will differ, but the number is the same. Block mode compression (or no compression) results in one file per compute node that has data at the top of the plan tree. If a compute node is not used in the plan at all or produces no data at the top of the plan, it does not contribute a file. *Yellowbrick recommends the use of block mode GZIP options in most cases.*

The `GZIP_STREAM_*` options operate in stream mode. Stream mode compression results in potentially more files being exported: typically one file per compute node per core. The `GZIP_STREAM_*` options are intended to be used *only* if your downstream workflow tools cannot handle `gzip` files containing multiple compression blocks. Additionally, the `GZIP_STREAM_*` options consume significantly more network connections than the `GZIP*` options, meaning many network routers won't be able to handle the increased number of connections reliably.

Where possible, all workers and CPU cores are used for unload queries. Regardless of the compression option that you use, the following exceptions apply:

- Single-compute node queries: Only one compute node executes the plan, so only as many files as there are cores on that compute node are exported.
- Queries that select only part of a data set: If the unload query constrains a narrow segment of data (such as a date range or a set of low-cardinality values), only a subset of the workers may contain any data to be unloaded, resulting in fewer output files.
- Queries involving sorts that do not specify the `--parallel` option in the `ybunload` command. In this case, a final sort on a single compute node and a single core will yield only one output file. (To guarantee a single-file unload, make sure the unload query has an `ORDER BY` clause and do not specify the `--parallel` option in the command.)
- Queries involving aggregates. The final aggregation phase of the plan is done on one compute node, resulting in a reduced number of files, similar to the sort query case.

To summarize, in most cases the number of output files produced by an unload query will be one file per compute node that has data at the top of the plan. If you change the `max_file_size` setting, the unload may generate more files or fewer files. For example, if you are doing a 6TB unload (uncompressed) from a 15-node cluster with perfect data distribution, that would be ~410GB per blade. With a default `max_file_size` of 50GB, the unload would produce 9 files per blade for a total of 135 files. To reduce the number of output files for the unload to 15, you could specify a larger `max_file_size`, such as 500GB.

Specific Compression Options

Block mode:

- `GZIP` and `GZIP_FAST` are synonyms (for the fastest compression option).
- `GZIP_MORE` provides better compression, but slower unload performance
- `GZIP_BEST` provides the best compression but much slower performance.

Stream mode:

- `GZIP_STREAM` and `GZIP_STREAM_FAST` are synonyms (for the fastest compression option).
- `GZIP_STREAM_MORE` provides better compression, but slower unload performance
- `GZIP_STREAM_BEST` provides the best compression but much slower performance.

Setting Up an Unload

This section summarizes the steps required to unload data from a Yellowbrick database.

1. Make sure the target Yellowbrick database is up and running.
2. Determine the output destination and the format of the unloaded files.
3. Run the `ybunload` command as a user with `SELECT` privileges on the tables referenced in the command.

You must define a few essential pieces of information in the command. You can also specify a range of other processing, formatting, and logging options, as needed.

```
ybunload [options] -t table --format -o destination
```

- `[options]`: You will need to set some database connection values if they are not already set with environment variables. Other options depend on the nature of the data and related formatting requirements for the load.
- `table`: the target table for the unload. Alternatively, you can unload the results of a query, using the `-s` (select) option instead of `-t`.
- `--format`: `csv` (comma-delimited by default), `text` (tab-delimited by default), or `parquet` (binary)

  Note: Azure object storage only supports unloaded data in `parquet` format.
- `-o destination`: A local file system where the client user has write permissions or a supported object storage system where the client user has credentials to upload files. (See Unloading Data to Object Storage. You can also unload data to `stdout`.

  You can specify a prefix for the output files so that they are named consistently and uniquely for a given unload.

  Compressed GZIP (`.gz`) output files are supported.

For more details, see ybunload Options, Common Options in ybtools, and ybunload Examples.

Tip: If you are regularly unloading data into the same directory location with the same prefix, use the `--truncate-existing` option to remove files from previous runs. Otherwise, the output directory may contain a mix of unload files from different runs, or the unload may fail with an error.

4. Query the following system views to track unload progress and look at completed unload operations:

- sys.log_unload
- sys.unload

Client Tools and Drivers

Yellowbrick Documentation > How-to Guides > Work with Tools & Drivers

Platforms: All platforms

Parent topic: How-to Guides

In this section:

- Installing ybtools
- Setting Up a Database Connection
- Downloading Drivers
- Installing the Java Runtime Environment
- Opening Network Ports for Clients
- Uninstalling ybtools
- Using the ADO.NET Driver

Yellowbrick provides ODBC, JDBC, and ADO.NET drivers and a suite of client tools. For information about supported drivers, see Downloading Drivers.

The `ybtools` package consists of the following client tools:

- ybsql
- ybload
- ybunload
- ybbackup, ybrestore, and ybbackupctl
- ybdumpschema
- ybdumproles
- ybrelay

Supported Client Platforms

The client tools are supported on the following platforms:

- Ubuntu 18.04 LTS, 20.04 LTS, 22.04 LTS and 24.04 LTS
- RHEL and CentOS 7, 8, and 9
- AIX 7.2
- macOS 10.10 and later
- Windows Server 2012 and later, and Windows 10

Important: `ybrelay` is not supported on Windows clients.

OpenSSL and OpenLDAP Libraries

The `ybtools` package on Linux client platforms depends on:

- OpenSSL: `libssl 1.0`
- OpenLDAP: `libldap-2.4-2`

If necessary, you can install these libraries as follows:

- Ubuntu: `apt-get install openssl`

- RHEL and CentOS: `yum install openssl`

Client Downloads

You can download the appropriate `ybtools` package for your operating system either from Yellowbrick Manager or from the Yellowbrick Customer Support site. Check your Yellowbrick server version first so you can download the corresponding version of the tools. If you do not have access to either of these portals, contact your customer sucess manager.

To download `ybtools` from Yellowbrick Manager, navigate to *Support → Drivers and Tools*:

Note: The `Generic` package consists of the Java-based bulk load and backup tools. The other tools are platform-specific and cannot be part of this Generic package. This package may be useful if your client platform is not among those listed or is incompatible with those provided.

The tools can also now be downloaded as a docker container that you can spin up in your environment. For more information on installation and usage instructions, please refer to installing ybtools on docker.

Installing ybtools

Platforms: All platforms

Parent topic: Work with Tools & Drivers

In this section:

The following sections explain how to install `ybtools` on the following supported client platforms:

Platform	Installer File Name	Supported OS Versions
AIX 7	`ybtools-*.aix7.2.ppc.rpm`	AIX 7.2
EL7	`ybtools-*.el7.x86_64.rpm`	Red Hat and CentOs Enterprise Linux 7 (64-bit)
EL8	`ybtools-*.el8.x86_64.rpm`	Red Hat and CentOs Enterprise Linux 8 (64-bit)
EL9	`ybtools-*.el9.x86_64.rpm`	Red Hat and CentOs Enterprise Linux 9 (64-bit)
Ubuntu 18	`ybtools-*.ubuntu.18.04.x86_64.deb`	Ubuntu 18.04 LTS (64-bit)
Ubuntu 20	`ybtools-*.ubuntu.20.04.x86_64.deb`	Ubuntu 20.04 LTS (64-bit)
Ubuntu 22	`ybtools-*.ubuntu.22.04.x86_64.deb`	Ubuntu 22.04 LTS (64-bit)
Ubuntu 24	`ybtools-*.ubuntu.24.04.x86_64.deb`	Ubuntu 24.04 LTS (64-bit)
macOS	`ybtools-*.macos.x86_64.dmg`	macOS 10.10 and later
Windows	`ybtools-*.windows.x86_64.exe`	Windows Server 2012 and later, and Windows 10 (64-bit)
Generic	`ybtools-*.generic.noarch.tar.gz`	Other 64-bit UNIX-like environments, including Alpine Linux and Cygwin. Java tools only: `ybsql`, `ybdumpschema`, and `ybdumproles` are not included.

Installing the "generic" ybtools

You can install `ybtools` on certain platforms that do not have a dedicated operating-system package. The "generic" package may work for you if your client platform is not listed as one of the supported operating systems or is incompatible with those listed.

This package does not provide an installer; it is simply a zipped `ybtools` directory. Therefore, it does not update any environment variables (such as `PATH` or `LD_LIBRARY_PATH`). The individual `ybtools` scripts will update the `LD_LIBRARY_PATH` as needed when they are run.

1. Download the generic package from the Yellowbrick Manager. Go to **Support** > **Drivers and Tools** > **Client Tools**. For example: `ybtools-6.1.2-a8363b32.1234.generic.noarch.tar.gz`
2. If necessary, copy the downloaded package to the client system.
3. Decompress and extract the `generic.noarch.tar.gz` file to a directory of your choice. The result will be a top-level `ybtools` directory with some subdirectories. (For example, use the `gunzip` and `tar -xvf` commands.)

- If you want `ybtools` to be accessible to all users, Yellowbrick recommends using the `usr/bin/` directory. Then you can create soft links to the unzipped location for specific tools. For example:

```
% ln -s /usr/bin/ybtools/bin/ybload /usr/bin/ybload
```

- If `ybtools` is intended for your individual use, Yellowbrick recommends using your home directory. Then add the unzipped location of the `~/ybtools/bin` directory to your path in your `.profile` or `.bashrc` file. For example:

```
% export PATH=$PATH:~/ybtools/bin
```

- If you need to use multiple versions of `ybtools` under UNIX/Linux environments, you can use the generic package for the versions that you do not want to be the default for all users.

Installing ybtools on AIX

Follow these steps to install the client tools on an AIX client system:

1. Download the package for AIX from the Yellowbrick Manager. Go to **Support** > **Drivers and Tools** > **Client Tools**.
2. Yellowbrick client tools are installed by default at `/opt/ybtools` and require at least 200MB of free space. Ensure there is enough space on the `/opt` directory for the installation:

```
$ df -g
Filesystem    GB blocks      Free %Used    Iused %Iused Mounted on
/dev/hd4           0.09      0.06   36%     2420    15% /
/dev/hd2           2.06      0.23   89%    37171    40% /usr
/dev/hd9var        0.19      0.16   17%      950     3% /var
/dev/hd3           0.22      0.21    2%       33     1% /tmp
/dev/hd1           0.03      0.03    2%        7     1% /home
/dev/hd11admin      0.12      0.12    1%        5     1% /admin
/proc                 -         -    -         -      - /proc
/dev/hd10opt       0.41      0.03   92%    14452    63% /opt
/dev/livedump      0.25      0.25    1%        4     1% /var/adm/ras/livedump
```

If necessary, increase the amount of space on the `/opt` directory:

```
$ chfs -a size=4G /opt
Filesystem size changed to 8388608
```

```
$ df -g
Filesystem    GB blocks      Free %Used    Iused %Iused Mounted on
/dev/hd4           0.09      0.06   36%     2420    15% /
/dev/hd2           2.06      0.23   89%    37171    40% /usr
/dev/hd9var        0.19      0.16   17%      950     3% /var
/dev/hd3           0.22      0.21    2%       33     1% /tmp
/dev/hd1           0.03      0.03    2%        7     1% /home
/dev/hd11admin      0.12      0.12    1%        5     1% /admin
/proc                 -         -    -         -      - /proc
/dev/hd10opt       4.00      3.63   10%    14452     2% /opt
/dev/livedump      0.25      0.25    1%        4     1% /var/adm/ras/livedump
```

Note: On RHEL and AIX clients, you can install `ybtools` in a different preferred location. You do not have to use the default installation directory.

3. Check the `md5sum` of the RPM package:

```
$ csum -h MD5  ybtools-6.1.2-a8363b32.1234.aix7.1.ppc.rpm
ebb49ee3ed738d6471cc34bf8fdfd3ea  ybtools-6.1.2-a8363b32.1234.aix7.1.ppc.rpm
```

4. Install the `ybtools` package:

```
$ yum install ybtools-6.1.2-a8363b32.1234.aix7.1.ppc.rpm
```

where the package name is `ybtools-6.1.2-a8363b32.1234.aix7.1.ppc.rpm` in this example.

5. If you see the following error when you run `ybsql`, set the `LIBPATH` environment variable:

```
$ ybsql --help
Could not load program ybsql:
    Dependent module libldap.a(libldap-2.4.so.2) could not be loaded.
Could not load module libldap.a(libldap-2.4.so.2).
System error: No such file or directory
```

```
$ export LIBPATH="/opt/freeware/lib:/opt/freeware/lib64:/usr/lib64:/usr/lib:/opt/freeware/64/lib:/usr/linux/lib"
```

```
$ ybsql --help
ybsql is the Yellowbrick Database interactive terminal.
```

For example, the following installation was run on an AIX 7.1 client system:

```
$ yum install ybtools-6.1.2-a8363b32.1234.aix7.1.ppc.rpm
Failed to set locale, defaulting to C
AIX_Toolbox                                                                                 | 2.9 kB  00:00:0
AIX_Toolbox/primary_db                                                                      | 1.4 MB  00:00:0
AIX_Toolbox_71                                                                              | 2.9 kB  00:00:0
AIX_Toolbox_71/primary_db                                                                   |  23 kB  00:00:0
AIX_Toolbox_noarch                                                                          | 2.9 kB  00:00:0
AIX_Toolbox_noarch/primary_db                                                               |  62 kB  00:00:0
Setting up Install Process
Examining ybtools-6.1.2-a8363b32.1234.aix7.1.ppc.rpm: ybtools-6.1.2-a8363b32.1234.aix7.1.ppc
Marking ybtools-6.1.2-a8363b32.1234.aix7.1.ppc.rpm to be installed
Resolving Dependencies
--> Running transaction check
---> Package ybtools.ppc 0:6.1.2-a8363b32.1234 will be installed
--> Processing Dependency: libldap.a(libldap-2.4.so.2) for package: ybtools-6.1.2-a8363b32.1234.aix7.1.ppc
--> Running transaction check
---> Package openldap.ppc 0:2.4.48-1 will be installed
--> Processing Dependency: cyrus-sasl >= 2.1.26-2 for package: openldap-2.4.48-1.ppc
--> Processing Dependency: libgcc >= 6.3.0-1 for package: openldap-2.4.48-1.ppc
--> Processing Dependency: libsasl2.a for package: openldap-2.4.48-1.ppc
--> Processing Dependency: libgcc_s.a(shr.o) for package: openldap-2.4.48-1.ppc
--> Running transaction check
---> Package cyrus-sasl.ppc 0:2.1.26-3 will be installed
---> Package libgcc.ppc 0:6.3.0-2 will be installed
--> Finished Dependency Resolution
Dependencies Resolved
...                                                                                 4.9 MB/s | 6.0 MB     00:0
Running Transaction Check
Running Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing : libgcc-6.3.0-2.ppc                                                                  1/4
  Installing : cyrus-sasl-2.1.26-3.ppc                                                             2/4
  Installing : openldap-2.4.48-1.ppc                                                               3/4
Group "ldap" does not exist.
  Installing : ybtools-6.1.2-a8363b32.1234.aix7.1.ppc                                                      4/4
Installed:
  ybtools.ppc 0:6.1.2-a8363b32.1234
Dependency Installed:
```

```
  cyrus-sasl.ppc 0:2.1.26-3                libgcc.ppc 0:6.3.0-2                openldap.ppc 0:2.4.48-1
Complete!
```

Installing ybtools on Docker

To run ybtools using docker, ensure that Docker is installed on your client system. See Intalling Docker Desktop

Once Docker is installed and verified as running, you can use the following commands to run the tools container:

```bash
# Pull the stable tools image
docker pull yellowbrickdata/ybd-tools:stable

# Run the container in interactive mode, using bash as the entrypoint
docker run --network host -it yellowbrickdata/ybd-tools:stable bash

# Run ybsql and connect to your instance
ybsql -h <instance-url/localhost> -U ybdadmin -d yellowbrick -W
```

Installing ybtools on Linux

Make sure that Java 1.8 64-bit is installed on your client system. See Installing the Java Runtime Environment.

Follow these steps to install the client tools on an Ubuntu, Debian, or CentOS Linux client system:

1. Download the package for your client operating system from the Yellowbrick Manager. Go to **Support** > **Drivers and Tools** > **Client Tools**.
2. If necessary, copy the downloaded package to the client system.
3. From the client directory where the download file was copied, run the appropriate installation commands, which must be run as `sudo`:

- **Ubuntu 18.04 LTS**, **20.04 LTS**, **22.04 LTS**, **24.04 LTS**: Install the `ybtools` package:

  ```
  $ sudo apt-get install ./ybtools-6.1.12-7a4d6a43.3360.ubuntu.20.04.x86_64.deb
  ```

  where the package name is `ybtools-6.1.12-7a4d6a43.3360.ubuntu.20.04.x86_64.deb` in this example.

- **RHEL/CentOS 7.0**, **8.0**, **9.0**:

  1. Remove the old version of `ybtools`:

     ```
     $ sudo yum remove ybtools
     ```

  2. Install the new `ybtools` package:

     ```
     $ sudo yum install ybtools-6.1.2-a8363b32.1234.el7.x86_64.rpm
     ```

     where the package name is `ybtools-6.1.2-a8363b32.1234.el7.x86_64.rpm` in this RHEL 7.0 example.

For example, the following installation was run on an Ubuntu 16 client system:

```
$ sudo apt-get install ./ybtools-6.1.12-7a4d6a43.3360.ubuntu.20.04.x86_64.deb
Reading package lists... Done
Building dependency tree
Reading state information... Done
Note, selecting 'ybtools' instead of './ybtools-6.1.12-7a4d6a43.3360.ubuntu.20.04.x86_64.deb'
The following packages were automatically installed and are no longer required:
  bsh libasm3-java libganymed-ssh2-java libisl13 libjackrabbit-java libjline-java libjna-java libjna-jni libjna-platform-java libj
  libjsoncpp0v5 liblldb-3.8 libllvm3.8 libmaven-scm-java libnetbeans-cvsclient-java libruby2.2 libsisu-guice-java llvm-3.8 llvm-3.
  nagios-plugins-common python3-bcdoc ruby2.1-dev
Use 'sudo apt autoremove' to remove them.
The following packages will be upgraded:
  ybtools
1 upgraded, 0 newly installed, 0 to remove and 346 not upgraded.
Need to get 0 B/185 MB of archives.
After this operation, 0 B of additional disk space will be used.
Get:1 /home/brumsby/ybtools-6.1.12-7a4d6a43.3360.ubuntu.20.04.x86_64.deb ybtools amd64 6.1.12-7a4d6a43.3360 [185 MB]
```

```
(Reading database ... 229176 files and directories currently installed.)
Preparing to unpack .../ybtools-6.1.12-7a4d6a43.3360.ubuntu.20.04.x86_64.deb ...
Unpacking ybtools (6.1.12-7a4d6a43.3360) over (5.4.0-20220620182315) ...
Setting up ybtools (6.1.12-7a4d6a43.3360) ...
update-alternatives: setting up automatic selection of ybbackup
update-alternatives: using /opt/ybtools/bin/ybbackup to provide /usr/bin/ybbackup (ybbackup) in auto mode
update-alternatives: setting up automatic selection of ybbackupctl
update-alternatives: using /opt/ybtools/bin/ybbackupctl to provide /usr/bin/ybbackupctl (ybbackupctl) in auto mode
update-alternatives: setting up automatic selection of ybdumproles
update-alternatives: using /opt/ybtools/bin/ybdumproles to provide /usr/bin/ybdumproles (ybdumproles) in auto mode
update-alternatives: setting up automatic selection of ybdumpschema
update-alternatives: using /opt/ybtools/bin/ybdumpschema to provide /usr/bin/ybdumpschema (ybdumpschema) in auto mode
update-alternatives: setting up automatic selection of ybload
update-alternatives: using /opt/ybtools/bin/ybload to provide /usr/bin/ybload (ybload) in auto mode
update-alternatives: setting up automatic selection of ybrelay
update-alternatives: using /opt/ybtools/bin/ybrelay to provide /usr/bin/ybrelay (ybrelay) in auto mode
update-alternatives: setting up automatic selection of ybrestore
update-alternatives: using /opt/ybtools/bin/ybrestore to provide /usr/bin/ybrestore (ybrestore) in auto mode
update-alternatives: setting up automatic selection of ybsql
update-alternatives: using /opt/ybtools/bin/ybsql to provide /usr/bin/ybsql (ybsql) in auto mode
update-alternatives: setting up automatic selection of ybtoken
update-alternatives: using /opt/ybtools/bin/ybtoken to provide /usr/bin/ybtoken (ybtoken) in auto mode
update-alternatives: setting up automatic selection of ybunload
update-alternatives: using /opt/ybtools/bin/ybunload to provide /usr/bin/ybunload (ybunload) in auto mode
update-alternatives: setting up automatic selection of ybrelay-init
update-alternatives: using /opt/ybtools/config/ybrelay-init to provide /usr/bin/ybrelay-init (ybrelay-init) in auto mode
```

On Linux clients, the tools are installed by default in `/opt/ybtools`. For example:

```
$ cd /opt/ybtools
me@yb:/opt/ybtools$ ls
bin  config  integrations  lib  license.txt
```

Note: On RHEL and AIX clients, you can install `ybtools` in a different preferred location. You do not have to use the default installation directory.

If the locale on the client system is not compatible with the locale set for the database, you may see an error of the following form when you connect to a database with the `ybsql` client:

```
$ ybsql -d yellowbrick -h vm100.yb.io -U yellowbrick
Password for user yellowbrick:
ybsql: FATAL:  conversion between LATIN1 and LATIN9 is not supported
```

Make sure your client locale is consistent with the database locale. You can use the `locale` command to return client settings:

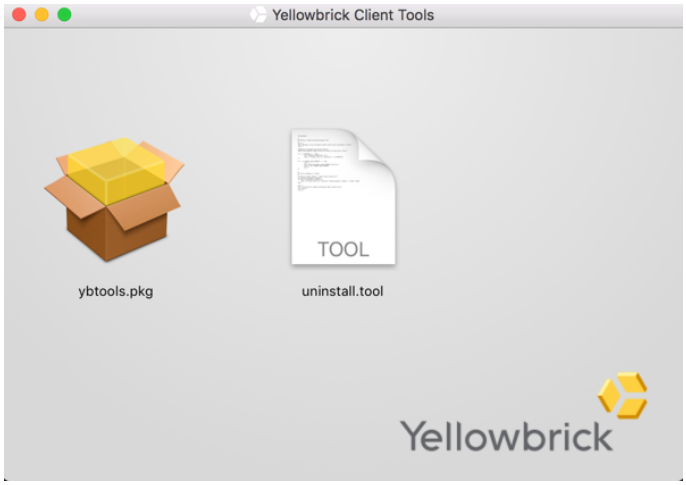
```
$ locale
LANG="en_US.UTF-8"
LC_COLLATE="en_US.UTF-8"
LC_CTYPE="en_US.UTF-8"
LC_MESSAGES="en_US.UTF-8"
...
```

To check which locales are installed on your client system, use the `locale -a` command.

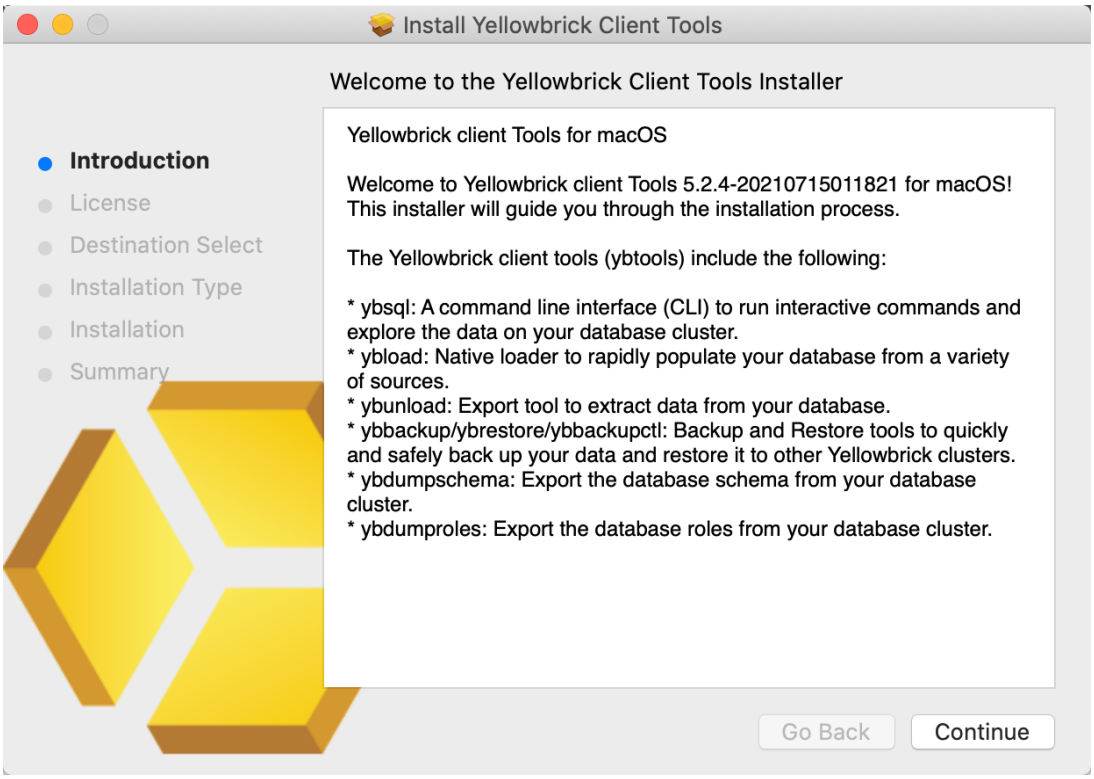
Installing ybtools on macOS

You can install `ybtools` on macOS versions 10.10 and later.

1. Download the macOS package from the Yellowbrick Manager. Go to **Support** > **Drivers and Tools** > **Client Tools**. For example: `ybtools-6.1.2-a8363b32.1234.macos.x86_64.dmg`.
2. If necessary, copy the downloaded package to the macOS client system.
3. Double-click the `.dmg` file. The Yellowbrick Client Tools installer opens.

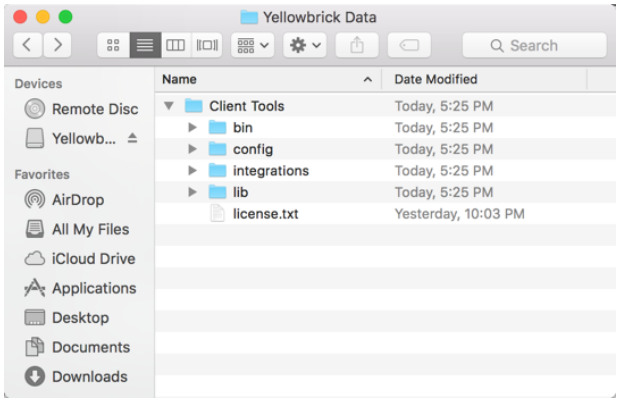


4. Double-click the `ybtools.pkg` icon and step through the installation program as directed. Be sure to accept the license agreement.



Note: You can change the default installation directory if required.

5. When the installation is complete, check the contents of the installation directory. For example:



Installing ybtools on Windows

Make sure that Java 1.8 64-bit is installed on your client system. See Installing the Java Runtime Environment.

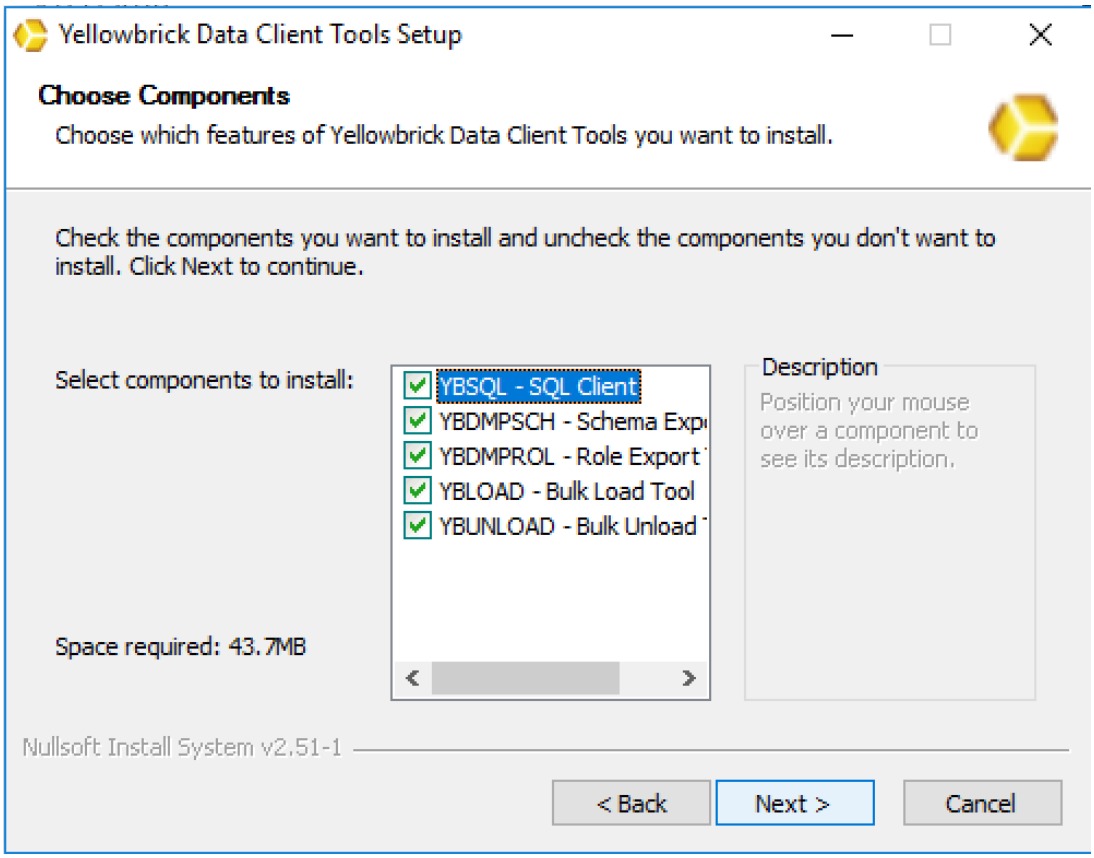
Follow these steps to install the client tools on a Windows client system:

1. Download the Windows package from the Yellowbrick Manager. Go to **Support** > **Drivers and Tools** > **Client Tools**.
2. If necessary, copy the downloaded package to the client system.
3. As an Administrator user, double-click the downloaded file to start the installation program (for example, `ybtools-6.1.2-a8363b32.1234.windows.x86_64.exe`).

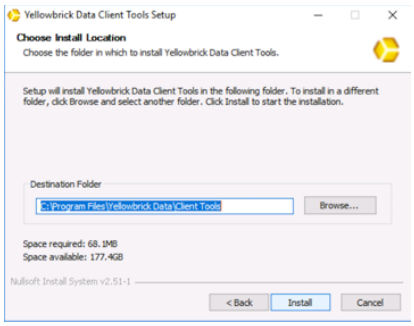
The Client Tools Setup Wizard starts up.

4. Click through the Setup Wizard and respond to all of the prompts. Note the following options:

You can select the tools to install:



You can choose an installation directory:



The default installation location is as follows:

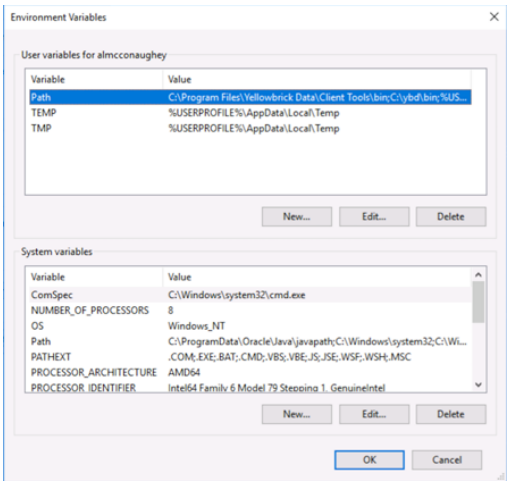
```
C:\Program Files\Yellowbrick Data\Client Tools
```

5. Optionally, set the `Path` variable to point to the installation directory.

6. Right-click the **Start** menu.

7. Select **System** > **Advanced System Settings** > **Environment Variables...**

8. Edit the Path variable and add the client tools location to the beginning of the string. In this example, the default location is added.



Changing the Console Code Page

After installing `ybtools`, run a simple ybsql command and check for the following `WARNING` message:

```
WARNING: Console code page (437) differs from Windows code page (1252)
        8-bit characters might not work correctly....
```

To change the console code page:

1. Each time you open a new command line console, set the code page by entering (for example):

```
chcp 1252
```

at the command prompt, where `1252` is the Windows Latin 1 (ANSI) code page. Set the code page for your environment, as indicated by the warning message. If you are using Cygwin, you can put this `chcp` command in `/etc/profile`.

2. Set the console font to `Lucida Console`; the raster font does not work with the ANSI code page.

JVM Settings on Windows Clients

Under normal circumstances, the default JVM settings for Yellowbrick client tools should be appropriate for most workloads. However, you may need to increase these settings because of memory requirements, for example. If so, here is an example you can follow for `ybload`:

```
@echo off

: Set base options for JVM
if not defined JVMOPT (
   set JVMOPT=-Xmx16g -Xms1g
)

: Override launch using JAVA_HOME/JAVACMD
if defined JAVA_HOME (
   set JAVACMD="%JAVA_HOME%\bin\java"
)
if defined JAVACMD (
   %JAVACMD% %JAVAOPT% %JVMOPT% -classpath "%~dp0ybload.exe%" io.yellowbrick.ybload.Main %*
) else (
   "%~dp0ybload.exe%" %*
)
```

Save these settings to a file such as `ybload.bat` and place the file next to `ybload.exe` in your `ybtools` installation.

Setting Up a Database Connection

Platforms: All platforms

Parent topic: Work with Tools & Drivers

In this section:

To run the client tools, you have to start a database session on the data warehouse instance where the target database, tables, and other objects reside. This session requires connection details and other login information that you can provide either as command-line options or as current values for environment variables.

Configuring SSL/TLS for Tools and Drivers

Platforms: All platforms

Parent topic: Setting Up a Database Connection

In this section:

This section explains how to configure secure connections, using SSL/TLS encryption, for both Yellowbrick tools and third-party tools that connect via ODBC and JDBC.

Secure Connections for ODBC/JDBC Clients and ybsql

This section covers SSL connection requirements and methods that are specific to the PostgreSQL ODBC and JDBC drivers and the `ybsql` client tool, which are based on the same `libpq` implementation. This section assumes that you are working with the PostgreSQL-developed ODBC and JDBC drivers. For drivers provided by other vendors, the concepts are the same but the specific implementation details, such as property names, are likely to be different.

The PostgreSQL drivers and `ybsql` use the PostgreSQL model of SSL modes to determine whether TLS/SSL is used when they connect to the server and, if so, how and under what restrictions. You can set the SSL mode with the operating-system environment variable `YBSSLMODE` or `PGSSLMODE`, or with an `sslmode` connection property, depending on how you are connecting.

If you choose an `SSLMODE` requiring verification, the certificate(s) of one or more trusted CAs must be located in `~/.yellowbrick/root.cert` on Linux clients or `%APPDATA%\yellowbrick\root.crt` on Windows clients. These modes are used to prevent being a victim of server identity spoofing.

SSL Mode for ODBC/JDBC Clients and ybsql

Important: SSL-only connectivity is enabled for all data warehouse instances. Therefore, all client connections must use SSL.

The default Yellowbrick SSL mode is `prefer`, which supports encryption of all network communications between the client and the server instance but does not check certificates to verify that the host system is trusted. If you want to use the default `prefer` mode, you do not need to specify an SSL mode when you log in to an instance from `ybsql`. The `allow` and `require` modes also attempt SSL connections and will fail if an SSL connection cannot be made. For example:

```
% export YBSSLMODE=require
% ybsql
Password:
ybsql (6.1.0)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)
Type:  \h for help with SQL commands
       \? for help with ybsql commands
       \g or terminate with semicolon to execute query
       \q to quit
...
```

If you want to use one of the SSL modes that verify trust in addition to encrypting network communication (`verify-ca` or `verify-full`), you may need to obtain and export the "chain of trust" from the Yellowbrick instance, make sure the certificates in the chain are valid, and place them in the correct location on the client system. A verified SSL connection checks the signature of the certificate to verify that it comes from a trusted certificate authority (CA), and that the hostname used in the connection matches the subject alternative name (SAN) in the certificate.

For details about obtaining certificates and troubleshooting SSL connections, see Obtaining a Certificate Chain.

If you don't set up the chain of trust correctly, you will see the following error:

```
% export YBSSLMODE=verify-ca
% ybsql
ybsql: root certificate file "*******/.yellowbrick/root.crt" does not exist
Either provide the file or change sslmode to disable server certificate verification.
```

SSL Modes

Whether you use the environment variable or a connection property, `YBSSLMODE` accepts the following values:

SSL Mode	Description
`allow`	Attempt a non-SSL connection. If it fails, attempt an SSL connection. *All non-SSL connections will fail in Version 6.x.*

`disable`	Only attempt a non-SSL connection. *Do not use this option; all non-SSL connections will fail in Version 6.x.*Because all client connections must use SSL, setting `YBSSLMODE=disable` always results in a failure to connect:
	`% export YBSSLMODE=disable`
	`% ybsql`
	`ybsql: FATAL: SSL connection is required`

`prefer`	Attempt an SSL connection, but do not verify trust. If the connection fails, attempt a non-SSL connection. `prefer` is the default option.
`require`	Only attempt an SSL connection. If a root CA file is present, verify the certificate as if `verify-ca` is specified.
`verify-ca`	Attempt an SSL connection and verify that the server certificate is issued by a trusted CA. Verify that the server is trustworthy by checking the certificate chain up to a trusted certificate authority (CA).
`verify-full`	Only attempt an SSL connection. Verify that the server certificate is issued by a trusted CA and that the server host name matches the name in the certificate. Verify that the server host name matches its certificate's Common Name (CN) attribute. If the CN attribute starts with an asterisk (*), it is a wildcard and will match all characters except a dot (.). This means the certificate will not match subdomains and will not match if an IP address is used instead of a host name. `verify-full` is recommended in most security-sensitive environments.

Creating a Client-Side root CA File

If you are using an SSL mode that verifies trust, you need to have a root CA file on your client host that contains the needed certificates. The file will typically be provided by your DBA or IT staff. If that is not the case, you can manually create the file yourself. Note that this file:

- Must contain the intermediary certificate(s) and root certificate for the server(s) you are connecting to.
- May contain intermediary and root certs for connecting to multiple different servers.
- Must be a text file that contains PEM certificates (base 64-encoded text).

You can explicitly include the root CA bundle file name as a connection property or simply use the default file name and path for your specific client operating system, as shown in the following table:

Operating System	Location	Notes
Linux, AIX, macOS	`~/.postgresql/root.cert` `~/.yellowbrick/root.cert`	Root certs are needed for `ybsql` but not for Java-based `ybtools`. Intermediary certs may be needed in both cases.

Operating System	Location	Notes
Windows	`%APPDATA%\postgresql\root.crt` `%APPDATA%\yellowbrick\root.crt`	

Yellowbrick and PostgreSQL diverge slightly in terms of SSL support. Yellowbrick does not support two-way trust; therefore, only the `root.cert` or `root.crt` files can be used when connecting to Yellowbrick.

Environment Variables and Connection Properties

You can set the defaults for most connection properties by using environment variables. You can also explicitly set them and override the defaults or environment variable settings as part of the connection. The following table lists the environment variables and their `libpq` connection property names.

Yellowbrick Environment Variable	PostgreSQL Environment Variable	Connection Property	Notes
`YBHOST`	`PGHOST`	`host`	Use a fully qualified host name if you are connecting with an SSL `verify` trust mode.
`YBHOSTADDR`	`PGHOSTADDR`	`hostaddr`	Do not use with an SSL `verify` trust mode.
`YBPORT`	`PGPORT`	`port`	If no port is specified, the default is `5432`.
`YBDATABASE`	`PGDATABASE`	`dbname`	
`YBUSER`	`PGUSER`	`user`	
`YBPASSWORD`	`PGPASSWORD`	`password`	
`YBAPPNAME`	`PGAPPNAME`	`application_name`	
`YBSSLMODE`	`PGSSLMODE`	`sslmode`	See SSL Modes for details.
`YBSSLROOTCERT`	`PGSSLROOTCERT`	`sslrootcert`	
	`PGREQUIRESSLMODE`	`requiressl`	Deprecated in favor of `YBSSLMODE/PGSSLMODE`.

Setting Environment Variables

The most common way to set the TLS/SSL connection options you want to use is through the environment variables. Using the environment variables will set the default values for all drivers and applications that use `libpq`. These values can then be overridden at the application or driver level when connecting. When these properties are evaluated, the order of precedence is as follows:

- Connection property (if set)
- YB* environment variable
- PG* environment variable

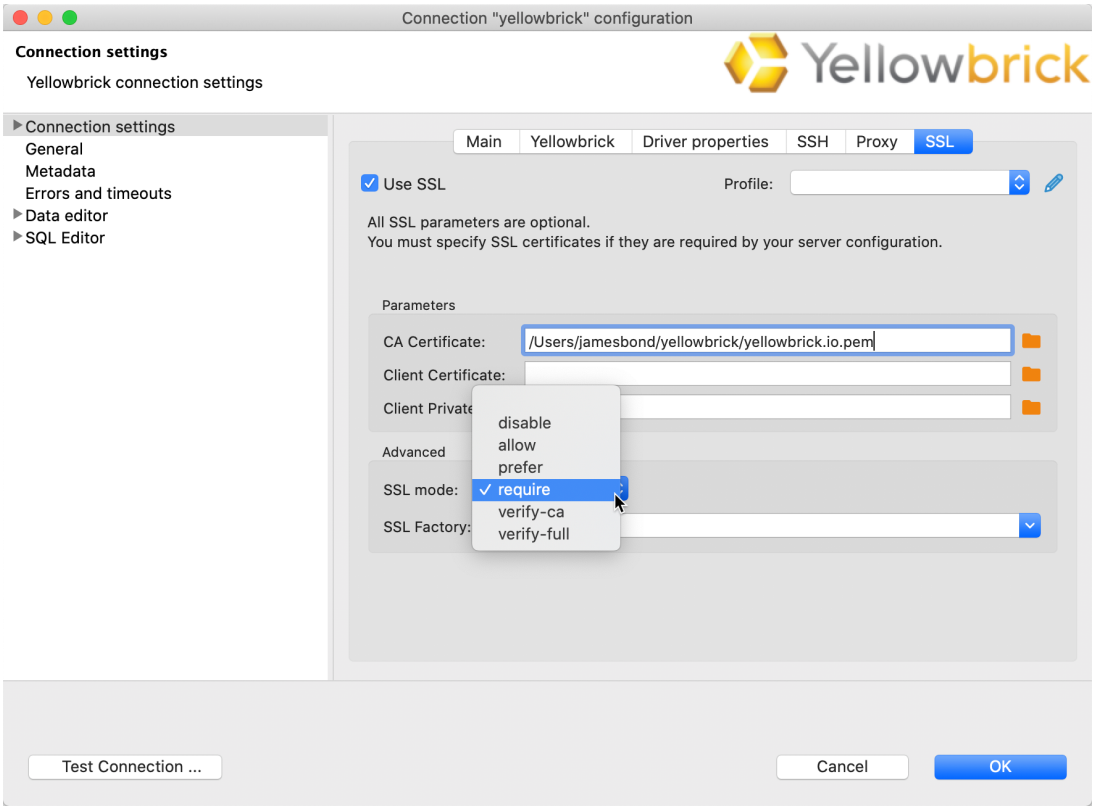
The command syntax for setting environment variables differs by operating system and command shell. For example, to set `YBSSLMODE` to `require`, use:

Operating System	Statement
Linux, AIX, and macOS	`export YBSSLMODE=require`
Windows CMD	`set YBSSLMODE=require`

Operating System	Statement
Windows PowerShell	`$env:YBSSLMODE = 'require'`

Secure Connections for DBeaver Clients

By default, the DBeaver client application does not enable SSL. You need to explicitly enable SSL in the Yellowbrick connection settings. The user interface for setting up an SSL configuration for DBeaver varies by version; however, the main steps are the same across all versions. The following example is for DBeaver Version 7.3 on macOS.



DBeaver uses the PostgreSQL-developed JDBC drivers for Yellowbrick connections. Therefore:

- DBeaver uses the same defaults as a JDBC connection.
- If you have not explicitly set the `YBSSLMODE` or `PGSSLMODE` environment variable and do not require trust verification, you do not need to set any properties.
- DBeaver uses the Java certificate keystore. This means that even if you set the SSL mode to `verify-ca` or `verify-full`, you may not need to provide a certificate file.
- The same two properties discussed in previous sections are the only properties you may need to set: the SSL mode and the root certificate, which only needs to be set if you are not using the default root certificate file.

Secure Connections for JDBC Clients

By default, the PostgreSQL JDBC driver uses the `YBSSLMODE` or `PGSSLMODE` environment variable to determine if SSL is going to be used. When SSL is used, the connection uses the root certificate file pointed to by the `YBSSLROOTCERT` environment variable, if specified. If this variable is not specified, the connection uses the default `root.cert` or `root.crt` file in the default location (if it exists).

Example: JDBC Connection String

In JDBC, you can set or override the values by specifying the properties in the URL. For example:

```
jdbc:postgresql://yb14.slc.yellowbrick.io:5432/yellowbrick?sslmode=require&sslrootcert=c:/ybd/star.slc.yellowbrick.io.pem&Applicat
```

These instructions pertain to the PostgreSQL JDBC driver. If you are using the `pgjdbc-ng` driver, see http://impossibl.github.io/pgjdbc-ng/docs/current/user-guide/#connection-ssl

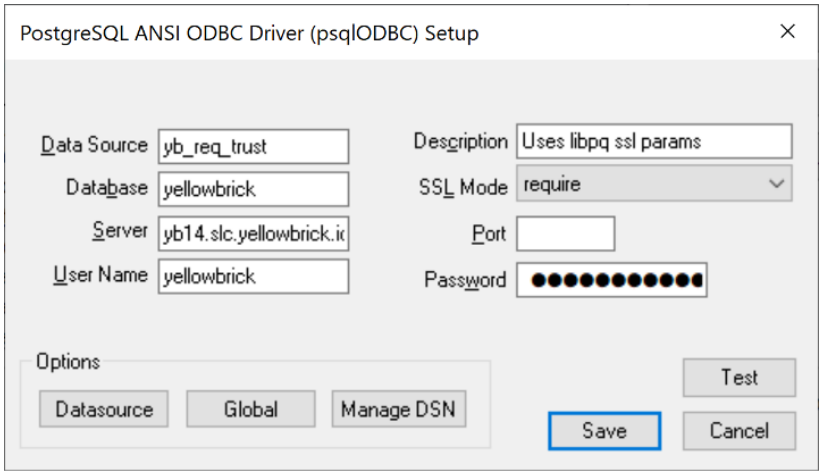
Secure Connections for ODBC Clients

By default, the PostgreSQL ODBC driver uses the `YBSSLMODE` or `PGSSLMODE` environment variable to determine if SSL is to be used. If SSL is going to be used, the connection uses the `root.crt` file in the default location (if it exists) or the `root.crt` file pointed to by the `YBSSLROOTCERT` environment variable.

If you are not using environment variables for SSL-related connection settings or want to override the values in use, you can set the connection properties in the Windows DSN GUI, in the `odbc.ini` file, or via "DNS-less" ODBC `libpq` connection properties (depending upon your connection type).

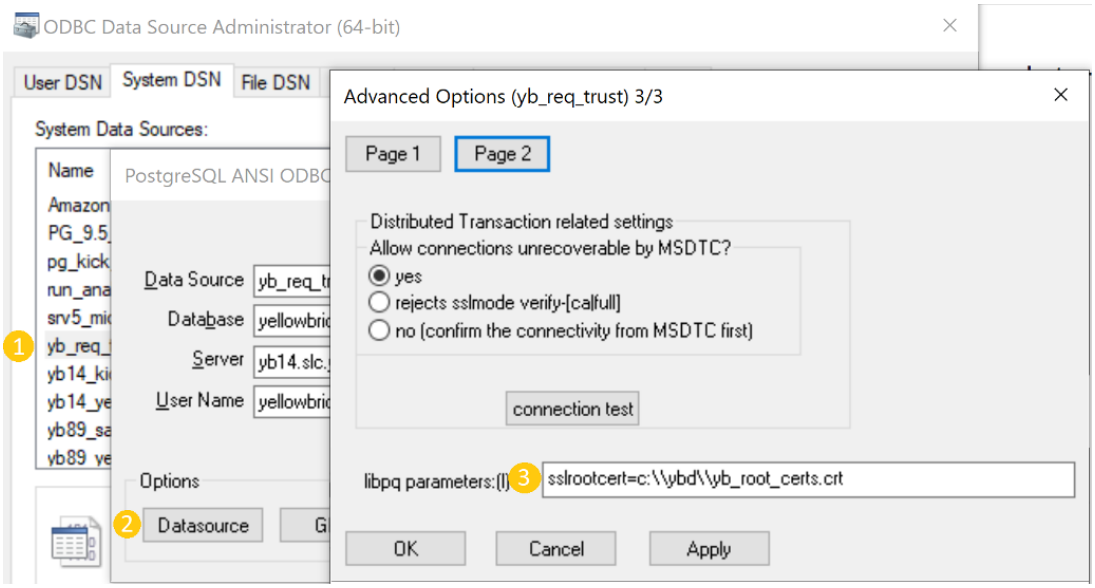
Example: Windows UI

You can specify the SSL mode to use in the ODBC driver setup:



You can specify an alternative `sslrootcert` by setting the `libpq` connection parameters with `conninfo` style strings in the ODBC driver configuration UI (Tab3 `libpq` parameters). For example:

```
sslrootcert=c:\\ybd\\yb_root_certs.crt
```



Note that the property **Allow connections unrecoverable by MSDTC?** is set to **yes**. Do not use the **connection test** option; you will get a misleading error. Instead, use the **Test** button in the general **Setup** dialog (as shown earlier).

Example: ODBC DSN-less Connection String

This example shows a DNS-less connection string with `require` SSL mode and a specific root certificate.

```
Driver={PostgreSQL ANSI};Server=IP address;Port=5432;Database=myDataBase;
Uid=myUsername;Pwd=myPassword;sslmode=require;sslrootcert=~/ybd/yb_root_certs.crt
```

Example: odbc.ini DSN

This example shows an `odbc.ini` DSN with `verify-ca` trust and a specific root certificate.

```
[YB14YellowbrickSSLWithRootCRT]
Driver = /usr/local/lib/psqlodbcw.so
Description = YB14
Database = yellowbrick
Servername = yb14
SSLMode = verify-ca
SSLRootCert = /home/my_user_name/ybd/yb14_root.crt
```

Secure Connections for ybsql (libpq)

Assuming you are using the default locations, you can test the basic connection by using a `ybsql` login command like this:

```
ybsql "sslmode=<ssl_mode> host=<hostname> dbname=<dbname> user=<username>"
```

For example, from a Linux command prompt:

```
 % ybsql "sslmode=prefer host=**********.yellowbrickcloud.com dbname=yellowbrick user=ybdadmin@yellowbrickcloud.com"
Password:
ybsql (6.1.0)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)
Type:  \h for help with SQL commands
       \? for help with ybsql commands
       \g or terminate with semicolon to execute query
       \q to quit
yellowbrick=>
```

You will be prompted for your password and a secure connection is attempted.

Example: ybsql on Windows (verify-ca, using a non-default root certificate)

You can precede the `libpq` connection string with options such as `-c` to execute a command. For example, this connection runs a query against the `sys.session` view and returns the secure details for the session:

```
ybsql -c "select secure_details from sys.session where process_id = pg_backend_pid()"
"sslmode=verify-ca sslrootcert=/ybd/star.nyc.yellowbrick.io.pem host=yb007.nyc.yellowbrick.io dbname=yellowbrick user=yellowbrick"
...
secure_details
-----------------------------------------------
TLSv1.2/ECDHE-RSA-AES256-GCM-SHA384/256 bits
(1 row)
```

Obtaining a Certificate Chain

If you require certificates to be included in your root certificate file and your DBA or IT staff has not provided them, you can often import them yourself by using a web browser or the `openssl` application. If you do this, remember that while it does no harm to include them, you do not need to include the leaf certificates in the file. You only need the intermediary certificates and, for `ybsql` connections, root certificates.

Import the chain of trust certs using a browser

Mozilla Firefox has the most convenient interface for importing certificates to a file because it has an option to generate the entire certificate chain with a single click.

1. Click the lock icon next to the URL for the Yellowbrick Manager login, then click the **Connection secure** expansion arrow.
2. Click **More Information** to go to the "Page Info."
3. Click **Security**, then **View Certificate** to see the certificates in the chain of trust.
4. Click the PEM (chain) link, which generates the certificate bundle for the entire chain of trust.

In other browsers, the steps are similar but you may have to repeat them for each certificate in the chain (unlike Mozilla Firefox, where you can generate the entire certificate chain at once).

Using an openssl Command to import a certificate

To show all certificate information, including the metadata, echo the results of an `openssl` command. For example:

```
$ echo -n | openssl s_client -connect ************.yellowbrickcloud.com:443 -showcerts
```

In place of `************.yellowbrickcloud.com`, substitute your Yellowbrick instance name, as copied from the Yellowbrick Manager Dashboard.

The following `openssl` command shows how to import the trusted certificate for an instance and save it to a `.pem` file:

```
% openssl s_client -servername ************.yellowbrickcloud.com -connect ************.yellowbrickcloud.com:443
</dev/null | sed -ne '/-BEGIN CERTIFICATE-/,/-END CERTIFICATE-/p' >cacert.pem
depth=2 C = US, O = Internet Security Research Group, CN = ISRG Root X1
verify return:1
depth=1 C = US, O = Let's Encrypt, CN = R3
verify return:1
depth=0 CN = ************.yellowbrickcloud.com
verify return:1
poll error%
```

Check the contents of the `.pem` file:

```
% more cacert.pem
-----BEGIN CERTIFICATE-----
MIIFbDCCBFSgAwIBAgISAz9hxAT2s+dCrx6C4bbW0NHpMA0GCSqGSIb3DQEBCwUA
```

```
MDIxCzAJBgNVBAYTAlVTMRYwFAYDVQQKEw1MZXQncyBFbmNyeXB0MQswCQYDVQQD
EwJSMzAeFw0yMjA4MDIyMDQ0MDNaFw0yMjEwMzEyMDQ0MDJaMD4xPDA6BgNVBAMT
M2JvYnItaW5zdDEtMDgwMjIyLjk4NzZyYjIyLmRldi55ZWxsb3dicmlja2Nsb3Vk
LmNvbTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBANJMz56R9JSDWODX
bx4qBX2cvr7yHI8RrGYtFathJ1dq+wXcF8TeR+vm2rto0ncoqfhXKV8WFc6PypmS
+ps0XrT7uEkUgW0/R4ZS5X9f5mfxNJ5lov6rKkxzJqc7+YuCi8Ydb59xt0sdcVzt
bekOpx4WtJFRUbjFj1V2JRpyOdPxlDQIAeWbUrAQBRBlgliXSWYmA8gk0sW0COkM
p04QSFQTerUsNuRUvH00nyS0l016SgBVWk16q1GGlnEd7uJpHvY1DJj7TrD0dcNQ
CK7Rnl1gtL9ZksZA/izCpKcuBMZVhpqnHRoGp00LdKpxy+EscQtq31lO8Fp0psv7
stNV9JECAwEAAaOCAm4wggJqMA4GA1UdDwEB/wQEAwIFoDAdBgNVHSUEFjAUBggr
BgEFBQcDAQYIKwYBBQUHAwIwDAYDVR0TAQH/BAIwADAdBgNVHQ4EFgQUOuOh0v+J
nAzohypaHsQiznpp/PowHwYDVR0jBBgwFoAUFC6zF7dYVsuuUAlA5h+vnYsUwsYw
VQYIKwYBBQUHAQEESTBHMCEGCCsGAQUFBzABhhVodHRwOi8vcjMuby5sZW5jci5v
cmcwIgYIKwYBBQUHMAKGFmh0dHA6Ly9yMy5pLmxlbmNyLm9yZy8wPgYDVR0RBDcw
NYIzYm9ici1pbnN0MS0wODAyMjIuOTg3NnJiMjIuZGV2LnllbGxvd2JyaWNrY2xv
dWQuY29tMEwGA1UdIARFMEMwCAYGZ4EMAQIBMDcGCysGAQQBgt8TAQEBMCgwJgYI
KwYBBQUHAgEWGmh0dHA6Ly9jcHMubGV0c2VuY3J5cHQub3JnMIIBBAYKKwYBBAHW
eQIEAgSB9QSB8gDwAHcA36Veq2iCTx9sre64X04+WurNohKkal6OOxLAIERcKnMA
AAGCYIQOYgAABAMASDBGAiEAsjbYmBOx9bDxWlmC2BOcghE9MNs+UsjsMUZQsDAU
IfICIQD5JfSX8IGr2aahOI5IFOdEf+l1pw9cBpjHt+JYbADNEAB1AEalVet1+pEg
MLWiiWn0830RLEF0vv1JuIWr8vxw/m1HAAABgmCEDqIAAAQDAEYwRAIgbbFIyfiv
gt9FxQvPNINUBoj4jV4+gwW5tCfsgtPkBHkCIApckKCAXUK0C/DWpgSDgEeMm2Jx
NMj6UBMxvC2xT7ztMA0GCSqGSIb3DQEBCwUAA4IBAQBUwQ9ZgjUKwMTHIZdGPLdX
Gf48W9u3ywlHPSvjaQtiWqtuN7EiO5LNIVHjei/fRjLYo7Jq8GX0ko/p7d43n4tJ
IflKPbAHcIyO/vQ9D6t7PTctpKQV+AMwY2vxRgHoHigNPopFxydzNL2ciUt58HXp
0rZIJyKKqyoJrTHFRGDeU/FI7WlpkTVIXtKwNxXYXJCGD34tnvz/G5WIjyvw4gjh
tyrqD9b5K7Dx33cAN2p4kUsls1FvDz/xvVMJJdYS/gsZIY9tLAAX08W89MZ966G9
JY+yIaemJynh7yJTlxTQWaI9eZURKeX9YmtCmA+m/FC3rQ0f7qC7GRb6q8/P0Z3y
-----END CERTIFICATE-----
```

Using an imported .pem file with the --cacert option in ybtools

In turn, you can pass an imported `.pem` file name in as the value for the `--cacert` option in a `ybtools` command. For example, this `ybload` command makes a secure connection:

```
% ybload -t premdb.match --format csv -W --secured --cacert cacert.pem /mydata/premdb/match.csv
Password for user trebor@yellowbrickcloud.com:
16:19:49.087 [ INFO] ABOUT CLIENT:
   app.cli_args         = YBHOST=<HOSTNAME> (from ENV_VAR) YBDATABASE=premdb (from ENV_VAR) YBUSER=<USERNAME> (from ENV_VAR) --sec
   app.name_and_version = ybload 6.1.0-a8363f32.1792
...
```

Secure Connections for Java-based ybtools

Connections to the following Java-based client tools support and require Secure Sockets Layer (SSL) and Transport Layer Security (TLS) encryption:

- `ybload`
- `ybunload`
- `ybbackup`, `ybrestore`
- `ybrelay`: for details, see Setting Up the ybrelay Service

Note: `ybsql` also supports and requires SSL/TLS but uses some different command-line options. See ybsql Connections.

Java-based clients provide the following security options:

- `--secured`: require SSL, which has the default SSLMODE of require root CA verification. When `--secured` is set, SSL/TLS encryption is used to secure all communication. *If this option is not set, secure connections are attempted anyway.*
- `--cacert`: supply a custom root CA bundle for trusting the certificate installed under Yellowbrick. Note that this is not a server certificate to be used in two-way trust.
- `--disable-trust`: do not require root certificate verification. `--disable-trust` is significant because it turns off the SSL/TLS root CA certification, not SSL/TLS. The bulk data tools require root CA certification by default. However, `ybsql` and many client tools do not require root CA certification.

See Opening Network Ports for Clients for a list of the port numbers that the client tools use for data control and data transfer.

When you are making a secure connection with Java-based `ybtools`:

- Remember that all connections to data warehouse instances must be secure SSL connections.
- By default, secure connections require trust verification. To disable trust verification, use the `--disable-trust` option within each Java-based tool.
- Because the Java-based tools use the Java certificate keystore, you will typically need to have a root certificate bundle file only for intermediary certificates issued by your organization.
- If you do need certificates included in a custom root certificate file, the order of precedence for the file to use is the option `--cacert`, the environment variable `YBCACERT` (recognized only by the Java tools), then the environment variables `YBSSLROOTCERT` and `PGSSLROOTCERT`.
- The `--cacert` option has an alternative syntax for the Java keystore format file that is not found in the drivers or `ybsql`. For Java keystore files for certificates, use `--cacert yellowbrick.jks:mypassword`, where the `:` character separates the file name from the password of the keystore.
- If you are using a Java application/JDBC driver, you can update the Java SSL trust keystore to avoid providing a root certificate bundle, but this approach is not generally recommended.

Summary of Options for Java-based ybtools

Property	Value / From	Notes
TLS/SSL port	`11112`	Regular bulk traffic port is `11111` Regular TLS database traffic port is `443`

Property	Value / From	Notes
Default TLS mode	Yellowbrick SSL/TLS mode	If Yellowbrick TLS/SSL only is not enabled, TLS is off by default.
Alternative TLS mode	`--secured`	Enable TLS/SSL even if not required.
Default trust mode	Verify CA mode	If TLS is enabled, CA verification is required by default.
	`--disable-trust`	
Default root certificate file	`root.cert` or `root.crt`	See Creating a Client-Side root CA File.

Alternative root certificate file	`--cacert`
	`YBCACERT`
	`YBSSLROOTCERT`
	`PGSSLROOTCERT`
		Alternative root certificate will be looked for in this order of preference.

To connect securely to an instance with `ybload` or other Java-based client tools, you use the `--secured` connection option. By default, loads are "promoted" to secure SSL connections even if `--secured` is not specified, and an SSL connection is always attempted. You will see this message:

```
14:25:12.107 [ INFO] Connection to **************.yellowbrickcloud.com cannot be made without SSL/TLS; attempting upgrade to secur
```

Explicit SSL connections take one of the following options (or combinations of options) in the command:

- `--secured` for encryption without explicit trust verification
- `--secured` and `--disable-trust` (or `-k`) for encryption without trust verification
- `--secured` and `--cacert` for encryption *and* explicit trust verification

For example, the syntax in all of the following commands is valid:

```
ybload -t premdb.match --format --secured csv /mydata/premdb/match.csv
ybload -t premdb.match --format --secured --disable-trust csv /mydata/premdb/match.csv
ybload -t premdb.match --format --secured --cacert /mydata/ybcloud.pem csv /mydata/premdb/match.csv
```

Note that the `--cacert` option requires the path to a local `.pem` file that contains the trusted certificate. If necessary, you can use an `openssl` command to import the certificate and save it to a file.

SSL/TLS Overview

Platforms: All platforms

Parent topic: Configuring SSL/TLS for Tools and Drivers

To use SSL/TLS for secure connections to Yellowbrick data warehouses, you need to complete the following tasks (and understand some related concepts):

- Decide the level of trust verification required.
- Obtain certificates if needed based on the required trust level.
- Set the defaults or explicit connection properties for the required level of trust and certificate locations.
- Use the proper server host name in your connections.
- Make sure the correct set of network ports is available for client connections.

SSL/TLS, Certificates, and Trust

Most problems encountered by users attempting SSL connections are a result of not having some basic understanding of TLS/SSL, certificates, and verification of trust.

SSL (Secure Sockets Layer) and TLS (Transport Layer Security) are protocols to encrypt and authenticate data transfer between applications. SSL was the original protocol that has since been replaced by TLS. SSL is often used to refer to a secure protocol although TLS is typically what is being used.

Using a TLS/ SSL connection provides two separate pieces of functionality:

- Encryption of traffic between your client application (`ybsql` , ODBC driver, `ybload` , and so on) and the server application (the Yellowbrick data warehouse)
- Verification that the server is who it claims to be

When a TLS/SSL connection is established, the data will always be encrypted, even on an untrusted connection. If a connection is trusted, this means that the client application has been able to verify, using certificates, that the server host and domain names match the names passed through in the connection. This verification is achieved through the SSL *chain of trust*, as discussed later in this section.

The options to enable or disable trust requirements differ among the drivers, `ybsql` , and the Java-based `ybtools` . If trust verification is required, the decision to verify trust is a *client-side* function. Yellowbrick instances are configured to allow TLS/SSL connections only, but the server cannot require the client to do trust verification.

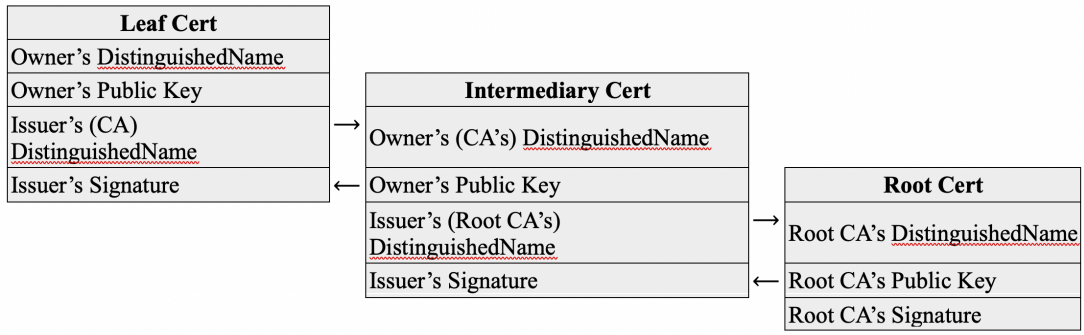
SSL Chain of Trust

A chain of trust is an ordered list of certificates, available on the client host:

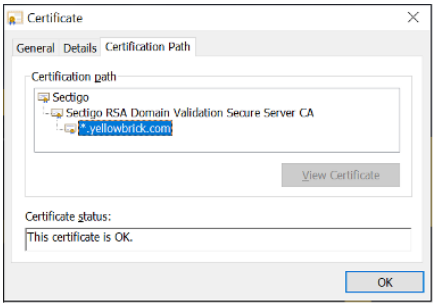
- An initial leaf certificate, which is the certificate installed on your Yellowbrick data warehouse
- Intermediate certificates, which are issued to your organization by a trusted Certificate Authority (CA)
- The root certificate of the CA, which is publicly available and typically included in the trusted certificate store for your operating system and Java installation.

This chain enables the client application to verify that the certificate is trustworthy; each certificate in the chain can be verified through its parent. In turn, server identity spoofing can be prevented.

The following diagram is a logical view of the chain of trust:



To see a specific chain of trust, you can use a web browser. The following example, using Microsoft Edge, shows a chain of three certificates:



- **Sectigo**: root certificate from the trusted Certificate Authority (CA)
- **Sectigo RSA Domain Validation Secure Server**: Intermediary certificate from the certificate signer that:
- Can be verified using the root certificate
- Issued the following leaf certificate
- ***.yellowbrick.com**: Leaf certificate, a wildcard certificate that applies to any host in the domain `yellowbrick.com` instead of a specific host name only

Self-signed certificates cannot be verified because they do not provide a chain of trust. The self-signed certificates that a Yellowbrick data warehouse includes are an example of this. Many organizations have policies requiring verification of the chain of trust when users connect to databases, web servers, and so on. Therefore, these organizations install their own trusted certificates on their Yellowbrick data warehouses.

The security policies of many companies do not allow the use of self-signed certificates or client settings that ignore certificate root verification for use with clusters storing confidential or personally identifiable information (PII). In these environments, the self-signed certificate must be replaced with a certificate from a trusted signing authority, using a PEM-encoded file or a Java KeyStore (JKS).

SSL/TLS Ports

Database client connections, bulk load and unload options, Yellowbrick Manager access, and other SSL/TLS communication with the Yellowbrick cluster go through specific ports. Additional network ports in the corporate firewall may need to be opened; otherwise, users will receive errors and communication will fail. For a list of port numbers used for data control and transfer, see Opening Network Ports for Clients.

Troubleshooting SSL Issues

This section identifies some common error conditions that may arise when you are setting up secure connections for Yellowbrick data warehouses.

"Server certificate...does not match hostname"

Most server certificates are based on host names. If you are using an SSL mode that requires trust, you need to know the correct host and domain name for the connection, not the IP address. Consider the following examples for a host named `yb14.yellowbrick.io`, using a wildcard certificate where `CN=*.slc.yellowbrick.io` and `verify-full` SSL mode. The host name used in the connection must be a name in the domain and cannot be an IP address:

```
ybqsl "sslmode=verify-full sslrootcert=/mnt/c/ybd/my.pem host=10.10.114.10 dbname=yellowbrick user=yellowbrick"
ybsql: server certificate for "*.slc.yellowbrick.io" (and 1 other name) does not match host name "10.10.114.10"
..
ybsql "sslmode=verify-full sslrootcert=/mnt/c/ybd/my.pem host=yb14.yellowbrick.io dbname=yellowbrick user=yellowbrick"
ybsql: server certificate for "*.slc.yellowbrick.io" (and 1 other name) does not match host name "yb14.yellowbrick.io"
```

"root.crt does not exist"

If you are using an SSL mode that requires trust verification, you may see a message like this one:

```
ybsql: root certificate file "/home/user_1/.postgresql/root.crt" does not exist
```

To fix the problem, either provide the file or change the SSL mode to disable server certificate verification. Verify that your `root.crt` file is in the correct location and that you have adequate permissions on it. In particular, remember that the location and names differ by operating system.

- Linux, AIX, macOS:

```
~/.postgresql/root.cert
```

```
~/.yellowbrick/root.cert
```

- Windows:

```
%APPDATA%\postgresql\root.crt
```

```
%APPDATA%\yellowbrick\root.crt
```

“SSL error: certificate verify failed”

This type of error occurs if you attempt to use an SSL mode that requires trust verification with an untrusted certificate, and you do not have the intermediary and/or root certificate in your root certificate file.

For example, a Yellowbrick data warehouse includes a self-signed certificate so it is not trusted. Attempting a connection using an SSL mode of `verify-ca` results in an error:

```
ybsql "sslmode=verify-full sslrootcert=/~/my_root.pem host=yb.my.com dbname=dev user=me"
ybsql: SSL error: certificate verify failed
```

Verifying SSL/TLS Encryption

Yellowbrick data warehouse instances support and enforce SSL/TLS encryption for client communications and passwords. All client connections from all tools, including connections to Yellowbick Manager, the front-end PostgreSQL database, and `ybtools`, must connect via HTTPS. All non-SSL connections are rejected.

Connection Prerequisites

TLS/SSL version 1.2 is required, as provided by `openssl` version 1.0.2 and later, which exists on almost all current operating systems. To check your version on Linux/UNIX, macOS, and Windows platforms, run the `openssl version` command. For example:

```
$ openssl version
OpenSSL 1.0.2g-fips  1 Mar 2016
```

```
% openssl version
LibreSSL 2.8.3
```

Verifying Secure Connections

Administrators can verify that clients are connecting securely by running a query against the sys.session view. For example, this query joins `sys.session` and `sys.user` to get the name of users and their secure connection details:

```
premdb=> select ss.session_id, ss.application_name, ss.user_id, su.name, ss.secure_details
from sys.user su, sys.session ss where su.user_id=ss.user_id and secure_connection = true;
 session_id | application_name | user_id |            name             |                secure_details
------------+------------------+---------+-----------------------------+-----------------------------------------------
      93192 | ybsql            |   16399 | trebor@yellowbrickcloud.com | TLSv1.2/ECDHE-RSA-AES256-GCM-SHA384/256 bits
(1 row)
```

Common Environment Variables in ybtools

Platforms: All platforms

Parent topic: Setting Up a Database Connection

The following environment variables may be set for Yellowbrick client tools to expedite and simplify database connections. See also ybsql Environment Variables.

Note that the following examples are for Linux platforms. On Windows platforms, omit the `export` keyword.

Environment Variable	Equivalent Options	Description	Example
`YBDATABASE`	`-d` or `--dbname`	Destination database name. Default: `yellowbrick` See also SQL Identifiers.	`--dbname premdb` `export YBDATABASE=premdb`
`YBHOST`	`-h` or `--host`	Destination server host name. Default: `localhost`	`-h test.mysandbox.dev.yellowbrickcloud.com` `export YBHOST=test.mysandbox.dev.yellowbrickcloud.com`
`YBCLUSTER`	`--cluster`	Virtual compute cluster to use.	`--cluster "small-default-cluster"` `export YBCLUSTER=small-default-cluster`
`YBPORT`	`-p` or `--port`	Destination server port number. Default: `5432`	`--port 5433` `export YBPORT=5433`

Environment Variable	Equivalent Options	Description	Example

`YBUSER`	`-U` or `--username`	Database login username. The default user is the current OS user on the client system.	`-U bobr@yellowbrickcloud.com`
			
			`export YBUSER=bobr@yellowbrickcloud.com`

`YBPASSWORD`	`-W` or `--password`	Interactive prompt for the database user's password. No default.	`--password`
			
			`export YBPASSWORD=********`

`YBAUTHISSUER`	`--auth-issuer`	URL of the OAuth/OIDC issuer.	`--auth-issuer https://cn.beta.yellowbrickcloud.com/auth/realms/mycompany.com`
			
			`export YBAUTHISSUER=https://cn.beta.yellowbrickcloud.com/auth/realms/mycompany.`

`YBAUTHTOKEN`	`--auth-token`	JWT string	`--auth-token ***********`
			
			`export YBAUTHTOKEN=***********`

`YBCLIENTID`	`--client-id`	Yellowbrick client ID (`ybauth`)	`--client-id ybauth`
			
			`export YBCLIENTID=ybauth`

Common Options in ybtools

Platforms: All platforms

Parent topic: Setting Up a Database Connection

The Java-based client tools, and to a certain extent, `ybsql`, share some common options and equivalent environment variables. These options define connectivity settings, logging behavior, and security credentials. For information about options that are specific to one client tool, see:

- ybsql Reference
- ybload Options
- ybunload Options
- ybbackup Options
- ybrestore Options

See also Object Storage Options.

General Options

-?, --help

Return basic usage information and command options.

--help-advanced

Return usage information for more advanced command options.

--java-version

Return the Java version that is running on the client system. The client tools require the 64-bit version of Java 8 (also known as Java 1.8). Java 9 and 10 are not supported.

--version

Display the version of each tool you are running (as part of `ybtools`). This option is not intended to be combined with other options. For example:

```
$ ./ybunload --version
ybunload version 6.1.12-7a4d6a43.3360
...
$ ./ybload --version
ybload version 6.1.12-7a4d6a43.3360
```

Connection and Authentication Options

Yellowbrick client tools support the Open Authorization (OAuth 2.0) and OpenID Connect (OIDC) standard for database connections.

--auth-browser

Authenticate the connection by opening a browser window on the client machine and entering a password. `-U username` is required when you use this option. You may need to make sure that your client system is configured to open a default browser. If a browser is not available (ie. restricted console, ssh), fallback to device login if the authorization endpoint supports it.

--auth-device

Authenticate the connection by device login flow, utilizing a browser URL. You may use this option on a restricted console, copy the URL given and code, and proceed to authenticate with a separate browser or device that has a browser.

--auth-direct

Authenticate the connection directly from the `ybtools` command (as opposed to opening a browser). The `-U username` option and `YBPASSWORD`, `--password`, or `-W` are required when you use direct authentication.

--auth-issuer

Specify the URL of the OAuth/OIDC issuer. This URL belongs to the identity provider (IDP) that issues JWT tokens for user connections. The IDP checks that the user exists in the specified realm, then returns a JWT token, which is a short-lived password for authentication for that user. Alternatively, you can set the `YBAUTHISSUER` environment variable.

This option must be specified for any kind of OAuth/OIDC connection.

For example:

```
--auth-issuer https://cn.beta.yellowbrickcloud.com/auth/realms/mycompany.com
```

where `cn.beta.yellowbrickcloud.com/auth` is the Yellowbrick IDP, which has some number of realms, such as `mycompany.com`.

--auth-scopes

Supply override for scopes requested when authenticating. Some IDPs such as Azure AD require this to be specified as <client_id>/.default, which returns an access token that can be validated by an external resource owner, such as Yellowbrick Database. Alternatively, you can set the `YBAUTHSCOPES` environment variable.

--auth-token AUTH_TOKEN

Supply a JWT string in the command line explicitly. Alternatively, you can set the `YBAUTHTOKEN` environment variable.

--client-id

Specify the Yellowbrick Data Warehouse client ID, which is `ybauth`. Alternatively, you can set the `YBCLIENTID` environment variable.

This option is required for OAuth/OIDC authentication.

--cluster

Name of the virtual compute cluster to use.

--dbname, -d

Name of the database used for the connection. The default is `yellowbrick`.

--host

Host name.

--initial-connection-timeout NUMBER

Number of seconds to wait for initial connections to the database. The default is `120`. This timeout option ensures that an operation does not wait too long when there is a basic problem with incorrect connection parameters, or a firewall is preventing connection errors from reaching the client. To allow an unlimited wait time, set this option to `0`.

-p, --port

Specify the database server port. Alternatively, set this value with the `YBPORT` environment variable. Default: `5432`

-U, --username

Specify the database user who is running the command. Alternatively, set this value with the `YBUSER` environment variable.

If this option is not specified, the default user is the current OS user on the client system. Users who connect to tools such as `ybload` via Kerberos SSO, for example, can do so directly without specifying a username or a password.

Note: The `ybload` user must have `INSERT` privileges on the target table but does not have to own the table.

-W, --password

Interactively prompt for the database user's password. Do not enter a password on the command line if you use this option. For non-interactive password entry, set the `YBPASSWORD` environment variable.

Logging Options

--log-level OFF | ERROR | WARN | INFO | DEBUG | TRACE

Specify the logging level for the default console output. The default level is `INFO`. (Use the `--logfile-log-level` option to specify the logging level for a named log file.)

--logfile STRING

Specify the name and location of a log file. If the specified file already exists, it will be truncated. If this option is not specified, no log file is written. When you specify this `--logfile` option, also specify a `--logfile-log-level` value other than `OFF`.

Note: When object storage is used for loading or unloading data, logs must be written to the local file system. Specifying a log file in an object storage location, such as an S3 bucket, is not supported.

--logfile-log-level OFF | ERROR | WARN | INFO | DEBUG | TRACE

Specify the logging level for a given log file (as defined with the `--logfile` option). If the level is not specified, it defaults to the `--log-level` value. You must specify a `--logfile-log-level` value other than `OFF` when you specify the `--logfile` option.

-q, --quiet

Do not write any output to the screen. This option is suitable for `cron` invocations. If this option is specified, you must also specify `--logfile` and `--logfile-log-level`.

Security Options

--cacert STRING

Customize trust with secured communication; use this option in combination with the `--secured` option. Enter the file name of a custom PEM-encoded certificate or the file name and password for a Java KeyStore (JKS).

For PEM format, the file must be named with a `.pem`, `.cert`, `.cer`, `.crt`, or `.key` extension. For example:

```
--cacert cacert.pem
```

For JKS format, files are always password-protected. Use the following format:

```
--cacert yellowbrick.jks:changeit
```

where the `:` character separates the file name from the password of the keystore.

See also Verifying SSL/TLS Encryption and

Secure Connections for Java-based ybtools.

--disable-trust, -k

Disable SSL/TLS trust when using secured communications. Trust is enabled by default. See also Verifying SSL/TLS Encryption.

Important: This option is not supported for use on production systems and is only recommended for testing purposes. It may be useful to disable trust during testing, then enable it when a formal signed certificate is installed on the cluster.

--secured

Use SSL/TLS to secure all communications. By default, loads are "promoted" to secure SSL connections even if `--secured` is not specified, and an SSL connection is always attempted.

Examples of OAuth 2.0/OIDC Connections

The examples in this section show how to use `ybtools` options that support OAuth/OIDC connectivity.

Example with --auth-direct

The following `ybload` command connects to a specific Yellowbrick instance by using the `--auth-direct` option with `--username` and `-W`:

```
$ ./ybload -h myybdinstance.elb.us-east-1.amazonaws.com -d premdb -t premdb.match --format csv
--auth-issuer https://cdwm.beta.yellowbrickcloud.com/auth/realms/my_realm.com --auth-direct --username me.myself@myrealm.com -W
--client-id ybauth /home/premdata/match.csv
Password for user me.myself@myrealm.com:
...
```

Example with --auth-browser

In this example, a browser window opens and requests the user's password. A `Login Successful` message is displayed when authentication succeeds.

```
$ ./ybload -h myybdinstance.elb.us-east-1.amazonaws.com -d premdb -t premdb.match --format csv
--auth-issuer https://cdwm.beta.yellowbrickcloud.com/auth/realms/my_realm.com --auth-browser --username me.myself@myrealm.com
--client-id ybauth /home/premdata/match.csv
...
```

Object Storage Options

The following options are used in `ybload`, `ybunload`, `ybbackup`, `ybrestore`, and `ybbackupctl` commands. These options specify connectivity, credentials, and configuration settings for object storage systems, including Azure Blob Storage and S3 object stores. The usage for these options depends on the object store provider.

For more details about supported object storage authentication methods and URIs, see Loading from Amazon S3 and Loading from Azure Blob Storage.

--object-store-anonymous, --no-object-store-anonymous

Connect to the object store anonymously, instead of providing a credential. For example, you can connect anonymously when you are accessing an AWS S3 bucket that is publicly available. You cannot specify `--object-store-anonymous` and credential values in the same command.

--object-store-credential STRING

The credential to use when connecting to an object store.

For AWS S3, this credential corresponds to your AWS Secret Access Key.

For Azure, this credential is a storage account access key or a generated SAS token.

--object-store-endpoint VALUE

The endpoint URI to use when connecting to an object store.

For AWS S3, if you specify the `--object-store-region` option, you can omit the `--object-store-endpoint` option. An example of an AWS S3 endpoint is:

```
https://s3.us-east-2.amazonaws.com
```

An example of an on-premises S3-compatible endpoint is:

```
http://ybload1.yellowbrick.io:9000
```

where the host name is a system where the S3 provider is installed. (The port number varies by S3 provider.)

For Azure, if you specify the `--object-store-identity` option, you can omit the `--object-store-endpoint` option. An example of an Azure endpoint is:

```
https://ybbobr.blob.core.windows.net
```

where `ybbobr` is the storage account name.

You can only access one object store per operation. For example, you cannot specify connection details for both an Azure client and an S3 client in the same command, and you cannot load or unload data using multiple Azure endpoints or storage accounts.

--object-store-identity STRING

The identity to use when connecting to an object store.

For AWS S3, this identity corresponds to your AWS Access Key ID. For example:

```
ABCD1EF2GHIJKLMN3OPQ
```

For Azure, this identity is a storage account name. If you specify this option, you can omit the `--object-store-endpoint` option because the storage account name is always part of the standard endpoint pattern (`https://<storage account>.blob.core.windows.net`). For example:

```
https://ybbobr.blob.core.windows.net
```

where `ybbobr` is the storage account.

--object-store-profile STRING

The credential profile to use when connecting to an object store. This option overrides the settings for `--object-store-identity` and `--object-store-credential` .

This option is not supported for Azure object stores.

--object-store-provider-config JAVA_PROPERTIES_FILE

The name of a Java properties file that contains provider-specific configuration settings for an object store. For example: `ybbobr_az.properties` or `ybbobr_s3.properties` . Default: `{}` .

--object-store-region STRING

The region to use when connecting to an object store.

For AWS S3, an example of a region is `us-east-2` .

For on-premises, S3-compatible object storage providers, a region value must be specified even if the provider does not support or require regions. You can specify any string for the region, such as `noregion` or `dummy` . Alternatively, you can set the region in an AWS configuration file:

```
~/.aws/config

[default]
region = dummy
```

The region option is not supported for Azure Blob storage.

Java Properties File

If you choose to specify object storage options in a properties file (`--object-store-provider-config` option), drop the `--object-store-` prefix from the option names. You can either add a `yb.object_store.` prefix to the entries in the file, or you can specify the entries without a prefix.

Do not quote any of the values in the file. For example, specify `us-east-2` , not `'us-east-2'.`

AWS S3

For example:

```
$ more ybbobrS3.properties

yb.object_store.region = us-east-2
yb.object_store.identity = ABCD1EF2GHIJKLMN3OPQ
yb.object_store.credential = ****************************************
```

```
```

Alternatively, you can specify:

```
region = us-east-2
identity = ABCD1EF2GHIJKLMN3OPQ
credential = ****************************************
```

This properties file is distinct from the `~/.aws/*` configuration files that the S3 SDK consults.

Azure Blob Storage, Azure Data Lake Gen 2

```
endpoint = https://mystorageaccount.blob.core.windows.net
identity = mystorageaccount
credential = myStorageAccountKey
```

For example:

```
$ more ybbobrAz.properties

yb.object_store.endpoint = https://ybbobr.blob.core.windows.net
yb.object_store.identity = ybbobr
yb.object_store.credential = ************************************
```

S3 Pass-Through Options

The following pass-through options are supported for S3 clients. You can specify them either in a Java properties file (referenced by `--object-store-provider-config`) or as URI parameters in the *first* S3 URI passed to the `ybload` command. (The defaults listed here are as defined by the AWS Java SDK 2.7.31.)

accelerateModeEnabled
: Defaults to `false`.

checksumValidationEnabled
: Defaults to `true`.

chunkedEncodingEnabled
: Defaults to `true`.

dualStackEnabled
: Defaults to `false`.

Downloading Drivers

You can connect ODBC, JDBC, and ADO.NET applications to Yellowbrick databases using drivers downloaded from some or all of the following sources (depending on the driver type and client operating system):

- Yellowbrick System Manager (under Cloud Native 7.x)
- Yellowbrick System Management Console (SMC) (under Yellowbrick appliances)
- Your Linux or MacOS package manager from their default or PostgreSQL repository
- Directly downloading from the PostgreSQL site

Note that installing the ODBC or ADO.NET drivers typically requires Administrator privileges.

ADO.NET:

Connecting to a Yellowbrick database from a .NET application requires an ADO .NET driver for PostgreSQL. Npgsql is the PostgreSQL developed ADO.NET Data Provider. Yellowbrick supports the 7.x and 8.x driver versions.

These Npgsql drivers are installed using the NuGet client tools. NuGet is the central package repository for .NET authors and consumers.

For further information, including connection strings, see:

- Using the ADO.NET Driver in this documentation section
- Npgsql Getting Started Guide
- Npgsql home

JDBC:

The most recent supported PostgreSQL JDBC driver for Java 8 and later is 42.7. The same driver is used across all platforms and no installer is needed. Download the driver jar file from the Yellowbrick SMC or System Manager, or directly using the following link postgresql-42.7.4.jar.

ODBC:

The currently supported ODBC drivers for Yellowbrick are the PostgreSQL developed version 12 drivers. Many enterprise ETL and BI tools ship with DataDirect ODBC drivers for PostgreSQL which can also be used.

Linux and MacOS clients require unixODBC to already be installed. For general instructions on using ODBC on Linux and macOS platforms, go to unixODBC.

ODBC for Windows:

You can download PostgreSQL ODBC drivers for Windows from the Yellowbrick SMC or System Manager, or directly using the following links:

- 64-bit MSI: https://ftp.postgresql.org/pub/odbc/versions.old/msi/psqlodbc_12_02_0000-x64.zip
- 32-bit MSI: https://ftp.postgresql.org/pub/odbc/versions.old/msi/psqlodbc_12_02_0000-x86.zip

- Combined Installer (64-bit and 32-bit): https://ftp.postgresql.org/pub/odbc/versions.old/msi/psqlodbc_12_02_0000.zip

ODBC for RHEL/CentOS Linux

The currently supported EL 7 and 8 PostgreSQL driver version is 12.x. The EL 9 recommended driver version is 13.x. You can obtain the drivers from your EL distribution, your package manager (using the PostgreSQL EPEL repository), the Yellowbrick SMC or System Manager, or download the RPM directly using the links provided below.

- EL 7: https://ftp.postgresql.org/pub/repos/yum/12/redhat/rhel-7.9-x86_64/postgresql12-odbc-12.01.0000-1PGDG.rhel7.x86_64.rpm
- EL 8: https://ftp.postgresql.org/pub/repos/yum/12/redhat/rhel-8.12-x86_64/postgresql12-odbc-12.01.0000-1PGDG.rhel8.x86_64.rpm
- EL 9: https://ftp.postgresql.org/pub/repos/yum/12/redhat/rhel-9.12-x86_64/postgresql12-odbc-13.02.0000-2PGDG.rhel9.x86_64.rpm

The recomended method is to use your package manager with the EPEL repository; it ensures the additional required packages are installed.

Note: Driver version 13 is the oldest PG 12 compatible driver under EL 9.

Ubuntu/Debian Linux

The Ubuntu PostgreSQL drivers can be installed from your distribution or the PostgreSQL APT repository using the package manager, or directly from PostgreSQL (based on your Ubuntu version). If downloading directly from PostgreSQL, the top-level PostgreSQL ODBC driver URL is: https://ftp.postgresql.org/pub/repos/apt/dists/. The recomended method is to use your package manager with the EPEL repository; it ensures the additional required packages are installed.

ODBC for macOS

There are many ways to install the PostgreSQL ODBC drivers on macOS. The recommended method is by using Homebrew with the following command:

```bash
brew install unixodbc psqlodbc
```

Installing the Java Runtime Environment

From version 7.3 and above, the following client tools require the 64-bit version of Java 11 or Java 17:

- `ybbackup`
- `ybbackupctl`
- `ybrestore`
- `ybload`
- `ybunload`
- `ybrelay`

Client tools versions 6.7 and above required 64-bit version of Java 11.

Client tools versions 6.6 or earlier required Java 8, (also known as Java 1.8).

Java 9 and 10 are not supported. Only the JRE (Java Runtime Environment) is required, but you can install the complete JDK if you prefer. The JRE is required on both Linux and Windows platforms.

Follow these steps to check that the correct version of Java is installed and install it or upgrade it if necessary. (You can install the client tools if Java is not found, or if the wrong Java version is found, but you cannot run them.)

1. Check the Java version on your client system by using the `java -version` command. For example, on an Ubuntu Linux client:

```
me@yb.io:~$ java --version
openjdk 11.0.20.1 2023-08-24
OpenJDK Runtime Environment (build 11.0.20.1+1-post-Ubuntu-0ubuntu122.04)
OpenJDK 64-Bit Server VM (build 11.0.20.1+1-post-Ubuntu-0ubuntu122.04, mixed mode, sharing)
```

The build numbers may vary; what you are looking for is `11` as first element in the version string and a reference to `64-Bit`. In this example, the correct version of Java is already installed. You would be able to proceed with the client tools installation.

2. If your client system does not have the JRE installed (or has the wrong version), go to the Java SE site and download Java 11.
3. Follow the Oracle installation instructions for your client platform.
4. Check the Java version again when the installation is complete.

Locale compatibility

Java has introduced locale changes as outlined in JEP 252. When transitioning from Java 8 to Java 11, which was implemented with the release of **ybtools 6.7**, it's important to note that certain locale behaviors may have changed.

If you have never used ybtools in a version before 6.7, these changes do not affect you.

However, for users of ybtools versions 6.6 or earlier, particularly those employing non-English locales, there may be potential issues with data formats no longer being accepted. An illustrative example is found in the Greek language, where the morning/afternoon time indicator has shifted from πμ/μμ to π.μ./μ.μ.

To address this, a compatibility mechanism has been added to the latest version of ybtools, allowing the continued use of the previous format.

Compatibility mode on Linux/MacOS

On Linux distributions and MacOS, you can trigger the compatibility mode by setting the environment variable `YB_LOCALE_COMPATIBILITY` to `true`. This will revert to the locale handling that was present with ybtools before version 6.7.

Example:

```
export YB_LOCALE_COMPATIBILTY=true
```

If you want to stop the compatibility mode, either delete the `YB_LOCALE_COMPATIBILITY` variable or set it to anything other than `true`.

Compatibility mode on Windows

The compatibility mode on Windows works differently and requires the use of the `_JAVA_OPTIONS` environment variable. To revert to the legacy behaviour, set the `_JAVA_OPTIONS` to `-Djava.locale.providers=COMPAT,CLDR,SPI`.

Powershell example:

```
$Env:_JAVA_OPTIONS='-Djava.locale.providers=COMPAT,CLDR,SPI'
```

Please be aware that `_JAVA_OPTIONS` may be used for other tools and options. Therefore, it is not recommended to overwrite its contents with only the Yellowbrick compatibility option. If the variable was not empty before, it's advised not to delete its contents.

Extent of the locale differences

The exhaustive list of changes introduced by JEP 252 is not well-documented. If you encounter issues with the format of your data that previously worked with earlier ybtools versions, we recommend checking if enabling the locale compatibility mode resolves the issue.

Opening Network Ports for Clients

The Yellowbrick client applications (`ybtools`) run on customer computers and need to communicate with Yellowbrick instances in order to send or receive data, using the customer's existing network infrastructure.

The network infrastructure is usually protected by firewalls (and/or Layer 3 switches) that filter traffic, mainly for security or performance reasons. These firewalls are typically configured to allow access only to critical services, such as web servers, mail servers, and existing databases. Therefore, you will need to open up access to Yellowbrick traffic through specific ports. If you don't make these firewall changes, you are likely to see the following errors or other "could not connect" errors:

- `Connection Refused` : the firewall rejected the traffic and sent a response to the client indicating that it was rejected.
- `Connection Timeout` : the firewall dropped the traffic without sending a response to the client.

Configure the network firewall to allow traffic from the client computer to the cluster on these ports (for example, by opening up the ports using source and destination filters in routing tables). The corporate IT staff should be familiar with this process and should only need to be informed about the port requirements listed on this page. In most cases, a VPN should not be necessary to send Yellowbrick client traffic through corporate firewalls.

Yellowbrick client tools communicate with the cluster by specifying both a host (instance name or `YBHOST` value) and port `5432` (`YBPORT`), which cannot be changed.

SSL-only mode is used for client connections.

All client application and user connections to Yellowbrick instances occur via the following reserved ports.

Port	Purpose	Protocol	Notes
80	Yellowbrick Manager	HTTP	HTTP server port (redirects to 443)
443	Yellowbrick Manager	HTTPS	TLS versions 1.1 and 1.2 only
5432	Yellowbrick database	TCP	Default port for database connections for all protocols: ODBC, JDBC, libpq, and so on.
11111, 11112	`ybtools` control port	TCP	Control ports used for `ybload` , `ybunload` , `ybbackup` , and `ybrestore` . (11111 redirects to 11112.)
31000 and 31001	`ybtools` data transfer	TCP	`31000` and `31001` : two ports for sending and receiving data (used by `ybload` , `ybunload` , `ybbackup` , `ybrestore` , and so on)

Uninstalling ybtools

Platforms: All platforms

Parent topic: Work with Tools & Drivers

If you need to remove client tools from your system, follow these steps.

Uninstall on Linux

To uninstall the client tools on Linux, run the following command:

- Ubuntu:

```
$ sudo apt-get remove -y ybtools
```

- CentOS:

```
$ sudo yum remove ybtools
```

Note: You can use the `rpm -qa` command to list all installed RPMs.

For example, on an Ubuntu client:

```
$ sudo apt-get remove -y ybtools
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  bsh libasm3-java libganymed-ssh2-java libisl13 libjackrabbit-java libjline-java libjna-java libjna-jni libjna-platform-java libj
  libmaven-scm-java libnetbeans-cvsclient-java libruby2.2 libsisu-guice-java llvm-3.8 llvm-3.8-dev llvm-3.8-runtime nagios-plugins
Use 'sudo apt autoremove' to remove them.
The following packages will be REMOVED:
  ybtools
0 upgraded, 0 newly installed, 1 to remove and 325 not upgraded.
After this operation, 0 B of additional disk space will be used.
(Reading database ... 153987 files and directories currently installed.)
Removing ybtools (6.1.2-a8363b32.1234) ...
```

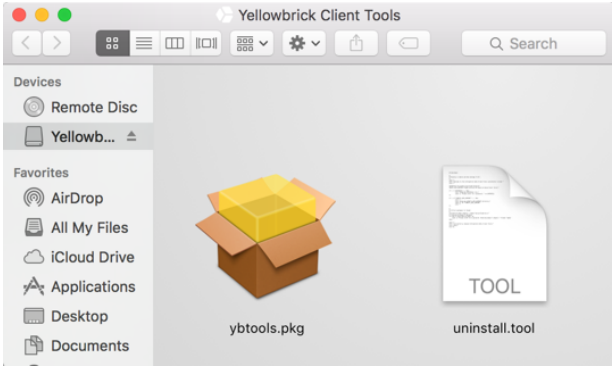
Uninstall on Windows

On the **Start** menu, go to **Control Panel** > **Uninstall a Program**, then select the tools you want to remove.

Uninstall on macOS

Follow these steps:

1. Open **Yellowbrick Client Tools**:



2. Run the `uninstall.tool` script.

```
Welcome to the Yellowbrick Data Client Tools uninstaller script.

Removing /etc/paths.d/yellowbricktools file.
Please enter <user>'s password: <password>

Removing /Applications/Yellowbrick Data/Client Tools/ directory.

Forgot package 'com.yellowbrick.ybtools.' on '/'.

Successfully removed Yellowbrick Data Client Tools.
Done.

[Process completed]
```

Uninstall on Docker

To uninstall the tools, simply remove the tools container using the following commands:

```
# Check if your container is running
docker ps

# If it shows up in the list, use the following command; otherwise,skip this step
docker stop <container-name>

# Remove the container
docker rm <container-name>

# List all images present in your system
docker images

# Find the image id of tools images and remove it
docker rmi <image-name:tag>
```

Using the ADO.NET Driver

`Npgsql` is an ADO.NET data provider that you can use to access a Yellowbrick database server from a program developed for the ADO.NET framework. You can develop ADO.NET applications that create tables, load data from local files, and run queries.

To connect to a Yellowbrick database from an ADO.NET application:

1. Download the `npgsql` driver.
2. Develop your ADO.NET program by following the guidelines in the npgsql documentation.
3. Define a connection to the Yellowbrick server. The connection string must contain the following parameters:

- Host
- Port (5432)
- Database
- User
- Password For example:

```
Host=yb007;Port=5432;Username=ybd;Database=premdb";Password=*******;
```

Administering Yellowbrick

In this section:

Enterprise Edition runs on all Yellowbrick appliances, as well as all public and private cloud platforms. The database engine is identical between platforms, but the infrastructure and storage architecture are different, leading to some differences in system views, SQL utility statements and administration.

Appliance System Administration

Platforms: EE: All appliance platforms

Parent topic: Administer Yellowbrick

In this section:

This section covers Yellowbrick appliance tasks from a system management perspective, including details on maintenance and failover operations.

Appliance: Disk Encryption

Platforms: EE: All appliance platforms

Parent topic: Appliance

In this section:

This section describes how to set up and enable hardware-based encryption on the disk drives installed in the appliance. Encryption works on disk drives for both the manager nodes and the compute blades.

For information about encrypting the data selected from specific columns and tables, see Encrypting Sensitive Data. For information about using SSL/TLS encryption for client connections and passwords, see Enabling and Verifying SSL/TLS Encryption.

Encryption Concepts

The purpose of hardware data encryption is to prevent data visibility in case of unauthorized access or theft. Yellowbrick system hardware encryption protects *data at rest* on SSDs. Data is not encrypted when it is being processed in memory or moved over the internal network on the appliance (such as during query processing and other database administration operations).

Automatic encryption of data on disk is a function of a particular type of SSD known as a *self-encrypting drive* (SED). All data stored on all such disks in the system is encrypted automatically, using the AES256 encryption algorithm (Opal 2.0-compliant). Users do not (and cannot) choose which tables or columns to encrypt, or how to encrypt the data. If you are using encryption, you have to enable it on all of the drives on all of the blades. (When you add or replace a blade, you can enable encryption on the new blade specifically by using a combination of `ybcli encryption` and `blade` commands.)

Encryption setup involves creating a *keystore* on the primary manager node. The keystore, which resides on a manager node OS disk, contains and manages *encryption keys* for the drives (these keys are generated by a `ybcli` command when you enable encryption). A drive encryption key is a mechanism for accessing the encrypted data on the drive (enabling I/O to the drives). An attempt to steal data from an SED will fail because an encryption key is required to unlock the disk. *SEDs cannot read their own data without the provision of the correct key.*

When you set up the keystore, the system provides:

- An *authentication key* for secure access to the keystore. Think of this key simply as a system-generated password. The `ybcli` user must provide this key in order to run any encryption commands.
- One or more *unlock keys* for securely unlocking the keystore. An *unlocked keystore* is a prerequisite to running database operations on encrypted drives.
- Generated encryption keys for each drive on the appliance.

You can "re-key" the unlock keys and the authentication key at any time by using the `keystore rotate` command. You can "re-key" the drives by using the `encryption rotate` command.

Encryption Setup and Enablement

To start using encryption on your appliance, you need to set up the keystore, generate keys, and unlock the keystore and drives. You can use `ybcli` commands to complete these steps. For details, see Setting Up Encryption.

Warning: Do not enable encryption on a Yellowbrick appliance if you do not have sufficient understanding of the process. You must be able to understand the `ybcli` command output for setting up and enabling encryption, and you must keep the generated encryption keys in a safe and secure location. *If you lose your encryption keys, you will not be able to recover the data on the drives. You will have to work with Yellowbrick Customer Support to do an "emergency unlock" of the drives, which results in complete data loss.*

Encryption Maintenance

Once the appliance is set up to use encryption, in general it does not require any management. Encryption is always on. However, there are a few scenarios in which you may have to intervene to unlock the keystore and the drives or complete other maintenance tasks. See also Managing Encrypted Drives.

The keystore is automatically locked whenever a failover occurs or the system is power-cycled, either intentionally or because of a power or hardware failure. The same requirement applies anytime that hardware is removed from the system.

A manager node failover does not trigger any change to the encryption setup, except that the keystore is moved to the other manager node. Encryption will continue to operate as normal during and after the failover operation.

You can enhance the security of the system by periodically "rotating" keys:

- The `encryption rotate` command generates new keys on all of the drives.
- The `keystore rotate` command generates new unlock keys and a new authentication key.

Rotating keys on the drives requires the database to be shut down. The keystore itself can have its keys rotated while the database is online.

It is rarely necessary to disable encryption, but you can do it with the `encryption disable` command, which requires a database shutdown. When you disable encryption, no data is removed from the drives.

Setting Up Encrypted Drives

Platforms: EE: All appliance platforms

Parent topic: Appliance: Disk Encryption

In this section:

This section summarizes the steps for setting up encryption for the first time on a new appliance. When these steps are complete, the data at rest on the manager node drives and the compute blade drives will be encrypted, but the drives will be unlocked and accessible for routine I/O operations during loading and query processing.

See also Example: Set Up the Keystore and Enable Encryption, which contains the complete output for this procedure in a single `ybcli` session.

The main steps are creating a keystore, unlocking it, and enabling encryption on the drives.

1. Start a ybcli session on the primary manager node.

2. Create a new keystore by running the following command and responding to the prompts.

```
YBCLI (PRIMARY)> keystore setup

Setting up the keystore
...
```

This command creates a new keystore and generates a set of keys:

- A single 36-byte authentication key, which is a required "password" that you have to enter before you can run any `encryption` commands.
- One or more 64-byte keys to unlock the keystore At least one of each type of key is required.

Note: You can request up to 5 unlock keys. In this way, multiple administrators can unlock the keystore using a combination of keys. No single key has to be distributed to a single administrator.

The `keystore setup` command also asks if you want to back up the keystore. Backing up the keystore is recommended. If you choose not to do a backup now, you can run the `keystore backup` command manually later.

3. Store all of the generated keys in a safe and secure location that is physically separate from the appliance. When prompted, you must enter all keys exactly as they were printed to the screen when you ran the `keystore setup` command.

4. Unlock the new keystore by entering one or more of the unlock keys that were generated:

```
YBCLI (PRIMARY)> keystore unlock
...
```

Here is an example of an unlock key:

```
9d046e746d09e264d61533a4a0c3a8ca2709561c86e42fd74376afed98b101b8
```

5. Enable encryption on all of the drives by running the following command and entering the authentication key:

```
YBCLI (PRIMARY)> encryption enable
...
```

Here is an example of an authentication key:

```
8b6e8b44-9ad5-5e75-36ce-7ed961a46453
```

After you enter the authentication key, the `ybcli` session remains authenticated for 10 minutes. After 10 minutes, or if you exit from the `ybcli` session and start a new one, you will be prompted to enter the key again.

You will also be prompted to either print or redact the generated key values on the screen. You may want to redact them for security reasons. The `ybcli` keeps a history of commands that have been run; however, this history does not include the key values.

Enabling encryption on all of the drives takes a few minutes.

6. Check the encryption status of the drives.

```
YBCLI (PRIMARY)> encryption status
```

The status information reports whether encryption was enabled successfully on the drives on each blade.

Example: Force Keystore Setup

Platforms: EE: All appliance platforms

Parent topic: Setting Up Encrypted Drives

The following output shows an example of the output and prompts when you use the `force` option with the keystore setup command.

CAUTION:

This command erases all of the drive keys in the keystore, then re-creates it.

```
YBCLI (PRIMARY)> keystore setup force

You have entered the 'setup' command with the 'force' option, which will erase all drive keys in the keystore.
WARNING: Potential data loss. Please read the following information carefully.

 -> If encryption is not enabled on any of the drives, you are safe to proceed.
 -> If encryption is enabled, all data on the encrypted drives will be lost unless you can restore the keystore from a backup.
 -> If you cannot restore the keystore, each encrypted drive will have to be manually unlocked by using the key printed on its lab

If you do not want to run the command with the force option, enter 'no' below.

Are you sure you want to do this?: yes
Forcing keystore initialization
Stopping keystore and reinitializing. Standby...  Done
Starting keystore. Standby...  Done
Setting up the keystore. Standby...
The system will generate two types of keys:
  -> A single authentication key
  -> One or more keys to unlock the keystore
At least one of each type of key is required.

Note: You can request up to 5 keystore unlock keys. In this way, multiple administrators can unlock the keystore
using a combination of keys. No single key has to be distributed to a single administrator.

How many unlock keys should be generated for the keystore? (1 to 5): 1
1 key will be required to unlock the keystore

Successfully initialized the keystore. Please store the keys listed below in a secure location.

The following 1 key is used to unlock the keystore after system bootup or failover:
Note: 1 of 1 key(s) are required for unlocking the keystore.
Keystore unlock key 1: 1c980bf78d027facb30e5ca2df401c1227d3292690a527566e24098955e02c07

The following key is used to authenticate to the keystore (required by any encryption command):
Authentication key: e8a96833-c8a8-065e-7927-e91f4f932e08

Keys have been generated. Please store them in a safe place.

Do you want to create a backup of the keystore?

Type yes to continue: yes

Stopping the keystore service before backup. Standby...  Done
Backing up keystore. Standby... Done
Starting the keystore service after backup. Standby...  Done
```

```
The keystore has been backed up successfully to:
    /tmp/ybd-ks-11-09-2019-19-48-56.tar.gz
Please copy the backup to another machine. The backup is located on this system at:
    yb00-mgr0.yellowbrick.io:/tmp/ybd-ks-11-09-2019-19-48-56.tar.gz
    MD5: 038e793f0571e32f34b4d554cf2cec66


Do you want to unlock the keystore (not required)?

Type yes to continue: no
Keystore will not be unlocked
```

Example: Set Up the Keystore and Enable Encryption

Platforms: EE: All appliance platforms

Parent topic: Setting Up Encrypted Drives

The following output is an example of a complete `ybcli` session in which the keystore is created, new keys are generated, the keys and a backup of the keystore are copied to a secure location, and encryption is enabled.

---

1. Set Up the Keystore

Run the `keystore setup` command:

```
YBCLI (PRIMARY)> keystore setup

Setting up the keystore. Standby...
The system will generate two types of keys:
  -> A single authentication key
  -> One or more keys to unlock the keystore
At least one of each type of key is required.

Note: You can request up to 5 keystore unlock keys. In this way, multiple administrators can unlock the keystore
using a combination of keys. No single key has to be distributed to a single administrator.

How many unlock keys should be generated for the keystore? (1 to 5): 1
1 key will be required to unlock the keystore

Successfully initialized the keystore. Please store the keys listed below in a secure location.

The following key is used to authenticate to the keystore (required by any encryption command):
Authentication key: 5f2cb5e7-0e08-897e-1119-4ec98c8814a0

The following 1 key is used to unlock the keystore after system bootup or failover:
Note: 1 of 1 key(s) are required for unlocking the keystore.
Keystore unlock key 1: 5278a49d2180e1b2fa841eebc01a1ba9ad551e33ebef1f218fcef3db9f28ede6

Keys have been generated. Please store them in a safe place.

Do you want to create a backup of the keystore?

Type yes to continue: yes

Stopping the keystore service before backup. Standby...  Done
Backing up keystore. Standby... Done
Starting the keystore service after backup. Standby...  Done

The keystore has been backed up successfully to:
    /tmp/ybd-ks-11-03-2019-11-27-59.tar.gz
Please copy the backup to another machine. The backup is located on this system at:
    yb00-mgr0.yellowbrick.io:/tmp/ybd-ks-11-03-2019-11-27-59.tar.gz
    MD5: 272e2231990294380d9366a91b21e6e1


Do you want to unlock the keystore (not required)?
```

```
Type yes to continue: no
Keystore will not be unlocked
```

Note: You can use the log keystore command to see audit log entries for keystore operations.

2. Store the Keys in a Safe Place

Copy and paste the keys from the screen output into a text file, then store it in a safe location:

```
The following key is used to authenticate to the keystore (required by any encryption command):
Authentication key: 5f2cb5e7-0e08-897e-1119-4ec98c8814a0

The following 1 key is used to unlock the keystore after system bootup or failover:
Note: 1 of 1 key(s) are required for unlocking the keystore.
Keystore unlock key 1: 5278a49d2180e1b2fa841eebc01a1ba9ad551e33ebef1f218fcef3db9f28ede6
```

3. Copy the Keystore Backup File to a Secure Location

For example:

```
[ybdadmin@yb007-mgr0 ~]$ scp /tmp/ybd-ks-11-03-2019-11-27-59.tar.gz keysadmin@keysadmin.yellowbrick.io:/home/keysadmin/backups

keysadmin@keysadmin.yellowbrick.io's password:
ybd-ks-11-03-2019-11-27-59.tar.gz                                     100%   22KB  22.0KB/s   00:00
```

4. Enable Encryption

Run the `encryption enable` command:

```
YBCLI (PRIMARY)> encryption enable

Are you sure you want to enable encryption system wide?
WARNING: Enabling encryption means this system will have to be unlocked after every power-cycle.
Also make sure to have the keys for the keystore available in a secure place. These are needed to unlock both the system and the d

Type yes to continue: yes

Do you want this command to show the actual drive keys generated?
WARNING: The keys will be shown in clear text. Saying anything but yes below will not display the keys.

Type yes to show drive keys: no
Not displaying keys in clear text
WARNING: The keystore is locked. Initiating unlocking
Please enter an unlock key to begin unlocking the keystore:
Key ->

Verifying keystore status. Standby...
Keystore locked: NO

Keystore was successfully unlocked

Authenticating to the keystore for this YBCLI session. Standby...
Please enter keystore authentication key:
Key ->

Key accepted. This session is authenticated for the next 10 minutes or until YBCLI exits.
```

```
Determining system status
Retrieving blade details. Standby...

No blades are currently using encryption. Enabling encryption initially will take the database stack down.

Type yes to continue: yes

Stopping YBD services before enabling encryption. Standby...  Done

Enabling encryption on manager node(s). Standby...

Manager: yb00-mgr0.yellowbrick.io (local node)
    Drive-sda - Serial: 17381914CAF2 -> Generating key. Standby...  KEY: REDACTED
        Enabling encryption. Standby... Encryption enabled
    Drive-sdb - Serial: 17381914CB3A -> Generating key. Standby...  KEY: REDACTED
        Enabling encryption. Standby... Encryption enabled

Manager: yb00-mgr1.yellowbrick.io (remote node)
    Drive-sda - Serial: S3D2NX0J507797Y -> Generating key. Standby...  KEY: REDACTED
        Enabling encryption. Standby... Encryption enabled
    Drive-sdb - Serial: S3D2NX0J507787F -> Generating key. Standby...  KEY: REDACTED
        Enabling encryption. Standby... Encryption enabled

Enabling encryption on blades. Standby...
Stopping YBD services before enabling encryption on blades. Standby...  Done
Retrieving blade details. Standby...

Blade in bay: 1 - UUID: 00000000-0000-0000-0000-38B8EBD00483
    Drive-0 - Serial: S3EWNWAJ318665L -> Generating key. Standby...  KEY: REDACTED
        Enabling encryption. Standby... Encryption enabled
    Drive-1 - Serial: S3EWNWAJ318656W -> Generating key. Standby...  KEY: REDACTED
        Enabling encryption. Standby... Encryption enabled
    Drive-2 - Serial: S3EWNWAJ318660F -> Generating key. Standby...  KEY: REDACTED
        Enabling encryption. Standby... Encryption enabled
    Drive-3 - Serial: S3EWNWAJ318658M -> Generating key. Standby...  KEY: REDACTED
        Enabling encryption. Standby... Encryption enabled
    Drive-4 - Serial: S3EWNWAJ318653X -> Generating key. Standby...  KEY: REDACTED
        Enabling encryption. Standby... Encryption enabled
    Drive-5 - Serial: S3EWNWAJ318652L -> Generating key. Standby...  KEY: REDACTED
        Enabling encryption. Standby... Encryption enabled
    Drive-6 - Serial: S3EWNWAJ318657P -> Generating key. Standby...  KEY: REDACTED
        Enabling encryption. Standby... Encryption enabled
    Drive-7 - Serial: S3EWNWAJ318651N -> Generating key. Standby...  KEY: REDACTED
        Enabling encryption. Standby... Encryption enabled

...

One or more drives had encryption enabled. Initiating keystore backup.

Stopping the keystore service before backup. Standby...  Done
Backing up keystore. Standby... Done
Starting the keystore service after backup. Standby...  Done

The keystore has been backed up successfully to:
    /tmp/ybd-ks-11-03-2019-12-03-01.tar.gz
Please copy the backup to another machine. The backup is located on this system at:
    yb00-mgr0.yellowbrick.io:/tmp/ybd-ks-11-03-2019-12-03-01.tar.gz
    MD5: 613d33018355d4c7f19e7d3921b075c7


Do you want to unlock the keystore (not required)?

Type yes to continue: no
Keystore will not be unlocked

YBCLI (PRIMARY)>
```

5. Copy the Keystore Backup Again

Repeat step 3, using the new backup file, which now contains the encryption keys for the drives.

6. Unlock the Keystore

Alternatively, you can unlock the keystore when you enable encryption or when you run `encryption status`.

```
YBCLI (PRIMARY)> keystore unlock

Please enter an unlock key to begin unlocking the keystore:
Key ->

Key was submitted to the keystore but it is not yet fully unlocked.

Do you want to enter another key?
Note: The keystore maintains its state; if needed, you can enter the remaining key(s) later.
If the system restarts or fails over, you will have to enter all of the keys again.

Type yes to continue: yes

Please enter another keystore key to continue unlocking the keystore (Progress: 1/2):
Key ->

Verifying keystore status. Standby...
Keystore locked: NO

Keystore was successfully unlocked
```

7. Check Encryption Status

Run the `encryption status` command.

```
YBCLI (PRIMARY)> encryption status

Do you want this command to show the actual drive keys?
WARNING: The keys will be shown in clear text. Saying anything but yes below will not display the keys

Type yes to show drive keys: no
Not displaying keys in clear text

Authenticating to the keystore for this YBCLI session. Standby...
Please enter keystore authentication key:
Key ->

Key accepted. This session is authenticated for the next 10 minutes or until YBCLI exits.

Retrieving manager node encryption status. Standby...

Local Manager drive encryption status:
    Supports encryption: 2/2
    Encryption enabled : 2/2
    Drives locked      : 0/2
        DRIVE-sda - Serial: 17381914CAF2 -> KEY PRESENT
        DRIVE-sdb - Serial: 17381914CB3A -> KEY PRESENT

Remote Manager drive encryption status:
    Supports encryption: 2/2
    Encryption enabled : 2/2
    Drives locked      : 0/2
```

```
        DRIVE-sda - Serial: S3D2NX0J507797Y -> KEY PRESENT
        DRIVE-sdb - Serial: S3D2NX0J507787F -> KEY PRESENT


Retrieving blade encryption status. Standby...
Retrieving blade details. Standby...

Blade in bay: 1 - UUID: 00000000-0000-0000-0000-38B8EBD00483
    Supports encryption: 8/8
    Encryption enabled : 8/8
    Drives locked      : 0/8
        DRIVE-0 - Serial: S3EWNWAJ318665L -> KEY PRESENT
        DRIVE-1 - Serial: S3EWNWAJ318656W -> KEY PRESENT
        DRIVE-2 - Serial: S3EWNWAJ318660F -> KEY PRESENT
        DRIVE-3 - Serial: S3EWNWAJ318658M -> KEY PRESENT
        DRIVE-4 - Serial: S3EWNWAJ318653X -> KEY PRESENT
        DRIVE-5 - Serial: S3EWNWAJ318652L -> KEY PRESENT
        DRIVE-6 - Serial: S3EWNWAJ318657P -> KEY PRESENT
        DRIVE-7 - Serial: S3EWNWAJ318651N -> KEY PRESENT

Blade in bay: 2 - UUID: 00000000-0000-0000-0000-38B8EBD00CBC
    Supports encryption: 8/8
    Encryption enabled : 8/8
    Drives locked      : 0/8
        DRIVE-0 - Serial: S3EWNWAJ316127E -> KEY PRESENT
        DRIVE-1 - Serial: S3EWNWAJ330178F -> KEY PRESENT
        DRIVE-2 - Serial: S3EWNWAJ330176R -> KEY PRESENT
        DRIVE-3 - Serial: S3EWNWAJ330173Y -> KEY PRESENT
        DRIVE-4 - Serial: S3EWNWAJ338486F -> KEY PRESENT
        DRIVE-5 - Serial: S3EWNWAJ338488E -> KEY PRESENT
        DRIVE-6 - Serial: S3EWNWAJ316128T -> KEY PRESENT
        DRIVE-7 - Serial: S3EWNWAJ338479X -> KEY PRESENT
...
```

Example: Use Multiple Unlock Keys

Platforms: EE: All appliance platforms

Parent topic: Setting Up Encrypted Drives

The following output is an example of a `ybcli` session in which the keystore is created and enabled with `multiple unlock keys`.

In this example, the system generates 5 unlock keys but only 3 are required to unlock the keystore. This means that you can distribute the 5 keys to 5 different people in your organization, then 3 of these people must be present when the keystore needs to be unlocked. You will not be able to unlock the keystore with less than 3 keys.

For an example of the complete encryption setup procedure, see Example: Set Up the Keystore and Enable Encryption.

---

1. Set Up the Keystore

Run the `keystore setup` command:

```
YBCLI(8183) (PRIMARY - yb100-mgr0)> keystore setup

Setting up the keystore. Standby...
The system will generate two types of keys:
  -> A single authentication key
  -> One or more keys to unlock the keystore
At least one of each type of key is required.

Note: You can request up to 5 keystore unlock keys. In this way, multiple administrators can unlock the keystore
using a combination of keys. No single key has to be distributed to a single administrator.

How many unlock keys should be generated for the keystore? (1 to 5): 5
How many unlock keys should be required to unlock the keystore? (2 to 5): 3

Successfully initialized the keystore. Please store the keys listed below in a secure location.

The following 3 keys are used to unlock the keystore after system bootup or failover:
Note: 3 of 5 key(s) are required for unlocking the keystore.
Keystore unlock key 1: 6bfeae96d9953905ba36a7683b7c2c76648cc91d71adff29ce3155089d06c2bcec
Keystore unlock key 2: 514f10d8aec2d6719bbaf898fdb236b6eedb716a2df8d04e05fb076631db2159b1
Keystore unlock key 3: 69cea90c851a8bf7abab7684677f8fa6db036c1794f03da6137ce084a0c1c09dc2
Keystore unlock key 4: c50b770062675d6716e5313390dc6e29077e87eb388dacc8ed1d75a6cddd26877a
Keystore unlock key 5: a8127657aa0b4a973e151351b9c511e2f112ff2fdd1056cb25b6bf055113eb32b0

The following key is used to authenticate to the keystore (required by any encryption command):
Authentication key: 52ea8341-aa63-fe31-deb2-49dd34340864

Keys have been generated. Please store them in a safe place.

Do you want to create a backup of the keystore?
Response (yes/no): yes

Stopping the keystore service before backup. Standby...  Done
Backing up keystore. Standby... Done
Starting the keystore service after backup. Standby...  Done

The keystore has been backed up successfully to:
    /tmp/ybd-ks-09-18-2019-11-50-20.tar.gz
Please copy the backup to another machine. The backup is located on this system at:
```

```
    yb98-mgr0:/tmp/ybd-ks-09-18-2019-11-50-20.tar.gz
    MD5: 2dc47c7203682ee6327d090624d0fcc8
```

Note: You can use the log keystore command to see audit log entries for keystore operations.

2. Store the Keys Safely and Copy the Backup File

Copy and paste all of the generated keys from the screen output into a text file, then store the file in a safe location. Also copy the keystore backup file to a secure location.

3. Enable Encryption

Run the `encryption enable` command:

```
YBCLI(8183) (PRIMARY - yb100-mgr0)> encryption enable

Are you sure you want to enable encryption system wide?
WARNING: Enabling encryption means this system will have to be unlocked after every power-cycle.
Also make sure to have the keys for the keystore available in a secure place. These are needed to unlock both the system and the d
Response (yes/no): yes

Do you want this command to show the actual drive keys generated?
WARNING: The keys will be shown in clear text. Saying no below will not display the keys.
Response (yes/no): yes
WARNING: The keystore is locked. Initiating unlocking
Please enter an unlock key to begin unlocking the keystore:
Key ->

Key was submitted to the keystore but it is not yet fully unlocked.

Do you want to enter another key?
Note: The keystore maintains its state; if needed, you can enter the remaining key(s) later.
If the system restarts or fails over, you will have to enter all of the keys again.
Response (yes/no): yes

Please enter another keystore key to continue unlocking the keystore (Progress: 1/3):
Key ->

Key was submitted to the keystore but it is not yet fully unlocked.

Do you want to enter another key?
Note: The keystore maintains its state; if needed, you can enter the remaining key(s) later.
If the system restarts or fails over, you will have to enter all of the keys again.
Response (yes/no): yes

Please enter another keystore key to continue unlocking the keystore (Progress: 2/3):
Key ->

Verifying keystore status. Standby...
Keystore locked: NO

Keystore was successfully unlocked

Authenticating to the keystore for this YBCLI session. Standby...
Please enter keystore authentication key:
Key ->

Key accepted. This session is authenticated for the next 10 minutes or until YBCLI exits.
```

Example: Rotate the Keystore with Multiple Unlock Keys

Platforms: EE: All appliance platforms

Parent topic: Appliance: Disk Encryption

The following `ybcli` output shows an example of the complete process for refreshing the keys for the keystore (authentication and three unlock keys). You cannot generate a new authentication key separately.

```
YBCLI (PRIMARY)> keystore rotate

Rotating the keystore will generate new authentication and unlock keys.
Note: A backup of the current keystore will be performed prior to rotating the keystore.
Keystore rotation requires both the authentication and unlock keys to be present.

Do you want to rotate the keystore?

Type yes to continue: yes

Stopping the keystore before backup. Standby... Done
Backing up keystore. Standby...Done
Starting the keystore after backup. Standby... Done

The keystore has successfully been backed up to: /tmp/ybd-ks-10-25-2017-16-14-20.tar.gz
Please copy the backup to another machine. The backup is located on this system at:
    yb00-mgr1.yellowbrick.io:/tmp/ybd-ks-10-25-2017-16-14-20.tar.gz
    MD5: 7ab839eabcad275161a4b006d5d00ffc

Please enter a keystore unlock key to begin unlocking the keystore for rotation (ctrl-c to cancel):
Key ->

Please enter another keystore unlock key to unlock the keystore (Progress: 1/2):
Key ->

Authenticating to the keystore for keystore rotation. Standby...
Please enter keystore authentication key:
Key ->

Rotating the keystore. Standby...

Note: You can request up to 5 keystore unlock keys. In this way, multiple administrators can unlock the keystore
using a combination of keys. No single key has to be distributed to a single administrator.

How many unlock keys should be generated for the keystore? (1 to 5): 3
How many unlock keys should be required to unlock the keystore? (2 to 3): 2

The following unlock key(s) were generated. A total of 2 key(s) is required to unlock the keystore:
New keystore unlock key 1: fc243401f27cb29d8da94ff03df9facc5f1806511a3b05f02971d4957335b4a093
New keystore unlock key 2: 778d2a3724b7f4421c54d3dc4323809c6253b5cea706478c1160702b3f334cb765
New keystore unlock key 3: 5c83c05756909d209b130210a39b2693316c17a0ec55684f98ff28dff956d14d11

Rotating new authentication key. Standby...
New authentication key: 7c4012c2-0975-f93c-aedc-645564fe629c

Keys have been generated. Please store them in a safe place.

Do you want to create a backup of the keystore?
```

```
Type yes to continue: yes

Stopping the keystore before backup. Standby... Done
Backing up keystore. Standby...Done
Starting the keystore after backup. Standby... Done

The keystore has successfully been backed up to: /tmp/ybd-ks-10-25-2017-16-16-23.tar.gz
Please copy the backup to another machine. The backup is located on this system at:
    yb00-mgr1.yellowbrick.io:/tmp/ybd-ks-10-25-2017-16-16-23.tar.gz
    MD5: 6665faf86d1224be3aa2541d2f85ba64

Note: When restoring this backup, the keystore will have to be unlocked again by providing the unlock key(s)

Do you want to unlock the keystore again?
Note: This is not required unless you intend to run additional encryption commands. Any encryption
command will automatically prompt for keystore unlocking should it be required.

Type yes to continue: yes
Please enter a keystore unlock key to begin unlocking the keystore:
Key ->

The keystore is not yet fully unlocked.
Do you want to continue entering another key?
Note: Keystore unlocking is stateful and the remaining key(s) can be provided at a later time.
If the system is restarted or failed over, all keys will have to be provided again.

Type yes to continue: yes

Please enter another keystore unlock key to unlock the keystore (Progress: 1/2):
Key ->

Verifying keystore status. Standby...
Keystore locked: NO

Keystore was successfully unlocked

Keystore rotation successfully completed
```

Managing Encrypted Drives

Platforms: EE: All appliance platforms

Parent topic: Appliance: Disk Encryption

The following procedures explain what you need to do to return encrypted drives to an operational state in case of power failures, blade replacements, upgrades, and failover operations. This section also covers two periodic maintenance tasks: backing up the keystore and rotating the keys.

Warning: If you lose your encryption keys, you need to open a case with Yellowbrick Customer Support and run an "emergency unlock" command. Although this procedure will bring your encryption setup back to a clean state, your data will be lost.

After a power failure (total system)

If there is a power failure on the whole appliance, follow these steps to recover access to the drives:

1. Restart the system. (Not everything will be operational.)
2. Start the `ybcli`.
3. Respond to the prompt to unlock the keystore and enter the authentication key.
4. Power on the blades: `blade poweron all`
5. Run the `encryption unlock` command.

After a power failure (manager nodes only)

If power fails only on the manager nodes, run the first three steps from the previous section.

After a power failure (blades only)

If power fails only on the compute blades, follow these steps:

1. Power on the blades: `blade poweron all`
2. Run the `encryption unlock` command.
3. Respond to the prompt to unlock the keystore and enter the authentication key.

During an upgrade

During an upgrade on a system with encrypted drives, the Yellowbrick Installer will prompt you to unlock the keystore and run the `encryption unlock` command.

Replacing a blade

The steps for replacing a compute blade depend on whether you are going to use the same blade or a new one:

- If you are replacing a failed blade with the same blade (after fixing the problem), run the `encryption unlock` command. If the command fails for some reason, you will probably need to replace the blade with another blade.
- If you are replacing a failed blade with a new blade, run the `encryption enable` command and respond to the prompts to unlock the keystore and authenticate.

Adding one or more blades (expansion)

You can add one or more new blades to expand the capacity of the system. On a system that uses encryption:

1. Make sure the new blade is a supported SED blade before inserting it into the chassis.
2. Power on the new blade, using the blade poweron command.

3. Run the `encryption enable` command and respond to the prompts.
4. Run the blade add command.

After a manager node failover (or a reboot of the manager nodes)

The keystore is replicated on both manager nodes, so failing over does not require any user intervention. Encryption continues to work as normal.

If you need to run any encryption commands, you will see a prompt to unlock the keystore.

Key rotation

As an additional security measure, you can periodically generate new encryption keys for the drives. See the encryption rotate command. You can do the same thing with the unlock keys and authentication key by using the keystore rotate command.

Backup and restore operations

Whenever the keystore is modified in response to `encryption enable` and `encryption rotate` operations, a keystore backup happens automatically. Backup files are saved to the `ybcli` user's home directory. The files are very small and have a timestamped name.

Note: Always move new backup files to a remote system for secure storage.

You can back up the keystore manually at any time by using the `keystore backup` command.

Remote Diagnostics

Platforms: EE: All appliance platforms

Parent topic: Appliance

In this section:

The remote diagnostics system is used to collect critical log, telemetry, crash dump, performance, and monitoring/alerting data from the Yellowbrick cluster so Customer Support can proactively identify, respond to, and repair issues on the cluster. Diagnostics data is sent to support automatically, either using a web service with connectivity from the customer site to the remote diagnostics server (`phonehome.yellowbrick.io` , port `443`) or via an SMTP email channel.

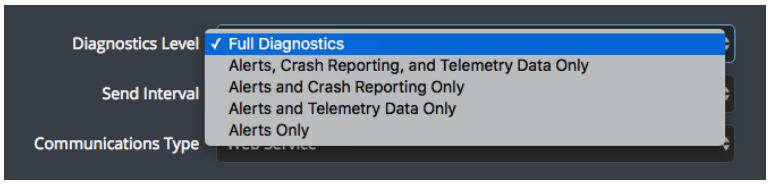
Configuring Remote Diagnostics

Platforms: EE: All appliance platforms

Parent topic: Remote Diagnostics

Use the SMC to configure remote diagnostics by choosing a service level and defining connectivity settings. You may not need to follow these steps if you are going to use the default Full Diagnostics service level via the web service.

1. Log in to the SMC and go to **Configure** > **System Settings** > **Diagnostics Data**.
2. Select a diagnostics service level. See Diagnostics Service Levels for details about the differences.



3. Optionally, set the send interval by selecting from the drop-down list. (Every 15 minutes is the default interval.)
4. Select the communications type: **Web Service** or **Email**.

Note: Select **Disabled** if you want to disable all diagnostics activity on your appliance.

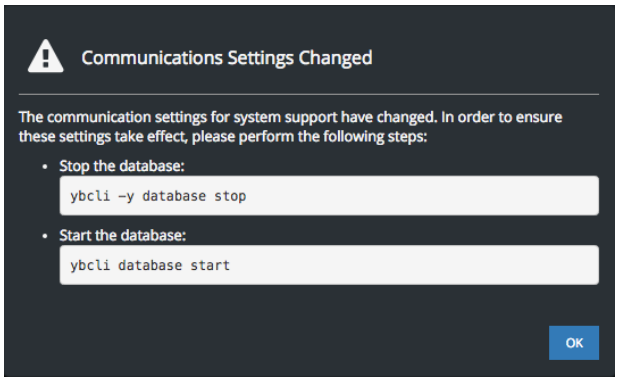
5. Enter the required information for the communications type you chose:

- If you selected **Web Service**, you may need to enable and define a proxy host, port, and type (HTTP or SOCKS) and add a new trusted certificate. If you have a proxy, select **Enable Proxy** and fill out the required fields. If your proxy requires its own certificate, you can clear existing certificates, add new ones, and trust them pending a save to your new diagnostics settings. (If you do not have a proxy or you have a transparent proxy, do not select **Enable Proxy**.)
- If you selected **Email**, fill out the requested details, including the SMTP server and the recipient.

  Note: When the **Recipients** field is left blank, `phonehome@yellowbrick.io` is the default recipient. If you enter any other email addresses where you want the diagnostics data to be sent, you must also manually enter `phonehome@yellowbrick.io` (assuming that you want Customer Support to receive the diagnostics information automatically). The list of recipients must be separated by commas.

6. Test your configuration by clicking **Test Alert**, then **Send Alert**. A default test message will be sent; optionally, you can edit the message text and severity. If the alert fails to send, go back and check your settings.
7. Click **Save Settings** in the upper right corner of the screen.

A pop-up message provides instructions on stopping and starting the database in order for the configuration changes to take effect.


Communications Settings Changed
The communication settings for system support have changed. In order to ensure these settings take effect, please perform the following steps:
Stop the database:
ybcli -y database stop
Start the database:
ybcli database start
OK

Diagnostics Service Levels

Platforms: EE: All appliance platforms

Parent topic: Remote Diagnostics

The *full diagnostics* service level, as defined in the following table, is enabled for new appliances when they are shipped. Depending on the specific support requirements at your site, you can configure remote diagnostics to operate at a "lower" service level. Note the main differences between full diagnostics and lower service levels:

- Full diagnostics data includes system log files, which may contain SQL query text and schema DDL information (table and column names).
- No email channel is available for full diagnostics; the web service must be used.

Both predefined system alerts and user-defined alerts are captured in diagnostics data. For information about configuring alerts, see System Alerts. Note that user-defined alert messages may get sent over the network as part of remote diagnostics data.

Service Level	Description
Full Diagnostics	System log files, full crash dumps, telemetry data, and system alerts are all included in the full set of diagnostics data that is sent to support. Log files may contain SQL query text and schema DDL information. The email channel is not available for this service level.
Alerts, Crash Reporting, and Telemetry Data Only	Crash dumps, telemetry data, and system alerts are included. No logs are included. Web service and email channels are both available.
Alerts and Crash Reporting Only	Crash dumps and system alerts are included. No logs are included. Web service and email channels are both available.
Alerts and Telemetry Data Only	Telemetry data and system alerts are included. No logs are included. Web service and email channels are both available.
Alerts Only	Only system alerts are included. No logs are included. Web service and email channels are both available.
Disabled (Communication Type in SMC)	Disabling the communication type in the SMC disables remote diagnostics support completely. In this case customers open support tickets as needed and provide diagnostics information on request from Customer Support.

Note: Collecting and sending diagnostics data to support with the `ybcli system diagnostics` command is a completely separate process from diagnostics data collection. The data that `ybcli` collects may contain SQL query text and schema DDL information.

Enabling Remote Diagnostics

Before you can configure remote diagnostics, make sure that connectivity is established with the diagnostics server and that the appliance is registered for diagnostics support.

1. Check remote diagnostics server connectivity. (If your appliance is going to use email-based diagnostics, you can skip this step.)

Test connectivity to the diagnostics server by doing an `ssh` to the manager node on the appliance, then running the following `telnet` command. .

```
$ telnet phonehome.yellowbrick.io 443
Trying 10.30.20.160...
Connected to phonehome.yellowbrick.io.
Escape character is '^]'.
```

This output shows that the manager node can connect to the server at port 443. You can enter `CTRL+C` to escape. If this command hangs, you will need to get connectivity resolved before you can proceed.

2. Make sure the appliance is registered, or re-register it if necessary.

Yellowbrick always ships appliances that are already registered. Verify this by using the `ybcli system status` command. If for some reason the registration is lost, follow these steps to re-register the cluster with Yellowbrick.

Use `ybcli system register` to execute the registration process and get the Yellowbrick cluster certificates. Running this command makes a registration request using REST, generates the certificates, and returns them in the REST response:

```
YBCLI (PRIMARY)> system register <tenant-name> <system-name>
```

where `<tenant-name>` is the company name and `<system-name>` is the system name. Yellowbrick has so far named all systems `Yellowbrick1` to `YellowbrickN`. This step requires connectivity to the Yellowbrick diagnostics web service on port 443. If connectivity is blocked at the customer site and registration is lost, contact Yellowbrick support for a manual registration.

System Alerts

Yellowbrick Documentation > How-to Guides > Administer Yellowbrick > Appliance > System Alerts

Platforms: EE: All appliance platforms

Parent topic: Appliance

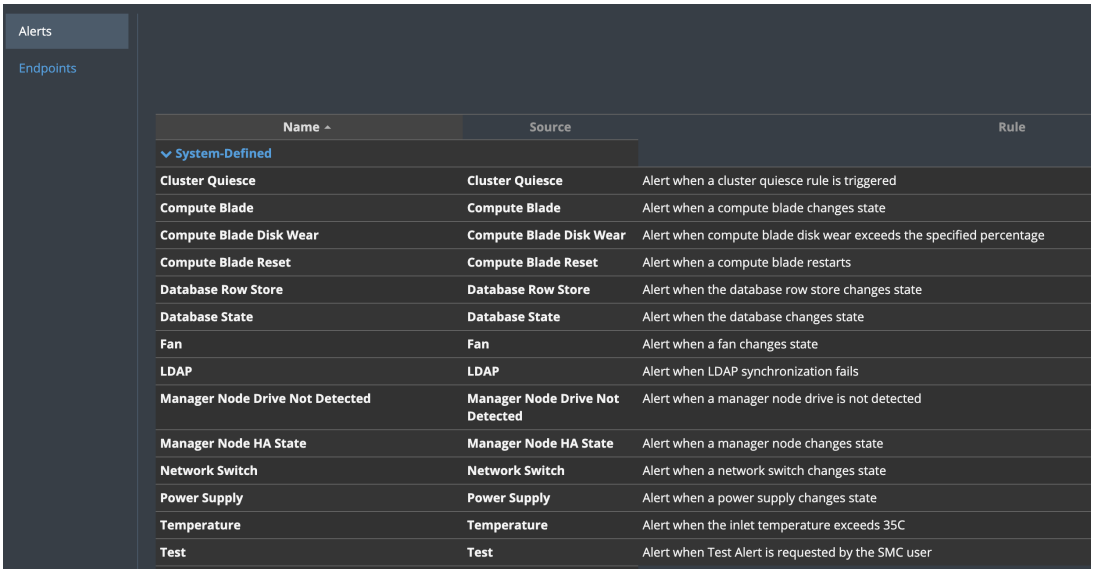
In this section:

You can use the SMC to configure alerts that are triggered when specific conditions occur on the system, such as database state changes or elevated temperature readings on hardware components. Some alerts are system-defined and ready for use, but for others you can define specific thresholds and rules. For all alerts, you can create "endpoints" that define how alert messages are published to users and what they contain.

To start using the alerting feature, log in to the SMC and go to **Configure** > **Alerting**. To see active and logged alerts, go to **Manage** > **Cluster Alerts** or query the `sys.alert` and `sys.log_alert` views.

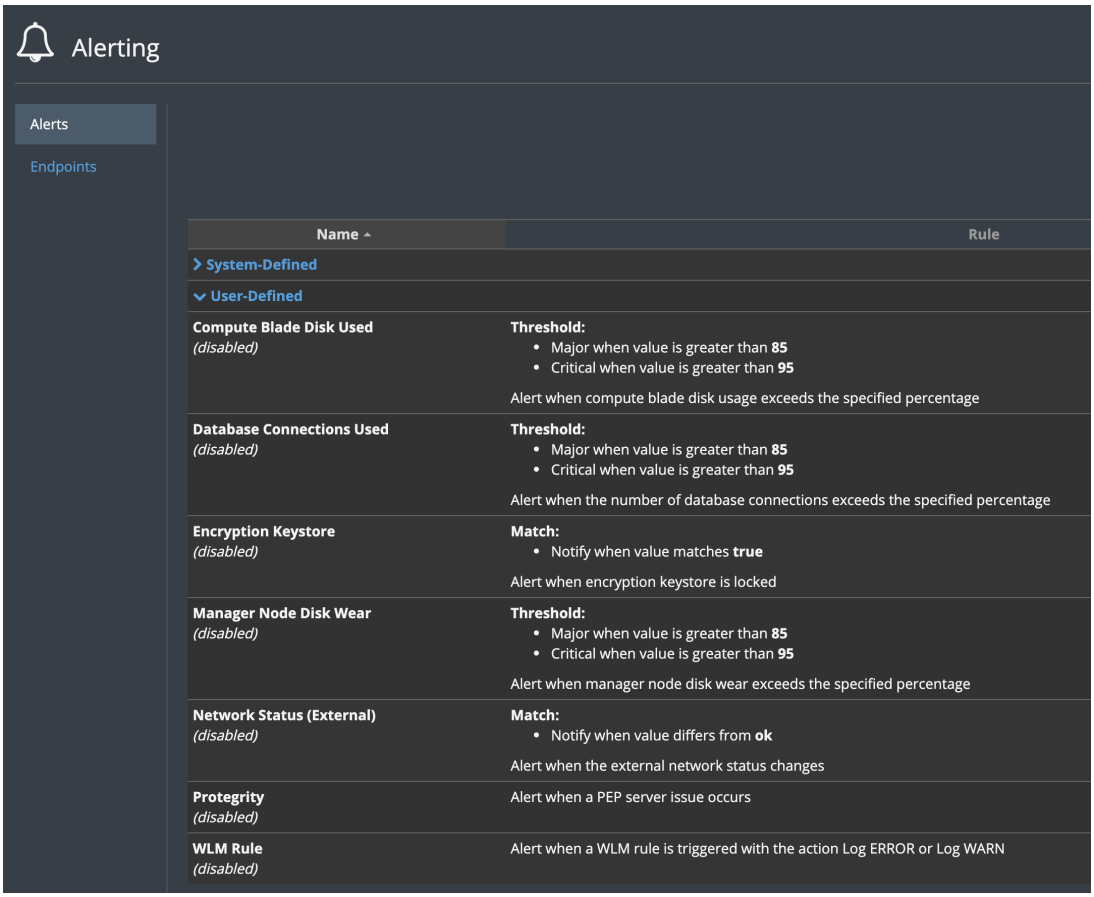
System-Defined and User-Defined Alerts

System-defined alerts cannot be changed. They are always enabled and are triggered according to a specific set of rules that you cannot edit.



Name	Source	Rule
System-Defined		
Cluster Quiesce	Cluster Quiesce	Alert when a cluster quiesce rule is triggered
Compute Blade	Compute Blade	Alert when a compute blade changes state
Compute Blade Disk Wear	Compute Blade Disk Wear	Alert when compute blade disk wear exceeds the specified percentage
Compute Blade Reset	Compute Blade Reset	Alert when a compute blade restarts
Database Row Store	Database Row Store	Alert when the database row store changes state
Database State	Database State	Alert when the database changes state
Fan	Fan	Alert when a fan changes state
LDAP	LDAP	Alert when LDAP synchronization fails
Manager Node Drive Not Detected	Manager Node Drive Not Detected	Alert when a manager node drive is not detected
Manager Node HA State	Manager Node HA State	Alert when a manager node changes state
Network Switch	Network Switch	Alert when a network switch changes state
Power Supply	Power Supply	Alert when a power supply changes state
Temperature	Temperature	Alert when the inlet temperature exceeds 35C
Test	Test	Alert when Test Alert is requested by the SMC user

User-defined alerts have default behavior that you can change by setting a threshold or by creating and using WLM rules. You cannot create completely new user-defined alerts. User-defined alerts must be enabled individually. Disabled alerts are not triggered. In this example, some of the alerts are disabled.



For more details about all of these alerts, see Alert Types.

Endpoints

You configure how alerts are published to consumers by defining endpoints:

- Notify Yellowbrick support (phonehome)
- Send email
- Post to web service (http)
- Send an SNMP trap

See Creating an Alert Endpoint.

Creating an Alert Endpoint

Platforms: EE: All appliance platforms

Parent topic: System Alerts

In this section:

Endpoints define the method that will be used to publish alerts, who the recipients or subscribers will be, and how the contents of alert messages will be formatted.

1. In the SMC, go to **Configure** > **Alerting**.
2. Click **Endpoints**, then **Create Alert Endpoint**.
3. Enter a name and optionally a description. Leave the **Enabled** checkbox selected (or leave it disabled if you are not ready to turn on alerts via this endpoint).
4. Click **Next** and choose an endpoint type.

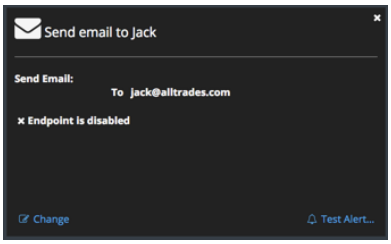
- Send Email
- Send an SNMP Trap
- Post to Web Service (http)
- Notify Yellowbrick Support

5. Fill out the details for the endpoint type you selected, then click **Finish**. For example, to send email, you need to supply information about the email server and at least one email address. For details about each type, see Endpoint Types.

All new endpoints are automatically added for all system-defined and user-defined alerts. If you do not want to use an endpoint, disable it. You can still test an alert for a specific endpoint when it is disabled.

6. Send a test alert to verify that the endpoint is configured correctly. (You can test an alert for a specific endpoint when it is enabled or disabled.)
7. Click **Test Alert** in the bottom-right corner of the endpoint summary screen.

   For example:



   Note: If you click **Test Alert** in the upper right corner of the main Alerting screen, this alert will attempt to go to all of the enabled endpoints that have been configured for the system.

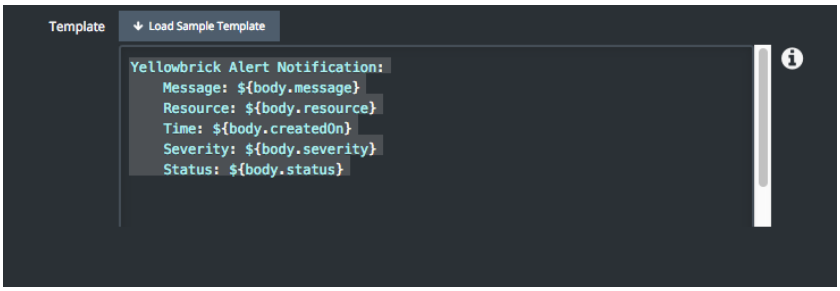
8. Enter a message.
9. Optionally, change the severity and enter a resource.
10. Click **Send**.

Alert Templates

When you create endpoints, you can load templates that describe the structure and contents of the alert messages that are sent. The following templates are supported:

- Text
- HTML
- JSON
- XML
- Form Encoded

Having selected a template, you can customize it. For example, the default text template looks like this:



You can add to or modify this template by using attributes from the list of template attributes. This list is displayed when you click the information icon next to the **Template** field. Add entries with the following form:

```
${body.attribute}
```

For example, this modified template has a different title and two additional components (`Appliance` and `Alert Name`):

```
Yellowbrick Database Alert:
   Appliance: ${body.origin}
   Alert Name: ${body.alertDefinitionName}
   Message: ${body.message}
   Resource: ${body.resource}
   Time: ${body.createdOn}
   Severity: ${body.severity}
   Status: ${body.status}
```

The resulting messages look like these examples:

```
Yellowbrick Database Alert:
   Appliance: yb100
   Alert Name: WLM Rule
   Message: Alert for premdb queries (291285)
   Resource: database:wlm:Bob's premdb rule
```

```
   Time: Wed Aug 22 17:24:54 PDT 2018
   Severity: 50
   Status: OPEN
```

```
Yellowbrick Database Alert:
   Appliance: yb100
   Alert Name: Database State
   Message: The database is degraded due to: missing compute nodes (1). Performance may be affected.
   Resource: database:state
   Time: Wed Aug 22 17:27:59 PDT 2018
   Severity: 75
   Status: CLOSED
```

Note: Message severity is a gradient, where `25` =Informational, `50` =Minor, `75` =Major, and `100` =Critical.

Endpoint Types

Platforms: EE: All appliance platforms

Parent topic: Creating an Alert Endpoint

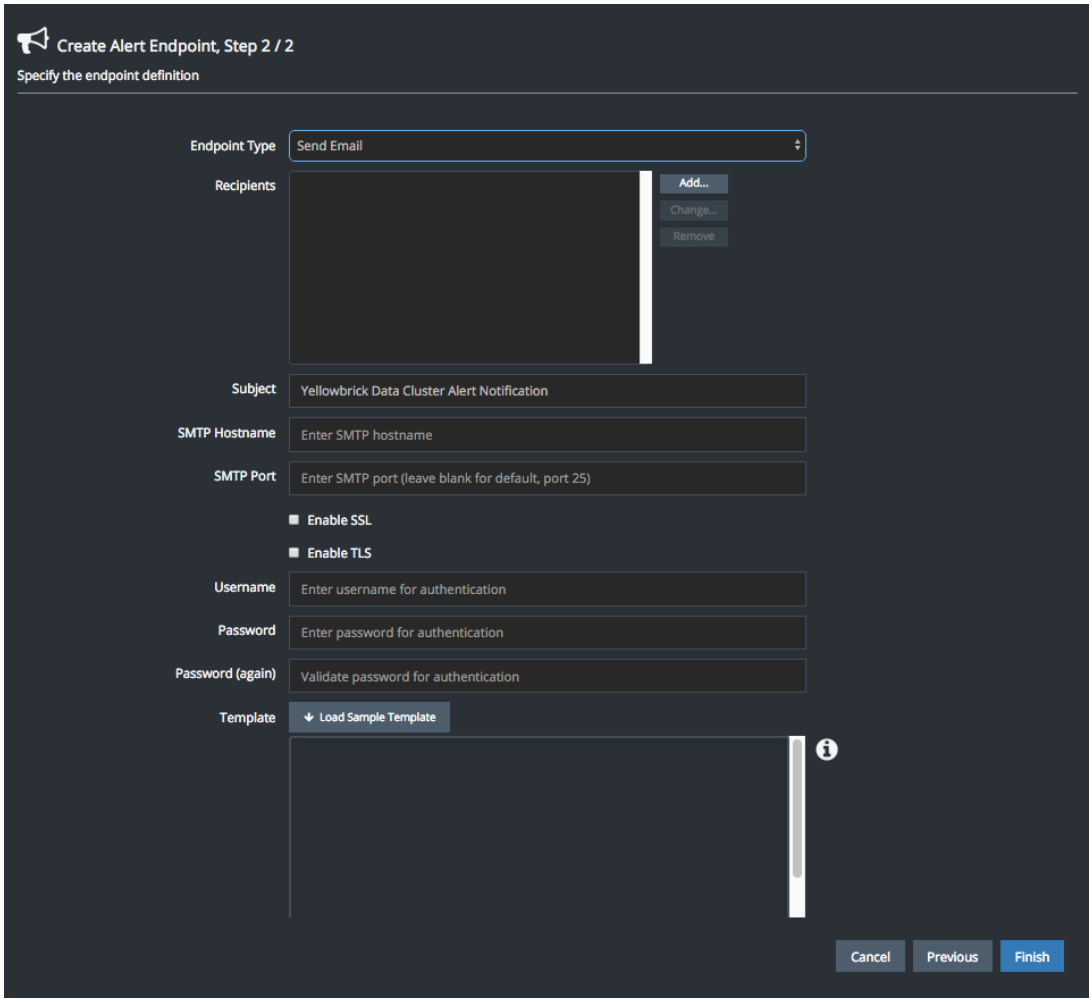
Yellowbrick supports four different endpoint types:

- Notify Yellowbrick support (phonehome)
- Send email
- Post to web service (http)
- Send an SNMP trap

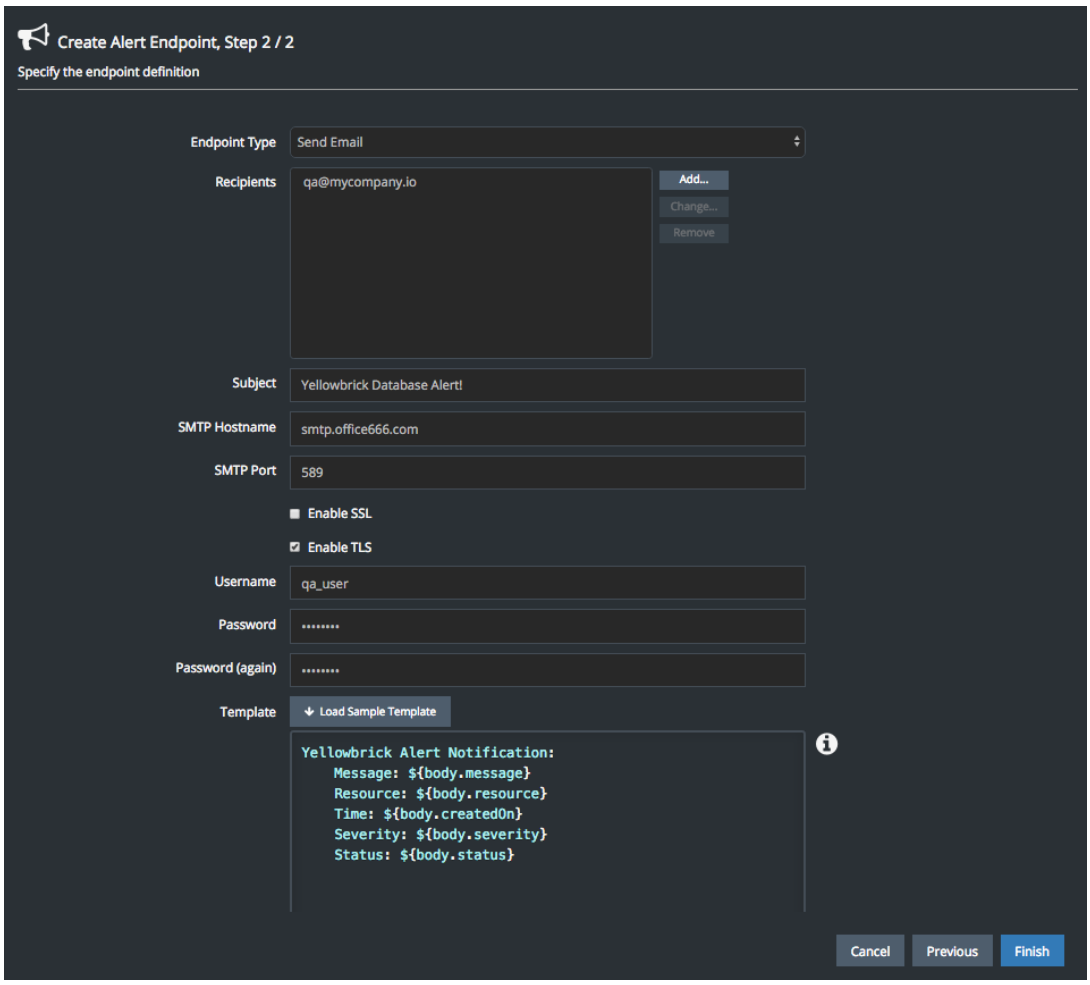
You can configure and enable many different endpoints, but they are all tied to the same set of alerts.

Send Email

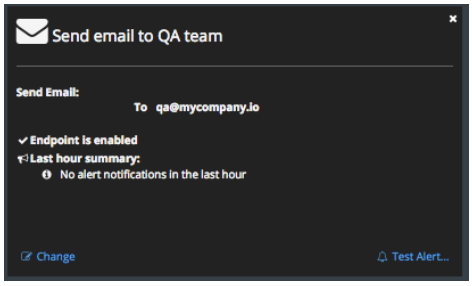
The following configuration details are required for this endpoint type:



Add email addresses by clicking the **Add** button. You cannot type into the `Recipients` field. Be sure to provide accurate credentials (username/password) and the correct SMTP hostname and port number for your mail server. You can change the subject line that will be used for email alerts. For example:

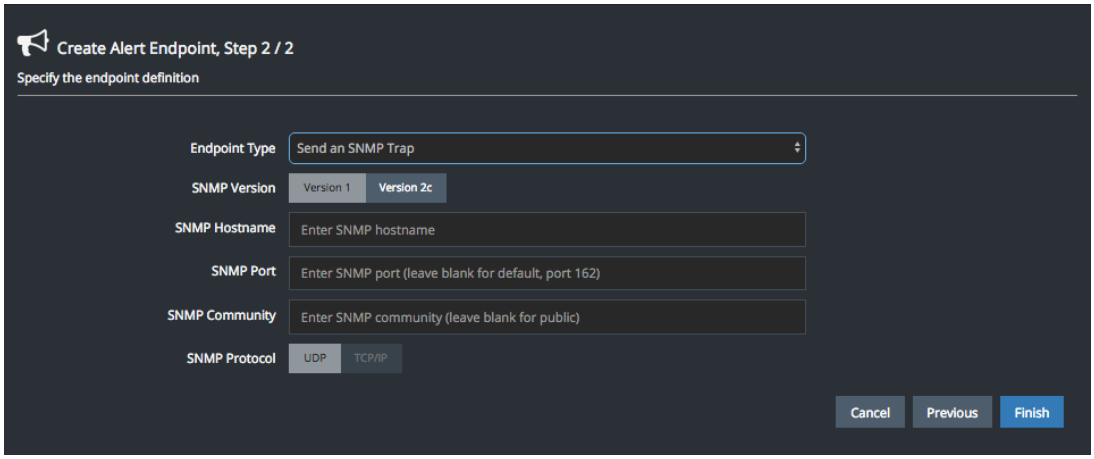


For information about Load Sample Template choices, see Alert Templates. After clicking **Finish** on this screen, you can see the endpoint on the **Endpoints** tab of the main **Alerting** screen:



Send an SNMP Trap

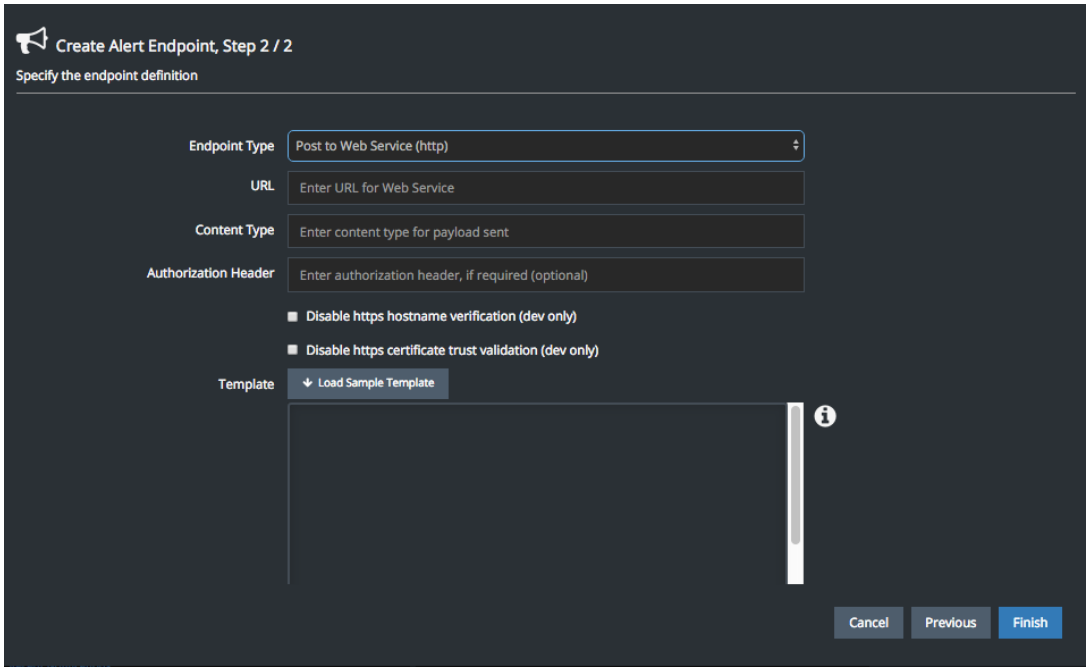
The following configuration details are required for this endpoint type:



SNMP Versions 1 and 2c are supported. Version 1 supports the UDP protocol. Version 2c supports UDP or TCP/IP. Be sure to provide the correct SMTP hostname and port number.

Post to Web Service (http)

The following configuration details are required for this endpoint type:



Notify Yellowbrick Support

No configuration is required or allowed for this endpoint.

Alert Types

Platforms: EE: All appliance platforms

Parent topic: System Alerts

This section describes the specific system-defined and user-defined alerts you can configure on your cluster.

Alerts return two messages reflecting an `OPEN` status and a `CLOSED` status for the event that triggered the alert. `OPEN` alerts are considered *active* until `CLOSED`. An alert is closed when the alert event is resolved, such as when the database or a compute blade comes back online. Test alerts also have `OPEN` and `CLOSED` messages, which follow each other within about 10 seconds.

Alert messages provide specific details about the error conditions that triggered the alert. For example, for a Database State alert you may see one of the following messages:

```
The database stopped running: too many missing compute nodes
The database is degraded due to: rebuilding. Performance may be affected.
```

System-Defined Alerts

The following system-defined alerts are supported. (You cannot create new alert types.)

Alert Name	Rule	Resource ID in Alert Messages
Cluster Quiesce	Alert when the database is quiesced or an attempt to quiesce it fails. When the database is quiesced, active queries are canceled. Queries that were queued start running when the database comes back online.	`database:event`
Compute Blade	Alert when a compute blade changes state. For example, a blade may be powered off, causing an alert.	Chassis number and blade number. For example: `chassis0:blade10`
Compute Blade Disk Wear	Alert when compute node disk wear indicates a hardware replacement will be necessary in due time. Thresholds are 85% wear for major and 90% for critical severity.	Chassis number and blade number. For example: `chassis0:blade10`
Compute Blade Filesystem Usage	Alert when a blade filesystem usage gets too high. Default is 95% and can be controlled via a configuration parameter. This generates a critical alert.	`bbfs:drive:usage` "Blade UUID" - Filesystem(s): 1 at 99%,5 at 91%;
Compute Blade Reset	Alert when a compute blade restarts.	Chassis number and blade number. For example: `chassis2:blade14`

Alert Name	Rule	Resource ID in Alert Messages
Database State	Alert when the database changes state. For example, the database may be degraded because a compute node is offline.	`database:state`
Database Row Store	Alert when the database row store changes state.	`database:rowstore`
Fan	Alert when a fan changes state. For example, a fan may have failed or have been removed from the appliance.	Chassis number and fan number. For example: `chassis0:fan2`
LDAP	Alert when LDAP synchronization fails.	`database:LDAPSynchronizer`
Manager Node Disk Wear	Alert when manager node disk wear indicates a hardware replacement will be necessary in due time. Thresholds are 85% wear for major and 90% for critical severity.	Manager node number, drive name, then `wear`. For example: `manager2:mgmt2-/dev/nvme2n1:wear`
Manager Node Drive Not Detected	Alert when a manager node drive is not detected. For example, a specific drive may not be installed.	Manager node number and drive ID. For example: `manager1:drive:nvme4n1`
Manager Node HA State	Alert when a manager node changes state. For example, one of the manager nodes may be offline, and failover is temporarily not supported.	`database:ha_state`
Network Switch	Alert when a network switch changes state.	Chassis number and switch number. For example: `chassis0:switch2`
Power Supply	Alert when a power supply changes state.	Chassis number and power supply number. For example: `chassis0:power2`
Temperature	Alert when the inlet temperature for the system exceeds 35C.	`database:temperature`

Alert Name	Rule	Resource ID in Alert Messages
Test	Alert when **Test Alert** is requested by the SMC user.	`database:test`

Test Alerts

Test alerts are system-defined, but you can trigger them in two different ways:

- After finishing the creation of a new endpoint, you can send a test alert for that specific endpoint. In this case, the endpoint may be enabled or disabled. Click **Test Alert** within the summary screen for the endpoint.
- You can send an alert to all *enabled* endpoints via **Configure** > **Alerting** > **Test Alert**). Disabled endpoints will not receive the alert.

User-Defined Alerts

The following user-defined alerts are supported. By default, they are all disabled. You can enable all of them or any subset.

You cannot create new alert types.

The alerts with numeric thresholds have default values for `Major` and `Critical` severity alerts. You can define additional thresholds for `Informational` and `Minor` severity alerts.

Alert Name	Rule	Resource ID in Messages
Backup Chain Age	Alert when there are backup chains older than a configurable threshold (controlled by configuration parameter `old_chain_threshold_days`). - Default threshold: 30 days	`database:old_backup_chains`
Compute Blade Disk Used	Alert when compute blade disk usage exceeds the specified percentage. One alert is triggered per cluster, when any one drive exceeds the threshold. Default thresholds: - Major severity when value is greater than 85 - Critical severity when value is greater than 95	Chassis number, blade number, drive number, then `usage`. For example: `chassis0:blade9:drive3:usage`
Compute Blade Disk Wear	Alert when compute blade disk wear exceeds the specified percentage. Default thresholds: - Major severity when value is greater than 85 - Critical severity when value is greater than 95	Chassis number, blade number, drive number, then `wear`. For example: `chassis0:blade9:drive3:wear`
Database Connections Used	Alert when the number of database connections exceeds the specified percentage. Default thresholds: - Major severity when value is greater than 85 - Critical severity when value is greater than 95	`database:connections`
Encryption Keystore	Alert when the encryption keystore is locked.	`mgmt0:vault` or `mgmt1:vault`, depending on which manager node is the primary.

Alert Name	Rule	Resource ID in Messages
Network Status (External)	Alert when the external network status changes.	`manager#:external_bond`
WLM Rule	Alert when a WLM rule is triggered with the action `Log ERROR` or `Log WARN`. See Rule Actions. WLM alerts are based on workload management rules rather than alerting rules. The message for a WLM alert contains the query ID that triggered the WLM rule in parentheses.	For example: `database:wlm:SELECT * rule` where `SELECT * rule` is the name of a WLM rule that was triggered and in turn triggered the alert.
Yellowbrick Row Store Data Files	Alert when the number of data files exceeds one of the configurable thresholds. - Minor severity when the number exceeds the value of configuration parameter `yrs_data_files_count_minor_threshold` - Major when the number exceeds the value of `yrs_data_files_count_major_threshold` - Critical when the number exceeds the value of `yrs_data_files_count_critical_threshold`	`database:yrs`
Yellowbrick Row Store Commit Records	Alert when the number of commit records exceeds one of the configurable thresholds. - Minor severity when the number exceeds the value of configuration parameter `yrs_commit_records_count_minor_threshold` - Major when the number exceeds the value of `yrs_commit_records_count_major_threshold` - Critical when the number exceeds the value of `yrs_commit_records_count_critical_threshold`	`database:yrs`
Yellowbrick Row Store Delete Records	Alert when the number of delete records exceeds one of the configurable thresholds. - Minor severity when the number exceeds the value of configuration parameter `yrs_delete_records_count_minor_threshold` - Major when the number exceeds the value of `yrs_delete_records_count_major_threshold` - Critical when the number exceeds the value of `yrs_delete_records_count_critical_threshold`	`database:yrs`
Yellowbrick Row Store Unused Files	Alert when the number of unused files exceeds one of the configurable thresholds. - Minor severity when the number exceeds the value of configuration parameter `yrs_unused_files_count_minor_threshold` - Major when the number exceeds the value of `yrs_unused_files_count_major_threshold` - Critical when the number exceeds the value of `yrs_unused_files_count_critical_threshold`	`database:yrs`

Query Alerts

To see active and logged *query alerts*, go to **Manage** > **Query Alerts**. WLM alerts appear under **Query Alerts** by default; if they are enabled in the **Configure Alerting** screen, WLM alerts also appear under **Cluster Alerts**.

Managing Alerts

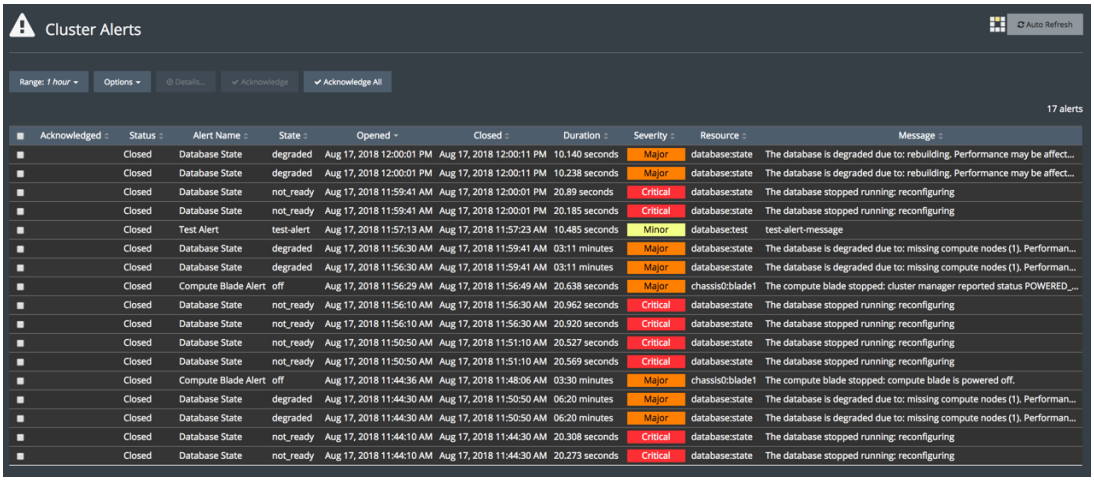
Platforms: EE: All appliance platforms

Parent topic: System Alerts

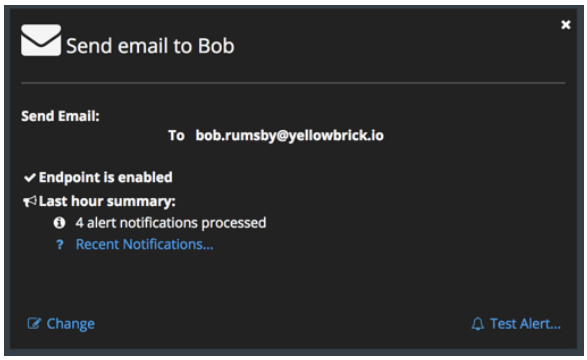
To manage active and logged alerts, you can use the SMC views, or you can query system views.

SMC Views

Go to **Manage** > **Cluster Alerts**. To change the display, click **Options**. For example:



Go to **Configure** > **Alerting** > **Endpoints** to see a quick summary of the number of alerts that have been processed recently or that have failed with errors. For example:



You can click **Recent Notifications** to see a list of recent alert messages for that endpoint.

System Views

To see active alerts (alerts in `OPEN` state), query the sys.alert view. For example:

```
yellowbrick=# select * from sys.alert;
-[ RECORD 1 ]-----+-----------------------------------------------------
alert_id          | c3346760-51c9-4b23-b5b7-bcb1cc00d262
alert_name        | Power Supply Alert
open_time         | 2018-08-16 15:32:34-07
close_time        | [NULL]
last_update_time  | 2018-08-16 15:32:34-07
duration_ms       | 74426598.612
resource          | chassis0:power4
severity          | CRITICAL
status            | OPEN
state             | error
message           | The power supply stopped: power supply status is error.
acknowledged      | f
acknowledged_time | [NULL]
acknowledged_by   | [NULL]
acknowledged_text | [NULL]
...
```

To see all logged alerts that have been `CLOSED` and any alerts that remain `OPEN`, query the sys.log_alert view. For example:

```
yellowbrick=# select * from sys.log_alert;
-[ RECORD 1 ]-----+-----------------------------------------------------------------------------------------
alert_id          | 180f13ae-3d4d-4452-9433-e44b1673b6c1
alert_name        | Database State
open_time         | 2018-08-14 01:16:37-07
close_time        | 2018-08-14 01:16:51.073-07
last_update_time  | 2018-08-14 01:16:51-07
duration_ms       | 14073.000
resource          | database:state
severity          | CRITICAL
status            | CLOSED
state             | offline
message           | The database stopped running: not enough compute nodes to start
acknowledged      | f
acknowledged_time | [NULL]
acknowledged_by   | [NULL]
acknowledged_text | [NULL]
-[ RECORD 2 ]-----+-----------------------------------------------------------------------------------------
alert_id          | bdf4f281-5eec-4321-81a2-a3dc20b49178
alert_name        | Database State
open_time         | 2018-08-14 01:16:51-07
close_time        | 2018-08-14 01:18:59.148-07
last_update_time  | 2018-08-14 01:18:59-07
duration_ms       | 128148.000
resource          | database:state
severity          | INFORMATIONAL
status            | CLOSED
state             | stopped
message           | The database was stopped by an administrator
acknowledged      | f
acknowledged_time | [NULL]
acknowledged_by   | [NULL]
acknowledged_text | [NULL]
...
```

Using the System Management Console

Yellowbrick Documentation > How-to Guides > Administer Yellowbrick > Appliance > Using the System Management Console

Platforms: EE: All appliance platforms

Parent topic: Appliance

In this section:

Configuration Settings in the SMC

Introduction to the SMC

SMC Console Login

The Yellowbrick System Management Console (SMC) is an intuitive and largely self-explanatory graphical user interface, mainly used for administration and monitoring. This section summarizes the main uses of the SMC and provides basic startup and login instructions.

Yellowbrick recommends the use of an *evergreen browser* (a browser that is automatically updated with new features, performance improvements, and security fixes). The SMC is supported on the following web browsers, all of which are classified as evergreen:

- Google Chrome: Version 40 and up
- Mozilla Firefox: Version 35 and up
- Microsoft Edge: Version 12 and up
- Safari: Version 9 and up

Note: The SMC is not supported on IE11 or any other version of Internet Explorer.

In This Section

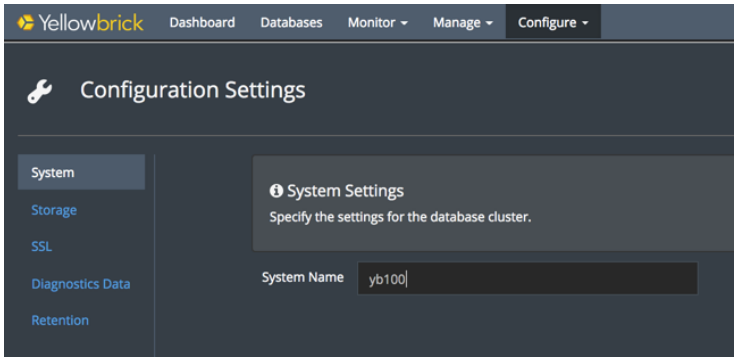
- Configuration Settings in the SMC

Configuration Settings in the SMC

Platforms: EE: All appliance platforms

Parent topic: Using the System Management Console

You can use the SMC **Configure Settings** screen to change the system configuration in a few ways.



After changing any of these values, click **Save Settings** in the upper right corner of the screen.

System Name

You can provide a system name that is different from the registered appliance name. For example, if you have multiple registered appliances (`yellowbrick1` , `yellowbrick2`), you may want to provide a more descriptive business name for each appliance, such as `Palo Alto Dev` and `Palo Alto QA` . These business names are basically nicknames that are visible in SMC browser tabs but do not interfere with the original registered names for each appliance.

Storage: Temporary Space Reservation

Set reserved temporary space as a percentage of total system capacity.

Warning: Changing this value cancels all running queries and reconfigures the database. This operation may take a while, during which time the database is unavailable.

For information about the use of temporary space in queries that spill to disk, see Allocating Query Memory.

SSL

See Enabling and Verifying SSL/TLS Encryption for information about SSL-only mode and importing certificates.

Diagnostics Data

Configure phonehome diagnostics data. See Remote Diagnostics.

Retention

Set the log retention time period for query statistics. The default is 30 days. You can change the setting to a minimum of 1 day or a maximum of 2 years (and specific time periods in between). A database restart is required in order for the change to take effect.

This time period sets the duration of history that is retained in the following `sys.log_*` views:

- sys.log_audit
- sys.log_authentication

- sys.log_error
- sys.log_query
- sys.log_query_alert
- sys.log_query_explain
- sys.log_session
- sys.log_warning

See also System Views.

Introduction to the SMC

This section describes the main features and functions of the SMC. The main menu has the following tabs:

Dashboard

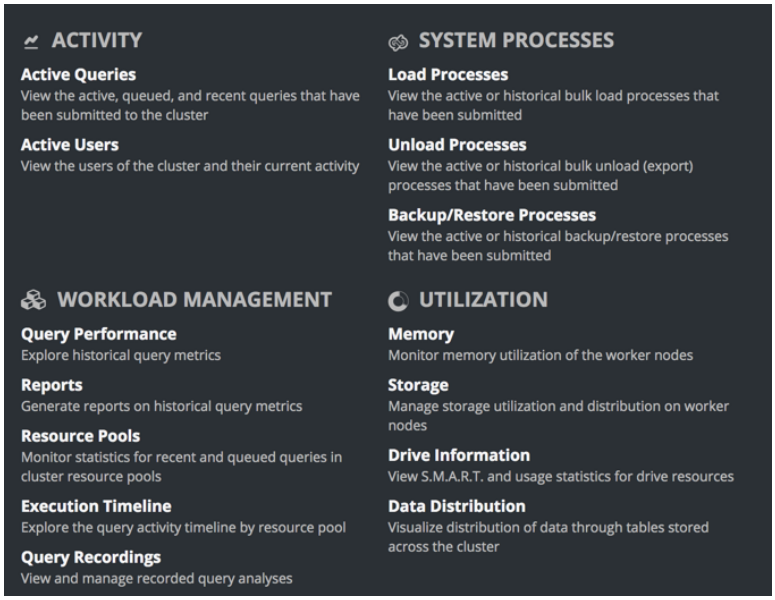
The Dashboard is the first screen you see in the SMC (after logging in). It displays eight different panels that show the current state of the system. These panels are updated dynamically and mainly focus on resource utilization at the cluster and compute node level.

Databases

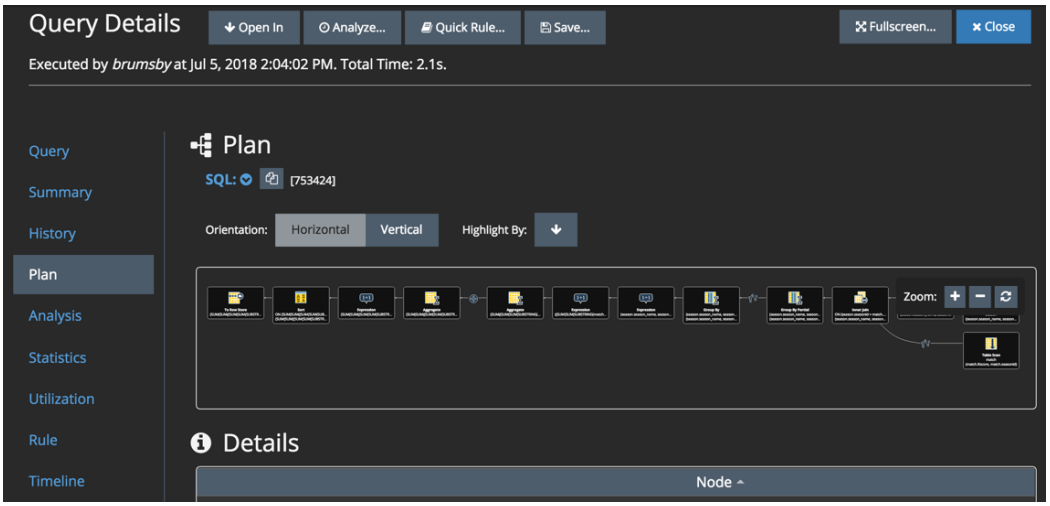
Connect to existing databases and create new databases. Look at database users and groups and the data distribution statistics for tables.

Monitor

Monitor active queries, users, and bulk loads, and monitor resource utilization in terms of memory and storage. Drill down into specific queries to look at performance statistics and query plans. Monitor the behavior of your workload management strategy and activity per resource pool.

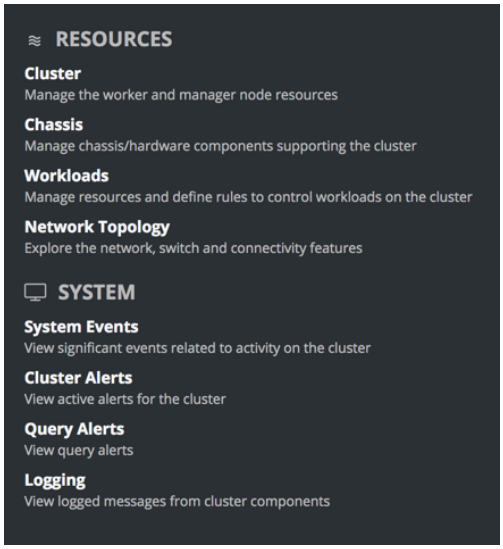


For example, here is the graphical query plan for a specific query that was run by a client user (**Monitor** > **Query Performance** > **Query Details** > **Plan**):



Manage

Manage cluster resources, including the manager and compute nodes, hardware components, and network connectivity. Manage and configure the profiles, resource pools, and rules that define your workload management strategy. Look at system events, alerts, and database logs, using filters to find the information you need.



Configure

Configure system settings.

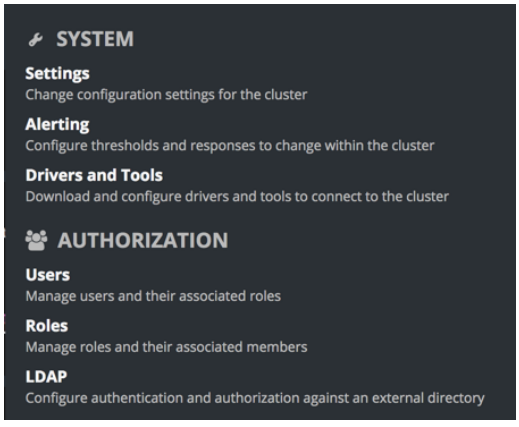
Configure alerts.

Download the following drivers and tools:

- JDBC driver
- ODBC driver
- ADO.NET driver
- `ybtools` (including `ybsql`, `ybload`, and so on)

Manage users and roles, and configure LDAP access.

Note: Yellowbrick Data recommends that you log in to the SMC only with a database `superuser` account that is configured as a member of the Administrator role (`sys_ybd_smc_admin`). Use of the other SMC roles (Manager, Reporter, and Viewer) is not recommended and may result in unpredictable behavior.



Passwords for SMC users must be at least 8 characters long and contain at least one uppercase character, one lowercase character, one numeric character, and one punctuation character. Special characters are also allowed (such as `@` and `$`).

SMC Console Login

To log in to the Yellowbrick System Management Console (SMC):

1. In a browser, go to `https://hostname`

You do not need to specify a port. For example: `https://yb100.mycompany.io`

2. Enter the username and password. On a new system, the only valid initial login is `yellowbrick` for both the username and the password.

Note: The SMC is started automatically as part of the database software stack when the database is started with:

```
ybcli database start
```

See ybcli Reference.

ybcli Reference

Yellowbrick Documentation > How-to Guides > Administer Yellowbrick > Appliance > ybcli Reference

Platforms: EE: All appliance platforms

Parent topic: Appliance

In this section:

The `ybcli` tool is the command-line interface (CLI) for Yellowbrick system management. It is the interface that you enter when you connect to the manager nodes. This CLI is used for managing the appliance at the system and hardware level; it is not for managing the database, except for starting and stopping the database or validating its runtime status.

`ybcli` is used to control and manage system components and is also very useful for system diagnostics, troubleshooting, and general interaction with the Yellowbrick appliance.

Some `ybcli` commands may fail if the hardware required to execute a given command is down.

```
Yellowbrick Data CLI v5.2.12-220217233906 (Tinman)
Copyright (c) 2016-2022 Yellowbrick Data, Inc.
All rights reserved
----------------------------------------
Redundant manager node detected
YBCLI is currently running on the PRIMARY manager node
Local manager node : yb100-mgr0.yellowbrick.io (PRIMARY)
Remote manager node: yb100-mgr1.yellowbrick.io (SECONDARY)

Type 'help' for a list of supported commands

YBCLI (PRIMARY - yb00-mgr0)>
```

The manager node that is currently servicing database operations and queries is considered the *primary manager node*. The other manager node is considered the *secondary manager node*. When you have logged into a manager node, the node where you are currently connected is known as the *local* manager node, and the other node is known as the *remote* manager node.

Most `ybcli` commands are issued against the local manager node (for example, creating users and changing passwords) and as needed, are automatically replicated to the remote node when needed. However, some commands must be run from the primary node (for example, tailing the logs). Some commands (such as `manager status`) allow you to specify the manager node where the command is run (all, local, or remote).

Type `help` for a list of supported commands:

```
YBCLI (PRIMARY - yb00-mgr0)> help
```

```
Available YBCLI commands. Type 'help <command>' for details
========================================================================
blade  config    debug       exit    help      log      quit    system
clear  database  encryption  health  keystore  manager  status
```

For more details, see help.

Tip: You can use tab completion to fill out keywords on the screen and return help information. When you reach the end of a keyword, press the Tab key twice. For example, after typing `system` and a space, press Tab twice to return the following output:

```
YBCLI (PRIMARY - yb00-mgr0)> system
appliance    blackout     diagnostics  factory      failover     maintenance  register     shutdown     status
```

Tip: You can see a history of `ybcli` commands that were run in the current session or a previous session by entering the up and down arrows at the prompt.

ybcli: config

Platforms: EE: All appliance platforms

Parent topic: ybcli Reference

In this section:

Configure or list various system settings, including network settings, time zones, and user accounts.

```
config bmc password [ all | local | remote ] <password>
config bmc log [ show | clear ]
config cert get
config cert create-selfsigned <hostname>
config dba password <username>
config firewall list
config firewall open [ all | HTTP | 80 | HTTPS | 443 | DATABASE [ <port> ] ]
config firewall close [ all | HTTP | 80 | HTTPS | 443 | DATABASE ]
config hostname get
config hostname set [ local | remote ] <hostname>
config network bonding [ all | local | remote ] get
config network bonding [ local |remote ] set [ layer2 | layer2+3 | layer3+4 | active-backup ]
config network bmc [ all | local | remote ] get
config network bmc [ local | remote ] set [ dhcp | <ipaddress> <subnet> <gateway> ]
config network database <port>
config network log disable
config network log enable <ipaddress> | <hostname> [ <port> ]
config network log get
config network manager [ all | local | remote ] get
config network manager [ local | remote ] set <ipaddress> <subnet> <gateway> <dns1> [ <dns2> ] <search-domain>
config network mtu get
config network mtu set <mtu>
config network system get
config network system set <ipaddress> <subnet>
config network route create [ all | local | remote ] default [vlanid] <gateway>
config network route create [ all | local | remote ] [vlanid] <ip> <subnet> <gateway>
config network route delete [ all | local | remote ] default [vlanid]
config network route delete [ all | local | remote ] [vlanid] <ip> <subnet> <gateway>
config network route list [ all | local | remote ]
config network vlan create <vlanid> <ipaddress> <subnet>
config network vlan delete <vlanid>
config network vlan list
config network vlan manager [ all | local | remote ] get <vlanid>
config network vlan manager [ local | remote ] set <vlanid> <ipaddress> <subnet> <gateway|none> <dns1|none> [ <dns2|none> ] <searc
config network vlan system get <vlanid>
config network vlan system set <vlanid> <ipaddress> <subnet>
```

```
config network trace ph <enable|disable>
config network trace get
config network trace set <bulk|sql|all> <enable|disable>
config network trace set <bulk|sql|all> size <MiBs>
config network trace set <bulk|sql|all> duration <minutes>
config network trace set default
config ntp add [ <ipaddress> | <hostname> ]
config ntp delete [ <ipaddress> | <hostname> ]
config ntp disable
config ntp enable
config ntp list
config ntp status
config passwords get
config passwords set [min_length]
config phonehome [ alerts | disabled | full ]
config timezone get
config timezone list
config timezone set <country> <city>
config user create <username>
config user delete <username>
config user key generate
config user key show
config user list
config user password <username>
config user policy <username>
```

For details about these commands, see the following sections.

config bmc

Platforms: EE: All appliance platforms

Parent topic: ybcli: config

Set the BMC password, show hardware event logs, or clear hardware even logs.

```
config bmc password [ all | local | remote ] <password>
config bmc log [ show | clear ]
```

password

Set the BMC password of the local, remote, or all manager nodes for the `BMC ADMIN` user. For example:

```
YBCLI(13077) (PRIMARY - yb00-mgr0)> config bmc password all ybd


Successfully set new BMC ADMIN password

Remote manager node
-------------------


Successfully set new BMC ADMIN password
```

log

Return or clear all low-level hardware event logs from both manager nodes. For example:

```
YBCLI(48888) (PRIMARY - y2b05-mgr1)> config bmc log show

y2b05-mgr1 - HW event log
-----
   1 | 02/24/2020 | 14:32:37 | Unknown #0xff | | Asserted
   2 | 02/24/2020 | 14:33:11 | Physical Security #0xaa | General Chassis intrusion () | Asserted
   3 | 02/24/2020 | 14:44:32 | Physical Security #0xaa | General Chassis intrusion () | Asserted
   4 | 02/24/2020 | 14:44:39 | Temperature #0x28 | Lower Critical going low | Asserted
   5 | 02/24/2020 | 23:53:47 | Unknown #0xff | | Asserted
...
```

For example:

```
YBCLI(274107) (PRIMARY - y2b23-mgr0)> config bmc log clear

Successfully cleared the HW event log of the primary manager node.
Successfully cleared the HW event log of the secondary manager node.
```

ybcli: config cert

Platforms: EE: All appliance platforms

Parent topic: ybcli: config

Get status of the current Yellowbrick Data certificate or, if it is self-signed, change the current Yellowbrick Data certificate.

```
config cert get
config cert create-selfsigned <hostname>
```

get

Return the status of the current Yellowbrick Data certificate. For example, if the certificate is self-signed:

```
YBCLI(46888) (PRIMARY - yb97-mgr0)> config cert get

Current certificate's hostname is yb97.slc.yellowbrick.io
```

For example, if the certificate is not self-signed:

```
YBCLI(47552) (PRIMARY - yb97-mgr0)> config cert get

Certificate is not self-signed.
Subject's hostname is yb97.slc.yellowbrick.io
Issuer's hostname is ybinit_dev_ca
```

create-selfsigned <hostname>

Return the status and, if it is self-signed, change the current Yellowbrick Data certificate. For example:

```
YBCLI(46888) (PRIMARY - yb97-mgr0)> config cert create-selfsigned yb97.yellowbrick.io

Current certificate's hostname is yb97.slc.yellowbrick.io

Changing the certificate's hostname requires an update of the system's DNS server so that its IP is referring to the new host
Are you sure you want to change it (yes/no)? yes

Current certificate's hostname is yb97.yellowbrick.io

Changing the certificate's hostname was successfull.
You need to update your system's DNS server to reflect your latest hostname change.
```

If the certificate is not self-signed, you will receive a message informing you that changing the hostname is not possible. For example:

```
YBCLI(47552) (PRIMARY - yb97-mgr0)> config cert create-selfsigned yb97.slc.yellowbrick.io

Certificate is not self-signed.
Subject's hostname is yb97.slc.yellowbrick.io
Issuer's hostname is ybinit_dev_ca

This certificate does not support changing its hostname.
```

Attempting to change a self-signed certificate to the hostname already in use will return a message informing you that the desired hostname is already in use. For example:

```
YBCLI(46888) (PRIMARY - yb97-mgr0)> config cert create-selfsigned yb97.slc.yellowbrick.io

Current certificate's hostname is yb97.slc.yellowbrick.io

Current certificate is already using the desired hostname
```

ybcli: config dba

Platforms: EE: All appliance platforms

Parent topic: ybcli: config

Change the password of any database user account, except reserved system users (`sys_ybd*`) and `ybdadmin` .

```
config dba password <username>
```

For example:

```
YBCLI(44766) (PRIMARY - yb98-mgr0)> config dba password ybd_test

Enter password for dba user ybd_test:
Password:
Enter password for dba user ybd_test again:
Password:
DBA user password has been updated successfully.

yellowbrick=# \c yellowbrick_test ybd_test
Password for user ybd_test:
psql (9.5.15, server 9.5.2)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)
You are now connected to database "yellowbrick_test" as user "ybd_test".
```

ybcli: config firewall

Platforms: EE: All appliance platforms

Parent topic: ybcli: config

Open or close a specific port on the system, including ports for Yellowbrick services and any custom ports. These commands operate on both manager nodes if both are available.

```
config firewall list
config firewall open all
config firewall open <HTTP | 80>
config firewall open <HTTPS | 443>
config firewall open DATABASE [port]
config firewall open tcp <port>
config firewall open udp <port>
config firewall close all
config firewall close all_custom
config firewall close <HTTP | 80>
config firewall close <HTTPS | 443>
config firewall close DATABASE
config firewall close tcp <port>
config firewall close udp <port>
```

list

List the ports on the system that are currently open or closed, including all the custom ports that are currently open.

open | close

Open or close a specific port, open or close all ports on the system (including custom ports), or close all custom ports only.

The default database port, if not specified in the `open` command, is `5432`. The following values are alternatives for the same port:

- `HTTP` and `80`
- `HTTPS` and `443`

Use the `tcp` and `udp` options to open or close specific custom ports on your system. Use the `all_custom` option to close all custom ports and leave the standard Yellowbrick service ports as they were.

You can use custom port numbers from 1 to 65535, except for the following ports, which are reserved:

- TCP: 22, 80, 443, 2049, 5432, 8182, 9443, 11111, 11112, and 31000-41000
- UDP: 546

Examples

Close the HTTP service port:

```
YBCLI(25344) (PRIMARY - yb100-mgr0)> config firewall close http

Successfully closed firewall for service HTTP

Remote manager node
```

```
------------------

Successfully closed firewall for service HTTP
```

Open the database service port:

```
YBCLI(25344) (PRIMARY - yb100-mgr0)> config firewall open database

No custom database port enabled. Using default port 5432
Successfully opened firewall for service DATABASE (5432)

Remote manager node
------------------

No custom database port enabled. Using default port 5432
Successfully opened firewall for service DATABASE (5432)
```

Open a custom TCP port:

```
YBCLI(25344) (PRIMARY - yb100-mgr0)> config firewall open tcp 65535

Successfully opened firewall port 65535/tcp

Remote manager node
------------------

Successfully opened firewall port 65535/tcp
```

Close a custom TCP port:

```
YBCLI(25344) (PRIMARY - yb100-mgr0)> config firewall close tcp 65535

Successfully closed firewall port 65535/tcp

Remote manager node
------------------

Successfully closed firewall port 65535/tcp
```

List all open and closed ports:

```
YBCLI(2820) (PRIMARY - yb100-mgr0)> config firewall list

The following services are open:        HTTP(80) HTTPS(443) Database(5432)
The following custom ports are open:    107/tcp 115/tcp 162/udp 631/udp
The following services are closed:

Remote manager node
------------------

The following services are open:        HTTP(80) HTTPS(443) Database(5432)
The following custom ports are open:    107/tcp 115/tcp 162/udp 631/udp
The following services are closed:
```

ybcli: config hostname

Platforms: EE: All appliance platforms

Parent topic: ybcli: config

Get or set host names on the manager nodes.

```
config hostname get
config hostname set [ local | remote ] <hostname>
```

get

Return the current manager node hostnames. For example:

```
YBCLI(13325) (PRIMARY - yb100-mgr0)> config hostname get

Local node : yb100-mgr0.ybroad.io
Remote node: yb100-mgr1.ybroad.io
```

set

Change the hostname of the local or remote manager node. For example:

```
YBCLI(13325) (PRIMARY - yb100-mgr0)> config hostname set local yb100-mgr0.ybroad.io

Changing the hostname of a manager node will cause the network stack to be reconfigured.
This may cause some services to be temporarily unavailable.
Are you sure you want to set the hostname of the local node to: yb100-mgr0.ybroad.io
Response (yes/no): yes
Setting the new hostname. Standby...
Restarting HA cluster services. Standby... Done
Waiting for cluster to be updated. Standby... Done
Successfully set hostname to: yb100-mgr0.ybroad.io
```

ybcli: config keystore

Platforms: EE: All appliance platforms

Parent topic: ybcli: config

Unlock and configure keystore for SQL access. Note that you must unlock the keystore file and enter the authentication key to run this command. See ybcli encryption commands.

```
config keystore sql enable
```

For example:

```
YBCLI(12545) (PRIMARY - yb98-mgr0)> config keystore sql enable

Are you sure you want to configure keystore for SQL integration?
Response (yes/no): yes

Authenticating to the keystore for this YBCLI session. Standby...
Please enter keystore authentication key:
Key ->

Key accepted.
SQL keystore has been successfully configured for vault.
```

ybcli: config network

Platforms: EE: All appliance platforms

Parent topic: ybcli: config

View or define the system network configuration.

```
config network bonding [ all | local | remote ] get
config network bonding [ local | remote ] set [ layer2 | layer2+3 | layer3+4 | active-backup ]
config network bmc [ all | local | remote ] get
config network bmc [ local | remote ] set [ dhcp | <ipaddress> <subnet> <gateway> ]
config network database <port>
config network log disable
config network log enable <ipaddress> | <hostname> [ <port> ]
config network log get
config network manager [ all | local | remote ] get
config network manager [ local | remote ] set <ipaddress> <subnet> <gateway> <dns1> [ <dns2> ] <search-domain>
config network mtu get
config network mtu set <mtu>
config network system get
config network system set <ipaddress> <subnet>
config network route create [ all | local | remote ] default [vlanid] <gateway>
config network route create [ all | local | remote ] [vlanid] <ip> <subnet> <gateway>
config network route delete [ all | local | remote ] default [vlanid]
config network route delete [ all | local | remote ] [vlanid] <ip> <subnet> <gateway>
config network route list [ all | local | remote ]
config network trace ph <enable|disable>
config network trace get
config network trace set <bulk|sql|all> <enable|disable>
config network trace set <bulk|sql|all> size <MiBs>
config network trace set <bulk|sql|all> duration <minutes>
config network trace set default
config network vlan create <vlanid>
config network vlan delete <vlanid>
config network vlan list
config network vlan manager [ all | local | remote ] get <vlanid>
config network vlan manager [ local | remote ] set <vlanid> <ipaddress> <subnet> <gateway|none> <dns1|none> [ <dns2|none> ] <searc
config network vlan system get <vlanid>
config network vlan system set <vlanid> <ipaddress> <subnet>
```

bonding

Get or set the network bonding configuration on the manager nodes. For example:

```
YBCLI (PRIMARY - yb00-mgr0)> config network bonding all get

Network bonding configuration:
Active    : layer3, layer4
Configured: layer3, layer4

Remote manager node
-------------------

Network bonding configuration:
Active    : layer3, layer4
Configured: layer3, layer4
```

The `set` command specifies the hash algorithm that allocates network traffic to clients (a network client may receive all its traffic on one interface or via both interfaces). Layer 2, Layer 2+3, Layer 3+4, and active-backup hash policies are supported.

- `layer2`: the default setting. This option uses the `XOR` of hardware MAC addresses to generate the hash. All traffic is placed on a particular network peer on the same network client. This algorithm is 802.3ad-compliant.
- `layer2+3`: a combination of `layer2` and `layer3` protocol information is used to generate the hash. All traffic is placed on a particular network peer on the same network client. For non-IP traffic, the formula is the same as for the `layer2` transmit hash policy. This policy provides a more balanced distribution of traffic than `layer2`, especially in environments where a `layer3` gateway device is required to reach most destinations. This algorithm is 802.3ad-compliant.
- `layer3+4`: this policy uses upper-layer protocol information (when available) to generate the hash. Traffic to a particular network peer may span multiple clients; however, a single connection will not span multiple clients. This algorithm is *not* 802.3ad-compliant.
- `active-backup`: this policy allows redundancy between the ports but not load balancing or increased bandwidth. Only one connection in the bond is active at any time. A different connection becomes active if, and only if, the active connection fails. This mode provides fault tolerance, without needing special support from the connected switch. This algorithm is *not* 802.3ad-compliant or compatible. 802.3ad mode must be disabled on the corresponding ports of the connected switch.

For example:

```
YBCLI (PRIMARY - yb00-mgr0)> config network bonding local set layer2
...
```

bmc

Get or set the BMC configuration on the local or remote manager node. To set the network configuration, enter `dhcp` or specify the IP address. Here is an example of the `get` command:

```
YBCLI (PRIMARY)> config network bmc all get

BMC network configuration
-------------------------
IP address  : 10.20.100.13
Subnet mask : 255.255.0.0
Gateway     : 10.20.0.1

Remote manager node
-------------------

BMC network configuration
-------------------------
IP address  : 10.20.100.13
Subnet mask : 255.255.0.0
Gateway     : 10.20.0.1
```

database <port>

Set the port number that is used for external client access to the database (the default is `5432`).

log disable

Disable sending logs to a remote syslog server. For example:

```
YBCLI(26355) (PRIMARY - yb99-mgr1)> config network log disable

Successfully disabled remote logging on this manager node
Restarting logging... Done
```

log enable <ipaddress> | <hostname> [<port>]

Enable sending the following logs to the specified remote syslog server:

- System log
- Executed `ybcli` commands
- Executed encryption and keystore commands
- All log data from blades and query execution
- User log in/out of system

The port must be 514 or a number greater than 1024 but less than or equal to 65535. Port `udp/514` is the default if a port number is not provided. For example:

```
YBCLI(6350) (PRIMARY - yb99-mgr1)> config network log enable 10.20.100.10 514

Successfully enabled remote logging to 10.20.100.10:514 on this manager node
Restarting logging... Done
```

```
YBCLI(50902) (PRIMARY - y2b23-mgr0)> config network log enable yellowbrick.io

Successfully enabled remote logging to yellowbrick.io:514 on this manager node
Restarting logging... Done
```

log get

Get the IP address and the port used for sending logs to a remote server. For example:

```
YBCLI(6350) (PRIMARY - yb99-mgr1)> config network log get

Remote network logging configured to use:
IP address: 10.20.100.10
Port      : 514
```

manager get | set

Get or set the network configuration for the local or remote manager node. For example:

```
YBCLI(9460) (PRIMARY - yb100-mgr0)> config network manager local get

Manager IP     : 10.20.100.12/255.255.0.0 - Link UP
Manager gateway: 10.20.0.1
Manager DNS    : DNS1: 10.20.12.12 - DNS2: NOT SET
```

mtu get | set

Get or set the network MTU of the customer-facing network adapters. The MTU setting must be between 1500 and 9000, and the specified size that you set must be supported on the network. For example:

```
YBCLI (PRIMARY)> config network mtu get

Customer network MTU: 1500 bytes

Remote manager node
-------------------

Customer network MTU: 1500 bytes
```

```
YBCLI (PRIMARY)> config network mtu set 9000
```

```
Changing the network MTU can be a disruptive process and possibly disconnect both manager nodes from the network if not suppo
Before changing the MTU, please verify the size chosen is supported on this network

Type yes to continue:
...
```

system get | set

Get or set the floating IP address for the system. For example:

```
YBCLI (PRIMARY)> config network system get

System floating IP: 10.30.111.10/24 - Link UP
```

route create [all | local | remote] default [vlanid] <gateway>

Set the default gateway for the network interface. For example:

```
YBCLI(1484) (PRIMARY - yb100-mgr1)> config network route create local default 4000 172.16.155.1

Successfully set default gateway
```

route create [all | local | remote] [vlanid] <IP address> <subnet> <gateway>

Create a static route for the network. For example:

```
YBCLI(1484) (PRIMARY - yb100-mgr1)> config network route create local 4000 172.16.104.1 255.255.255.0 172.16.40.10

Successfully added new static route
```

route delete [all | local | remote] default [vlanid]

Delete the default gateway for the network interface.

route delete [all | local | remote] [vlanid] <IP address> <subnet> <gateway>

Delete a static route for the network. For example:

```
YBCLI(1484) (PRIMARY - yb99-mgr1)> config network route delete local 4000 172.16.104.1 255.255.255.0 172.16.40.10

Successfully deleted static route
```

route list [all | local | remote]

List all default gateways and static routes for each interface. For example:

```
YBCLI(1484) (PRIMARY - yb100-mgr1)> config network route list local

    Main:
        Gateway: 10.20.0.1
        Static Routes: None
    VLAN ID:36:
        Gateway: 172.16.36.221
        Static Routes: 192.168.227.0/255.255.255.0 : 172.16.36.227
                       192.168.217.0/255.255.255.0 : 172.16.36.217
    VLAN ID:4000:
        Gateway: 172.16.155.1
        Static Routes: 172.16.104.1/255.255.255.0 : 172.16.40.10
```

trace ph <enable|disable>

Enable or disable sending network traces to Phonehome. By default, `config network trace ph` is disabled. For example:

```
YBCLI(26269) (PRIMARY - yb00-mgr0)> config network trace ph disable

Network tracing successfully disabled for phonehome (ph)
```

trace get

Get the current configuration for network trace. For example:

```
YBCLI(36333) (PRIMARY - yb98-mgr0)> config network trace get

Network Trace configuration:
       Bulk Ports: TCP[31000-41000] disabled, max 10240 MiB, duration 1440 minutes
       SQL Ports: TCP[5432] disabled, max 10240 MiB, duration 1440 minutes
       Upload to phonehome: disabled
```

trace set <bulk|sql|all> <enable|disable>

Enable or disable network trace for a specified trace type. By default, `config network trace set` is disabled. For example:

```
YBCLI(36333) (PRIMARY - yb98-mgr0)> config network trace set all disable

Network tracing successfully disabled for bulk port

Files are stored in /tmp/network_trace/*network_trace_bulk*.pcap*

Network tracing successfully disabled for SQL port

Files are stored in /tmp/network_trace/*network_trace_sql*.pcap*
```

trace set <bulk|sql|all> size <MiBs>

Set overall trace size for a specified trace type. By default, trace size is 10240 MiB. For example:

```
YBCLI(26269) (PRIMARY - yb00-mgr0)> config network trace set sql size 5120

Network tracing size set successfully for SQL port
```

```
YBCLI(26269) (PRIMARY - yb00-mgr0)> config network trace set all size 5120

Network tracing size set successfully for bulk port

Network tracing size set successfully for SQL port
```

trace set <bulk|sql|all> duration <minutes>

Set the duration for a specified network trace run. By default, traces run for 1440 minutes (24 hours). For example:

```
YBCLI(26269) (PRIMARY - yb00-mgr0)> config network trace set bulk duration 5

Network tracing duration set successfully for bulk port
```

```
YBCLI(26269) (PRIMARY - yb00-mgr0)> config network trace set sql duration 5

Network tracing duration set successfully for SQL port
```

```
YBCLI(26269) (PRIMARY - yb00-mgr0)> config network trace set all duration 5

Network tracing duration set successfully for bulk port

Network tracing duration set successfully for SQL port
```

trace set default

Reset all network trace settings to the default values. For example:

```
YBCLI(36333) (PRIMARY - yb98-mgr0)> config network trace set default

Network tracing successfully set defaults
```

vlan create

Create a VLAN. For example:

```
YBCLI(1484) (PRIMARY - yb100-mgr1)> config network vlan create 1000

Are you sure you want to create a VLAN with ID:1000?
Response (yes/no): yes

Successfully created vlan with ID 1000

Remote manager node
-------------------

Creating VLAN. Standby...

Successfully created vlan with ID 1000
```

vlan delete

Delete a VLAN. For example:

```
YBCLI(1484) (PRIMARY - yb100-mgr1)> config network vlan delete 1000

Successfully deleted vlan with ID 1000

Remote manager node
-------------------

Successfully deleted vlan with ID 1000
```

vlan list

Return a list of current VLANs. For example:

```
YBCLI(26899) (PRIMARY - yb100-mgr1)> config network vlan list

VLAN IDs configured:    1000, 4000, 36
```

```
Remote manager node
-------------------

VLAN IDs configured:    1000, 4000, 36
```

vlan manager [all | local | remote] get

Get the manager node network configuration for the VLAN. For example, get the network configuration on both manager nodes for the VLAN with ID 36:

```
YBCLI(8206) (SECONDARY - yb100-mgr0)> config network vlan manager all get 36

Manager IP     : 172.16.36.12/255.255.255.0 - Link UP
Manager gateway: NOT SET
Manager DNS    : DNS1: 1.1.1.1 - DNS2: NOT SET
Manager Domain : blah.com

Remote manager node
-------------------

Manager IP     : UNKNOWN/UNKNOWN - Link UP
Manager gateway: NOT SET
Manager DNS    : DNS1: NOT SET - DNS2: NOT SET
Manager Domain : NOT SET
```

vlan manager [local | remote] set

Set the manager node network configuration for the VLAN. For example, set the network configuration for a VLAN with ID 1000 on the local manager node:

```
YBCLI(1484) (PRIMARY - yb100-mgr1)> config network vlan manager local set 1000 172.16.200.12 255.255.255.0 172.16.200.1 8.8.8

The local manager node VLAN ID:1000 network will be set to:
IP     : 172.16.200.12
Subnet : 255.255.255.0
Gateway: 172.16.200.1
DNS1   : 8.8.8.8
DNS2   : none
Domain : example.com
Continue (yes/no)? yes

Successfully set new VLAN ID:1000 network configuration

Should the new settings be applied immediately?
Warning: This may cause loss of connectivity if settings of the local system are changed and YBCLI is run remotely.
Typing 'no' will cause settings to take effect on next reboot.
Response (yes/no): yes
Restarting network interface. This can take 20s. Standby...
Network has been restarted with the new configuration
```

vlan system get | set

Get or set the system IP address configuration (cluster floating IP address) for the VLAN. For example:

```
YBCLI(1484) (PRIMARY - yb100-mgr1)> config network vlan system set 1000 172.16.200.10 255.255.255.0

Are you sure you want to set the VLAN ID:1000 floating system IP to: 172.16.200.10/255.255.255.0
Response (yes/no): yes
Verifying IP is not in use on network. Standby...
IP address is available

Creating cluster IP configuration for VLAN ID:1000. Standby...
Successfully created cluster IP configuration for VLAN ID:1000 with address: 172.16.200.10/255.255.255.0
```

```
Waiting for IP address to become active. This can take 90s. Standby... Done
```

```
YBCLI(8206) (SECONDARY - yb100-mgr0)> config network vlan system get 4000

VLAN ID: 4000 System floating IP: UNKNOWN/UNKNOWN - Link UP
YBCLI(8206) (SECONDARY - yb100-mgr0)> config network vlan system get 36

VLAN ID: 36 System floating IP: UNKNOWN/UNKNOWN - Link UP
```

ybcli: config ntp

View or define Network Time Protocol (NTP) servers on the manager nodes.

```
config ntp add [ <ipaddress> | <hostname> ]
config ntp default
config ntp delete [ <ipaddress> | <hostname> ]
config ntp disable
config ntp enable
config ntp list
config ntp status
```

add

Add an NTP server for use by both manager nodes for time synchronization. Then restart NTP services. Use the hostname or IP address to identify the server. For example:

```
YBCLI (PRIMARY)> config ntp add 10.21.41.139

Successfully added 10.21.41.139 as NTP server. Restarting NTP services. Standby...Done

Remote manager node
-------------------

Successfully added 10.21.41.139 as NTP server. Restarting NTP services. Standby...Done
```

default

Set the NTP servers back to system defaults. For example:

```
YBCLI(14689) (PRIMARY - yb00-mgr0)> config ntp default

This will reset the NTP configuration back to default.
Continue (yes/no)? yes
NTP configuration reset to default.

Remote manager node
-------------------

This will reset the NTP configuration back to default.
NTP configuration reset to default.
```

delete

Delete an NTP server from both manager nodes. Then restart NTP services. Use the hostname or IP address to identify the server. For example:

```
YBCLI (PRIMARY)> config ntp delete 10.21.41.139

Removed 10.21.41.139 from NTP configuration. Restarting NTP services. Standby...Done

Remote manager node
```

```
------------------

Removed 10.21.41.139 from NTP configuration. Restarting NTP services. Standby...Done
```

disable

Disable use of NTP time synchronization on both manager nodes.

```
YBCLI (PRIMARY)> config ntp disable

Disabled NTP on this manager node

Remote manager node
------------------

Disabled NTP on this manager node
```

enable

Enable use of NTP time synchronization.

```
YBCLI (PRIMARY)> config ntp enable

Enabled NTP on this manager node

Remote manager node
------------------

Enabled NTP on this manager node
```

list

List the current NTP servers being used by both manager nodes. For example:

```
YBCLI (PRIMARY)> config ntp list

Current NTP servers configured:
Server: 1 -> 0.rhel.pool.ntp.org
Server: 2 -> 1.rhel.pool.ntp.org
Server: 3 -> 2.rhel.pool.ntp.org
Server: 4 -> 3.rhel.pool.ntp.org

Total: 5

Remote manager node
------------------

Current NTP servers configured:
Server: 1 -> 0.rhel.pool.ntp.org
Server: 2 -> 1.rhel.pool.ntp.org
Server: 3 -> 2.rhel.pool.ntp.org
Server: 4 -> 3.rhel.pool.ntp.org

Total: 5
```

status

Check the status of the NTP servers on the manager nodes. For example:

```
YBCLI (PRIMARY)> config ntp status

NTP service is enabled

Remote manager node
-------------------

NTP service is enabled
```

ybcli: config passwords

Configure user password length requirements:

```
config passwords get
config passwords set [min_length]
```

config passwords get

Read current setting for password length requirement. For example:

```
YBCLI(32239) (PRIMARY - yb99-mgr1)> config passwords get

The password length requirement is 14 characters
YBCLI(32239) (PRIMARY - yb99-mgr1)>
```

config passwords set

Change setting for password length requirement. For example:

```
YBCLI(32239) (PRIMARY - yb99-mgr1)> config passwords set

This command configures the global appliance system settings for password length
Do you want to alter these settings?
Continue (yes/no)? yes

Please enter the minimum password lengths required for new users: 10
Set the minimum password length to 10 characters

Remote manager node
-------------------

Set the minimum password length to 10 characters

YBCLI(32239) (PRIMARY - yb99-mgr1)>
```

ybcli: config phonehome

Platforms: EE: All appliance platforms

Parent topic: ybcli: config

Change the phonehome setting for sending alerts.

```
config phonehome [ alerts | disable | full ]
```

where `alerts`, `disable`, and `full` correspond to the Alerts Only, Disabled, and Full service levels for remote diagnostics that you can set in the SMC. See Diagnostics Service Levels.

Changing this setting requires a database restart, as prompted when you enter the `config phonehome` command. For example:

```
YBCLI(19084) (PRIMARY - yb100-mgr0)> config phonehome alerts
WARNING: Changing phonehome setting will restart YBD services.
Continue (yes/no)? yes
Configuration has been successfully updated.
Restarting YBD services to apply the new phonehome configuration...
Stopping YBD services...
 Done
Starting YBD services...
 Done
YBD services have been successfully restarted.
...
```

ybcli: config timezone

Platforms: EE: All appliance platforms

Parent topic: ybcli: config

List the supported time zones, get the currently set time zone, or set a new time zone.

```
config timezone get
config timezone list
config timezone set <country> <city>
```

timezone get

: Return the current time zone that is in effect on both manager nodes.

```
YBCLI (PRIMARY)> config timezone get

Current timezone: America/Los_Angeles

Remote manager node
-------------------

Current timezone: America/Los_Angeles
```

timezone list

: List all supported time zones.

```
YBCLI (PRIMARY)> config timezone list

Africa  Abidjan
Africa  Accra
Africa  Addis_Ababa
Africa  Algiers
...
```

Note that the entries are separated by spaces. When you set the time zone, use `country <space> city`. The list is very long so you may want to save it to a file.

```
[ybdadmin@yb100-mgr0 ~]$ ybcli config timezone list > ybtz.txt
[ybdadmin@yb100-mgr0 ~]$
```

timezone set

: Set the time zone on the manager nodes. When you set the time zone, use a space character, not a backslash, between the country or region and the city name. For example:

`config timezone set Europe Paris`

Changing the time zone requires the database to be stopped and restarted, as shown here:

```
YBCLI (PRIMARY)> config timezone set America Los_Angeles
```

```
Changing the timezone requires the database to be stopped. Are you sure you want to change the timezone?
Note: After this command is run, the database can be started again using the 'database start' command

Type yes to continue: yes

Stopping YBD services for timezone change. Standby...  Done
Successfully set timezone to: America/Los_Angeles

Remote manager node
-------------------

Successfully set timezone to: America/Los_Angeles


YBCLI (PRIMARY)> database status

Database system running        : NO
Database system ready          : N/A
Database system read-only      : NO
Database system uptime         : N/A
Database system users connected: 4 (including system users)

YBCLI (PRIMARY)> database start

Starting Yellowbrick database services. Standby...  Done

YBCLI (PRIMARY)>
```

ybcli: config user

Platforms: EE: All appliance platforms

Parent topic: ybcli: config

Create a user, delete a user, list all users, or change a user's password:

```
config user create <username>
config user delete <username>
config user key generate
config user key show
config user list
config user password <username>
config user policy <username>
```

Note:

- These user accounts are appliance user accounts that can log in to the command line interface (CLI) on the manager nodes. They are not Yellowbrick database users.
- Appliance users are automatically replicated across manager nodes.
- User passwords criteria:
  - must meet user settable minimum length (default=8 character)
  - must contain at least one uppercase letter
  - must contain at least one lowercase letter
  - must contain at least one number
  - must contain at least one special character
  - must not contain the user name in any form
  - must not reuse previous 4 passwords
  - must not be based on a dictionary word

user create

Create a new appliance user. For example:

```
YBCLI(32239) (PRIMARY - yb99-mgr1)> config user create bobr

Should the user have full administrator access at the system level?
Note: This will give the user account full access to all parts of the system and is generally not recommended.
Continue (yes/no)? no

Not enabling full administrator access for this user account.


Successfully created user: bobr

Remote manager node
-------------------


Successfully created user: bobr

Do you want to set a password for this user? This can also be set afterwards with the 'config user password' command
```

```
Continue (yes/no)? yes

Password requirements:
- Must be at least 14 characters long
- Must contain at least one uppercase letter
- Must contain at least one lowercase letter
- Must contain at least one number
- Must contain at least one special character
- Must not contain the user name in any form
- Must not reuse previous 4 passwords
- Must not be based on a dictionary word

Enter password for user bobr:
Password:
Enter password for user bobr again:
Password:

Successfully changed the password for the user bobr on the local manager node

Changing the password on the remote manager node for the user bobr

Successfully changed the password for the user bobr on the remote manager node

Do you want to set the password policy for this user? This can also be set afterwards with the 'config user policy' command
Continue (yes/no)? yes

This command allows setting individual password policies for users on this system

The current values for user bobr are:
   Password expires at:              never
   Password warn time before expire:  never

Do you want to change these values (yes/no)? yes
Please enter the number of days between password change (entering 0 disables this check): 180
Please enter the WARN time reminder (in days) for password changes (entering 0 disables this check): 30
Changing user policy for user bobr

Successfully set the password policy for user bobr

Current values are:
   Password expires at:              Sep 02, 2024
   Password expiration interval:     180 day(s)
   Password warn time before expire: 30 day(s)

Remote manager node
-------------------

Changing user policy for user bobr

Successfully set the password policy for user bobr

Current values are:
   Password expires at:              Sep 02, 2024
   Password expiration interval:     180 day(s)
   Password warn time before expire: 30 day(s)

YBCLI(32239) (PRIMARY - yb99-mgr1)>
```

user delete

Delete a user on both manager nodes. For example:

```
YBCLI(4901) (PRIMARY - yb100-mgr0)> config user delete bobr

Are you sure you want to delete user bobr from the manager nodes?
WARNING: This will delete all the data for this user on both manager nodes
```

```
Response (yes/no): yes
Successfully deleted user: bobr

Remote manager node
-------------------

Successfully deleted user: bobr
```

user key generate

Generate a new ssh public/private key pair for the current user. If the user is `ybdadmin`, key-based authentication will be set up between the manager nodes. For example:

```
YBCLI (PRIMARY)> config user key generate

Are you sure you want to generate a new system key for user ybdadmin?

Type yes to continue: yes

Copying key to remote manager node
Enter current password for the ybdadmin user:
ybdadmin@nnn.nnn.n.n's password:

Created a new ybdadmin key and set up key-based authentication between the manager nodes
```

user key show

Return the current user's public key.

```
YBCLI (PRIMARY)> config user key show

The public key for user: ybdadmin
ssh-rsa ...

Remote manager node
-------------------

The public key for user: ybdadmin
ssh-rsa ...
```

user list

List the users on the manager nodes. For example:

```
YBCLI (PRIMARY)> config user list

Users:
    ybdadmin
    bobr
    bobr2

Remote manager node
-------------------

Users:
    ybdadmin
    bobr
    bobr2
```

user password

Change the password for a user. The change is automatically propogated across both manager nodes. For example:

```
YBCLI(32239) (PRIMARY - yb99-mgr1)> config user password bobr

Password requirements:
- Must be at least 14 characters long
- Must contain at least one uppercase letter
- Must contain at least one lowercase letter
- Must contain at least one number
- Must contain at least one special character
- Must not contains the user name in any form
- Must not reuse previous 4 passwords
- Must not be based on a dictionary word

Enter password for user bobr:
Password:
Enter password for user bobr again:
Password:

Successfully changed the password for the user bobr on the local manager node

Changing the password on the remote manager node for the user bobr

Successfully changed the password for the user bobr on the remote manager node
YBCLI(32239) (PRIMARY - yb99-mgr1)>
```

user policy

Change the policy for password expiration for a user. The change is automatically propogated across both manager nodes. For example:

```
YBCLI(32239) (PRIMARY - yb99-mgr1)> config user policy bobr

This command allows setting individual password policies for users on this system

The current values for user bobr are:
   Password expires at:              Sep 02, 2024
   Password expiration interval:     180 day(s)
   Password warn time before expire: 30 day(s)

Do you want to change these values (yes/no)? yes
Please enter the number of days between password change (entering 0 disables this check): 180
Please enter the WARN time reminder (in days) for password changes (entering 0 disables this check): 10
Changing user policy for user bobr

Successfully set the password policy for user bobr

Current values are:
   Password expires at:              Sep 02, 2024
   Password expiration interval:     180 day(s)
   Password warn time before expire: 10 day(s)

Remote manager node
-------------------

Changing user policy for user bobr

Successfully set the password policy for user bobr

Current values are:
   Password expires at:              Sep 02, 2024
   Password expiration interval:     180 day(s)
   Password warn time before expire: 10 day(s)

YBCLI(32239) (PRIMARY - yb99-mgr1)>
```

ybcli: blade

Platforms: EE: All appliance platforms

Parent topic: ybcli Reference

Run operations on the compute blades.

```
blade add [ <chassis>/<bay> | <uuid> [ [ <chassisN>/<bayN> | <uuidN> ] ]
blade beacon [ <chassis>/<bay> ] [ on | off ]
blade disable [ <chassis>/<bay> | <uuid> ]
blade enable [ <chassis>/<bay> | <uuid> ]
blade erase [ <chassis>/<bay> | <uuid> ]
blade erase clear [ <chassis>/<bay> | <uuid> ]
blade health [ <chassis>/<bay> | all ]
blade poweroff [ <chassis>/<bay> | all ]
blade poweron [ <chassis>/<bay> | all ]
blade reboot [ <chassis>/<bay> | all ]
blade reset  [ <chassis>/<bay> | all ]
blade status [ <chassis>/<bay> | all ]
```

Note: Specify a chassis number (or `all`) and a bay number (or `all`). For example: `2/1`, `all/1`, `all/all`, `1/all` (without spaces). For some commands, you can enter `all` instead of a `chassis/bay` combination. For example:

```
blade health all
```

add

Expand the capacity of the cluster by adding one or more compute blades. Specify each blade by its bay number or its UUID. In general, the cluster must be online and in a healthy operational state when you run this command. Be sure to add blades that are already physically installed in the chassis and fully powered on. If you specify a list of blades to add and one or more of them does not exist or is not powered on, the command will fail for all of the blades. You can run the `status blade all` command to see the current state of the system before running the `blade add` command.

Note: Capacity expansion results in table rows being rewritten and redistributed on the storage system. If any *manual ordering* was applied to the data before the expansion, there is no guarantee that the data will remain sorted in that order. In turn any query performance advantage associated with that order will be lost (such as skipping blocks of data when tables are scanned).

beacon

Turn the beacon LED on or off either for all blades or for a specific blade (identified by its bay number). Use this command to make the blue beacon LED flash on a blade that needs to be swapped out.

disable

Disable a blade so it no longer participates in the system as a compute node. Specify the bay number or the UUID of the blade.

```
YBCLI (PRIMARY)> blade disable 0/10

Are you sure you want to disable blade with UUID: 00000000-0000-0000-0000-38B8EBD00C35

Type yes to continue: yes
```

```
Should the blade be powered off after being disabled?

Type yes to power off blade after disable: no
Blade will not be powered off.
Blade 00000000-0000-0000-0000-38B8EBD00C35 has been disabled.
It takes about 30 seconds to change the blade state to disabled.
```

To see which blades are currently disabled, use the `blade status all` command.

enable

Enable a blade that is currently not in use (make it a compute node). Specify the bay number or the UUID of the blade.

```
YBCLI (PRIMARY)> blade enable 0/10

Blade 00000000-0000-0000-0000-38B8EBD003DE has been enabled
It can take 30s before the blade state changes to enabled
```

If the blade is already provisioned for use, you will see the following message:

```
Blade 00000000-0000-0000-0000-38B8EBD003DE has already been enabled
```

erase

Erase all data from a single specified blade. Specify the bay number or the UUID of the blade. When you run the `blade erase` command, you see a series of warnings and prompts for your protection. Type `yes` at each prompt to continue.

```
YBCLI(37002) (PRIMARY - yb98-mgr0)> blade erase 1/1

Erasing a blade will destroy all data on the blade. The data will not be recoverable
This command will perform a SECURE erase of all drives on the blade
Continue (yes/no)? yes

All data on this blade will now be deleted
Please verify again that you want to complete this operation.
Continue (yes/no)? yes

Should the blade be powered off after being erased?
Response (yes/no): yes
Powering blade off after being erased

Starting to erase blade in chassis: 1 bay: 1
Erasing blade in chassis: 1 bay:  1 ->
       Using Yellowbrick Secure Erase Utility v4.0.0-1168-release
       Retrieving drive details. Standby... Done
       DRIVE-0 - Serial: S3EWNX0JC12534E -> 3 pass erase starting [2019-09-19 13:54:57 host:yb98-mgr1]
              Pass 1 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-88 PURGE and CLEAR
              Pass 2 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-88 PURGE and CLEAR
              Pass 3 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-88 PURGE and CLEAR
              DRIVE-0 - Serial: S3EWNX0JC12534E -> ALL 3 ERASE PASSES SUCCEEDED [2019-09-19 13:54:57 host:yb98-mgr1]
       DRIVE-1 - Serial: S3EWNX0K200853L -> 3 pass erase starting [2019-09-19 13:54:57 host:yb98-mgr1]
              Pass 1 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-88 PURGE and CLEAR
              Pass 2 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-88 PURGE and CLEAR
              Pass 3 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-88 PURGE and CLEAR
              DRIVE-1 - Serial: S3EWNX0K200853L -> ALL 3 ERASE PASSES SUCCEEDED [2019-09-19 13:54:58 host:yb98-mgr1]
       DRIVE-2 - Serial: S3EWNX0K202426R -> 3 pass erase starting [2019-09-19 13:54:58 host:yb98-mgr1]
              Pass 1 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-88 PURGE and CLEAR
              Pass 2 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-88 PURGE and CLEAR
              Pass 3 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-88 PURGE and CLEAR
              DRIVE-2 - Serial: S3EWNX0K202426R -> ALL 3 ERASE PASSES SUCCEEDED [2019-09-19 13:54:59 host:yb98-mgr1]
       DRIVE-3 - Serial: S3EWNX0K200684P -> 3 pass erase starting [2019-09-19 13:54:59 host:yb98-mgr1]
```

```
               Pass 1 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-88 PURGE and CLEAR
               Pass 2 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-88 PURGE and CLEAR
               Pass 3 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-88 PURGE and CLEAR
               DRIVE-3 - Serial: S3EWNX0K200684P -> ALL 3 ERASE PASSES SUCCEEDED [2019-09-19 13:54:59 host:yb98-mgr1]
       DRIVE-4 - Serial: S3EWNX0JC12542B -> 3 pass erase starting [2019-09-19 13:54:59 host:yb98-mgr1]
               Pass 1 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-88 PURGE and CLEAR
               Pass 2 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-88 PURGE and CLEAR
               Pass 3 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-88 PURGE and CLEAR
               DRIVE-4 - Serial: S3EWNX0JC12542B -> ALL 3 ERASE PASSES SUCCEEDED [2019-09-19 13:55:00 host:yb98-mgr1]
       DRIVE-5 - Serial: S3EWNX0K100445J -> 3 pass erase starting [2019-09-19 13:55:00 host:yb98-mgr1]
               Pass 1 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-88 PURGE and CLEAR
               Pass 2 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-88 PURGE and CLEAR
               Pass 3 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-88 PURGE and CLEAR
               DRIVE-5 - Serial: S3EWNX0K100445J -> ALL 3 ERASE PASSES SUCCEEDED [2019-09-19 13:55:00 host:yb98-mgr1]
       DRIVE-6 - Serial: S3EWNX0K100372P -> 3 pass erase starting [2019-09-19 13:55:00 host:yb98-mgr1]
               Pass 1 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-88 PURGE and CLEAR
               Pass 2 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-88 PURGE and CLEAR
               Pass 3 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-88 PURGE and CLEAR
               DRIVE-6 - Serial: S3EWNX0K100372P -> ALL 3 ERASE PASSES SUCCEEDED [2019-09-19 13:55:01 host:yb98-mgr1]
       DRIVE-7 - Serial: S3EWNX0JC12602K -> 3 pass erase starting [2019-09-19 13:55:01 host:yb98-mgr1]
               Pass 1 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-88 PURGE and CLEAR
               Pass 2 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-88 PURGE and CLEAR
               Pass 3 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-88 PURGE and CLEAR
               DRIVE-7 - Serial: S3EWNX0JC12602K -> ALL 3 ERASE PASSES SUCCEEDED [2019-09-19 13:55:02 host:yb98-mgr1]
       All 8 drives on blade in chassis: 1 bay:  1 successfully erased with 3 passes of NIST 800-88 PURGE and CLEAR [2019-09-
Blade has been erased
Powering off blade. Standby...
Blade(s) in chassis: 1 were instructed to power off
Waiting for blade(s) to power off
Blades powered off: 0/1
Blades powered off: 1/1
Blade will have to be reset to participate in the cluster.
```

The `blade erase` command cannot be run on the secondary manager node.

If you need to erase data from the whole system, you can use the `system factory` command.

erase clear

Erase all data from a single specified blade through the block-by-block writing of zeros in addition to the standard secure erase process of `blade erase`. Specify the bay number or the UUID of the blade.

Note that the blade erase clear command must be run in a screen or tmux session because the process takes longer to complete than the `blade erase` command. Additionally, the `blade erase clear` command cannot be run on the secondary manager node. When you run the `blade erase clear` command, you see a series of warnings and prompts for your protection. Type `yes` at each prompt to continue.

If you need to erase data from the entire system, you can use the `system factory` command.

```
YBCLI(64318) (PRIMARY - yb98-mgr0)> blade erase clear 0/1

Execution Start: Thu Dec  5 14:30:14 PST 2019
Erasing a blade will destroy all data on the blade. The data will not be recoverable
Clear erase will destroy all data on the blade by writing zeros to all bytes on the drive
This command will perform a SECURE erase of all drives on the blade
Continue (yes/no)? yes

All data on this blade will now be deleted
Please verify again that you want to complete this operation.
Continue (yes/no)? yes

Should the blade be powered off after being erased?
Response (yes/no): yes
Powering blade off after being erased
```

```
Starting to erase blade in chassis: 0 bay: 1
Erasing blade in chassis: 0 bay:  1 ->  Using Yellowbrick Secure Erase Utility v3.3.0-20502-release
        Retrieving drive details. Standby... Done
        Affected Inventory of Disks found on the Yellowbrick Appliance host:yb98-mgr0:
        Chassis 0 bay: 1
        DRIVE-0 - Serial: S469NF0K510756B
        DRIVE-1 - Serial: S469NF0K510762E
        DRIVE-2 - Serial: S469NF0K501083Z
        DRIVE-3 - Serial: S469NF0K501067B
        DRIVE-4 - Serial: S469NF0K501097L
        DRIVE-5 - Serial: S469NF0K510767N
        DRIVE-6 - Serial: S469NF0K510759X
        DRIVE-7 - Serial: S469NF0K501064H

        Clear erasing drives, it may take an extended amount of time. Erasing...
        DRIVE-0, Chassis 0, Bay 1 - Serial: S469NF0K510756B -> 3 pass erase starting [2019-12-05 14:30:27 host:yb98-mgr0]
                Pass 1 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-88 PURGE and CLEAR
                  512110190592 bytes written to Drive 0 (Clear) completed at 2019-12-05 14:49:34
                  Cryptographic Key Delete and Disk Erase (Purge) completed at 2019-12-05 14:49:34
                Pass 2 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-88 PURGE and CLEAR
                  1024220381184 bytes written to Drive 0 (Clear) completed at 2019-12-05 15:11:51
                  Cryptographic Key Delete and Disk Erase (Purge) completed at 2019-12-05 15:11:51
                Pass 3 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-88 PURGE and CLEAR
                  1536330571776 bytes written to Drive 0 (Clear) completed at 2019-12-05 15:30:07
                  Cryptographic Key Delete and Disk Erase (Purge) completed at 2019-12-05 15:30:07
                DRIVE-0 - Serial: S469NF0K510756B -> ALL 3 ERASE PASSES SUCCEEDED [2019-12-05 15:32:38 host:yb98-mgr0]
        DRIVE-1, Chassis 0, Bay 1 - Serial: S469NF0K510762E -> 3 pass erase starting [2019-12-05 14:30:27 host:yb98-mgr0]
                Pass 1 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-88 PURGE and CLEAR
                  512110190592 bytes written to Drive 1 (Clear) completed at 2019-12-05 14:48:15
                  Cryptographic Key Delete and Disk Erase (Purge) completed at 2019-12-05 14:48:15
                Pass 2 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-88 PURGE and CLEAR
                  1024220381184 bytes written to Drive 1 (Clear) completed at 2019-12-05 15:07:40
                  Cryptographic Key Delete and Disk Erase (Purge) completed at 2019-12-05 15:07:40
                Pass 3 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-88 PURGE and CLEAR
                  1536330571776 bytes written to Drive 1 (Clear) completed at 2019-12-05 15:29:07
                  Cryptographic Key Delete and Disk Erase (Purge) completed at 2019-12-05 15:29:07
                DRIVE-1 - Serial: S469NF0K510762E -> ALL 3 ERASE PASSES SUCCEEDED [2019-12-05 15:32:38 host:yb98-mgr0]
        DRIVE-2, Chassis 0, Bay 1 - Serial: S469NF0K501083Z -> 3 pass erase starting [2019-12-05 14:30:27 host:yb98-mgr0]
                Pass 1 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-88 PURGE and CLEAR
                  512110190592 bytes written to Drive 2 (Clear) completed at 2019-12-05 14:47:57
                  Cryptographic Key Delete and Disk Erase (Purge) completed at 2019-12-05 14:47:57
                Pass 2 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-88 PURGE and CLEAR
                  1024220381184 bytes written to Drive 2 (Clear) completed at 2019-12-05 15:07:56
                  Cryptographic Key Delete and Disk Erase (Purge) completed at 2019-12-05 15:07:56
                Pass 3 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-88 PURGE and CLEAR
                  1536330571776 bytes written to Drive 2 (Clear) completed at 2019-12-05 15:30:26
                  Cryptographic Key Delete and Disk Erase (Purge) completed at 2019-12-05 15:30:26
                DRIVE-2 - Serial: S469NF0K501083Z -> ALL 3 ERASE PASSES SUCCEEDED [2019-12-05 15:32:38 host:yb98-mgr0]
        DRIVE-3, Chassis 0, Bay 1 - Serial: S469NF0K501067B -> 3 pass erase starting [2019-12-05 14:30:27 host:yb98-mgr0]
                Pass 1 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-88 PURGE and CLEAR
                  512110190592 bytes written to Drive 3 (Clear) completed at 2019-12-05 14:55:40
                  Cryptographic Key Delete and Disk Erase (Purge) completed at 2019-12-05 14:55:40
                Pass 2 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-88 PURGE and CLEAR
                  1024220381184 bytes written to Drive 3 (Clear) completed at 2019-12-05 15:17:39
                  Cryptographic Key Delete and Disk Erase (Purge) completed at 2019-12-05 15:17:39
                Pass 3 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-88 PURGE and CLEAR
                  1536330571776 bytes written to Drive 3 (Clear) completed at 2019-12-05 15:32:05
                  Cryptographic Key Delete and Disk Erase (Purge) completed at 2019-12-05 15:32:05
                DRIVE-3 - Serial: S469NF0K501067B -> ALL 3 ERASE PASSES SUCCEEDED [2019-12-05 15:32:38 host:yb98-mgr0]
        DRIVE-4, Chassis 0, Bay 1 - Serial: S469NF0K501097L -> 3 pass erase starting [2019-12-05 14:30:27 host:yb98-mgr0]
                Pass 1 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-88 PURGE and CLEAR
                  512110190592 bytes written to Drive 4 (Clear) completed at 2019-12-05 14:47:54
                  Cryptographic Key Delete and Disk Erase (Purge) completed at 2019-12-05 14:47:54
                Pass 2 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-88 PURGE and CLEAR
                  1024220381184 bytes written to Drive 4 (Clear) completed at 2019-12-05 15:08:15
                  Cryptographic Key Delete and Disk Erase (Purge) completed at 2019-12-05 15:08:15
                Pass 3 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-88 PURGE and CLEAR
```

```
                  1536330571776 bytes written to Drive 4 (Clear) completed at 2019-12-05 15:28:43
                  Cryptographic Key Delete and Disk Erase (Purge) completed at 2019-12-05 15:28:43
                DRIVE-4 - Serial: S469NF0K501097L -> ALL 3 ERASE PASSES SUCCEEDED [2019-12-05 15:32:38 host:yb98-mgr0]
        DRIVE-5, Chassis 0, Bay 1 - Serial: S469NF0K510767N -> 3 pass erase starting [2019-12-05 14:30:27 host:yb98-mgr0]
                Pass 1 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-88 PURGE and CLEAR
                  512110190592 bytes written to Drive 5 (Clear) completed at 2019-12-05 14:53:11
                  Cryptographic Key Delete and Disk Erase (Purge) completed at 2019-12-05 14:53:11
                Pass 2 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-88 PURGE and CLEAR
                  1024220381184 bytes written to Drive 5 (Clear) completed at 2019-12-05 15:17:58
                  Cryptographic Key Delete and Disk Erase (Purge) completed at 2019-12-05 15:17:58
                Pass 3 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-88 PURGE and CLEAR
                  1536330571776 bytes written to Drive 5 (Clear) completed at 2019-12-05 15:32:38
                  Cryptographic Key Delete and Disk Erase (Purge) completed at 2019-12-05 15:32:38
                DRIVE-5 - Serial: S469NF0K510767N -> ALL 3 ERASE PASSES SUCCEEDED [2019-12-05 15:32:38 host:yb98-mgr0]
        DRIVE-6, Chassis 0, Bay 1 - Serial: S469NF0K510759X -> 3 pass erase starting [2019-12-05 14:30:27 host:yb98-mgr0]
                Pass 1 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-88 PURGE and CLEAR
                  512110190592 bytes written to Drive 6 (Clear) completed at 2019-12-05 14:48:43
                  Cryptographic Key Delete and Disk Erase (Purge) completed at 2019-12-05 14:48:43
                Pass 2 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-88 PURGE and CLEAR
                  1024220381184 bytes written to Drive 6 (Clear) completed at 2019-12-05 15:07:40
                  Cryptographic Key Delete and Disk Erase (Purge) completed at 2019-12-05 15:07:40
                Pass 3 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-88 PURGE and CLEAR
                  1536330571776 bytes written to Drive 6 (Clear) completed at 2019-12-05 15:28:51
                  Cryptographic Key Delete and Disk Erase (Purge) completed at 2019-12-05 15:28:51
                DRIVE-6 - Serial: S469NF0K510759X -> ALL 3 ERASE PASSES SUCCEEDED [2019-12-05 15:32:38 host:yb98-mgr0]
        DRIVE-7, Chassis 0, Bay 1 - Serial: S469NF0K501064H -> 3 pass erase starting [2019-12-05 14:30:27 host:yb98-mgr0]
                Pass 1 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-88 PURGE and CLEAR
                  512110190592 bytes written to Drive 7 (Clear) completed at 2019-12-05 14:48:05
                  Cryptographic Key Delete and Disk Erase (Purge) completed at 2019-12-05 14:48:05
                Pass 2 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-88 PURGE and CLEAR
                  1024220381184 bytes written to Drive 7 (Clear) completed at 2019-12-05 15:07:48
                  Cryptographic Key Delete and Disk Erase (Purge) completed at 2019-12-05 15:07:48
                Pass 3 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-88 PURGE and CLEAR
                  1536330571776 bytes written to Drive 7 (Clear) completed at 2019-12-05 15:28:51
                  Cryptographic Key Delete and Disk Erase (Purge) completed at 2019-12-05 15:28:51
                DRIVE-7 - Serial: S469NF0K501064H -> ALL 3 ERASE PASSES SUCCEEDED [2019-12-05 15:32:38 host:yb98-mgr0]
        All 8 drives on blade in chassis: 0 bay:  1 successfully erased with 3 passes of Secure erase + format [2019-12-05 15

Summary of Yellowbrick Secure Erase Utility 2019-12-05 14:30:27
8 Drives Erased Successfully with NIST 800-88 PURGE and CLEAR
0 Drives Erased Failed
Blade has been erased
Powering off blade. Standby...
Blade(s) in chassis: 0 were instructed to power off
Waiting for blade(s) to power off
Blades powered off: 0 of 1
Blades powered off: 1 of 1
Blade will have to be reset to participate in the cluster.
```

health

Check the hardware health and boot status of one or all blades. For example:

```
YBCLI (PRIMARY)> blade health 0/10

Chassis:  0
-----------
Bay: 10 HW Status: ok    Power: on LED: [ OFF  ON   OFF  ] FRU: C4-0013-00 R01 Serial: TAA18040900C35 Temp: 49C CPU: Booted (

Retrieving blade alerts...

Blade alerts reported: None
```

Note: The `health blade` command provides the same information.

If a blade is powered off, it returns:

```
YBCLI (PRIMARY)> blade health 0/15

Bay: 15 Status: off
```

poweroff, poweron

: Power on or power off one blade or all blades. For example, power off blade 15:

```
YBCLI (PRIMARY)> blade poweroff 15

Blade in bay 15 was instructed to power off
Waiting for blade(s) to power off
Blades powered off: 0/1
Blades powered off: 1/1
```

When a blade is powered on, it passes through three states to come online:

- CPU: Powering up/Self-test
- CPU: Booting YBOS
- CPU: Booted (YBOS)

The `blade health` command reports these states. This process may take a few minutes. An attempt to reboot a blade before it is in the `Booted (YBOS)` state will fail. For example:

```
YBCLI (PRIMARY)> blade poweron 15

Blade in bay 15 instructed to power on
Waiting for blade(s) to power on
Blades powered on: 0/1
Blades powered on: 1/1

YBCLI (PRIMARY)> blade health 15

Bay: 15 Status: ok Power: on FRU: C4-0005-00     Serial: NOT READ YET   CPU: Powering up/Self-test
```

reboot

Reboot one blade or all blades. This operation is a graceful reboot; the Yellowbrick software stack is shut down first.

```
YBCLI (PRIMARY)> blade reboot all

Gracefully rebooting blade in bay:  1 -> OK
Gracefully rebooting blade in bay:  2 -> OK
Gracefully rebooting blade in bay:  3 -> OK
Gracefully rebooting blade in bay:  4 -> OK
Gracefully rebooting blade in bay:  5 -> OK
...
```

reset

Perform an immediate hardware reset on one blade or all blades.

```
YBCLI (PRIMARY)> blade reset 15

Blade in bay 15 was reset
```

status

Return the UUID, current status, CPU and memory usage, uptime, encryption status, and drive status/wear for one or all blades. For example, when run on Tinman:

```
YBCLI(8911) (PRIMARY - yb100-mgr0)> blade status 1/3

Chassis:  1
-----------
Blade Bay:  3 -> BOOTED  UUID: 00000000-0000-0000-0000-38B8EBD006F4 - Version: YBOS-2.0.4-DEBUG
         BIOS: v05.04.21.0038.00.011 - Memory total/free: 65587652/1870728 KiB - IB-FW: 2.36.5000
         CPU: Intel(R) Xeon(R) CPU E5-2618L v4 @ 2.20GHz - Cores: 10 - Load: 1%
         Address: 192.168.20.12 - Uptime: 0 day(s), 01:59:29 - Worker: Running
         Encryption Supported: YES - Encryption Enabled: NO - locked: N/A
               SSD: 4: S3EWNWAJ312492R    5: S3EWNWAJ330205H    6: S3EWNWAJ312493K    7: S3EWNWAJ330206Z
                       OK/0% 2B6QCXP7        OK/0% 2B6QCXP7        OK/0% 2B6QCXP7        OK/0% 2B6QCXP7
                    0: S3EWNWAJ311166N    1: S3EWNWAJ330207R    2: S3EWNWAJ311169V    3: S3EWNWAJ330208K
                       OK/0% 2B6QCXP7        OK/0% 2B6QCXP7        OK/0% 2B6QCXP7        OK/0% 2B6QCXP7
         Cluster status: OPERATIONAL - Cluster role: MEMBER - Last seen: just now
```

For example, when run on Andromeda:

```
YBCLI(416590) (PRIMARY - y2b21-mgr1)> blade status 0/1

Chassis:  0
-----------
Blade Bay:  1 -> YBOS READY UUID: 00000000-0000-0000-0000-38B8EBD20E88 - Version: YBOS-3.0.0-20200917181022-DEBUG
         BIOS: Athn_1.01.3 - Memory total/free: 1056744476/18735500 KiB
         BMC: 00.17.01 - Kalidah-1: 0x0DE38309 - Kalidah-2: 0x0DE38309 - IB-FW: 16.28.1002
         CPU: AMD EPYC 7702P 64-Core Processor - Microcode: 0x8301034 - Cores: 64 - Load: 1%
         Address: 192.168.10.10 - Uptime: 0 day(s), 00:15:14 - Worker: Running
         Encryption Supported: YES - Encryption Enabled: NO - locked: N/A
               SSD: 4: S439NE0MA00440     5: S439NE0MA00287     6: S439NE0MA00271     7: S439NE0MA00284
                       OK/0% EDA5502Q        OK/0% EDA5502Q        OK/0% EDA5502Q        OK/0% EDA5502Q
                    0: S439NE0MA00442     1: S439NE0MA00253     2: S439NE0MA00264     3: S439NE0MA00288
                       OK/0% EDA5502Q        OK/0% EDA5502Q        OK/0% EDA5502Q        OK/0% EDA5502Q
         Cluster status: OPERATIONAL - Cluster role: MEMBER - Last seen: just now
```

Note: If the amount of memory installed on the blades varies (based on the amount reported for the first blade), you will see red text for the memory information and a warning message at the end of the output. You will also see a warning if the BIOS is being flashed on one or more blades. These blades will automatically reset when BIOS flashing is complete.

The `status blade` command provides the same information.

If the database is not running, the output does not include cluster status information.

This example shows the status of one bay that does not have a blade installed:

```
YBCLI (PRIMARY)> blade status 1/14

Chassis:  1
-----------
Blade Bay: 14 -> BLADE NOT INSTALLED
```

ybcli: clear

Clear the screen where `ybcli` is running or clear the history of previously issued commands. For example, you may want to clear the history if you inadvertently paste an encryption key at the `ybcli` prompt.

```
clear [ screen ] | history
```

For example:

```
YBCLI (PRIMARY)> clear

YBCLI (PRIMARY)>
```

The `screen` keyword is optional; the `clear` command defaults to `clear screen`.

ybcli: database

Platforms: EE: All appliance platforms

Parent topic: ybcli Reference

Start or stop the Yellowbrick database software. These commands do not start up or shut down the system. The manager and compute nodes are not affected.

You can also see the current status of the database, which returns the same information as the `status database` command.

```
database start [ force ]
database status
database stop  [ force ]
```

For example:

```
YBCLI (PRIMARY)> database stop

This will stop the Yellowbrick database and disconnect all database clients
Are you sure you want to do this?
Type yes to continue: yes

Stopping Yellowbrick Database services. Standby... Done
```

```
YBCLI (PRIMARY)> database start

Starting Yellowbrick database services. Standby... Done
```

You can use the `force` option when the `ybcli` cannot detect whether the database is running or not. This may occur when there is a malfunction in the system.

ybcli: encryption

Platforms: EE: All appliance platforms

Parent topic: ybcli Reference

Enable and manage encryption for the drives on the manager nodes and compute blades.

```
encryption disable
encryption enable
encryption rotate
encryption status
encryption unlock
```

Encryption commands prompt for the authentication key that was generated when you set up the keystore. You may not see a prompt if you have already authenticated and the session has not timed out (it times out after 10 minutes or when you exit from the `ybcli`). Encryption commands also require the keystore to be unlocked.

Note: Encryption commands can only be run via the `ybcli` on the primary manager node.

encryption disable

Disable system-wide encryption on all drives, starting with the manager nodes, then proceeding to the compute blades. This command detects the drives that have encryption enabled and disables encryption only on those drives.

This command also detects the case where a drive key is missing from the keystore. If this occurs, you can try restoring the keystore from a backup, then run the disable command again.

```
YBCLI (PRIMARY)> encryption disable

Are you sure you want to disable encryption system wide?
WARNING: This will unlock all drives, disable encryption and delete the keys from the keystore.

Disabling encryption will take the database stack down.

Type yes to continue: yes
WARNING: The keystore is locked. Initiating unlocking
Please enter an unlock key to begin unlocking the keystore:
Key ->

Verifiying keystore status. Standby...
Keystore locked: NO

Keystore was successfully unlocked

Stopping YBD services before disabling encryption. Standby... Done

Disabling encryption on manager node(s). Standby...

Manager: yb00-mgr1.yellowbrick.io (local node)
    Drive-sda - Serial: S3D2NX0J507797Y -> Disabling encryption and removing key. Standby...Encryption disabled
    Drive-sdb - Serial: S3D2NX0J507787F -> Disabling encryption and removing key. Standby...Encryption disabled

...

Disabling encryption on blades. Standby...
Retrieving blade details. Standby...
```

```
Blade in bay: 1 - UUID: 00000000-0000-0000-0000-38B8EBD004F6
    Drive-0 - Serial: S3EWNWAJ318665L -> Disabling encryption and removing key. Standby...Encryption disabled
    Drive-1 - Serial: S3EWNWAJ318656W -> Disabling encryption and removing key. Standby...Encryption disabled
    Drive-2 - Serial: S3EWNWAJ318660F -> Disabling encryption and removing key. Standby...Encryption disabled
    Drive-3 - Serial: S3EWNWAJ318658M -> Disabling encryption and removing key. Standby...Encryption disabled
    Drive-4 - Serial: S3EWNWAJ318653X -> Disabling encryption and removing key. Standby...Encryption disabled
    Drive-5 - Serial: S3EWNWAJ318652L -> Disabling encryption and removing key. Standby...Encryption disabled
    Drive-6 - Serial: S3EWNWAJ318657P -> Disabling encryption and removing key. Standby...Encryption disabled
    Drive-7 - Serial: S3EWNWAJ318651N -> Disabling encryption and removing key. Standby...Encryption disabled

Blade in bay: 2 - UUID: 00000000-0000-0000-0000-38B8EBD0053C
    Drive-0 - Serial: S3EWNWAJ316127E -> Disabling encryption and removing key. Standby...Encryption disabled
    Drive-1 - Serial: S3EWNWAJ330178F -> Disabling encryption and removing key. Standby...Encryption disabled
    Drive-2 - Serial: S3EWNWAJ330176R -> Disabling encryption and removing key. Standby...Encryption disabled
...
```

encryption enable

Enable system-wide encryption on all drives. The keystore must be set up and unlocked, and you must enter the authentication key. When you run this command, it detects the drives that require encryption to be enabled and proceeds.

This command may fail for any system components that are not online. For example, if a compute blade is not booted, encryption will not be enabled on its drives. After booting the drive, you can run the command again.

This command returns detailed output per blade and drive, indicating whether encryption is enabled. See Example: Set Up the Keystore and Enable Encryption.

encryption rotate

Regenerate and rotate all encryption keys on all drives. Rotating keys on the drives requires the database to be shut down. You may want to run this command periodically to improve the security of the system. (This command does not generate new unlock and authentication keys for the keystore; see the `keystore rotate` command.) For example:

```
YBCLI (PRIMARY)> encryption rotate

WARNING: This command will rotate all keys of any encrypted drive in the system

Type yes to continue: yes

Do you want this command to show the actual drive keys generated?
WARNING: The keys will be shown in clear text. Saying anything but yes below will not display the keys.

Type yes to show drive keys: yes

Authenticating to the keystore for this YBCLI session. Standby...
Please enter keystore authentication key:
Password:

Key accepted. This session is authenticated for the next 10 minutes or until YBCLI exit.

Rotating keys on manager node(s). Standby...

Manager: yb00-mgr0.yellowbrick.io (local node)
    Drive-sda - Serial: S3D2NX0J601378A -> Generating key. Standby... KEY: TxXlBKj+E5N2QSoihv43pvynC7X+2BC0
        Rotating key. Standby...Rotated
    Drive-sdb - Serial: S3D2NX0J601380W -> Generating key. Standby... KEY: ZP2YumewQta+5kyIxWa1sYtVrY28OMaT
        Rotating key. Standby...Rotated

Manager: yb00-mgr1.yellowbrick.io (remote node)
    Drive-sda - Serial: S3D2NX0J507797Y -> Generating key. Standby... KEY: 84E3D+FbWShDszJfpSFR1z47rXEUzssn
        Rotating key. Standby...Rotated
    Drive-sdb - Serial: S3D2NX0J507787F -> Generating key. Standby... KEY: HmdiE0pk5K2R63bNmeolnKL97DBlzPF7
        Rotating key. Standby...Rotated
```

```
Rotating keys on blades. Standby...
...
```

encryption status

Return the encryption status of all of the drives on the manager nodes and the blades. After encryption is enabled, all of the drives should report that they are enabled and unlocked. For example:

```
YBCLI (PRIMARY)> encryption status

Do you want this command to show the actual drive keys?
WARNING: The keys will be shown in clear text. Saying anything but yes below will not display the keys

Type yes to show drive keys: no
Not displaying keys in clear text
WARNING: The keystore is locked. Initiating unlocking
Please enter a keystore unlock key to begin unlocking the keystore:
Password:

The keystore is not yet fully unlocked.
Do you want to continue entering another key?
Note: Keystore unlocking is stateful and the remaining key(s) can be provided at a later time.
If the system is restarted or failed over, all keys will have to be provided again.

Type yes to continue: yes

Please enter another keystore unlock key to unlock the keystore (Progress: 1/2):
Password:

Verifiying keystore status. Standby...
Keystore locked: NO

Keystore was successfully unlocked

Retrieving manager node encryption status. Standby...

Local Manager drive encryption status:
    Supports encryption: 2/2
    Encryption enabled : 2/2
    Drives locked      : 0/2
        DRIVE-sda - Serial: S3D2NX0J507797Y -> KEY PRESENT
        DRIVE-sdb - Serial: S3D2NX0J507787F -> KEY PRESENT


Retrieving blade encryption status. Standby...
Retrieving blade details. Standby...

Blade in bay: 1 - UUID: 00000000-0000-0000-0000-38B8EBD00483
    Supports encryption: 8/8
    Encryption enabled : 8/8
    Drives locked      : 0/8
        DRIVE-0 - Serial: S3EWNWAJ318665L -> KEY PRESENT
        DRIVE-1 - Serial: S3EWNWAJ318656W -> KEY PRESENT
        DRIVE-2 - Serial: S3EWNWAJ318660F -> KEY PRESENT
        DRIVE-3 - Serial: S3EWNWAJ318658M -> KEY PRESENT
        DRIVE-4 - Serial: S3EWNWAJ318653X -> KEY PRESENT
        DRIVE-5 - Serial: S3EWNWAJ318652L -> KEY PRESENT
        DRIVE-6 - Serial: S3EWNWAJ318657P -> KEY PRESENT
        DRIVE-7 - Serial: S3EWNWAJ318651N -> KEY PRESENT
...
```

encryption unlock

Unlock all locked drives on the system. If a blade (compute node) has one or more drives that are locked, that compute node cannot start. This command prompts for the authentication key, then unlocks the drives by using the keys in the keystore. This command also starts the database as part of its normal operation. A warning message is displayed before the database is started.

If the manager node drives are locked when the system is powered on, start the `ybcli`. The `ybcli` will detect that the drives are locked and automatically return a prompt, asking you to continue by attempting to unlock the drives with the keys in the keystore. You will not be able to start the database until the keys are unlocked.

If all of the drives on a manager node are locked, it cannot become the primary manager node.

ybcli: exit, quit

Platforms: EE: All appliance platforms

Parent topic: ybcli Reference

Exit the `ybcli` session. The `exit` and `quit` commands are synonymous.

```
YBCLI (PRIMARY)> exit
Exiting. Standby...
Bye

YBCLI (PRIMARY)> quit

Exiting. Standby...
Bye
```

ybcli: health

Platforms: EE: All appliance platforms

Parent topic: ybcli Reference

Retrieve health-related information for various system components, including any hardware-level alerts from those components.

```
health all
health blade [ <chassis>/<bay> ]
health cmp [ <chassis>/<cmp> ]
health fan [ <chassis>/<fan> | all ]
health fan chassis [ <chassis>/<fan> | all ]
health fan manager [ local | remote | all ]
health network
health psu [ <chassis>/<psu> | all ]
health psu chassis [ <chassis>/<psu> | all ]
health psu manager [ local | remote | all ]
health storage
```

Note: Specify a chassis number (or `all`) and a bay/cmp/psu/fan number (or `all`). For example: `0/1`, `all/1`, `all/all`, `1/all` (without spaces).

all

Return health information for all of the components on all chassis: blades, fans, power supplies, chassis processors, network bonds and NICs on both manager nodes, and so on. For example:

```
YBCLI (PRIMARY)> health all

Chassis:  0
-----------
Bay:  1 HW Status: ok    Power: on LED: [ OFF  ON   OFF  ] FRU: C4-0005-01 R01 Serial: TAA17031300483 Temp CPU: 43C Inlet: 2
Bay:  2 HW Status: ok    Power: on LED: [ OFF  ON   OFF  ] FRU: C4-0005-01 R01 Serial: TAA16080300CBC Temp CPU: 42C Inlet: 2
Bay:  3 HW Status: ok    Power: on LED: [ OFF  ON   OFF  ] FRU: C4-0005-01 R01 Serial: TAA17010900681 Temp CPU: 41C Inlet: 2
Bay:  4 HW Status: ok    Power: on LED: [ OFF  ON   OFF  ] FRU: C4-0005-01 R01 Serial: TAA17031300500 Temp CPU: 44C Inlet: 2
Bay:  5 HW Status: ok    Power: on LED: [ OFF  ON   OFF  ] FRU: C4-0005-01 R01 Serial: TAA170313004F6 Temp CPU: 40C Inlet: 2
Bay:  6 HW Status: ok    Power: on LED: [ OFF  ON   OFF  ] FRU: C4-0005-01 R01 Serial: TAA1703130053C Temp CPU: 41C Inlet: 2
...
Retrieving blade alerts...

Blade alerts reported: None

Chassis:  1
-----------
...
```

blade

Return health information for one blade (given a chassis number and a bay number) or all blades. For example:

```
YBCLI (PRIMARY)> health blade 1/10

Chassis:  1
-----------
Bay: 10 HW Status: ok    Power: on LED: [ OFF  ON   OFF  ] FRU: C4-0008-01 R01 Serial: TAA18040900C76 Temp: 45C CPU: Booted (
```

```
Retrieving blade alerts...

Blade alerts reported: None
```

Note: You will see a warning if the BIOS is being flashed on one or more blades. These blades will automatically reset when BIOS flashing is complete.

cmp

Return chassis management processor (CMP) health information for one or both CMP bays on one or all chassis. For example:

```
YBCLI (PRIMARY)> health cmp all

Remote manager node
-------------------

Chassis:  0
-----------
CMP1 Primary   Status: ok Temp: 58C LED: [ FAST ON   OFF  ] FRU: C4-0001-03 R02 Serial: TAB17080111110 - Uptime: 376405 secor
CMP2 Secondary Status: ok Temp: 58C LED: [ FAST SLOW OFF  ] FRU: C4-0001-03 R02 Serial: TAB17080111114 - Uptime: 376403 secor

CMP alerts reported: None

Chassis:  1
-----------
CMP1 Primary   Status: ok Temp: 45C LED: [ OFF  ON   OFF  ] FRU: C4-0001-01 R02 Serial: TAB1612081105A - Uptime: 324806 secor
CMP2 Secondary Status: ok Temp: 58C LED: [ OFF  SLOW OFF  ] FRU: C4-0001-04 R01 Serial: TAB17020811084 - Uptime: 376393 secor

CMP alerts reported: None
```

fan

Return health information for one of the five fans on one or all chassis, or all fans. For example:

```
YBCLI (PRIMARY)> health fan all

Chassis:  0
-----------
Fan: 1 Status: ok - RPM: 3744/3558 (20%/20%)
Fan: 2 Status: ok - RPM: 3752/3464 (20%/20%)
Fan: 3 Status: ok - RPM: 3727/3467 (20%/20%)
Fan: 4 Status: ok - RPM: 3709/3526 (20%/20%)
Fan: 5 Status: ok - RPM: 3759/3455 (20%/20%)

Fan alerts reported: None

Chassis:  1
-----------
Fan: 1 Status: ok - RPM: 3741/3532 (20%/20%)
Fan: 2 Status: ok - RPM: 3748/3507 (20%/20%)
Fan: 3 Status: ok - RPM: 3730/3489 (20%/20%)
Fan: 4 Status: ok - RPM: 3716/3473 (20%/20%)
Fan: 5 Status: ok - RPM: 3727/3532 (20%/20%)

Fan alerts reported: None
```

```
YBCLI (PRIMARY)> health fan 1/5

Chassis:  1
-----------
Fan: 5 Status: ok - RPM: 3727/3536 (20%/20%)
```

```
Fan alerts reported: None
```

fan chassis

Return health information for just the chassis fan, either individually (1 through 5) or all. For example:

```
YBCLI(49290) (PRIMARY - yb97-mgr0)> health fan chassis all

Chassis:  0
------------
FAN1 Status: ok - RPM: 3276/3086 (15%/15%)
FAN2 Status: ok - RPM: 3202/2974 (15%/15%)
FAN3 Status: ok - RPM: 3236/3008 (15%/15%)
FAN4 Status: ok - RPM: 3255/3036 (15%/15%)
FAN5 Status: ok - RPM: 3194/2981 (15%/15%)

Chassis FAN alerts reported: None
```

fan manager

Return health information for just the manager node fans. For example:

```
YBCLI(13171) (PRIMARY - yb98-mgr0)> health fan manager all

Manager node (yb98-mgr0)
------------
FAN1 Status: ok RPM: 5700
FAN2 Status: ok RPM: 6000
FAN3 Status: ok RPM: 6100
FAN4 Status: ok RPM: 6000
FAN5 Status: ok RPM: 5800
FAN6 Status: ok RPM: 5900
FAN7 Status: ok RPM: 5800
FAN8 Status: ok RPM: 5600


Remote manager node
-------------------

Manager node (yb98-mgr1)
------------
FAN1 Status: ok RPM: 5700
FAN2 Status: ok RPM: 6100
FAN3 Status: ok RPM: 6000
FAN4 Status: ok RPM: 5800
FAN5 Status: ok RPM: 6400
FAN6 Status: ok RPM: 6100
FAN7 Status: ok RPM: 6100
FAN8 Status: ok RPM: 6100
```

network

Return customer, blade, and HA network health information for both manager nodes: bonds, NICs, and overall status. The possible status values are `OK`, `WARNING`, and `ERROR`. For example:

```
YBCLI(19584) (PRIMARY - yb98-mgr0)> health network

Network health:

Customer network bond   : Link UP - Address: 10.10.198.12 - Speed: 20000 Mbit
```

```
    MAC: 90:e2:ba:a6:64:69
    RX errors 0  dropped 0
    TX errors 0  dropped 0

    VLANS:
        39   : Address: N/A
               RX errors 0  dropped 0
               TX errors 0  dropped 0

        198  : Address: 10.10.198.12
               RX errors 0  dropped 0
               TX errors 0  dropped 0

Customer network (NIC1) : Link UP
    MAC: 90:e2:ba:a6:64:68
    Transceiver type(s):
        Infiniband: 1X Copper Passive
        FC: Copper Passive

Customer network (NIC2) : Link UP
    MAC: 90:e2:ba:a6:64:68
    Transceiver type(s):
        Infiniband: 1X Copper Passive
        FC: Copper Passive

Customer network (NIC3) : Not present
Customer network (NIC4) : Not present

Blade network bond  : Link UP - Address: 192.168.10.2
Blade network (NIC1)    : Link UP
Blade network (NIC2)    : Link UP

HA network bond     : Link UP - Address: 192.168.1.1
HA network (NIC1)   : Link UP
HA network (NIC2)   : Link UP

Chassis-0 mgmt network  : Link UP - Address: 192.168.2.1
Chassis-0 cmp ip addr   : 192.168.2.3
Chassis-1 mgmt network  : Link UP - Address: 192.168.5.1
Chassis-1 cmp ip addr   : 192.168.5.4
Chassis-2 mgmt network  : Link UP - Address: 192.168.6.1
Chassis-2 cmp ip addr   : 192.168.6.4
Chassis-3 mgmt network  : Link DOWN (Not configured)
Chassis-3 cmp ip addr   : N/A

Network Ping Status :
    Floating system IP : OK - Address: 10.10.198.10
    Customer network IP: OK - Address: 10.10.198.12
    DNS                : OK - Address: 10.10.10.20
                       : OK - Address: 10.10.10.21
    NTP                : OK - Address: 10.10.10.20
                       : OK - Address: 10.10.10.21
    VLANS:
        39  : Floating system IP : N/A
              Customer network IP: N/A

        198 : Floating system IP : OK - Address: 10.10.198.10
              Customer network IP: OK - Address: 10.10.198.12

Overall network health  : OK


Remote manager node
-------------------

Network health:
...
```

psu

Return health information for both chassis and manager node power supplies. For example:

```
YBCLI(49290) (PRIMARY - yb97-mgr0)> health psu all

Chassis:  0
-----------
Power supply: 1 Status: ok - Input voltage: 207V
Power supply: 2 Status: ok - Input voltage: 207V
Power supply: 3 Status: ok - Input voltage: 208V
Power supply: 4 Status: ok - Input voltage: 208V

Chassis PSU alerts reported: None

Manager node (yb97-mgr0)
------------
Power supply: 2 Status: ok Presence: Installed - AC Present
Power supply: 1 Status: ok Presence: Installed - AC Present


Remote manager node
-------------------

Manager node (yb97-mgr1)
------------
Power supply: 1 Status: ok Presence: Installed - AC Present
Power supply: 2 Status: ok Presence: Installed - AC Present
```

psu chassis

Return health information for just the chassis power supplies, either individually (1 through 4) or all. For example:

```
YBCLI(49290) (PRIMARY - yb97-mgr0)> health psu chassis all

Chassis:  0
-----------
Power supply: 1 Status: ok - Input voltage: 207V
Power supply: 2 Status: ok - Input voltage: 206V
Power supply: 3 Status: ok - Input voltage: 209V
Power supply: 4 Status: ok - Input voltage: 208V

Chassis PSU alerts reported: None
```

psu manager

Return health information for just the manager node power supplies, either local, remote, or all. For example:

```
YBCLI(29029) (PRIMARY - yb98-mgr0)> health psu manager all

Manager node (yb98-mgr0)
------------
Power supply: 1 Status: ok Presence: Installed - AC Present
Power supply: 2 Status: ok Presence: Installed - AC Present


Remote manager node
-------------------

Manager node (yb98-mgr1)
------------
Power supply: 1 Status: ok Presence: Installed - AC Present
Power supply: 2 Status: ok Presence: Installed - AC Present
```

storage

Return storage health information for both manager nodes. For example:

```
YBCLI (PRIMARY)> health storage

Manager node OS RAID      : OK       Drive0: OK         Drive1: OK
Manager node HA RAID      : OK       Drive0: OK         Drive1: OK
Manager node SPOOL RAID   : OK       Drive0: OK         Drive1: OK
Manager node ROWSTORE RAID: OK       Drive0: OK         Drive1: OK
Encryption status         : No drives locked

OS drive0 (sdd) NAND spare    : 100 %
OS drive0 (sdd) NAND life used:   0 %

OS drive1 (sdc) NAND spare    : 100 %
OS drive1 (sdc) NAND life used:   0 %

...
```

ybcli: help

Platforms: EE: All appliance platforms

Parent topic: ybcli Reference

After starting the `ybcli`, you can type `help` to return a list of command categories. Type `help command_name` to see the syntax and description for a specific category of commands (such as `blade` or `manager`). Type the next keyword in the command to restrict the output to the syntax for that command (such as `manager reboot`).

```
help [ command ] [ keyword ]
```

For example:

```
YBCLI (PRIMARY)> help

Available YBCLI commands. Type 'help <command>' for details
================================================================================
blade  config    debug  health  log      quit    system
clear  database  exit   help    manager  status
```

```
YBCLI (PRIMARY)> help manager

      manager reboot [ local | remote ]
      manager reset [ local | remote ]
      manager resume [ local | remote ]
      manager shutdown [ local | remote ]
      manager standby [ local | remote ]
      manager status [ all | local | remote ]
```

```
YBCLI (PRIMARY)> help manager reboot

    manager reboot  [ local | remote ]
```

ybcli: keystore

Set up and manage the keystore for the encrypted drives.

```
keystore backup
keystore backup list
keystore lock
keystore restore <filename>
keystore rotate
keystore setup [ force ]
keystore status
keystore unlock
```

keystore backup

Back up the keystore to the `/tmp` directory. After backing up the keystore, move the backup file from the manager node to a safe and secure external storage server. Do not copy it to another location on the appliance. Any keys left on the manager node under `/tmp` for more than 10 days will be automatically deleted.

The `keystore restore` command, if you need to use it, expects the file to be in its original location, so you will need to secure-copy the backup file back to `/tmp` at that time. Keystore backup files are very small and have a timestamped name.

For example:

```
YBCLI (PRIMARY)> keystore backup

Are you sure you want to back up the keystore to: /tmp/ ?
NOTE: During the backup, the database remains online and users can run queries.
After the backup, you have to unlock the keystore before you can run any encryption commands.

Type yes to continue: yes

Stopping the keystore service before backup. Standby...  Done
Backing up keystore. Standby... Done
Starting the keystore service after backup. Standby...  Done

The keystore has been backed up successfully to:
    /tmp/ybd-ks-08-09-2019-19-41-23.tar.gz
Please copy the backup to another machine. The backup is located on this system at:
    yb00-mgr0.yellowbrick.io:/tmp/ybd-ks-08-09-2019-19-41-23.tar.gz
    MD5: 4d56f07d6dcba20c6bceb0495448f842


Do you want to unlock the keystore (not required)?

Type yes to continue: no
Keystore will not be unlocked
```

Note: Whenever the keystore is modified in response to `encryption enable` and `encryption rotate` operations, a keystore backup happens automatically.

keystore backup list

List all the keystore backups that exist on the system, beginning with the most recent. For example:

```
YBCLI(26168) (PRIMARY - yb100-mgr1)> keystore backup list

The following Yellowbrick keystore backups exist on this system:
    Yellowbrick keystore backup: /tmp/ybd-ks-08-06-2019-17-36-50.tar.gz
    Yellowbrick keystore backup: /tmp/ybd-ks-08-06-2019-11-21-02.tar.gz
    Yellowbrick keystore backup: /tmp/ybd-ks-08-05-2019-19-08-15.tar.gz
    Yellowbrick keystore backup: /tmp/ybd-ks-08-05-2019-16-40-01.tar.gz
```

keystore lock

Lock the keystore. Locking the keystore prevents encryption commands from being run until it has been unlocked; however, locking the keystore does not change the encryption status on the system. If encryption was enabled when you locked the keystore, encryption remains enabled.

```
YBCLI (PRIMARY)> keystore lock

Authenticating to the keystore for this YBCLI session. Standby...
Please enter keystore authentication key:
Key ->

Key accepted. This session is authenticated for the next 10 minutes or until YBCLI exits.
Successfully locked the keystore
```

keystore restore <filename>

Restore the keystore from a backup. Before running the command, secure-copy the backup file from its external location to the `/tmp` directory on the active manager node. Enter the name of the backup file to be restored. You can display a list by running the `keystore backup list` command.

Do not specify a path in the command, just the filename, and do not use quotes around the filename. The expected path to the backup file is `/tmp` on the manager node. After restoring, you have to unlock the keystore. For example:

```
YBCLI (PRIMARY)> keystore restore ybd-ks-08-09-2019-19-41-23.tar.gz

Are you sure you want to restore the keystore using:
    /tmp/ybd-ks-08-09-2019-19-41-23.tar.gz

NOTE: During the restore, the database remains online and users can run queries.

Type yes to continue: yes

Stopping the keystore service before restore. Standby...  Done
Verifying backup. Standby... Done
Restoring. Standby... Done
Starting the keystore service after restore. Standby...  Done

The keystore has been restored successfully using:
    /tmp/ybd-ks-08-09-2019-19-41-23.tar.gz

Do you want to unlock the keystore again (not required)?

Type yes to continue: no
Not unlocking keystore after restore
```

keystore rotate

Generate new unlock keys and a new authentication key for the keystore. You cannot generate unlock keys or the authentication key separately. For example:

```
YBCLI(63867) (PRIMARY - yb00-mgr0)> keystore rotate

Rotating the keystore will generate new authentication and unlock keys.
Note: A backup of the current keystore will be performed prior to rotating the keystore.
Keystore rotation requires both the authentication and unlock keys to be present.

Do you want to rotate the keystore?
Response (yes/no): yes

Stopping the keystore service before backup. Standby...  Done
Backing up keystore. Standby... Done
Starting the keystore service after backup. Standby...  Done

The keystore has been backed up successfully to:
       /tmp/ybd-ks-01-02-2020-08-26-14.tar.gz
Please copy the backup to another machine. The backup is located on this system at:
       yb00-mgr0.yellowbrick.io:/tmp/ybd-ks-01-02-2020-08-26-14.tar.gz
       MD5: 31724b6a57a4a788f47b5541b6012345

Please enter an unlock key to begin unlocking the keystore:
Key ->

Keystore was successfully unlocked

Authenticating to the keystore for key rotation. Standby...
Please enter keystore authentication key:
Key ->

Rotating the keystore. Standby...

Note: You can request up to 5 keystore unlock keys. In this way, multiple administrators can unlock the keystore
using a combination of keys. No single key has to be distributed to a single administrator.

How many unlock keys should be generated for the keystore? (1 to 5): 1
1 key will be required to unlock the keystore

The following unlock key(s) were generated. A total of 1 key(s) is required to unlock the keystore:
New keystore unlock key 1: 86fdbd9599bb83c732e3c4eeab2d47a1c6325c85d1c72547c4f61a13da212345

Rotating new authentication key. Standby...
New authentication key: c5e9efb6-401a-1da1-43f3-00aba2e12345

Keys have been generated. Please store them in a safe place.

Do you want to create a backup of the keystore?
Response (yes/no): yes

Stopping the keystore service before backup. Standby...  Done
Backing up keystore. Standby... Done
Starting the keystore service after backup. Standby...  Done

The keystore has been backed up successfully to:
       /tmp/ybd-ks-01-02-2020-08-32-23.tar.gz
Please copy the backup to another machine. The backup is located on this system at:
       yb00-mgr0.yellowbrick.io:/tmp/ybd-ks-01-02-2020-08-32-23.tar.gz
       MD5: 4ab76751319fc490ad8c5104a9aa41ac


Do you want to unlock the keystore (not required)?
Response (yes/no): no
Keystore will not be unlocked

Keystore rotation successfully completed
```

For an example with three unlock keys, see Example: Rotate the Keystore with Multiple Unlock Keys.

This command does not generate new encryption keys for the drives; see the `encryption rotate` command.

keystore setup

Set up a new keystore on a system that will use encrypted drives. You cannot enable encryption until the keystore is created. The `keystore setup` command generates a set of keys:

- A single 36-byte authentication key, which is a required "password" that you have to enter before you can run any `encryption` commands.
- One or more 64-byte keys to unlock the keystore

At least one of each type of key is required.

Note: You can request up to 5 unlock keys. In this way, multiple administrators can unlock the keystore at different times using a combination of keys. No single key has to be distributed to a single administrator.

During the setup, you will be prompted to run a backup of the keystore; this is a recommended part of the procedure.

For detailed examples, see Example: Set Up the Keystore and Enable Encryption and Example: Use Multiple Unlock Keys.

keystore setup force

CAUTION:

This command erases all of the drive keys in the keystore, then re-creates it. Read the command output on the screen carefully before proceeding.

If encryption is *not* enabled on any of the drives, you are safe to proceed. If encryption *is* enabled, all data on the encrypted drives will be lost unless you can restore the keystore from a backup. If you cannot restore the keystore, each encrypted drive will have to be manually unlocked by using the key printed on its label. All data will be lost. See Example: Force Keystore Setup.

keystore status

Show whether the keystore is locked or unlocked. For example:

```
YBCLI (PRIMARY)> keystore status

keystore status
Locked: NO
```

keystore unlock

Unlock the keystore. You must enter one or more 64-byte unlock keys, depending on the number of keys that were specified when the keystore was set up.

```
YBCLI (PRIMARY)> system keystore unlock

Please enter the first key to unlock the keystore:
Key ->

Verifying keystore status. Standby...
Keystore locked: NO

Keystore was successfully unlocked
```

ybcli: log

Return log file entries that were recorded in the past 1 to 24 hours or tail one of the logs. You can return logs specific to a single compute blade, the database, the keystore, or the local manager node.

```
log blade [ 1-24 | all | tail ] [ <chassis>/<bay> | <uuid> | all ] [ <keyword> | "<search_string>" ]
log clang [ 1-24 | all | tail ] [ <keyword> | “<search_string>” ]
log database [ 1-24 | all | tail ] [ <keyword> | "<search_string>" ]
log install [ <keyword> | "<search_string>" ]
log keystore [ 1-24 | all | tail ] [ <keyword> | "<search_string>" ]
log manager [ 1-24 | all | tail ] [ <keyword> | "<search_string>" ]
log ybcli [ 1-24 | all | tail ] [ <keyword> | "<search_string>" ]
```

Note: In all cases, `1-24` refers to a specific hour from 1 to 24, where `1` is the last hour and `24` is the last 24 hours. For `<chassis>/<bay>`, specify a chassis number and a bay number. For example: `2/1`, `1/15` (without spaces).

All of the `log` commands have a search feature. You can append a keyword or a text string to the end of any `log` command, and only the log lines that contain the keyword or string are returned. A single keyword does not require quotes, but a string with multiple words does require quotes. The search is not case-sensitive. A single-keyword search will find words with a matching stem. See Log Search Examples.

blade

Return the log for a specific blade or all blades. For a specific blade, use its chassis and bay number or its `UUID`. (Use the `status blade` command to get the UUID.) For example:

```
YBCLI (PRIMARY)> log blade tail 0/1

Press ctrl-c to stop tailing log....
2018-07-10T17:53:42.042404-07:00 ybos-00000000-0000-0000-0000-38B8EBD00C1C YBDB[25486]:  23465685 Warn          kernel.Work
2018-07-10T17:53:42.042423-07:00 ybos-00000000-0000-0000-0000-38B8EBD00C1C YBDB[25486]:  23465685 Warn          kernel.Work
2018-07-10T17:53:42.042434-07:00 ybos-00000000-0000-0000-0000-38B8EBD00C1C YBDB[25486]:  23465685 Warn          kernel.Work
...
```

To return logs for all blades (for all hours), use the following command:

```
log blade all all
```

You can also specify the number of hours for all blades. For example:

```
log blade 1 all
```

clang

Return the log for compiler-related events. This command must be run on the primary manager node. For example:

```
YBCLI(5797) (PRIMARY - yb97-mgr0)> log clang 1

2021-05-19T11:04:32.742150-07:00 yb97-mgr1 CLANG: ybstop    WARNING: -c CONFIG not specified.  Attempting to guess the DEFAU
2021-05-19T11:04:32.746774-07:00 yb97-mgr1 CLANG: ybstop    Detected default config: system-Linux.xml
2021-05-19T11:04:32.746904-07:00 yb97-mgr1 CLANG: ybstop    Using configuration system-Linux.xml...
2021-05-19T11:04:32.747364-07:00 yb97-mgr1 CLANG: ybstop    Stopping clang...
2021-05-19T11:04:33.855883-07:00 yb97-mgr1 CLANG: ybstop    Finding specific pids to kill: [clang]
2021-05-19T11:04:34.155364-07:00 yb97-mgr1 CLANG: ybstop    YB processes stopped
2021-05-19T11:04:34.155575-07:00 yb97-mgr1 CLANG: ybstop    Note: some YB processes remain:
2021-05-19T11:04:34.155702-07:00 yb97-mgr1 CLANG: ybstop    >>>     Worker:
2021-05-19T11:04:34.155804-07:00 yb97-mgr1 CLANG: ybstop    >>>     Lime: 62453
2021-05-19T11:04:34.155897-07:00 yb97-mgr1 CLANG: ybstop    >>>     Spiky: 4076
2021-05-19T11:04:34.156049-07:00 yb97-mgr1 CLANG: ybstop    >>>     Postgres:
2021-05-19T11:04:34.156175-07:00 yb97-mgr1 CLANG: ybstop    >>>     SMC: 60677
2021-05-19T11:04:34.156262-07:00 yb97-mgr1 CLANG: ybstop    >>>     DCS:
2021-05-19T11:04:34.156412-07:00 yb97-mgr1 CLANG: ybstop    >>>     External tables server:
2021-05-19T11:04:34.156526-07:00 yb97-mgr1 CLANG: ybstop    >>>     Bulk service:
2021-05-19T11:04:34.156586-07:00 yb97-mgr1 CLANG: ybstop    >>>     Clang server:
2021-05-19T11:04:34.156626-07:00 yb97-mgr1 CLANG: ybstop    YB stopped
```

Contact Yellowbrick Support for further assistance.

database

Return the log for the database. This command must be run on the primary manager node.

```
YBCLI (PRIMARY)> log database tail

Press ctrl-c to stop tailing log....
    LEFT OUTER JOIN sys."vt_flush_tables" f
      ON f."tableId" = c.oid
     AND f."databaseName" = current_database()
     AND NOT f."expired"
    LEFT OUTER JOIN pg_catalog.pg_stat_all_tables t
      ON t."relid" = c."oid"
    WHERE c."relkind" = 'r'
    ;
2017-03-14 15:05:03.788 PDT 26246 FATAL:  password authentication failed for user "ybd"
2017-03-14 15:05:03.788 PDT 26246 DETAIL:  Connection matched pg_hba.conf line 96: "host    all             all             0
...
```

install

Return the log for when the installation program was last run on the system. For example:

```
YBCLI(31444) (PRIMARY - yb100-mgr1)> log install

The following output is the contents of the log when the Yellowbrick installer was last run on this system.

Yellowbrick Data appliance installer v2.0.0
Installing: v0.0.1-11160
Installing using the following parameters: {'core': False, 'fieldreset': False, 'force': False, 'rpmreinstall': False, 'ignor

Blackout tenant  : YB
Blackout cluster : yb00
Blackout duration: 3600
stderr filename  : /root/YB-yb00-create-blackout-stderr.out

Blackout UUID    : 2d46bf92-d500-4e23-845e-1b4e1d6ff5fc

Starting backing up existing install
Backup location: /mnt/ybdata/backup/backup-2019_02_22-0534-59/
Backup complete
```

```
Installing YBD appliance version: 0.0.1-11160-release...
...
...
System upgrade complete
Install complete

End of the install log.
```

keystore

Return the keystore/encryption audit log (on systems with encrypted drives). For example:

```
YBCLI (PRIMARY)> log keystore tail

Press ctrl-c to stop tailing log....
2017-10-25T13:59:34.415317-07:00 yb00-mgr1 journal: YBCLI user ybdadmin (10.20.12.24): keystore: Executing -> keystore init
2017-10-25T13:59:34.419245-07:00 yb00-mgr1 journal: YBCLI user ybdadmin (10.20.12.24): keystore: Requires authentication
2017-10-25T13:59:57.254574-07:00 yb00-mgr1 journal: YBCLI user ybdadmin (10.20.12.24): keystore: Executing -> keystore init
2017-10-25T13:59:57.258328-07:00 yb00-mgr1 journal: YBCLI user ybdadmin (10.20.12.24): keystore: Requires authentication
2017-10-25T14:00:37.265908-07:00 yb00-mgr1 journal: YBCLI user ybdadmin (10.20.12.24): keystore: Executing -> keystore init
2017-10-25T14:00:37.270577-07:00 yb00-mgr1 journal: YBCLI user ybdadmin (10.20.12.24): keystore: Requires authentication
2017-10-25T14:04:32.013941-07:00 yb00-mgr1 journal: YBCLI user ybdadmin (10.20.12.24): keystore: Executing -> keystore init
2017-10-25T14:04:32.016892-07:00 yb00-mgr1 journal: YBCLI user ybdadmin (10.20.12.24): keystore: Requires authentication
2017-10-25T14:04:37.550317-07:00 yb00-mgr1 journal: YBCLI user ybdadmin (10.20.12.24): Keystore: Executing -> keystore unlock
2017-10-25T14:04:37.552872-07:00 yb00-mgr1 journal: YBCLI user ybdadmin (10.20.12.24): Keystore: Already unlocked
...
```

manager

Return or tail the log for the local manager node. For example:

```
YBCLI (SECONDARY)> log manager all

2017-10-31T13:01:01.639817-07:00 yb00-mgr1 rsyslogd: [origin software="rsyslogd" swVersion="7.4.7" x-pid="1383" x-info="http
2017-10-31T13:01:01.973358-07:00 yb00-mgr1 crmd[3756]:  notice: Operation ybdkeystore_stop_0: ok (node=yb00-mgr1.yellowbrick.
2017-10-31T13:01:01.974923-07:00 yb00-mgr1 crmd[3756]:  notice: Initiating action 122: start ybdkeystore_start_0 on yb00-mgr1
2017-10-31T13:01:01.975659-07:00 yb00-mgr1 systemd: Reloading.
...
```

```
YBCLI (SECONDARY)> log manager tail

Press ctrl-c to stop tailing log....
2017-11-27T19:27:54.168847+01:00 yb00-mgr1 crmd[3725]:  notice: Operation ybdfs_notify_0: ok (node=yb00-mgr1.yellowbrick.io,
2017-11-27T19:27:54.225340+01:00 yb00-mgr1 crmd[3725]:   error: pcmkRegisterNode: Triggered assert at xml.c:594 : node->type
2017-11-27T19:27:54.569690+01:00 yb00-mgr1 ntpd[7901]: Deleting interface #35 BLADE_bond, 192.168.10.1#123, interface stats:
2017-11-27T19:27:54.729028+01:00 yb00-mgr1 systemd: Stopped YellowBrick Vault.
...
```

ybcli

Return the `ybcli` command history. For example:

```
YBCLI (PRIMARY - yb100-mgr0)> log ybcli tail
Press ctrl-c to stop tailing log....
2018-12-05 05:35:38 :  INFO : Executing -> system appliance version
2018-12-05 05:35:38 :  INFO : Completed -> system appliance version
2018-12-05 05:35:43 :  INFO : Executing -> health cmp all
2018-12-05 05:35:45 :  INFO : Completed -> health cmp all
2018-12-05 11:44:38 :  INFO : YBCLI 2.0.0-9737 started
2018-12-05 11:44:47 :  INFO : Executing -> help log
```

```
2018-12-05 11:45:41 :  INFO : Executing -> log ybcli tail
...
```

Log Search Examples

The following examples show how you can search through the logs by specifying a single keyword or a longer search string. The first example searches the manager logs for the word `installed`. Quotes are not required:

```
YBCLI(16789) (PRIMARY - yb00-mgr0)> log manager 24 installed

2019-04-16T13:41:34.155927-07:00 yb00-mgr0 installer2[10836]: Current version installed: v0.0.1-12607
2019-04-16T14:41:37.304598-07:00 yb00-mgr0 installer2[10894]: Current version installed: v0.0.1-12607
2019-04-16T14:44:41.598975-07:00 yb00-mgr0 installer2[10894]: New YBD appliance package installed. Continuing
```

The second example searches the manager logs for the string `Current version installed`. Quotes are required:

```
YBCLI(16789) (PRIMARY - yb00-mgr0)> log manager 24 "Current version installed"

2019-04-16T13:41:34.155927-07:00 yb00-mgr0 installer2[10836]: Current version installed: v0.0.1-12607
2019-04-16T14:41:37.304598-07:00 yb00-mgr0 installer2[10894]: Current version installed: v0.0.1-12607
```

The third example searches the database logs for the string `System not ready`:

```
YBCLI(16789) (PRIMARY - yb00-mgr0)> log database all "System not ready"

2019-04-16 14:50:59.330 PDT 7581 0 WM002 WARNING:  System not ready
2019-04-16 14:50:59.601 PDT 7600 0 WM002 WARNING:  System not ready
2019-04-16 14:51:00.790 PDT 7869 0 WM002 WARNING:  System not ready
...
```

The fourth example searches the logs for a specific blade and returns all entries that contain `fail`, `Fail`, and words that begin with `fail` or `Fail`:

```
YBCLI(16789) (PRIMARY - yb00-mgr0)> log blade 24 1/1 fail

Chassis:  1
-----------
2019-04-16T14:44:42.960413-07:00 ybos-00000000-0000-0000-0000-38B8EBD0061D YBDB[4423]:         1 Inform    kernel.LoggingDumper: C
2019-04-16T14:44:42.960413-07:00 ybos-00000000-0000-0000-0000-38B8EBD0061D YBDB[4423]:         1 Inform    kernel.LoggingDumper: C
...
2019-04-16T14:47:52.951151-07:00 ybos-00000000-0000-0000-0000-38B8EBD0061D kernel:  [    0.699306] pci 0000:01:00.0: BAR 6: failed
2019-04-16T14:47:52.951151-07:00 ybos-00000000-0000-0000-0000-38B8EBD0061D kernel:  [    0.699402] pci 0000:ff:12.0: BAR 2: failed
...
2019-04-16T14:47:52.951730-07:00 ybos-00000000-0000-0000-0000-38B8EBD0061D kernel:  [    9.117878] mgmt_bond: Setting fail_over_ma
2019-04-16T14:47:52.951730-07:00 ybos-00000000-0000-0000-0000-38B8EBD0061D kernel:  [    9.176539] psmouse serio1: Failed to reset
Press any key for next section. 'q' to stop
...
```

ybcli: manager

Platforms: EE: All appliance platforms

Parent topic: ybcli Reference

Run operations on the manager nodes.

```
manager reboot [ local | remote ]
manager reset [ local | remote ]
manager resume [ local | remote ]
manager shutdown [ local | remote ]
manager standby [ local | remote ]
manager status [ all | local | remote ]
```

Note: The manager node where `ybcli` is running is considered the `local` node. The other manager node is considered `remote`. The nodes are clearly identified when you start `ybcli`:

```
Yellowbrick Data CLI vx.x.xx (Genx)
Copyright (c) 2016-2019 Yellowbrick Data, Inc.
All rights reserved
----------------------------------------
Redundant manager node detected
YBCLI is currently running on the PRIMARY manager node.
Local manager node : yb99-mgr0 -> (PRIMARY ACTIVE)
Remote manager node: yb99-mgr1 -> (SECONDARY ACTIVE)
```

reboot

Reboot a manager node. For example:

```
YBCLI (PRIMARY)> manager reboot remote

WARNING: The local node is currently the primary
Rebooting the remote node will disable system redundancy

Are you sure you want to do this?
Type yes to continue: yes

Remote manager node
-------------------

Initiating reboot. This process can take 60 seconds
```

reset

Hardware reset of a manager node in case of a system malfunction. Running `manager reset` on the primary manager node will be disruptive to the database and should only be used as a last resort if `manager reboot` is unsuccessful. For example:

```
YBCLI(2675) (PRIMARY - yb00-mgr1)> manager reset local

Warning: The local node is currently the primary node.
Resetting the local node initiates a system failover, causing all services to restart.
```

```
If the remote node is unavailable, resetting the local node causes a database shutdown.
Continue (yes/no)? yes
Initiating reset in 3 seconds
Manager will be reset in 3 seconds
```

resume

Bring a manager node that was on standby back into the HA cluster, enabling redundancy.

shutdown

Shut down one or both manager nodes. For example:

```
YBCLI (PRIMARY)> manager shutdown local
Warning: The local node is currently the primary node
Shutting down the local node will initiate a system fail-over
This will cause all services to restart or if the remote node is unavailable, a database shutdown
Continue (yes/no)?
...
```

standby

Remove a manager node from the HA cluster, disabling redundancy and putting the node on standby.

```
YBCLI (PRIMARY)> manager standby local

Warning: The local node is currently the primary
Enabling standby mode on the local node will initiate a system fail-over
This will cause all services to restart or if both nodes are in standby, a database shutdown

ERROR: YBCLI cannot perform a failover while connected to the floating cluster IP
To perform a cluster failover, please connect to the IP address of the current primary or secondary node

The IP addresses and hostnames of the manager nodes can be obtained using the following commands:
    health network
    config hostname get
```

status

Get status information about one or both of the manager nodes. For example:

```
YBCLI (PRIMARY)> manager status all

Host         : yb100-mgr0.yellowbrickroad.io
Uptime       : 40 days, 01:57:32
Users online : 2
CPU usage    : 0.2 %
Memory used  : 11G GiB
Memory free  : 232G GiB

Remote manager node
-------------------

Host         : yb100-mgr1.yellowbrickroad.io
Uptime       : 40 days, 02:01:34
Users online : 0
CPU usage    : 0.2 %
Memory used  : 2.8G GiB
Memory free  : 247G GiB
```

Parent topic: ybcli Reference

ybcli: status

Return the status of various system components.

```
status all
status blade [ <chassis>/<bay> ]
status cmp [ <chassis>/<cmp> ]
status database
status manager [ local | remote | all ]
status storage
status system
```

Note: For `status blade` and `status cmp`, specify a chassis number (or `all`) and a bay or cmp number (or `all`). For example: `0/1`, `all/1`, `all/all`, `1/all` (without spaces).

all

Return the status of all components.

blade

Return the UUID, current status, CPU and memory usage, uptime, encryption status, and drive status/wear for one or all blades. For example, when run on Tinman:

```
YBCLI(8911) (PRIMARY - yb100-mgr0)> blade status 1/3

Chassis:  1
-----------
Blade Bay:  3 -> BOOTED  UUID: 00000000-0000-0000-0000-38B8EBD006F4 - Version: YBOS-2.0.4-DEBUG
         BIOS: v05.04.21.0038.00.011 - Memory total/free: 65587652/1870728 KiB - IB-FW: 2.36.5000
         CPU: Intel(R) Xeon(R) CPU E5-2618L v4 @ 2.20GHz - Cores: 10 - Load: 1%
         Address: 192.168.20.12 - Uptime: 0 day(s), 01:59:29 - Worker: Running
         Encryption Supported: YES - Encryption Enabled: NO - locked: N/A
               SSD: 4: S3EWNWAJ312492R    5: S3EWNWAJ330205H    6: S3EWNWAJ312493K    7: S3EWNWAJ330206Z
                        OK/0% 2B6QCXP7        OK/0% 2B6QCXP7        OK/0% 2B6QCXP7        OK/0% 2B6QCXP7
                     0: S3EWNWAJ311166N    1: S3EWNWAJ330207R    2: S3EWNWAJ311169V    3: S3EWNWAJ330208K
                        OK/0% 2B6QCXP7        OK/0% 2B6QCXP7        OK/0% 2B6QCXP7        OK/0% 2B6QCXP7
         Cluster status: OPERATIONAL - Cluster role: MEMBER - Last seen: just now
```

For example, when run on Andromeda:

```
YBCLI(416590) (PRIMARY - y2b21-mgr1)> blade status 0/1

Chassis:  0
-----------
Blade Bay:  1 -> YBOS READY UUID: 00000000-0000-0000-0000-38B8EBD20E88 - Version: YBOS-3.0.0-20200917181022-DEBUG
         BIOS: Athn_1.01.3 - Memory total/free: 1056744476/18735500 KiB
         BMC: 00.17.01 - Kalidah-1: 0x0DE38309 - Kalidah-2: 0x0DE38309 - IB-FW: 16.28.1002
         CPU: AMD EPYC 7702P 64-Core Processor - Microcode: 0x8301034 - Cores: 64 - Load: 1%
         Address: 192.168.10.10 - Uptime: 0 day(s), 00:15:14 - Worker: Running
         Encryption Supported: YES - Encryption Enabled: NO - locked: N/A
               SSD: 4: S439NE0MA00440     5: S439NE0MA00287     6: S439NE0MA00271     7: S439NE0MA00284
```

```
                    OK/0% EDA5502Q        OK/0% EDA5502Q        OK/0% EDA5502Q        OK/0% EDA5502Q
                  0: S439NE0MA00442     1: S439NE0MA00253     2: S439NE0MA00264     3: S439NE0MA00288
                    OK/0% EDA5502Q        OK/0% EDA5502Q        OK/0% EDA5502Q        OK/0% EDA5502Q
          Cluster status: OPERATIONAL - Cluster role: MEMBER - Last seen: just now
```

Note: If the amount of memory installed on the blades varies (based on the amount reported for the first blade), you will see red text for the memory information and a warning message at the end of the output. You will also see a warning if the BIOS is being flashed on one or more blades. These blades will automatically reset when BIOS flashing is complete.

cmp

Return the status of the chassis management processors (CMPs) and InfiniBand switches on the manager nodes. The output includes link status for all of the InfiniBand ports. For example:

```
YBCLI (PRIMARY)> status cmp 1/1

Chassis:  1
-----------
CMP1 version: 2.53 Role: PRIMARY
Blade link status
Port 1 -> UP - Link: FDR,x4
Port 2 -> UP - Link: FDR,x4
Port 3 -> UP - Link: FDR,x4
Port 4 -> UP - Link: FDR,x4
Port 5 -> UP - Link: FDR,x4
Port 6 -> UP - Link: FDR,x4
Port 7 -> UP - Link: FDR,x4
Port 8 -> UP - Link: FDR,x4
Port 9 -> UP - Link: FDR,x4
Port10 -> UP - Link: FDR,x4
Port11 -> UP - Link: FDR,x4
Port12 -> BLADE NOT INSTALLED
Port13 -> BLADE NOT INSTALLED
Port14 -> BLADE NOT INSTALLED
Port15 -> BLADE NOT INSTALLED

External link status
Ext port 1 -> UP - Link: FDR,x4
Ext port 2 -> UP - Link: FDR,x4
Ext port 3 -> UP - Link: FDR,x4
Ext port 4 -> UP - Link: FDR,x4
Ext port 5 -> UP - Link: FDR,x4
Ext port 6 -> UP - Link: FDR,x4
Ext port 7 -> UP - Link: FDR,x4
Ext port 8 -> UP - Link: FDR,x4
```

database

Return the status of the database software stack, including uptime. For example:

```
YBCLI (PRIMARY)> status database

Database system running        : YES
Database system ready          : YES
Database system read-only      : NO
Database system uptime         : 02:32:49
Database system users connected: 4 (including system users)
```

manager

Return uptime and CPU and memory usage on one or both manager nodes. For example:

```
YBCLI (PRIMARY)> status manager all

Host         : yb1000-mgr0.ybd.io
Uptime       : 15 days, 05:55:48
Users online : 2
CPU usage    : 3.1 %
Memory used  : 10G GiB
Memory free  : 229G GiB

Remote manager node
-------------------

Host         : yb1000-mgr1.ybd.io
Uptime       : 15 days, 05:56:52
Users online : 0
CPU usage    : 0.0 %
Memory used  : 2.6G GiB
Memory free  : 246G GiB
```

storage

Return free and used storage space on the manager node file systems, including the database row store, and storage quotas on the primary manager node. For example:

```
YBCLI(912) (PRIMARY - yb00-mgr1)> status storage

OS filesystem       : 112G free -  39G used
HA filesystem       : 847G free - 941G used
Rowstore filesystem : 457G free -  21G used
Spool filesystem    : 954G free -  33M used

Storage quotas:
        Catalog rowstore   : 2.2G of 200G used
        Cluster manager    : 146.1M of 330G used
```

This command returns two entries for storage quotas:

- **Catalog rowstore**: Amount of storage space being used for the database system catalog.
- **Cluster manager**: Amount of storage space being used by the compiler.

system

Return the overall status of the system, indicating the status of the manager nodes and whether the database is running. For example, when run on Tinman:

```
YBCLI(30354) (PRIMARY - yb98-mgr0)> status system

Manager nodes configured: 2
---------------------------
Node 1 (PRIMARY - LOCAL NODE ) : yb98-mgr0 -> ONLINE
Node 2 (SECONDARY - REMOTE NODE ) : yb98-mgr1 -> ONLINE

HW Platform                       : Tinman
Database system running           : YES
Database system ready             : YES (Responding: YES)
Database system read-only         : NO
Database system catalog rowstore  : NORMAL
Database system rowstore          : NORMAL
Database system storage used      : 20%
Database system uptime            : 14:59:42
System work status                : gc:idle  parityrebuild:idle  analyzer:idle  system:idle  user:active,idle
Data collection running           : YES
Blade parity                      : Enabled
Blade parity rebuilding           : NO (Progress: N/A)
```

```
Blade data check in-progress       : NO
Shard rewriting                    : NO (0 shards pending)
Capacity expansion in-progress     : YES (5 Tables and 1752898 MBs remaining)
Cluster degraded mode              : YES (Reason: Drive(s) failed)
Maintenance mode                   : NO
Software update in-progress        : NO (Version: 4.1.1-24805)
Floating system IP                 : 10.10.102.10 - 255.255.255.0
System registered                  : NO
LDAP status                        : Not configured
Logging status                     : Enabled
Encryption keystore                : Available - Status: Not setup
Chassis configuration              : Found: 1 - Configured: 1
Add-ons                            : None
```

For example, when run on Andromeda:

```
YBCLI(51453) (PRIMARY - y2b23-mgr0)> system status

Manager nodes configured: 2
---------------------------
Node 1 (PRIMARY   - LOCAL NODE  ) : y2b23-mgr0.slc.yellowbrick.io -> ONLINE
Node 2 (SECONDARY - REMOTE NODE ) : y2b23-mgr1.slc.yellowbrick.io -> ONLINE

HW Platform                        : Andromeda
Database system running            : YES (9/9/9 active/powered/installed blades)
Database system ready              : YES (Responding: YES)
Database system read-only          : NO
Database system catalog rowstore   : NORMAL
Database system rowstore           : NORMAL
Database system storage used       : 20%
Database system uptime             : 00:36:20
System work status                 : gc:idle  parityrebuild:idle  analyzer:idle  system:idle  user:idle
Data collection running            : YES
Blade parity                       : Enabled
Blade parity rebuilding            : NO (Progress: N/A)
Blade data check in-progress       : NO
Shard rewriting                    : NO (0 shards pending)
Capacity expansion in-progress     : YES (3 Tables and 764036 MBs remaining)
Cluster degraded mode              : NO
Maintenance mode                   : NO
Software update in-progress        : NO (Version: 5.1.0-33380)
Floating system IP                 : 10.10.223.10 - 255.255.255.0
System registered                  : YES (Tenant: yb - System: y2b23)
LDAP status                        : Not configured
Logging status                     : Enabled
Encryption keystore                : Available - Status: READY - Locked: YES
Chassis configuration              : Found: 1 - Configured: 1
System FRU configuration           : All components match records
Add-ons                            : None
```

Note: In the Database system running line, the active/powered/installed blades text will be yellow if not all installed blades are active or online.

Attention: The statistics under `Capacity expansion in-progress` only update once a table has been completely moved. Therefore, it may appear at times as though no progress is being made.

ybcli: system

Platforms: EE: All appliance platforms

Parent topic: ybcli Reference

Run operations that affect the whole system or return overall status information.

```
system appliance update <image-name> [ ignoreoff ] [ factory ]
        WARNING: The 'factory' parameter will delete all data on this system
        NOTE: The image must be copied to /tmp on the primary manager node
system appliance update list
system appliance update reattach
system appliance version
system blackout <minutes>
system chassis beacon <chassis> on | off
system chassis setup
system chassis status
system cmp beacon <chassis>/<cmp> on|off
system diagnostics [ fast | full ]
system diagnostics event <event-time> [ event-window ]
system diagnostics list
system factory
     WARNING: The 'system factory' command will delete all data on this system
system failover [ force ]
system info [ get | read | list | compare | validate ]
system maintenance [ on | off ]
system register <tenant-name> <system-name>
system register list
system register <certificate>
system serial get
system session kill <id>
system session show
system shutdown
system status
system timestamp [ on | off ]
```

appliance update

Update the Yellowbrick appliance. Specify the software image, which must be copied to the `/tmp` directory on the primary manager node. The `ignoreoff` option causes any blades that are powered off to be ignored. No attempt is made to power-cycle them.

Note: Any permissions that were granted to users on the `sys` schema are not preserved when the software is upgraded. The DBA will need to reapply these permissions.

The `factory` option resets the system to its factory defaults.

Warning: Using the `factory` option will delete all of the data on the database system.

appliance update list

List the installers that exist on the system. For example:

```
YBCLI(29459) (PRIMARY - yb00-mgr0)> system appliance update list

The following Yellowbrick installers exist on this system:
    Yellowbrick installer: /tmp/ybd-3.0.0-12050-release
```

```
    Yellowbrick installer: /tmp/ybd-3.0.0-12086-release
...
```

appliance update reattach

Use the `reattach` option to reattach to an installation that was interrupted because of a loss of network connectivity. For example:

```
YBCLI(22827) (PRIMARY - yb100-mgr0)> system appliance update reattach

This command will look for previous system update sessions still running and attach to them.
Note: Only use this command if a previous install was interrupted due to loss of network connectivity.
Continue (yes/no)? yes
...
```

appliance version

Return the appliance and `ybcli` versions that are running. This command also returns the date and time of the last software upgrade. For example:

```
YBCLI(4230) (PRIMARY - yb99-mgr0)> system appliance version

YBD appliance version        : 5.1.0-32872-release-custom
        ybos                 : 2.0.8-debug   (build: 20201021112105)
        toolchain            : 4
        ybcli                : 5.0.0         (build: 20201106204904)
        ybd_sescripts        : 2.3.0         (build: 20201106204904)
        ybd_vault            : 2.2.0         (build: 20201106204904)
        installer2           : 3.0.0         (build: 20201106204904)
        ybd_config           : 3.0.0         (build: 20201106204904)
        ybd_sed              : 2.0.0         (build: 20201106204904)
        ybd_misc             : 2.2.0         (build: 20201106204904)
        firmware-switchx2    : 9.4.2000
        firmware-cmp         : 2.97
        ybd                  : 5.1.0-release (build: 20201106204904)

YBD appliance SHA            : 297117d8a47323f12ca4ae11b03c459bec0e2f38
Software update in-progress  : NO
Software last updated        : 11-06-2020 at 15:27:00

Software update history      :
       5.1.0-32641 -> 5.1.0-32655  (succeeded - FACTORY) - 10-28-2020 at 11:15:10
       5.1.0-32655 -> 5.1.0-32654  (aborted/failed) - 10-28-2020 at 13:57:21
       5.1.0-32655 -> 5.1.0-32691  (succeeded) - 10-29-2020 at 14:06:16
       5.1.0-32691 -> 5.1.0-32712  (succeeded - FACTORY) - 11-01-2020 at 09:49:13
       5.1.0-32712 -> 5.1.0-32733  (succeeded) - 11-02-2020 at 01:21:44
       5.1.0-32733 -> 5.1.0-32741  (succeeded) - 11-02-2020 at 10:54:31
       5.1.0-32741 -> 5.1.0-32747  (succeeded) - 11-02-2020 at 11:59:16
       5.1.0-32747 -> 5.0.0-32792  (succeeded) - 11-03-2020 at 19:30:25
       5.0.0-32792 -> 5.1.0-32781  (succeeded - FACTORY) - 11-03-2020 at 20:47:37
       5.1.0-32781 -> 5.1.0-32872  (succeeded - FACTORY) - 11-06-2020 at 15:27:00
```

blackout [minutes]

Suppress alerts on the system for 30 minutes by default.

```
YBCLI(16533) (PRIMARY - yb00-mgr0)> system blackout

Doing a system blackout means all alerts will be suppressed for 30 minutes.
Continue (yes/no)? yes

This system has entered an alert blackout period of 30 minutes
```

You can also specify the length of time for the blackout, using a range of 30 to 1440 minutes. For example:

```
YBCLI(17528) (PRIMARY - yb00-mgr0)> system blackout 60

Doing a system blackout means all alerts will be suppressed for 60 minutes.
Continue (yes/no)? yes

This system has entered an alert blackout period of 60 minutes
```

chassis beacon

Specify the chassis number (`0` , `1` , or `all`) and `on` or `off` :

- `on` : turns on the blue LEDs for all installed blades in the specified chassis. The blue LED on the front panel of the chassis is also turned on if it is not already blinking.
- `off` : turns off the blue LEDs.

This command syntax also accepts `chassis0` and `chassis1` as alternatives to `0` and `1` , and it accepts `0-1` as alternatives to `all` .

For example, turn on the LEDs for chassis `0` :

```
YBCLI (PRIMARY)> system chassis beacon 0 on

Chassis 0 beacon turned on
```

For example, turn off the LEDs on both chassis:

```
YBCLI (PRIMARY)> system chassis beacon all off

Chassis 0 beacon turned off
Chassis 1 beacon turned off
```

chassis setup

Detect and configure an additional chassis on the appliance. Yellowbrick appliances support single-chassis and dual-chassis configurations. After running this command, respond to the prompts, as shown in the following example.

Note: You cannot run this command by accessing the system through the floating IP address. Use the dedicated IP address for the primary manager node.

```
YBCLI (PRIMARY)> system chassis setup

This command detects and sets up additional chassis on an existing system.
Note: All chassis must be connected to the system and powered up.
WARNING: While a chassis is being configured, the database will be shut down.

Type yes to continue: yes
Running system chassis setup

Note: In the field, Yellowbrick appliances support expansion only (the addition of chassis or blades).
Are you sure you want to expand the number of chassis on this system?

Type yes to continue: yes

Stopping Yellowbrick services prior to system chassis setup. Standby...  Done
Preparing network for multi-chassis detection. Standby... Done

Detecting chassis configuration...

Manager node is ready for multi-chassis configuration
```

```
Remote manager node
-------------------

Manager node is ready for multi-chassis configuration

Configuring chassis on this cluster
Supported chassis : 2
Detected chassis  : 2
    Chassis: 0 - Address: 192.168.2.4
    Chassis: 1 - Address: 192.168.5.4
    Chassis: 2 - Not Installed
    Chassis: 3 - Not Installed

Configuring HA. Standby... Waiting for HA stack to initialize. Standby... Done

2 chassis have been configured successfully.

The database can now be started with the 'database start' command.
```

chassis status

Return the status of the chassis configuration on the appliance. Yellowbrick appliances support single-chassis and dual-chassis configurations. For example:

```
YBCLI (PRIMARY)> system chassis status

Chassis configuration
---------------------
Found: 2 - Configured: 2

Retrieving chassis wiring details...

Chassis processor wiring
------------------------
Chassis: 0 -> CMP1 - Serial: TAB18050311170 - MAC: 38:D2:69:45:65:7E
Chassis: 0 -> CMP2 - Serial: TAB18050311164 - MAC: 38:D2:69:44:C2:21
Chassis: 1 -> CMP1 - Serial: TAB1803281113C - MAC: 38:D2:69:45:56:3A
Chassis: 1 -> CMP2 - Serial: TAB18050311174 - MAC: 38:D2:69:45:65:27

Retrieving chassis blade details...

Chassis: 0
----------
Blades installed: 11

Chassis: 1
----------
Blades installed: 11
```

cmp beacon <chassis>/<cmp> on|off

Turn the CMP beacon on or off. For example:

```
YBCLI (PRIMARY)> system cmp beacon 0/1 on
```

diagnostics [fast | full]

Send a diagnostics report to Yellowbrick for support to investigate. In addition to the sent report, a copy is left in the `/tmp` directory on the manager node.

The `fast` option does not check the status of the hardware or the blades. Because the analysis covers fewer subsystems, the report is generated much faster. The `full` option retrieves more detailed diagnostics and can take a significant amount of time. (It may also disrupt other operations on the system.) Do not specify `fast` or `full` unless requested to do so by Customer Support.

For example:

```
YBCLI (PRIMARY)> system diagnostics

This command will gather system diagnostics information and send it to Yellowbrick Data.
Are you sure you want to do this?

Type yes to continue: yes

Retrieving system log...Done
Retrieving installer log...Done
Retrieving cluster manager log...Done
Retrieving front-end database log...Done
Retrieving kernel log...Done
Retrieving HW events from blades...Done
Retrieving HW events from managers...Done
Retrieving possible blade asserts/crashes...Done
Retrieving minidumps...Done
Retrieving cluster manager status...Done
Retrieving ybstack details...Done
Retrieving stack traces...Done
Retrieving YBDB contents...Done
Retrieving full hardware and system status...Done
Compressing data...Done

System diagnostics has been collected and submitted for phonehome.

The diagnostics data has been left on this system for manual copy at: /tmp/20180706155223-ybdiag-155223.tar.gz
```

Note: If the database is not running, only a partial report can be submitted.

diagnostics event

Return the logs from the specified timestamp. The `event-time` parameter is defined using the `YYYY-MM-DDTHH:MM:SS` format.

The optional `event-window` parameter specifies the number of minutes before and after the `event-time` during which logs should be collected. The `window-event` parameter has a default of 15 minutes and a maximum of 480 minutes (8 hours).

For example:

```
YBCLI(20258) (PRIMARY - yb00-mgr0)> system diagnostics event 2020-04-02T23:00:00

Starting dump for event with event time = 2020-04-02 23:00:00
Printing logs between 2020-04-02 22:45:00 and 2020-04-02 23:15:00
This command will gather system diagnostics information and send it to Yellowbrick Data.
Are you sure you want to do this?
Response (yes/no): yes
WARNING: The database may be unresponsive, and may reconfigure while full system diagnostics is executing.
Are you sure you want to do this?
Response (yes/no): yes

Retrieving system log...Done
Retrieving installer log...Done
Retrieving installer screen log...Done
Retrieving YBCLI log...Done
Retrieving DCS ybdiag details...Done
Retrieving SMC ybdiag details...Done
Retrieving LIME ybdiag details...Done
Retrieving cluster manager log...Done
Retrieving front-end database log...Done
Retrieving DCS log...Done
Retrieving external logs...Done
Retrieving SMC log...Done
Retrieving kernel log...Done
```

```
Retrieving blade (kernel.Worker) (HW-DAEMON) (REPRINT) logs...Done
Retrieving cmp logs...Done
Retrieving HW events from blades...Done
Retrieving HW events from managers...Done
Retrieving sensor readings from managers...Done
Retrieving possible blade asserts/crashes...Done
Retrieving minidumps...Done
Retrieving cluster manager...queries...status...meminfo...workers...ybstatus...smartdata...drives...Done
Retrieving database activity and locks...Done
Retrieving database session time out info...Done
Retrieving stack traces...Done
Retrieving replication details...Done
Skipping YBDB content
Retrieving full hardware and system status...Done
Retrieving networking stats...Done
Retrieving network routes...Done
Retrieving CPU/process stats...Done
Retrieving netstat status...Done
Retrieving manager nvme list (local) ...Done
Retrieving manager nvme list (remote) ...Done
Compressing data...Done
The size of the file /tmp/20200403001211-ybdiag-event-001211.tar.gz is 1.6 MB

System diagnostics has been collected and submitted for phonehome.

The diagnostics data has been left on this system for manual copy at: /tmp/20200403001211-ybdiag-event-001211.tar.gz
```

diagnostics list

Return a list of the diagnostics packages that exist on the system. For example:

```
YBCLI(31444) (PRIMARY - yb100-mgr1)> system diagnostics list

The following Yellowbrick diagnostics packages exist on this system:
    Diagnostics package: /tmp/20190222061328-ybdiag-061328.tar.gz
    Diagnostics package: /tmp/20190222061758-ybdiag-061758.tar.gz
    Diagnostics package: /tmp/20190222062303-ybdiag-062303.tar.gz
    Diagnostics package: /tmp/20190222062040-ybdiag-062040.tar.gz
    Diagnostics package: /tmp/20190222115628-ybdiag-115628.tar.gz
```

factory

Wipe the system of all data and return it to its factory defaults. Run a low-level format on all NVMe drives. When you run the `system factory` command, you see a series of warnings and prompts for your protection. The final protective prompt will ask you to enter the manager node's hostname to ensure you are wiping the correct node. Be aware that entering the incorrect hostname will cancel the `system factory` command.

Warning: This command will delete all of the data on the database system.

```
YBCLI(20440) (PRIMARY - yb98-mgr1)> system factory

WARNING: Performing a factory reset will delete all user data, including all
tables, databases, statistics, users, keys and configuration information.
This operation cannot be undone.
Are you sure you want to do this?
Response (yes/no): yes

All data on this system will now be deleted, and the system will be reset to its factory defaults.
Please verify again that you want to complete this operation.
Continue (yes/no)? yes

Please verify whether system factory should be run on this system:
System factory about to be performed on:
```

```
System IP                 : 10.10.198.10
Local manager node hostname: yb98-mgr1
Local manager node IP      : 10.10.198.14

Remote manager node hostname: yb98-mgr0
Remote manager node IP      : 10.10.198.12

Database running        : YES
Database ready          : YES
Database read-only      : NO
Database uptime         : 00:15:24
Database users connected: 4 (including system users)

Enter local manager node hostname to continue: yb98-mgr1
```

In the background, the `system factory` command performs the following actions:

- Runs a `ybinit` (database initialization command) to clean out the database.
- Removes old log files from the manager nodes.
- Removes all users (except `ybdadmin`) from the manager nodes.
- Does a low-level erase of all SSDs on the blades (if requested at the prompt).
- Generates new ssh keys for the manager nodes.

After this command is run, the system is fully clean. This command does not take the manager nodes fully back to their defaults (that is, it cannot detect if an administrator made custom changes to the manager nodes). The compute blades are provisioned as compute nodes and the system is ready for use.

failover [force]

Fail over to the other manager node. First, connect to one of the manager nodes directly. You cannot perform a failover while connected to the floating IP address for the HA cluster.

The `force` option forces a system failover regardless of other commands that are currently executing on the manager node. This option is needed occasionally in some emergency situations.

```
[ybdadmin@yb100-mgr0 ~]$ ybcli system failover

Current cluster roles:
Local node  : PRIMARY   - ACTIVE
Remote node : SECONDARY - ACTIVE
Failing over to another node is a disruptive process and not guaranteed to work
This should only be done if the current manager node is malfunctioning

Are you sure you want to do this?
Type yes to continue: yes

Initiating system failover
Monitoring completion. This can take 2 minutes. Notifications may appear. Standby...

System failover was successful. Yellowbrick database started.
Primary manager node is now: Remote node (yb100-mgr1.ybtest.io)

WARNING: A SYSTEM NODE ROLE CHANGE WAS DETECTED
Current roles
-------------
LOCAL NODE  : SECONDARY (ACTIVE)
REMOTE NODE : PRIMARY   (ACTIVE)
```

After the failover, logging into `ybcli` returns:

```
[ybdadmin@yb100-mgr1 ~]$ ybcli
...
No redundant manager node detected
YBCLI is currently running on the PRIMARY manager node.
Local manager node : yb100-mgr1.ybtest.io -> (PRIMARY ACTIVE)
Remote manager node: NOT PRESENT
```

system info [get | read | list | compare | validate]

Note: The `system info` commands are specific to Andromeda and will not work on Tinman.

Retrieve detailed information about the appliance components. The `system info` commands are used during the initial installation, system maintenance, and warranty replacements.

maintenance

Put the system into maintenance mode (`on`) or take it out of maintenance mode (`off`). In maintenance mode, the system does not accept database client connections (however, you can still log in to the manager nodes via `ssh`).

```
YBCLI (PRIMARY)> system maintenance on

Enabling system maintenance mode will also shut down the database layer

Are you sure you want to do this?
Type yes to continue: yes

Stopping YBD services for maintenance mode. Standby... Done
Successfully enabled system maintenance mode
```

Run `system status` to find out if the database is currently in maintenance mode.

register <tenant-name> <system-name>

Register the system for the Yellowbrick phonehome application. Enter a user-defined tenant name and the name of the system. For example:

```
system register YB yb100
```

Respond to the prompts and provide the requester name (your full name).

register list

List all compatible certificate files under `/tmp` for in-field registration. For example:

```
YBCLI(69050) (PRIMARY - yb98-mgr0)> system register list

The following p12 certificates exist on this system:
       newcustomer-newcluster1.p12
```

register <certificate>

Register the system with Phonehome using a certificate file for customers who do not have access to Phonehome. For example:

```
YBCLI(69050) (PRIMARY - yb98-mgr0)> system register newcustomer-newcluster1.p12

This cluster appears to already be registered. Do you want to re-register?
Response (yes/no): yes
```

```
Are you sure you want to register this system using:
Certificate bundle : /tmp/newcustomer-newcluster1.p12
Response (yes/no): yes

Performing system registration. Standby... Done
System registration was successful
The database must be restarted for the new certificate to take effect
```

serial get

Retrieve detailed information about the appliance components. See also `system info [ get | read | list | compare | validate ]` above.

session kill <id>

Kill a system session by providing the session ID. Use `session show` to list session IDs.

```
YBCLI(31444) (PRIMARY - yb100-mgr1)> system session kill 31444
...

YBCLI(31444) (PRIMARY - yb100-mgr1)> system session kill 31444

YBCLI cannot kill its own session.

YBCLI(31444) (PRIMARY - yb100-mgr1)> system session kill 40000

No YBCLI session with id: 40000 is currently executing.
```

session show

Return a list of active `ybcli` sessions and their session IDs:

```
YBCLI(31444) (PRIMARY - yb100-mgr1)> system session show

YBCLI ID:   3904 User: ybdadmin
YBCLI ID:   6123 User: user2
YBCLI ID:  22712 User: user2
YBCLI ID:  22893 User: ybdadmin
YBCLI ID:  24690 User: ybdadmin
YBCLI ID:  25025 User: user3
YBCLI ID: 31444 User: ybdadmin (this session)

7 YBCLI session(s) found running on this manager node.
```

shutdown

Shut down the entire appliance and all blades. See also Powering the Appliance Off and On.

```
YBCLI(70095) (PRIMARY - yb98-mgr0)> system shutdown

Shutting down the system will halt all blades and all manager nodes.
When in this state, all components may safely be powered off.
To start the system again, all components will have to be power cycled manually.
Continue (yes/no)? yes

Stopping services. Standby...  Done

Shutting down all blades...
Gracefully shutting down blade in bay:  1 -> OK
Gracefully shutting down blade in bay:  2 -> OK
Gracefully shutting down blade in bay:  3 -> OK
Gracefully shutting down blade in bay:  4 -> OK
Gracefully shutting down blade in bay:  5 -> OK
```

```
Gracefully shutting down blade in bay:  6 -> OK
Gracefully shutting down blade in bay:  7 -> OK
Gracefully shutting down blade in bay:  8 -> OK
Gracefully shutting down blade in bay:  9 -> OK
Gracefully shutting down blade in bay: 10 -> OK
Gracefully shutting down blade in bay: 11 -> OK
Gracefully shutting down blade in bay: 12 -> OK
Gracefully shutting down blade in bay: 13 -> OK
Gracefully shutting down blade in bay: 14 -> OK
Gracefully shutting down blade in bay: 15 -> OK
Gracefully shutting down blade in bay:  1 -> OK
Gracefully shutting down blade in bay:  2 -> OK
Gracefully shutting down blade in bay:  3 -> OK
Gracefully shutting down blade in bay:  4 -> OK
Gracefully shutting down blade in bay:  5 -> OK
Gracefully shutting down blade in bay:  6 -> OK
Gracefully shutting down blade in bay:  7 -> OK
Gracefully shutting down blade in bay:  8 -> OK
Gracefully shutting down blade in bay:  9 -> OK
Gracefully shutting down blade in bay: 10 -> OK
Gracefully shutting down blade in bay: 11 -> OK
Gracefully shutting down blade in bay: 12 -> OK
Gracefully shutting down blade in bay: 13 -> OK
Gracefully shutting down blade in bay: 14 -> OK
Gracefully shutting down blade in bay: 15 -> OK
Gracefully shutting down blade in bay:  1 -> OK
Gracefully shutting down blade in bay:  2 -> OK
Gracefully shutting down blade in bay:  3 -> OK
Gracefully shutting down blade in bay:  4 -> OK
Gracefully shutting down blade in bay:  5 -> OK
Gracefully shutting down blade in bay:  6 -> OK
Gracefully shutting down blade in bay:  7 -> OK
Gracefully shutting down blade in bay:  8 -> OK
Gracefully shutting down blade in bay:  9 -> OK
Gracefully shutting down blade in bay: 10 -> OK
Gracefully shutting down blade in bay: 11 -> OK
Gracefully shutting down blade in bay: 12 -> OK
Gracefully shutting down blade in bay: 13 -> OK
Gracefully shutting down blade in bay: 14 -> OK
Gracefully shutting down blade in bay: 15 -> OK

Powering off all blades...
Blade(s) in chassis: 0 were instructed to power off
Blade(s) in chassis: 1 were instructed to power off
Blade(s) in chassis: 2 were instructed to power off
Waiting for blade(s) to power off
Blades powered off: 0/45
Blades powered off: 1/45
Blades powered off: 30/45
Blades powered off: 34/45
Blades powered off: 45/45

Shutting down remote manager node. Standby...
Initiating shutdown. This process can take 60 seconds
Shutting down local manager node. Standby...
Initiating shutdown. This process can take 60 seconds
Shutting down YBD services if running
Shutting down cluster services. This can take up to 120 seconds
Stopping Cluster (pacemaker)... Stopping Cluster (corosync)...
Connection to yb98-mgr0 closed by remote host.
Connection to yb98-mgr0 closed.
```

status

Return overall system status information. For example, when run on Tinman:

```
YBCLI(30354) (PRIMARY - yb98-mgr0)> status system

Manager nodes configured: 2
---------------------------
Node 1 (PRIMARY - LOCAL NODE ) : yb98-mgr0 -> ONLINE
Node 2 (SECONDARY - REMOTE NODE ) : yb98-mgr1 -> ONLINE

HW Platform                       : Tinman
Database system running           : YES
Database system ready             : YES (Responding: YES)
Database system read-only         : NO
Database system catalog rowstore  : NORMAL
Database system rowstore          : NORMAL
Database system storage used      : 20%
Database system uptime            : 14:59:42
System work status                : gc:idle  parityrebuild:idle  analyzer:idle  system:idle  user:active,idle
Data collection running           : YES
Blade parity                      : Enabled
Blade parity rebuilding           : NO (Progress: N/A)
Blade data check in-progress      : NO
Shard rewriting                   : NO (0 shards pending)
Capacity expansion in-progress    : YES (5 Tables and 1752898 MBs remaining)
Cluster degraded mode             : YES (Reason: Drive(s) failed)
Maintenance mode                  : NO
Software update in-progress       : NO (Version: 4.1.1-24805)
Floating system IP                : 10.10.102.10 - 255.255.255.0
System registered                 : NO
LDAP status                       : Not configured
Logging status                    : Enabled
Encryption keystore               : Available - Status: Not setup
Chassis configuration             : Found: 1 - Configured: 1
Add-ons                           : None
```

when run on Andromeda:

```
YBCLI(51453) (PRIMARY - y2b23-mgr0)> system status

Manager nodes configured: 2
---------------------------
Node 1 (PRIMARY   - LOCAL NODE  ) : y2b23-mgr0.slc.yellowbrick.io -> ONLINE
Node 2 (SECONDARY - REMOTE NODE ) : y2b23-mgr1.slc.yellowbrick.io -> ONLINE

HW Platform                       : Andromeda
Database system running           : YES (9/9/9 active/powered/installed blades)
Database system ready             : YES (Responding: YES)
Database system read-only         : NO
Database system catalog rowstore  : NORMAL
Database system rowstore          : NORMAL
Database system storage used      : 20%
Database system uptime            : 00:36:20
System work status                : gc:idle  parityrebuild:idle  analyzer:idle  system:idle  user:idle
Data collection running           : YES
Blade parity                      : Enabled
Blade parity rebuilding           : NO (Progress: N/A)
Blade data check in-progress      : NO
Shard rewriting                   : NO (0 shards pending)
Capacity expansion in-progress    : YES (3 Tables and 764036 MBs remaining)
Cluster degraded mode             : NO
Maintenance mode                  : NO
Software update in-progress       : NO (Version: 5.1.0-33380)
Floating system IP                : 10.10.223.10 - 255.255.255.0
System registered                 : YES (Tenant: yb - System: y2b23)
LDAP status                       : Not configured
Logging status                    : Enabled
Encryption keystore               : Available - Status: READY - Locked: YES
```

```
Chassis configuration              : Found: 1 - Configured: 1
System FRU configuration           : All components match records
Add-ons                            : None
```

Note: In the Database system running line, the active/powered/installed blades text will be yellow if not all installed blades are active or online.

Attention: The statistics under `Capacity expansion in-progress` only update once a table has been completely moved. Therefore, it may appear at times as though no progress is being made.

timestamp on | off

Turn on timestamp display for `ybcli` commands. `Execution Start` and `Execution End` times are displayed for each command. This command is effective per user per `ybcli` session. The default is `off`. For example:

```
YBCLI(5422) (PRIMARY - yb100-mgr0)> system timestamp on

System execution timestamp has been turned on.
Execution End: Fri Apr  5 12:25:03 PDT 2019
...
YBCLI(5422) (PRIMARY - yb100-mgr0)> status blade 0/1

Execution Start: Fri Apr  5 12:51:13 PDT 2019
Chassis:  0
-----------
Blade Bay:  1 -> BOOTED  UUID: 00000000-0000-0000-0000-38B8EBD00578 - Version: YBOS-2.0.2-DEBUG
         BIOS: v05.04.21.0038.00.011 - Memory total/free: 65587652/1788928 KiB
         CPU: Intel(R) Xeon(R) CPU E5-2618L v4 @ 2.20GHz - Cores: 10 - Load: 66%
         Address: 192.168.10.10 - Uptime: 0 day(s), 01:06:45 - Worker: Running
         Encryption Supported: YES - Encryption Enabled: NO - locked: N/A
         Cluster status: OPERATIONAL - Cluster role: MEMBER - Last seen: just now

Execution End: Fri Apr  5 12:51:15 PDT 2019
```

Chassis Management

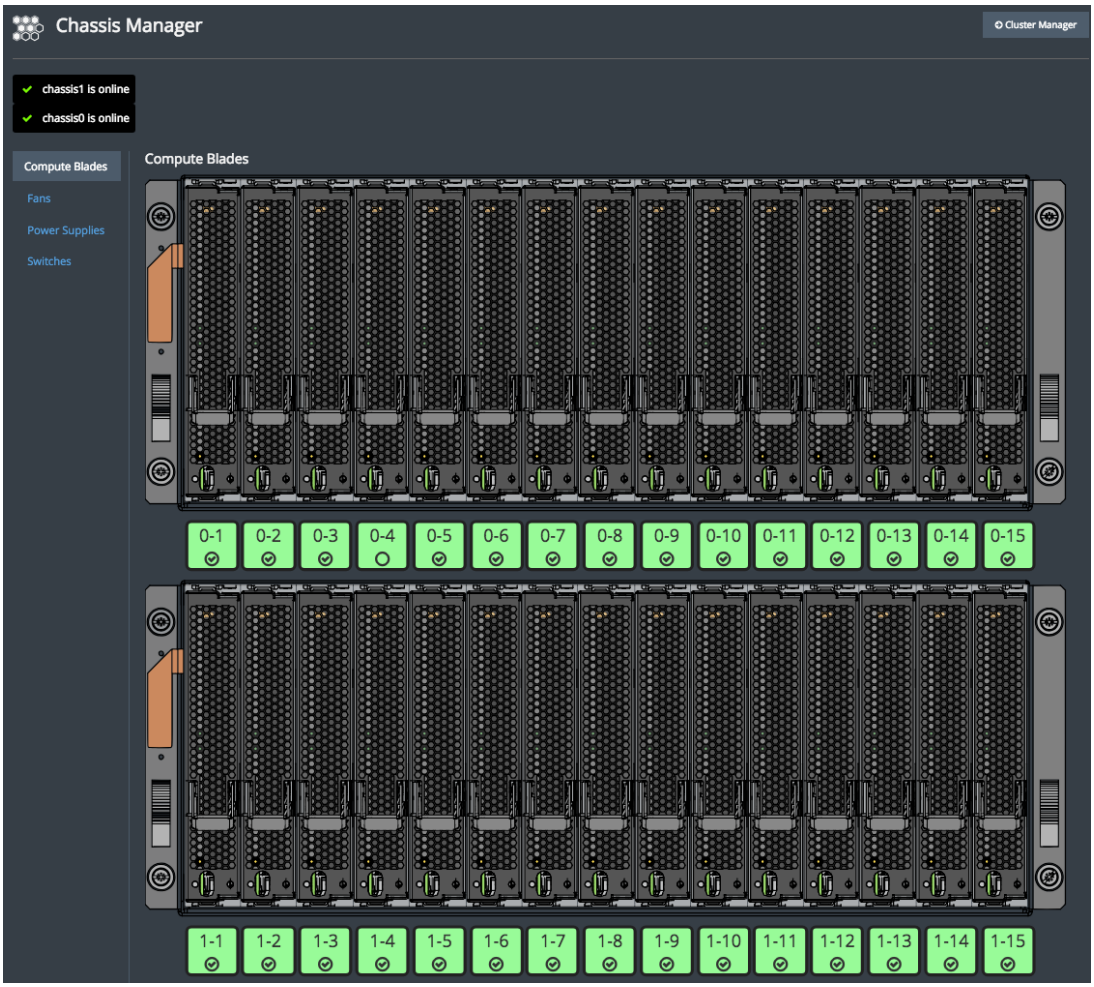
The Chassis Manager page displays the real-time status of each hardware component in the appliance (in the SMC, go to **Manage** > **Chassis**). You can use this page to monitor the current state of the appliance and investigate error conditions.

Note: Your Yellowbrick appliance may be a single-chassis or dual-chassis system. The examples in this section describe a dual-chassis system.

The Chassis Manager page has four main tabs.

Compute Blades

This tab provides a graphical view to assess the overall status of each compute blade on each chassis in the appliance, with information on both the physical status and power status:



Each vertical image displays the status of a single compute blade as it is physically inserted into an appliance chassis. This example shows a dual-chassis appliance. Each compute blade image provides information on its physical state, power status, light status, and software configuration status.

Physical Status and Power Status

The graphical image of the compute blades provides information on both the physical status and power status:

Graphic	Physical Status	Power Status
Image as shown above	Blade inserted	On
Image above with dark gray background	Blade inserted	Off
Image above as yellow	Blade inserted, powered on, being updated to match the cluster version level	On
Flat gray image (no graphic)	No blade inserted	N/A

LED Light Status

At the top of each compute blade image, the LED light status is shown. This status matches the physical LED lights that are visible when you look at the front of the flash itself. Each blade has three LED lights: green, red, and blue. The states of each light are: Off, On, Fast flash, Slow flash. The meaning of each light status is as follows:

LED Light Status	Description
All off	Power off
All fast blinking	Power on, blade starting
All slow blinking	Power on, OS or compute node software booting
Green on	Power on, booted
Red on	Hardware error
Blue on	Beacon activated

During normal operation, the green LED light is on. If the compute blade experiences a hardware error, the red LED light will turn on.

The blue LED light is a beacon light for administration purposes. Administrators can manually turn the blue beacon light on or off in either the Cluster Manager SMC page or by using the ybcli `blade` command. Turning on the beacon light visually indicates which compute blade needs servicing or replacement, avoiding operator errors during service events.

Note: On the real physical compute blades, there is an additional LED light at the bottom of each blade. This light, when flashing, indicates storage I/O activity. However, on this SMC page the light is shown statically as always on and simply shows the light's physical position.

Compute Blade Software Status

The graphical box below each compute blade (the green boxes in the screen capture) provides information on the software status for each blade. There are two indicators: the color of the box and the icon that is shown.

Color Indicator

|Color|Description| |-----|-----------| |Green|Software compute node online| |Red|Software compute node in error state| |Dark gray|Software compute node offline| |Light gray|Compute blade physically removed| |Blue|New compute blade physically inserted but not added to the cluster|

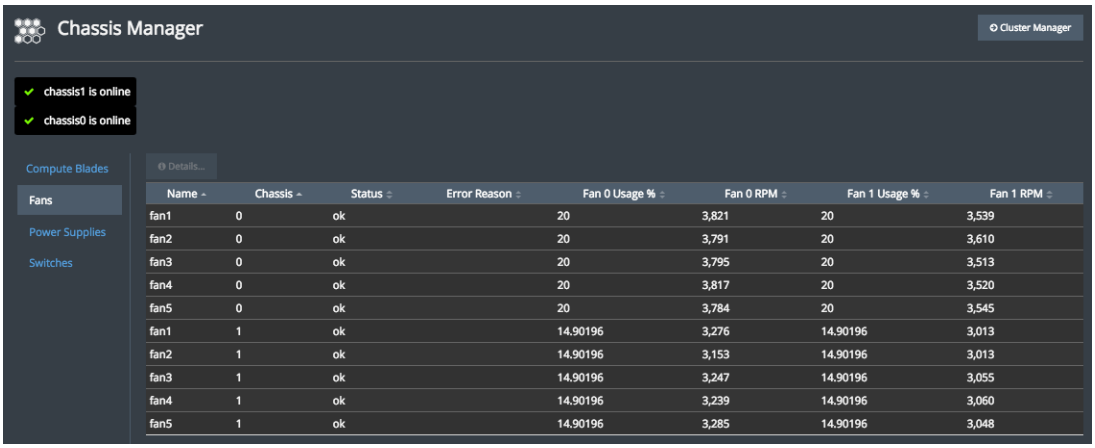
Icon Indicator

|Icon|Description| |----|-----------| |Circle with checkbox|Software compute node registered OK| |Circle with no checkbox|Software compute node registered as spare| |Circle with `x`|Cluster unable to communicate with software compute node on compute blade| |Triangle with `!`|Software compute node in error state|

Fans

This tab shows the status of each fan in the appliance.

The appliance has 5 fan slots that are hot-swappable, and each fan slot contains 2 fans. Fans are redundant and provide n+2 protection by slots. For example:



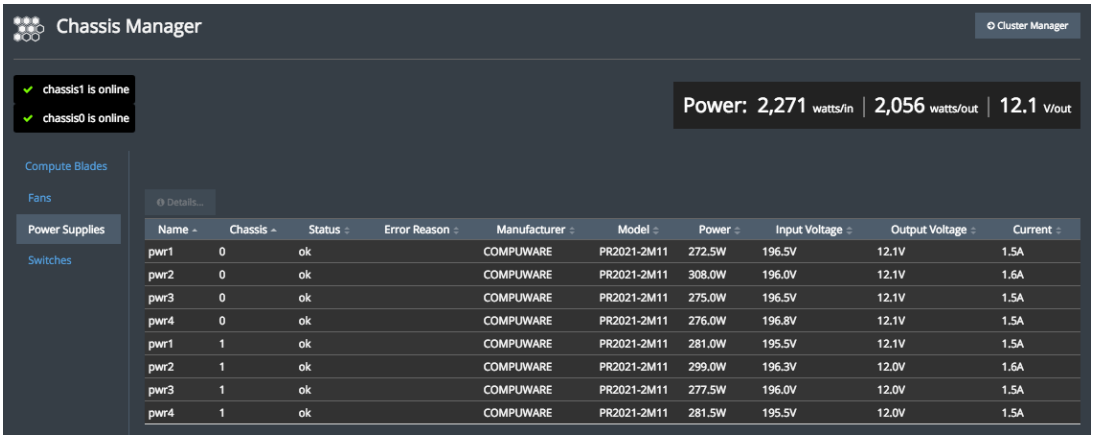
Name	Chassis	Status	Error Reason	Fan 0 Usage %	Fan 0 RPM	Fan 1 Usage %	Fan 1 RPM
fan1	0	ok		20	3,821	20	3,539
fan2	0	ok		20	3,791	20	3,610
fan3	0	ok		20	3,795	20	3,513
fan4	0	ok		20	3,817	20	3,520
fan5	0	ok		20	3,784	20	3,545
fan1	1	ok		14.90196	3,276	14.90196	3,013
fan2	1	ok		14.90196	3,153	14.90196	3,013
fan3	1	ok		14.90196	3,247	14.90196	3,055
fan4	1	ok		14.90196	3,239	14.90196	3,060
fan5	1	ok		14.90196	3,285	14.90196	3,048

The following table describes the information on the Fans tab:

Column	Description
Name	The physical fan slot
Chassis	Chassis number (0 or 1).
Status	Current status
Error Reason	Description if in error state
Fan 0 Usage %	For the 1st fan, the current % of max RPM
Fan 0 RPM	For the 1st fan, the current RPM
Fan 1 Usage %	For the 2nd fan, the current % of max RPM
Fan 1 RPM	For the 2nd fan, the current RPM

Power Supplies

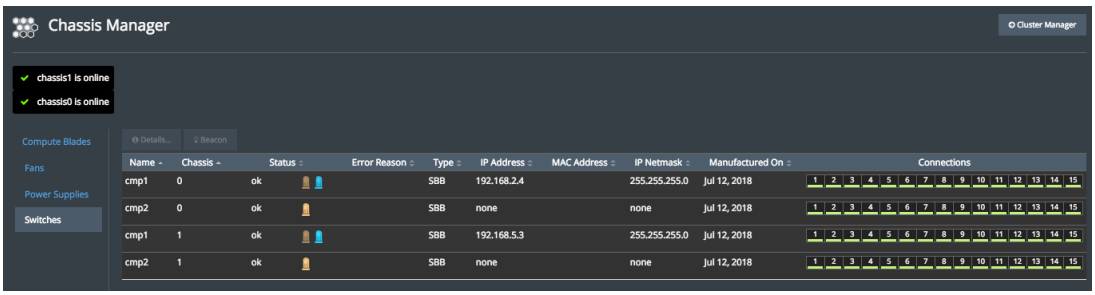
This tab shows the status of each power supply in the appliance. Appliances may have 2 to 4 power supplies per chassis depending on the number of compute blades that are installed. Power supplies are redundant and provide n+2 protection in 15-node configurations. Information on the status, manufacturer and model, and current power consumption, amps, and voltages is provided. For example:



Additionally, the current power consumption of the appliance is summarized at the top of the page. This power consumption is just for the compute blade chassis and does not include power drawn from the two manager nodes.

Switches

This page shows the status of each internal InfiniBand switch in the appliance. Appliances contain 2 HA redundant InfiniBand networks per chassis, with one in the active state and the other in standby state. The status, error reason, IP address, and port connections to each blade are listed. For example:



Checking the Logs

Platforms: EE: All appliance platforms

Parent topic: Appliance

When queries return warnings or errors, you can check the log files by using the SMC or ybcli commands.

In the SMC, go to **Manage** > **Logging**. The SMC provides controls for filtering the logs and setting the log level.

In the `ybcli`, use the following commands to tail the logs or return log entries for some number of hours up to the last 24:

- `log blade`: Return the log for one blade or all blades.
- `log database`: Return the log for database operations.
- `log manager`: Return the log for one or both manager nodes.

For examples and more details, see log.

Manager Node Failover

The Yellowbrick appliance was designed with redundancy in mind and has no single point of failure. Furthermore, the system can sustain multiple component failures and automatically fail over to redundant hardware that is detected to be operational. Depending on which component fails, the failover process may be fully transparent with no downtime, or it may require a small disruption to services (<60 seconds). Most failures are handled transparently with no disruption.

Failover Scenarios

You can inspect the system at any time to check the health of its components. You can use the System Management Console (SMC) or the Yellowbrick command line interface (ybcli) for this purpose.

Should a catastrophic event occur that causes a failover between the manager nodes, the victim of that process is taken out of the cluster and put into standby mode if it is still responding. Once the failure condition has been cleared, you can bring the manager node back into the cluster manually. This process is described in this section.

During system maintenance, you should put the cluster into maintenance mode. You can do this either via the SMC or by using the following `ybcli` command:

```
system maintenance on
```

This command stops the cluster from responding to changes in the environment that would otherwise cause node failover to happen. It will also shut down the Yellowbrick database stack. When maintenance is complete, you can bring the system back to normal operating mode by running the following `ybcli` command:

```
system maintenance off
```

HA Cluster Status

Run the following `ybcli` command at any time to check the status of the HA cluster:

```
system status
```

This command returns the status of various resources in the system. All of these resources always run on the same manager node: the primary manager node. For example, when run on Tinman:

```
YBCLI (PRIMARY)> system status

Cluster nodes configured: 2
---------------------------
Node 1 (PRIMARY   - LOCAL NODE  ) : yb100-mgr0.yellowbrickroad.io -> ONLINE
Node 2 (SECONDARY - REMOTE NODE ) : yb100-mgr1.yellowbrickroad.io -> ONLINE

HW Platform               : Tinman
Database system running   : YES
Database system ready     : YES (9/9 active/installed blades)
Database system read-only : NO
...
```

In this example, the `yb100-mgr0` node is currently the primary node running all database resources. Besides `yb100-mgr0`, the other manager node, with a hostname of `yb100-mgr1`, is also online and ready to take over.

For example, `yb100-mgr0` is the primary manager node and is serving all user requests; `yb100-mgr1` is the secondary manager node and will automatically take over resources if `yb100-mgr1` becomes unavailable.

Manual Failover

If it becomes necessary to manually fail over from one manager node to the other, you can use the `ybcli` to complete the task.

1. Start the `ybcli` as the `ybdadmin` user on either the primary or secondary manager node.
2. Run the `system status` command to verify that the secondary manager node is present.
3. Run the `system failover` command to perform a manual failover.

Note: For a manual failover, it can take 2 minutes before the system responds again at the floating IP address.

It is important to log in to the primary node directly when running the `system failover` command because the floating IP address will briefly be taken down as it moves between the nodes.

Using the `yb100-mgr0` and `yb100-mgr1` systems, failing over to `yb100-mgr1` would look like this:

```
Initiating system failover
Monitoring completion. This can take 2 minutes. Notifications may appear. Standby...

System failover was successful. Yellowbrick database started.
Primary manager node is now: Remote node (yb100-mgr1.yellowbrick.io)

WARNING: A SYSTEM NODE ROLE CHANGE WAS DETECTED
Current roles
------------
LOCAL NODE  : SECONDARY (ACTIVE)
REMOTE NODE : PRIMARY   (ACTIVE)
```

Now the `yb100-mgr1` node is running all resources. Once the failover is complete, you can fail back to the original node, which in this case was `yb100-mgr0`. `ybcli` will automatically determine when it is safe to do so. If the system is not capable of failing over, `ybcli` reports an error to explain the problem. This may happen if large amounts of data are being replicated between the nodes.

Use the `system status` command to determine when the process is complete.

Manager Node Ports

Platforms: EE: All appliance platforms

Parent topic: Appliance

The manager nodes have multiple ports for client application connections, system management connections, and connections to external services. The Manager nodes firewall is configured to allow these connections. To make changes to the manager node firewall settings, contact Technical Support; do not attempt to make changes to the firewall settings yourself.

Ports for Client Applications

All client application and user connections to Yellowbrick appliances occur only through the manager node via the following ports. (See also Opening Network Ports for Clients.)

Port	Purpose	Protocol	Notes
22	ssh	TCP	Used for database upgrades and administration.
80	SMC	HTTP	HTTP server port
8182	SMC	HTTP	Internal port for SMC. External connections can be blocked.
443	SMC	HTTPS	TLS versions 1.1 and 1.2 only
5432	Yellowbrick database	TCP	Default port for database connections for all protocols: ODBC, JDBC, libpq, and so on. This port can be changed.
11111	`ybtools` control port	TCP	Control port used for `ybload`, `ybunload`, `ybbackup`, and `ybrestore`.
11112	`ybtools` control port	TCP	Control port used when the `--secured` option is specified for `ybload`, `ybunload`, `ybbackup`, `ybrestore`.
31000 and 31001	`ybtools` data transfer	TCP	

BMC Ports

The Baseboard Management Controller (BMC) is the lights-out management application for the manager nodes. It may be on the same or a different network from the manager nodes. Only Yellowbrick system managers should need access to the BMC.

Port	Purpose	Protocol	Notes
22	ssh	TCP	Used for database upgrades and administration.
80	BMC Admin UI	HTTP	HTTP server port
443	BMC Admin UI	HTTPS	TLS versions 1.1 and 1.2 only
5900	BMC	TCP	Management port

Connections to External Services

By default, the manager node attempts to connect to the following external services.

Service	Host:port	Protocol	Purpose/Notes
DNS service	`…:53`	TCP	- Look up `phonehome.yellowbrick.io` - Resolve host names in authentication, etc. - Disabling creates pauses in operations
NTP time server service	`time.nist.gov:123`	UDP	- Time sync server - Can be disabled or configured to contact a different NTP server
Red Hat Subscription	`subscription.rhn.redhat.com:443`	TCP	Can be blocked; not needed.
Yellowbrick remote diagnostics (phonehome)	`phonehome.yellowbrick.com:443`	TCP	Only needed if phonehome is enabled; it is not enabled by default. See Remote Diagnostics for configuration instructions.

Powering the Appliance Off and On

Platforms: EE: All appliance platforms

Parent topic: Appliance

Please be aware that powering the appliance down and back up is not the same as restarting the database. Powering down and back up is typically only necessary for appliance cabinet maintenance or transport.

Shutting Down The Appliance

1. Start `ybcli` and run the following command to shut down the database and power off the blades and both manager nodes:

```
system shutdown
```

Powering Up An Appliance That Has Been Shut Down

1. If you have physical access to the system, power on manager node 0 using the power button and continue to step 3.
2. If you don't have physical access to the system, complete the following steps:
3. Connect to the BMC in your browser:

   ```
   http://<bmc_ip>
   ```

   Note: Replace `<bmc_ip>` with the IP address of your BMC.

   You will receive a certificate warning because BMC uses a self-signed certificate. You can ignore the certificate warning.

4. For Tinman, log in with username `YELLOWBRICK` and password `YELLOWBRICK`.

   Note: If the above credentials do not work, use `ADMIN` / `ADMIN` to log in.

   For Andromeda, log in with username `ADMIN` and password `Yellowbr1ck`.

5. Select **Remote Control** in the top menu bar.
6. Select **Power Control** in the left menu.
7. Power on manager node 0.
8. Wait at least one minute after starting manager node 0 before turning on manager node 1 by repeating step 1 or 2 for manager node 1.

The system should come back online within 5 - 10 minutes.

Validating The Appliance

1. Open a terminal session to manager node 0 either via `ssh` or the BMC remote console.

- To access the remote console, log in to the BMC at **Remote Control > iKVM/HTML5**.
- Log in as `ybadmin` (password: `YBDmgmt`).

2. Start `ybcli`. If you receive a notification that the system is still starting up, continue waiting.

Make sure both systems have been up for at least three minutes before continuing.

3. Power on all blades by running:

```
blade poweron all
```

4. Verify the system by running the following commands:

```
health blade all
```

```
status blade all
```

```
system status
```

5. When all blades are fully booted, start the database by running:

```
database start
```

Note: For systems using encryption, all blades and managers must be unlocked before the system is operational. When started after a total system shutdown, `ybcli` will ask for the encryption keys and perform encryption unlock automatically.

Starting and Stopping the Database

Platforms: EE: All appliance platforms

Parent topic: Appliance

If necessary, you can use the `ybcli` to start or stop the Yellowbrick database software independently. These commands do not start up or shut down the system as a whole. The manager nodes and compute nodes are not affected.

```
database start
database stop
```

For example:

```
YBCLI (PRIMARY)> database stop

This will stop the Yellowbrick database and disconnect all database clients
Are you sure you want to do this?
Type yes to continue: yes

Stopping Yellowbrick Database services. Standby... Done
```

See also ybcli Reference.

System Maintenance

Platforms: EE: All appliance platforms

Parent topic: Appliance

The Yellowbrick appliance requires very little maintenance; however, you may need to verify that various components are working as intended.

The best way to get an overview of the status of the system is to start the ybcli. On a healthy system, the `ybcli` returns information like this:

```
Yellowbrick Data CLI vx.x.x (Genx)
Copyright (c) 2016 Yellowbrick Data, Inc.
All rights reserved
----------------------------------------
Redundant manager node detected
YBCLI is currently running on the PRIMARY manager node.
Local manager node : yb100-mgr0.yellowbrick.io (PRIMARY ACTIVE)
Remote manager node: yb100-mgr1.yellowbrick.io (SECONDARY ACTIVE)

Type 'help' for a list of supported commands

YBCLI (PRIMARY)>
```

The output includes detailed information about which nodes are present in the system, their role, and current status. Any errors, such as when no primary node is found or if the system is in maintenance mode, are reported during `ybcli` startup.

The best way to get a complete overview of system health is to run the health all `and` status all commands. These two commands return the status of all components, including blades, power supplies, fans, external processors, and so on.

Each subsystem may also have one or more alerts. Alerts may describe increased thermal conditions, unstable power, or blades not operating nominally. For example, the following command retrieves any outstanding hardware alerts for the blade in bay 1:

```
YBCLI (PRIMARY) health blade 1
Bay:  1 Status: ok Power: on LED: [ OFF OFF ON  ] FRU: C4-0005-01 R01 Serial: TAA160802003DE CPU: Booted (YBOS)

Retrieving blade alerts...

Blade alerts reported: 1
Bay:  1: volt_00_pvccin_cpu_mV → 2234 (max error)
```

It is also useful to verify that the manager nodes are actively replicating their block devices to each other over the high-speed network. Note that the system software will make sure that only a node that has an up-to-date block device can ever be promoted to the primary manager node. To view the status of block replication, run the `health storage` command on any node that is not in standby state:

```
...
Replicated block device mounted: OK
Data replication active: OK
Data replication in progress: NO
...
```

In this example, the command was run on the primary node. It shows that block replication between the manager nodes is running and healthy, and that replication is not currently in progress.

You can take a manager node out of the cluster by putting it into standby mode. This means that the node will stop participating in the cluster and will not be considered for automatic system failover. However, you can still initiate a manual system failover to the node.

Note: After a node is put into standby mode, it will not actively replicate data blocks from the primary node. When it is brought out of standby mode, the node will resync its data store. Depending on how long the node was in standby mode and how much was written to the primary node in that time frame, it could take up to 2 minutes to fully synchronize the data store. The health storage command provides accurate information about the progress of resynchronizing the block device.

Disk Health

The manager nodes each have 4 block devices: 2 are mirrored for the operating system, and 2 are mirrored for the data in the Yellowbrick database. The latter is also actively synchronized to the secondary manager node.

You can check the status and health of all manager node disk drives by looking at the output of the `health storage` command.

A drive can be removed at any time and replaced with a drive provided by Yellowbrick. If possible, putting the system into maintenance mode is recommended before any planned hardware replacement.

Restarting the Database After a Power Failure

In case of a power failure that brings down both manager nodes, when the system is powered back on, you can use the system status command to check the overall state of the system. The database may not come back up, in which case you can start it with the database start command.

User Accounts

Platforms: EE: All appliance platforms

Parent topic: Appliance

This section describes the types of user accounts that exist on the Yellowbrick appliance, their usage, and the default accounts of each type that are created with a new installation.

User accounts fall into two main categories: appliance accounts and database accounts. Appliance accounts monitor and manage the appliance and hardware from the manager node command line interface; these accounts are not database users. Database user accounts exist only in the database and are not appliance users.

An initial appliance installation creates two login user accounts for administration. Their passwords can be changed but otherwise they are not configurable.

Usage	Default Username/Password
Appliance administrator superuser	`ybdadmin/YBDmgmt`
Database administrator superuser	`yellowbrick/yellowbrick`

Other appliance and database internal service accounts are created that the administrators should not modify. They are discussed in more detail later in this section.

Appliance User Accounts

Appliance users may execute appliance management commands within the Yellowbrick command line interface, `ybcli`, to perform actions such as starting the database and monitoring the appliance status. Appliance admins are also able to perform appliance configuration commands on the appliance hardware and users. See config. Additionally no-login accounts exist for Yellowbrick services.

Type	Username	Description
Administrator	`ybdadmin`	- Appliance administrator account for the Yellowbrick appliance. - Used for all aspects of appliance configuration and management. - Only account with permission to create other appliance users. - Also exists as a database no-login superuser.
User	Administrator-created	- Created by the administrator, `ybdadmin`. - Users can be created with or without administrator privileges.
Service	`ybddcs`, `ybdsmc`, and so on	- No-login accounts that own Yellowbrick services; System Management Console (SMC), phonehome Data Collection Services (DCS), and so on.

Database User Accounts

Database administration (superuser) and service accounts are created as part of the appliance initial installation and should not be modified except for changing the password of the `yellowbrick` user.

Type	Username	Description

Type	Username	Description
Administrator	`yellowbrick`	- Database administrator (superuser) with all privileges across all the databases in the Yellowbrick appliance. - Used to create additional superuser and end-user accounts, grant permissions, access all system catalog tables, and SMC. - Can use SQL-based administration tools and commands (such as `ybsql`). - Can use Java-based bulk operations including backup and restore operations (`ybbackup`, `ybrestore`), and load and unload operations (`ybload`, `ybunload`).
User	Administrator-created	- No default end-users or additional administrators are created during installation. - Database end-user accounts define an individual's access rights. See Managing Database Users and Roles. - You can create database users with SQL or in the SMC .
Roles	Administrator-created	- The same applies to "roles" (similar to groups), which are users without the login attribute.
Service	`sys_ybd_*`	- Several no-login internal superuser accounts used by Yellowbrick services. - These services run background operations, such as flushing and analyzing tables.
SMC Roles	`sys_ybd_smc`, `sys_ybd_smc_admin`, `sys_ybd_smc_manager`, `sys_ybd_smc_reporter`, `ys_ybd_smc_viewer`.	- The SMC has the `sys_ybd_smc` superuser and four non-login, non-superuser accounts. - You can create database users in the SMC with "console access" privileges at these four levels.

Using Network Tracing

One of the diagnostic features Yellowbrick provides is network tracing to assist in troubleshooting network issues. When enabled on an appliance, the network tracing feature captures data packets sent and received on the network interface port, including header, timestamp, source, and destination information. Note that the network tracing feature is disabled by default. Files produced by network tracing are all in pcap format and stored on the disk in `/tmp/network_trace`.

For more specific captures, filters are applied to network tracing. Captures can target packets on the SQL port, bulk ports, or a combination of SQL and bulk. Within the context of network tracing, SQL refers to SQL queries and responses and bulk refers to bulk loads, unloads, restores, and backups. Note that applying both results in two captures: one for bulk and one for SQL. Network tracing can be further modified by setting either the capture size or the duration of the trace. For appliances using Phonehome, enabling the Phonehome network tracing option uploads the captured files directly to the Phonehome servers while also continuing to store them locally.

For a list of all network trace commands, see config network.

Administer Cloud

Platforms: EE: All cloud platforms

Parent topic: Administer Yellowbrick

In this section:

This collection of how-to guides is designed to help you perform specific tasks related to installing, configuring and administering Enterprise Edition in public clouds.

Cloud: Configuration

Platforms: EE: All cloud platforms

Parent topic: Cloud

In this section:

After Yellowbrick has been installed, several platform-level configuration steps can be performed.

For example, if you chose during installation to use a custom object storage bucket, instructions on how to do so are here. If the instance is to be used in a production context or with sensitive data, we highly recommend you set up Single Sign-On as soon as possible.

Other common tasks, including setting up vanity DNS (to give your instance a nice URL) and SSL, are also detailed in this section.

Vanity DNS

Yellowbrick Documentation > How-to Guides > Administer Yellowbrick > Cloud > Configuring > Vanity DNS

Platforms: EE: All cloud platforms

Parent topic: Configuring

In this section:

- Configuring DNS Records for AWS
- Configuring DNS Records for Azure
- Configuring DNS Records for GCP

Yellowbrick does not manage DNS records on your behalf; after installation with the Deployer web application, you'll note that an IP address is used to access Yellowbrick Manager.

It's a best practice to configure DNS for your instance and this document explains how. You'll need to be familiar with basic DNS administration in order to do so.

It's likely that you are deploying Yellowbrick 7.x into a separate account/project/subscription from where your DNS zone is hosted. This setup is assuming you are using DNS delegation. That is, you host your root DNS zone (e.g. `mydomain.com`) in a separate account and have the child zone's (e.g. `subdomain.mydomain.com`) NS records configured into it. This child zone will be in the same resource group / project / account as your Yellowbrick install.

For example, let's say you have account **A** that hosts `mydomain.com`. Account **B**, the account that you deployed Yellowbrick into, will host your child DNS zone (`subdomain.mydomain.com`).

If you don't have a root zone or domain setup, you will need to set one up. We recommend setting up a separate, dedicated account/subscription/project for your root zone.

If you are hosting your DNS zone outside of a cloud platform, such as GoDaddy, you will need to add your child zone's NS records there instead.

Configuring DNS records for AWS

Platforms: EE: AWS cloud

Parent topic: Vanity DNS

Prerequisites

- The AWS CLI.
- Necessary AWS permissions for DNS administration.

You may also choose to use `kubectl` to find your Yellowbrick IP address in the example below.

Instructions

Step 1: Create a 'child' hosted zone for your Yellowbrick environment and retrieve the NS records. This will be the account of your Yellowbrick deployment.

```bash
export AWS_PAGER=""
export AWS_PROFILE=$YOURPROFILE
aws sts get-caller-identity

## Update childZoneDnsName to your environment
childZoneDnsName=""
    ## Example: childZoneDnsName="xyz.dev.yellowbrickcloud.com"

## Create the hosted zone
aws route53 create-hosted-zone --name $childZoneDnsName --caller-reference $(date +%Y%m%dT%H%M%SZ)
    ## Example:
        aws route53 create-hosted-zone --name xyz.dev.yellowbrickcloud.com --caller-reference $(date +%Y%m%dT%H%M%SZ)

## Save the NS records of the child zone for reference for step 2.
childZoneId=$(aws route53 list-hosted-zones --query "HostedZones[?Name == '${childZoneDnsName}.'].Id | [0]" --output text | cut -d

aws route53 list-resource-record-sets --hosted-zone-id $childZoneId --query "ResourceRecordSets[?Type == 'NS'].ResourceRecords[].V
```

Step 2: In the account of your root DNS zone, delegate and add the NS (name server) records of your child hosted zone to your domain. You will need to have access to the account of your root DNS zone.

```bash
## Turn off the cli pager and ensure you are pointing to the correct aws account (root dns zone account) if not already.
export AWS_PROFILE=$YOUR-ROOT-DNS-Zone-ACCOUNT
    ## Example: AWS_PROFILE=yb-dns
aws sts get-caller-identity

rootZoneDnsName="${YOUR-ROOT-DNS-ZONE-NAME}"
    ## Example: rootZoneDnsName=dev.yellowbrickcloud.com
childZoneDnsName="${YOUR-CHILD-DNS-ZONE-NAME}"
    ## Example: childZoneDnsName=xyz.dev.yellowbrickcloud.com

cat /tmp/nsrecords

vim /tmp/ns.json

## Update the following template to reflect your 4 ns record 'values'. Also change the ${childZoneDnsName} value. Ensure the "." i
```

```
{
    "Comment": "Add NS records",
    "Changes": [
        {
            "Action": "UPSERT",
            "ResourceRecordSet": {
                "Name": "${childZoneDnsName}.",
                "Type": "NS",
                "TTL": 60,
                "ResourceRecords": [
                    { "Value": "${nsRecord1}." },
                    { "Value": "${nsRecord2}." },
                    { "Value": "${nsRecord3}." },
                    { "Value": "${nsRecord4}." }
                ]
            }
        }
    ]
}

    ## Example:
    # {
    #     "Comment": "Add NS records",
    #     "Changes": [
    #         {
    #             "Action": "UPSERT",
    #             "ResourceRecordSet": {
    #                 "Name": "${childZoneDnsName}.",
    #                 "Type": "NS",
    #                 "TTL": 60,
    #                 "ResourceRecords": [
    #                     { "Value": "ns-852.awsdns-42.net." },
    #                     { "Value": "ns-174.awsdns-21.com." },
    #                     { "Value": "ns-1857.awsdns-40.co.uk." },
    #                     { "Value": "ns-1384.awsdns-45.org." }
    #                 ]
    #             }
    #         }
    #     ]
    # }

## Add the NS records of your child zone to your root zone
rootZoneId=$(aws route53 list-hosted-zones --query "HostedZones[?Name == '${rootZoneDnsName}.'].Id | [0]" --output text | cut -d'/

echo "${rootZoneId}"

aws route53 change-resource-record-sets --hosted-zone-id $rootZoneId --change-batch file:///tmp/ns.json
    ## Example output:
    # {
    #     "ChangeInfo": {
    #         "Id": "/change/C08335323T0L071YY3EW9",
    #         "Status": "PENDING",
    #         "SubmittedAt": "2024-09-05T00:39:24.469000+00:00",
    #         "Comment": "Add NS records"
    #     }
    #  }

## Confirm child zone ns records were added to root zone:
aws route53 list-resource-record-sets \
    --hosted-zone-id $rootZoneId \
    --query "ResourceRecordSets[?Name == '${childZoneDnsName}.' && Type == 'NS']" \
    --output json

    ## Example output:
    # [
    #     {
    #         "Name": "xyz.dev.yellowbrickcloud.com.",
    #         "Type": "NS",
```

```
    #         "TTL": 60,
    #         "ResourceRecords": [
    #             {
    #                 "Value": "ns-2029.awsdns-61.co.uk."
    #             },
    #             {
    #                 "Value": "ns-562.awsdns-06.net."
    #             },
    #             {
    #                 "Value": "ns-190.awsdns-23.com."
    #             },
    #             {
    #                 "Value": "ns-1322.awsdns-37.org."
    #             }
    #         ]
    #     }
    # ]
```

Step 3: Test DNS resolution of the child hosted zone with dig or nslookupv

bash

```
dig -t NS $childZoneDnsName
    ## EXAMPLE: dig -t NS mysubdomain.dev.mydomain.com
        ## You should see an ANSWER section
            # ;; ANSWER SECTION:
            # mysubdomain.dev.mydomain.com. 172800 IN NS   ns-1241.awsdns-27.org.
            # mysubdomain.dev.mydomain.com. 172800 IN NS   ns-1703.awsdns-20.co.uk.
            # mysubdomain.dev.mydomain.com. 172800 IN NS   ns-490.awsdns-61.com.
            # mysubdomain.dev.mydomain.com. 172800 IN NS   ns-527.awsdns-01.net.

nslookup -type=ns $childZoneDnsName
```

Step 4: In the account of your Yellowbrick deployment, add the Yellowbrick Manager IP address or DNS A record name to your hosted zone:

You will need to retrieve the Yellowbrick Manager IP address and update the 'ip' variable. This record can be anything you desire (manager, mgr, yb-manager, etc). We will be creating a simple A record pointing to a name you desire (Example: 142.250.65.110 referencing manager.yellowbrick.com).

bash

```
## Ensure aws cli is pointed to the correct aws account of the Yellowbrick install. The account where the childZone is located
export AWS_PROFILE=$YOURPROFILE

## Update the dnsName to reflect your Yellowbrick Manager IP
dnsName=""
    ## Example: dnsName=afc7bd334fd5d4c5cbd2e7b2533e0e4b-e1f916f82bad35e7.elb.us-east-1.amazonaws.com

    ## You can also retrieve the dns via kubectl:
        ## dnsName=$(kubectl -n $NAMESPACE get service yb-manager-service -o json | jq '.status.loadBalancer.ingress[].hostname' |

nlbName=$(echo "$dnsName" | awk -F'-' '{print $1}') && echo $nlbName

nlbZoneId=$(aws elbv2 describe-load-balancers --names $nlbName --query 'LoadBalancers[0].CanonicalHostedZoneId' | sed 's/"//g') &&

childZoneId=$(aws route53 list-hosted-zones --query "HostedZones[?Name == '${childZoneDnsName}.'].Id | [0]" --output text | cut -d

## Edit the following json and add the A record to your child zone. You will need to change 'recordNameYm', 'childZoneDnsName', 'n

recordNameYm=""
    ## Example: recordNameYm="manager"

cat >/tmp/ym.json <<EOF
{
    "Comment": "Add A record for Yellowbrick Manager",
    "Changes": [
        {
            "Action": "CREATE",
```

```
            "ResourceRecordSet": {
                "Name": "${recordNameYm}.${childZoneDnsName}.",
                "Type": "A",
                "AliasTarget": {
                    "HostedZoneId": "${nlbZoneId}",
                    "DNSName": "${dnsName}.",
                    "EvaluateTargetHealth": true
                }
            }
        }
    ]
}
EOF

cat /tmp/ym.json
    ## Example json: The HostedZoneId is not the zone id of your childZone. It's the zone id of the NLB.
        {
            "Comment": "Add A record for Yellowbrick Manager",
            "Changes": [
                {
                    "Action": "CREATE",
                    "ResourceRecordSet": {
                        "Name": "manager.xyz.dev.yellowbrickcloud.com.",
                        "Type": "A",
                        "AliasTarget": {
                            "HostedZoneId": "Z26RNL4JYFTOTI",
                            "DNSName": "a5b093499a6cf43f08c56afb7dbcaf78-62235f00c1f799a3.elb.us-east-1.amazonaws.com.",
                            "EvaluateTargetHealth": true
                        }
                    }
                }
            ]
        }
```

Step 5: Apply the record for Yellowbrick Manager

```bash
aws route53 change-resource-record-sets --hosted-zone-id $childZoneId --change-batch file:///tmp/ym.json
```

Step 6: Check DNS resolution for the Yellowbrick Manager

```bash
dig -t A "${recordNameYm}.${childZoneDnsName}"
    ## Example: dig -t A manager.xyz.dev.yellowbrickcloud.com
    ## Example output:
    #   ;; ANSWER SECTION:
    #   manager.xyz.dev.yellowbrickcloud.com. 55 IN A   **.**.54.6
```

Step 7: Add the Yellowbrick instance IP address(es) or DNS name of the NLB to your child hosted zone You will need to retrieve your IP address associated with the instance. This can be found in the Yellowbrick Manager under "instances." For each instance, you will need to enter a DNS record by performing this step.

```bash
## Ensure aws cli is pointed to the correct aws account of the Yellowbrick install. The account where the childZone is located
export AWS_PROFILE=$YOURPROFILE

## Retrieve the NLB A record of the instance from the Yellowbrick Manager under 'Instances' > 'Host/Port'. **IMPORTANT: Perform th
dnsName=""
    ## Example: dnsName=afc7bd334fd5d4c5cbd2e7b2533e0e4b-e1f916f82bad35e7.elb.us-east-1.amazonaws.com
    ## You can also retrieve the dns via kubectl:
        ## dnsName=$(kubectl -n $NAMESPACE get service ybinst-${instanceName} -o json | jq '.status.loadBalancer.ingress[].hostnam

nlbName=$(echo "$dnsName" | awk -F'-' '{print $1}') && echo $nlbName
nlbZoneId=$(aws elbv2 describe-load-balancers --names $nlbName --query 'LoadBalancers[0].CanonicalHostedZoneId' | sed 's/"//g') &&
childZoneId=$(aws route53 list-hosted-zones --query "HostedZones[?Name == '${childZoneDnsName}.'].Id | [0]" --output text | cut -d
```

```
## Create the json for Route53. 'recordNameDw', 'childZoneDnsName', 'nlbZoneId', and 'dnsName' parameters should all be set. The v

recordNameDw=""
    ## Example: recordNameDw="dw"

cat >/tmp/dw.json <<EOF
{
    "Comment": "Add A record for Yellowbrick Manager",
    "Changes": [
        {
            "Action": "CREATE",
            "ResourceRecordSet": {
                "Name": "${recordNameDw}.${childZoneDnsName}.",
                "Type": "A",
                "AliasTarget": {
                    "HostedZoneId": "${nlbZoneId}",
                    "DNSName": "${dnsName}.",
                    "EvaluateTargetHealth": true
                }
            }
        }
    ]
}
EOF

cat /tmp/dw.json

    ## Example output: All of the parameters should be set from previous commands. The HostedZoneId is not the zone id of your chi
    #   {
    #       "Comment": "Add A record for Yellowbrick Manager",
    #       "Changes": [
    #           {
    #               "Action": "CREATE",
    #               "ResourceRecordSet": {
    #                   "Name": "dw.xyz.dev.yellowbrickcloud.com.",
    #                   "Type": "A",
    #                   "AliasTarget": {
    #                       "HostedZoneId": "***TOTI",
    #                       "DNSName": "**********99a3.elb.us-east-1.amazonaws.com.",
    #                       "EvaluateTargetHealth": true
    #                   }
    #               }
    #           }
    #       ]
    #   }
```

Step 8: Apply the record for the instance

bash

```
aws route53 change-resource-record-sets --hosted-zone-id $childZoneId --change-batch file:///tmp/dw.json
```

Step 9: Check DNS resolution

bash

```
dig -t A ${recordNameDw}.${childZoneDnsName}
    ## Example: dig -t A manager.xyz.dev.yellowbrickcloud.com
```

Configuring DNS records for Azure

Yellowbrick Documentation > How-to Guides > Administer Yellowbrick > Cloud > Configuring > Vanity DNS > Configuring DNS Records for Azure

Platforms: EE: Azure cloud

Parent topic: Vanity DNS

Prerequisites

- Azure `az` CLI.
- Necessary Azure permissions for DNS administration.

You may also choose to use `kubectl` to find some information in the example below.

Instructions

Step 1: Create a 'child' hosted zone for your Yellowbrick environment and retrieve the NS records.

This will be in the account of your Yellowbrick deployment. In an Azure installation of Yellowbrick, currently there are two resource groups: The default resource group you've provided and a subset resource group of the AKS cluster resources.

The name of the latter resource group starts with `MC_`. For example, if your default resource group is `rg-yb-install`, then the secondary resource group would be `MC_rg-yb-install_yb-install_eastus`. For the creation of the `childZoneDnsName`, we will choose to place that into the primary resource group (in this case, `rg-yb-install`).

```bash
## Ensure your azure cli is set to the proper subscription of your "child" dns zone. This is the account where you have installed
az login
az account set --subscription $subscriptionId
    ## Example az account set --subscription mysubscription

childZoneDnsName=""
    ## Example: childZoneDnsName="xyz.dev.yellowbrickcloud.com"

## Create zone and retrieve NS records. Choose the resource group of the yellowbrick install
az network dns zone create -g $childZoneResourceGroup -n $childZoneDnsName
    ## Example: az network dns zone create -g rg-yb-install -n xyz.dev.yellowbrickcloud.com

nsRecords=($(az network dns record-set ns list --resource-group $childZoneResourceGroup --zone-name $childZoneDnsName --query "[].
```

Step 2: In the account of your root DNS zone, delegate and add the NS (name server) records of your child hosted zone to your domain. You will need to have access to the account of your root DNS zone

```bash
## Switch your azure cli to the proper subscription (if not in same subscription as child hosted zone) of your root dns zone
az account set --subscription $subscriptionId
az account show

rootZoneDnsName="${YOUR-ROOT-DNS-ZONE-NAME}"
    ## Example: rootZoneDnsName=dev.yellowbrickcloud.com
childZoneDnsName="${YOUR-CHILD-DNS-ZONE-NAME}"
    ## Example: childZoneDnsName=xyz.dev.yellowbrickcloud.com

## Verify your list of records still exists in the variable
echo "${nsRecords[@]}"
```

```
## Add the 4 ns records of the child zone to your root zone. rootZoneResourceGroup is the resource group of your root hosted zone.
for nsRecord in "${nsRecords[@]}"; do
    az network dns record-set ns add-record \
        --resource-group $rootZoneResourceGroup \
        --zone-name $rootZoneDnsName \
        --record-set-name $childZoneDnsName \
        --nsdname $nsRecord \
        --ttl=3600
done
```

Step 3: Test DNS resolution of the child hosted zone with dig or nslookup

bash

```
dig -t NS $childZoneDnsName
    ## EXAMPLE: dig -t NS mysubdomain.dev.mydomain.com
        ## You should see an ANSWER section
            # ;; ANSWER SECTION:
            # mysubdomain.dev.mydomain.com. 172800 IN NS   ns-1241.awsdns-27.org.
            # mysubdomain.dev.mydomain.com. 172800 IN NS   ns-1703.awsdns-20.co.uk.
            # mysubdomain.dev.mydomain.com. 172800 IN NS   ns-490.awsdns-61.com.
            # mysubdomain.dev.mydomain.com. 172800 IN NS   ns-527.awsdns-01.net.

## Or you can use nslookup
nslookup -type=ns $childZoneDnsName
```

Step 4: In the account of your Yellowbrick deployment, add the Yellowbrick Manager IP address or DNS A record name to your hosted zone. You will need to retrieve the Yellowbrick Manager IP address and update the 'ip' variable. This record can be anything you desire (manager, mgr, yb-manager, etc). We will be creating a simple A record pointing to a name you desire (Example: 142.250.65.110 referencing manager.yellowbrick.com).

bash

```
## Ensure your azure cli is pointed to the proper subscription (child zone)
az account show

rootZoneDnsName="${YOUR-ROOT-DNS-ZONE-NAME}"
    ## Example: rootZoneDnsName=dev.yellowbrickcloud.com
childZoneDnsName="${YOUR-CHILD-DNS-ZONE-NAME}"
    ## Example: childZoneDnsName=xyz.dev.yellowbrickcloud.com

## Update the ip to reflect your Yellowbrick Manager IP
ip=""
    ## Or you can retrieve the same result with kubectl:
    ## ip=$(kubectl -n $NAMESPACE get service yb-manager-service -o json | jq '.status.loadBalancer.ingress[].ip' | sed 's/"//g')

## Add the Yellowbrick Manager ip to your child zone. You can change the 'recordNameYm' to a name you desire.
recordNameYm="manager"
az network dns record-set a add-record \
    --resource-group $childZoneResourceGroup \
    --zone-name $childZoneDnsName \
    --record-set-name "${recordNameYm}.${childZoneDnsName}" \
    --ipv4-address $ip

    ## Example output:
    #   [
    #       {
    #           "ARecords": [
    #               {
    #                   "ipv4Address": "111.222.41.113"
    #               }
    #           ],
    #           "TTL": 3600,
    #           "etag": "*************23",
    #           "fqdn": "xyz.dev.yellowbrickcloud.com.xyz.dev.yellowbrickcloud.com.",
    #           "id": "/subscriptions/************/resourceGroups/************/providers/Microsoft.Network/dnszones/xyz.dev.yellow
    #           "name": "manager.xyz.dev.yellowbrickcloud.com",
```

```
    #           "provisioningState": "Succeeded",
    #           "resourceGroup": "$YOUR_RESOURCE_GROUP",
    #           "targetResource": {},
    #           "type": "Microsoft.Network/dnszones/A"
    #       }
    #   ]
```

Step 5: Add the Yellowbrick instance IP address(es) or DNS name of the NLB to your child hosted zone. You will need to retrieve the IP address associated with the instance. This can be found in the Yellowbrick Manager under "instances." For each instance, you will need to enter a DNS record by performing this step.

bash

```
## Ensure your azure cli is pointed to the proper subscription
az account show

rootZoneDnsName="${YOUR-ROOT-DNS-ZONE-NAME}"
    ## Example: rootZoneDnsName=dev.yellowbrickcloud.com
childZoneDnsName="${YOUR-CHILD-DNS-ZONE-NAME}"
    ## Example: childZoneDnsName=xyz.dev.yellowbrickcloud.com

## Retrieve the NLB A record of the data warehouse instance from the Yellowbrick Manager under 'Instances' > 'Host/Port'.
dwInstanceIp=""
    ## Or you can retrieve the same result with kubectl:
    ## dwInstanceIp=$(kubectl -n $NAMESPACE get service ybinst-${instanceName} -o json | jq '.status.loadBalancer.ingress[].ip' |

## Add the data warehouse ip to your child zone. Change 'recordNameDw' to your desired name. For example, if your data warehouse n
recordNameDw="dw"
az network dns record-set a add-record \
    --resource-group $childZoneResourceGroup \
    --zone-name $childZoneDnsName \
    --record-set-name "${recordNameDw}.$childZoneDnsName" \
    --ipv4-address $dwInstanceIp

    ## Example output:
    #   [
    #       {
    #           "ARecords": [
    #               {
    #                   "ipv4Address": "***.***.***.124"
    #               }
    #           ],
    #           "TTL": 3600,
    #           "etag": "******79",
    #           "fqdn": "dw.xyz.dev.yellowbrickcloud.com.xyz.dev.yellowbrickcloud.com.",
    #           "id": "/subscriptions/**********/resourceGroups/**********/providers/Microsoft.Network/dnszones/xyz.dev.yellowbr
    #           "name": "dw.xyz.dev.yellowbrickcloud.com",
    #           "provisioningState": "Succeeded",
    #           "resourceGroup": "*************",
    #           "targetResource": {},
    #           "type": "Microsoft.Network/dnszones/A"
    #       }
    #   ]
```

Step 6: Check DNS resolution

bash

```
dig -t A ${recordNameDw}.${childZoneDnsName}
    ## Example: dig -t A dw.xyz.dev.yellowbrickcloud.com
```

Configuring DNS records for GCP

Platforms: EE: GCP cloud

Parent topic: Vanity DNS

Prerequisites

- GCP `gcloud` CLI
- Necessary GCP permissions for DNS administration.

You may also choose to use `kubectl` to find some information in the example below.

Instructions

Step 1: Create a 'child' hosted zone for your Yellowbrick environment and retrieve the NS records. This will be the account of your Yellowbrick deployment

```bash
gcloud auth login
gcloud config set project $projectId
    ## Example: gcloud config set project myProject

gcloud config get-value project

childZoneName="${YOUR-CHILD-ZONE-NAME}"
childZoneDnsName="${YOUR-CHILD-ZONE-DNS-NAME}"
    ## Examples:
    # childZoneName="xyz-dev-yellowbrickcloud-com"
    # childZoneDnsName="xyz.dev.yellowbrickcloud.com"

gcloud dns managed-zones create $childZoneName \
    --dns-name=$childZoneDnsName \
    --description=$YOUR-DESCRIPTION \
    --visibility=$visibility

        ## $visibility can be 'public' or 'private,' depending on type of yellowbrick install

        ## Example:
        # gcloud dns managed-zones create xyz-dev-yellowbrickcloud-com \
        #     --dns-name="xyz.dev.yellowbrickcloud.com" \
        #     --description="YB DNS Zone" \
        #     --visibility=public

## Retrieve the ns records for the next step
nsRecords=($(gcloud dns record-sets list --zone=$childZoneName --name=$childZoneDnsName --type=NS --format=json | jq -r '.[].rrdat
```

Step 2: In the account of your root DNS zone, delegate and add the NS (name server) records of your child hosted zone to your domain. You will need to have access to the account of your root DNS zone.

```bash
## Switch your gcloud cli to the proper gcp project of your root dns zone
gcloud config set project $projectId
    ## Example: gcloudconfig set project myRootDnsZoneProject
gcloud config get-value project
```

```
rootZoneDnsName="${YOUR-ROOT-DNS-ZONE-NAME}"
    ## Example: rootZoneDnsName=dev.yellowbrickcloud.com
rootZoneName="${YOUR-ROOT-DNS-NAME}"
    ## Example: rootZoneName=dev-yellowbrickcloud-com

## Prepare the transaction.yaml and add the NS records of your child zone to your root zone for delegation
gcloud dns record-sets transaction start --zone=$rootZoneName
    ## Example gcloud dns record-sets transaction start --zone=dev-yellowbrickcloud-com

echo "${nsRecords[@]}"
    ## Example output:
    # ns-cloud-b1.googledomains.com. ns-cloud-b2.googledomains.com. ns-cloud-b3.googledomains.com. ns-cloud-b4.googledomains.com.

## Change the ns records to reflect your output of the previous command. This command simply updates the transaction.yaml. Ensure
gcloud dns record-sets transaction add ${nsRecord1}. ${nsRecord2}. ${nsRecord3}. ${nsRecord4}. \
    --name="${childZoneDnsName}" \
    --ttl=3600 \
    --type=NS \
    --zone="${rootZoneDnsName}"

    ## Example:
    # gcloud dns record-sets transaction add ns-cloud-b1.googledomains.com. ns-cloud-b2.googledomains.com. ns-cloud-b3.googledomai
    #     --name=xyz.dev.yellowbrickcloud.com \
    #     --ttl=3600 \
    #     --type=NS \
    #     --zone=dev.yellowbrickcloud.com

## Delegate the zone
gcloud dns record-sets transaction execute --zone=$rootZoneName
    ## Output: "Executed transaction [transaction.yaml] for managed-zone [dev-yellowbrickcloud-com]."

## Check existence of subdomain's ns records
gcloud dns record-sets list --zone=$rootZoneName --name=$childZoneDnsName --type=NS --format=json

    ## You should see your child zone's NS records in your root zone. Example output:
    #   [
    #       {
    #           "kind": "dns#resourceRecordSet",
    #           "name": "xyz.dev.yellowbrickcloud.com.",
    #           "rrdatas": [
    #           "ns-cloud-b1.googledomains.com.",
    #           "ns-cloud-b2.googledomains.com.",
    #           "ns-cloud-b3.googledomains.com.",
    #           "ns-cloud-b4.googledomains.com."
    #           ],
    #           "ttl": 3600,
    #           "type": "NS"
    #       }
        #   ]

    ## You can also view this in the GCP console: GCP > {your-project} > Network Services > Cloud DNS > {your-root-zone}. Check fo

rm transaction.yaml
```

Step 3: Test DNS resolution of the child hosted zone with dig or nslookup.

bash

```
dig -t NS $childZoneDnsName
    ## EXAMPLE: dig -t NS mysubdomain.dev.mydomain.com
    ## You should see an ANSWER section
        # ;; ANSWER SECTION:
        # mysubdomain.dev.mydomain.com. 172800 IN NS   ns-1241.awsdns-27.org.
        # mysubdomain.dev.mydomain.com. 172800 IN NS   ns-1703.awsdns-20.co.uk.
        # mysubdomain.dev.mydomain.com. 172800 IN NS   ns-490.awsdns-61.com.
        # mysubdomain.dev.mydomain.com. 172800 IN NS   ns-527.awsdns-01.net.
```

```
## Or you can use nslookup
nslookup -type=ns $childZoneDnsName
```

Step 4: In the account of your Yellowbrick deployment, add the Yellowbrick Manager IP address or DNS A record name to your hosted zone. You will need to retrieve the Yellowbrick Manager IP address and update the 'ip' variable. This record can be anything you desire (manager, mgr, yb-manager, etc). We will be creating a simple A record pointing to a name you desire (Example: 142.250.65.110 referencing manager.yellowbrick.com).

bash

```
## Switch your gcloud cli to the proper gcp project of your child dns zone
gcloud config set project $projectId
    ## Example: gcloudconfig set project myChildDnsZoneProject
gcloud config get-value project

childZoneName="${YOUR-CHILD-ZONE-NAME}"
childZoneDnsName="${YOUR-CHILD-ZONE-DNS-NAME}"
    ## Examples:
        childZoneName="xyz-dev-yellowbrickcloud-com"
        childZoneDnsName="xyz.dev.yellowbrickcloud.com"

## Update the ip to reflect your Yellowbrick Manager IP
ip=""
    ## Or you can retrieve the same result with kubectl:
    ## ip=$(kubectl -n $NAMESPACE get service yb-manager-service -o json | jq '.status.loadBalancer.ingress[].ip' | sed 's/"//g')

## Prepare the transaction.yaml and add the A records of the Yellowbrick Manager ip to your child zone
gcloud dns record-sets transaction start --zone=$childZoneName

## Change the 'recordNameYm' to a name you desire. For example if your data warehouse name is 'dw', then you would use dw.
recordNameYm="manager"
gcloud dns record-sets transaction add $ip \
    --name="${recordNameYm}.${childZoneDnsName}." \
    --ttl=300 \
    --type=A \
    --zone=$childZoneName

    ## Example:
    #   gcloud dns record-sets transaction add $ip \
    #       --name="manager.xyz.dev.yellowbrickcloud.com." \
    #       --ttl=300 \
    #       --type=A \
    #       --zone=$childZoneName

gcloud dns record-sets transaction execute --zone=$childZoneName

## Check resolution
dig -t A ${recordName}.${childZoneDnsName}
    ## Example: dig -t A manager.xyz.dev.yellowbrickcloud.com
```

Step 5: Add the Yellowbrick instance IP address(es) or DNS name of the NLB to your child hosted zone. You will need to retrieve your IP address associated with the instance. This can be found in the Yellowbrick Manager under "instances." For each data warehouse instance, you will need to enter a DNS record by performing this step.

bash

```
## Retrieve the NLB A record of the data warehouse instance from the Yellowbrick Manager under 'Instances' > 'Host/Port'.
dwInstanceIp=""
    ## Example: dwInstanceIp=123.456.789.214

    ## Or you can retrieve the same result with kubectl:
    ## dwInstanceIp=$(kubectl -n $NAMESPACE ybinst-${instanceName} -o json | jq '.status.loadBalancer.ingress[].ip' | sed 's/"//g'

## Prepare the transaction.yaml and add the A records of the NLB to your child zone
gcloud dns record-sets transaction start --zone=$childZoneName

## Change the recordName to something unique for your data warehouse instance. It can be the data warehouse name you set in the Ye
recordNameDw="dw"
gcloud dns record-sets transaction add $dwInstanceIp \
```

```
    --name="${recordNameDw}.${childZoneDnsName}." \
    --ttl=300 \
    --type=A \
    --zone=$childZoneName

    ## Example with 'dw' as the data warehouse name:
    #   gcloud dns record-sets transaction add $dwInstanceIp \
    #       --name="dw.xyz.dev.yellowbrickcloud.com." \
    #       --ttl=300 \
    #       --type=A \
    #       --zone=$childZoneName

gcloud dns record-sets transaction execute --zone=$childZoneName
```

Step 6: Check DNS resolution

```bash
dig -t A ${recordNameDw}.${childZoneDnsName}
    ## Example: dig -t A dw.xyz.dev.yellowbrickcloud.com
```

Yellowbrick Manager

Platforms: EE: All cloud platforms

Parent topic: Configuring

In this section:

Authentication/Authorization Models

There are 3 types of a authentication/authorization models supported by Yellowbrick Manager:

1. Instance Login
2. Single Sign-On
3. Infrastructure Administrator

Determining the Hostname or IP Address of Yellowbrick Manager

During installation, the initial URL of the Yellowbrick Manager will be displayed for initial management. Afterwards, if the IP address or Hostname is not known, you may use Kubernetes to determine its value by following these steps:

Determining hostname or IP Address of Yellowbrick Manager with Kubernetes

To determine the hostname or IP Address using Kubernetes, first you must obtain a valid Kubernetes configuration to enable `kubectl` usage.

Using `kubectl`:

```bash
% kubectl -n ${namespace} get service yb-manager-service
NAME                TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
yb-manager-service   ClusterIP   100.79.201.16   34.99.251.19  80/TCP    1d
```

The value of `EXTERNAL-IP` in this case would be used to access Yellowbrick Manager.

To determine the hostname or IP Address of Yellowbrick Manager with cloud CLI or Console

Use one of the following documents to discover the load balancer hostname or IP Address, in each of the following cloud products:

- AWS
- Azure
- GCP

Configuring "Ask TK" in Yellowbrick Manager

This guide describes the steps needed to configure the API endpoints that the *Ask TK* function in Yellowbrick Manager is allowed to connect to. Only the consumeradmin role can change these endpoints.

To add or remove access to external large language model services in Yellowbrick Manager, navigate to *Configuration → Yellowbrick Manager → Ask TK Access*. Allowed third-party endpoints are specified as a comma-separated list, for example:

```
api.openai.com, api.deepseek.com
```

Note that the entries must be the fully-qualified domain name (FQDN) or IP address of the API endpoint. Specifying the domain alone, for example `openai.com`, will not work. The endpoints must also be compatible with the OpenAI API.

Also note that the FQDN must be accessible from the Yellowbrick Manager server. For example, if you want to connect Yellowbrick Manager to an instance of Ollama running on your local machine, you must ensure that the FQDN or IP address can be reached from Yellowbrick Manager.

Changes must be saved in order to have an effect. Be aware that any changes will require a Yellowbrick Manager restart. This will cause any connections from other users to be terminated. The reason for this is that the new endpoints are stored in a config map in the Kubernetes cluster and a change here will only be picked up during initialization of the Yellowbrick Manager pod.

Infrastructure Administrator

Platforms: EE: All cloud platforms

Parent topic: Yellowbrick Manager

The infrastructure administrator account is not enabled by default after initial installation. You may enable the infrastructure administrator account for cross-instance management, or to manage your instances and their lifecycle from Yellowbrick Manager when they are suspended and cannot be used to login using instance login.

Enabling Infrastructure Administrator

You may enable Infrastructure Administrator from Yellowbrick Manager, by navigating to *Configuration → Yellowbrick Manager* and configuring a password and account name, which defaults to `infraadmin`.

Instance Authorization

The infrastructure administrator user and account will be automatically configured to login to each instance that has been deployed. Its initial authorization is established by the same `consumer` role which is given to all users, as well as `consumeradmin` which allows for the management of instance lifecycle (suspend, resume, upgrade, create, destroy, etc.)

To set an infrastructure administrator account and password, login as a user to an instance with `SYSADMIN` privileges. Once established, changing the password will require the current password to be known, so take care to store and protect its credentials.

CAUTION

The infrastructure administrator configuration is stored in a Kubernetes secret called `yb-manager-secret` in the namespace where Yellowbrick Manager is deployed. The password is not stored directly; a hash of this password is stored and used for authentication challenges. Removing the secret's value for this hashed password will reset the feature to its initial value.

As is the case for all Kubernetes resources, take care to protect this secret's configuration from misuse.

Yellowbrick Manager Instance Login

Platforms: EE: All cloud platforms

Parent topic: Yellowbrick Manager

By default, Yellowbrick Manager authenticates with a single instance deployed during installation. This instance is an enterprise-class data warehouse with its own database catalog for users and roles. During installation, it is configured with an initial administrator password of your choice. Logging into Yellowbrick Manager with this username and password authenticates against this embedded database.

The login screen of Yellowbrick Manager, when configured with a single instance, will display a single username and password entry form. The options available on this login form will change when other login options are configured, or if secondary Yellowbrick instances are created.

Default Roles

There are two roles used by Yellowbrick Manager to control access to database resources and instance lifecycle. By default, every user is given the `consumer` role when logging into Yellowbrick Manager. In addition, if the user is granted `SYSADMIN` role in the database, users will be given `consumeradmin` role by Yellowbrick Manager, which controls instance lifecycle operations such as instance suspend, resume, create, delete, upgrade and diagnostics.

Additional Instances

Additional instances may be created using Yellowbrick Manager which will result in the login form displaying a selection on which instance to authenticate with. When only one instance is configured, this selection will not be displayed.

Multiple Instance Management

Yellowbrick Manager will allow multiple instances to be managed with a single login when using either the Single Sign-On option or Infrastructure Administrator options.

Resuming an Instance

The instance login feature allows for the resume of an instance if it is currently suspended, in order to facilitate the login authentication challenge to access the instance. If the instance you are attempting to login to is suspended, Yellowbrick Manager will offer to resume it for you.

Yellowbrick Manager on AWS

To find the hostname (DNS name) or IP address associated with the load balancer for **Yellowbrick Manager** in AWS, you can use either the AWS CLI or AWS Management Console. This guide will help you quickly locate this information based on specific tags associated with your load balancer.

Prerequisites

- Ensure you have the AWS CLI installed and configured with appropriate permissions.
- Access to the AWS Management Console.

Option 1: Using AWS CLI

Follow these steps to retrieve the DNS name or IP address using the AWS CLI.

1. **Filter Load Balancers by Tags**

   Use the following command to filter load balancers based on tags associated with Yellowbrick Manager. Replace `REGION` with your AWS region.

   ```bash
   aws elbv2 describe-tags --region REGION --resource-arns $(aws elbv2 describe-load-balancers --region REGION --query 'LoadBalanc
   ```

This command will return a list of load balancers that match any of the specified tags.

2. **Retrieve the DNS Name**

   Once you have identified the ARN or name of the load balancer, you can get its DNS name by using the command below:

   ```bash
   aws elbv2 describe-load-balancers --region REGION --load-balancer-arns LOAD_BALANCER_ARN
   ```

   The output will include a DNSName attribute, which provides the load balancer's hostname.

   If you need an IP address, you can resolve the DNS name using a tool like nslookup:

   ```bash
   nslookup LOAD_BALANCER_DNS_NAME
   ```

Option 2: Using AWS Management Console

Alternatively, you can use the AWS Management Console to find the hostname or IP address.

1. **Access Load Balancers**

Log in to the AWS Management Console and navigate to the EC2 Dashboard. From the left menu, select Load Balancers under the Load Balancing section.

2. **Filter Load Balancers by Tags**

In the Load Balancers list, use the search box to filter based on tag keys or values. You can use any of the following tags associated with Yellowbrick Manager:

Tag	Value
cluster_yellowbrick_io_owner	`yb-install`
kubernetes_io_service_name	`${instanceName}/yb-manager-service`

3. Identify the Hostname or IP Address

Once you've found the correct load balancer, click on it to view its details. In the Description tab, locate the DNS name field to find the load balancer's hostname.

Yellowbrick Manager on Azure

Platforms: EE: All cloud platforms

Parent topic: Yellowbrick Manager

To find the hostname (DNS name) or IP address associated with the load balancer for **Yellowbrick Manager** in Azure, you can use either the Azure CLI or Azure Portal. This guide will help you quickly locate this information based on specific tags associated with your load balancer.

Prerequisites

- Ensure you have the Azure CLI installed and configured with appropriate permissions.
- Access to the Portal.

Option 1: Using Azure CLI

Follow these steps to retrieve the DNS name or IP address using the Azure CLI.

1. **Filter Load Balancers by Tags**

   Use the following command to filter load balancers based on tags associated with Yellowbrick Manager. Replace RESOURCE_GROUP with your Azure resource group.

```bash
az network lb list --resource-group RESOURCE_GROUP --query "[?tags.kubernetes_io_service_name && contains(tags.kubernetes_io_se
```

This command will return a list of load balancers that match any of the specified tags.

2. **Retrieve the DNS Name**

   Once you have identified the load balancer, you can get its frontend IP address configuration details, which includes the DNS name or IP address:

```bash
az network lb show --resource-group RESOURCE_GROUP --name LOAD_BALANCER_NAME --query "frontendIPConfigurations[].dnsSettings.fq
```

   If you need an IP address, modify the query to:

```bash
az network lb show --resource-group RESOURCE_GROUP --name LOAD_BALANCER_NAME --query "frontendIPConfigurations[].privateIPAddre
```

Option 2: Using Azure Portal

Alternatively, you can use the Azure Management Console to find the hostname or IP address.

1. **Access Load Balancers**

Log in to the Azure Portal and navigate to "Load Balancers" under the "Networking" section.

2. **Filter Load Balancers by Tags**

In the Load Balancers list, use the search box or "Tags" filter to find load balancers associated with Yellowbrick Manager. Use any of the following tags:

Tag	Value
cluster_yellowbrick_io_owner	`yb-install`
kubernetes_io_service_name	`${instanceName}/yb-manager-service`

3. Identify the Hostname or IP Address

Once you've found the correct load balancer, click on it to view its details. Under "Settings," navigate to "Frontend IP configurations" to locate the DNS name or IP address.

Yellowbrick Manager on GCP

Platforms: EE: All cloud platforms

Parent topic: Yellowbrick Manager

To find the hostname (DNS name) or IP address associated with the load balancer for **Yellowbrick Manager** on GCP, you can use either the `gcloud` CLI or the Google Cloud Console. This guide will help you quickly locate this information based on specific tags associated with your load balancer.

Prerequisites

- Ensure you have the `gcloud` CLI installed and configured with appropriate permissions.
- Access to the Portal.

Option 1: Using `gcloud` CLI

Follow these steps to retrieve the DNS name or IP address using the `gcloud` CLI.

1. **Filter Load Balancers by Labels**

   Use the following command to filter load balancers based on labels associated with Yellowbrick Manager. Replace `PROJECT_ID` with your GCP project ID.

   ```bash
   gcloud compute forwarding-rules list --project=PROJECT_ID --filter="labels.kubernetes_io_service_name:*yb-manager*"
   ```

This command will return a list of forwarding rules (equivalent to load balancers in GCP) that match any of the specified labels.

2. **Retrieve the DNS Name**

   Once you have identified the forwarding rule, you can get its IP address:

   ```bash
   gcloud compute forwarding-rules describe FORWARDING_RULE_NAME --region=REGION --project=PROJECT_ID --format="get(IPAddress)"
   ```

   Replace `FORWARDING_RULE_NAME` with the name of your forwarding rule, and `REGION` with the appropriate region.

   If you need the DNS name, you can perform a reverse lookup on the IP address using a tool like nslookup or dig.

Option 2: Using Google Cloud Console

Alternatively, you can use the Using Google Cloud Console to find the hostname or IP address.

1. **Access Load Balancers**

Log in to the Google Cloud Console and navigate to "Network services" > "Load balancing."

2. **Filter Load Balancers by Tags**

In the load balancer list, use the search box or filter options to find forwarding rules associated with Yellowbrick Manager. Use any of the following labels:

Label	Value
cluster_yellowbrick_io_owner	`yb-install`
kubernetes_io_service_name	`${instanceName}/yb-manager-service`

3. **Identify the Hostname or IP Address**

Once you've found the correct forwarding rule, click on it to view its details. The IP address will be displayed under the "Frontend" section.

Alerting

Yellowbrick Documentation > How-to Guides > Administer Yellowbrick > Cloud > Configuring > Alerting

Platforms: EE: All cloud platforms

Parent topic: Configuring

Yellowbrick supports integration with Slack and Opsgenie for alerting. When unexpected issues are detected, an alert will be sent to one or both of these channels.

Configuration of alerting endpoints is accomplished through usage of `kubectl` commands. In a subsequent release a friendly user interface will be added to Yellowbrick Manager.

Yellowbrick currently alerts on the following unexpected exceptional conditions:

- Disc space low or exhausted
- Unexpected crashes or process exits
- Issues with background tasks
- File system consistency issues
- Quota exhaustion
- Row store volume exhaustion

For a detailed list of the alerts Yellowbrick currently includes see Observability Alerts.

The workload manager is also capable of generating rule-based custom alerts in the event of conditions such as queries running too long, users hogging resources or similar. Such alerts will also be dispatched through this mechanism. See the workload manager rule actions for more information.

Step 1: Collect Integration Information

Alerts can be sent to Slack, Opsgenie, or both tools. To configure alerting for Slack, you need to know the URL and channel name. For Opsgenie, you need an API key and, optionally, an API URL.

To find your slack URL, follow the instructions here. Make sure that a target Slack channel (beginning with a `#`) has been created in advance. To create an Opsgenie API key, and work out which URL is pertinent, follow the instructions here.

Step 2: Create a JSON Configuration File

To configure alerting, either the slack configuration, the Opsgenie configuration or both must be specified in a JSON document and uploaded to a Kubernetes secret. To do so, create a document called alert.json as follows:

```bash
echo '{
  "slackChannel": <slackChannelName>,
  "slackUrl": <slackURL>,
  "opsGenieKey": <opsGenieKey>
  "opsGenieUrl": <opsGenieURL>
}' > alert.json
```

For Slack configuration, both `slackChannel` and `slackURL` must be specified. The channel must be prefixed by a `#` character and must be created in advance.

The `opsGenieUrl` is an optional parameter, defaulting to the global `https://api.opsgenie.com` if omitted.

A fully formed example JSON file might look something like:

```bash
echo '{
  "slackChannel": "#alerts",
  "slackUrl": "https://hooks.slack.com/services/T00000000/B00000000/XXXXXXXXXXXXXXXXXXXXXXXX",
  "opsGenieKey": "12345-abcde-67890-fghij-12345"
}' > alert.json
```

Step 3: Install the JSON Configuration File

The JSON document must be installed into a Kubernetes secret called `yb-monitoring-secret`. To do so, use the following `kubectl` command:

```bash
kubectl create secret generic yb-monitoring-secret --from-file=state=alert.json -n monitoring
```

The JSON document must be well formed, and the endpoints specified correctly, with egress permitted if running in a fully private configuration.

Step 4: Generate a Test Alert

Alerts are based on Prometheus expressions or direct calls to Alertmanager. Here we will show an example of how to generate an alert using both methods.

Direct Call to Alertmanager

The following example sets up port-forwarding to the Kubernetes Alertmanager service and then uses `curl` to send an alert that will expire after 5 minutes. It can take a minute before you will see the alert displayed in Opsgenie or Slack but it will be immediately visible when querying the Alertmanager API directly. If you do not want an automatic resolve message to be sent to Opsgenie and/or Slack after 5 minutes have elapsed set `send_resolved` to `false`.

```bash
# In one terminal setup execute the following to setup port-forwarding
kubectl port-forward -n monitoring svc/loki-prometheus-alertmanager 9093:9093

# In a second terminal POST the alert
curl -X POST http://localhost:9093/api/v2/alerts \
  -H "Content-Type: application/json" \
  -d "[
    {
      \"labels\": {
        \"alert_name\": \"Yellowbrick Test Alert\",
        \"alert_type\": \"observability\",
        \"send_resolved\": \"true\",
        \"severity\": \"CRITICAL\",
        \"namespace\": \"test_namespace\"
      },
      \"annotations\": {
        \"summary\": \"This is a summary\",
        \"description\": \"This is a description\"
      },
      \"startsAt\": \"$(date -u +"%Y-%m-%dT%H:%M:%SZ")\",
      \"endsAt\": \"$(date -u -d '+5 minutes' +"%Y-%m-%dT%H:%M:%SZ")\"
    }
  ]"

# Optionally query the Alertmanager API directly to see the alert.
curl http://localhost:9093/api/v2/alerts

# Which will produce
[
  {
    "annotations": {
      "description": "This is a description",
      "summary": "This is a summary"
    },
    "endsAt": "...",
    "fingerprint": "...",
    "receivers": [
      {
```

```
        "name": "team-pager"
      },
      {
        "name": "yb-observability-slack-receiver-nosendresolved"
      }
    ],
    "startsAt": "...",
    "status": {
      "inhibitedBy": [],
      "silencedBy": [],
      "state": "active"
    },
    "updatedAt": "...",
    "labels": {
      "alert_name": "Yellowbrick Test Alert",
      "alert_type": "observability",
      "namespace": "test_namespace",
      "send_resolved": "false",
      "severity": "CRITICAL"
    }
  },
  ...
```

If you query the Alertmanager API directly you will also see the `Watchdog` alert from the Always Firing Alerts.

Prometheus-Based Alert

Prometheus-based alerts are specified in config maps stored in the `monitoring` namespace. In order for the alert to be picked up by Prometheus (and sent to Alertmanager) it must have the label `alert-rule` set to `true`. The yaml file defined in the config map will be mounted in the Prometheus server pod where it will be read by Prometheus automatically. The syntax and configuration for the alert follow standard Prometheus alerting rules.

The following is a trivial alert that will fire when any instance has been up for less than 600 seconds (10 minutes); suspending and resuming an instance will reset the `yb_system_uptime_seconds` measurement for that instance. Like the previous example, the alert will be shown in Opsgenie and/or Slack if they are configured, but can also be seen by querying the Alertmanager API directly. See Observability Metrics for other exposed metrics to base alerts on.

bash

```
cat <<'EOF' > alert-600-second-uptime.yaml
apiVersion: v1
data:
  alert-system-up-less-than-600-seconds.yaml: |
    groups:
      - name: example-alerts
        rules:
          - alert: "System Up Less Than 600 seconds"
            expr: yb_system_uptime_seconds < 600
            labels:
              severity: CRITICAL
              alert_type: observability
              send_resolved: true
              namespace: '{{ $labels.namespace }}'
              yellowbrick_io_instance_id: '{{ $labels.yellowbrick_io_instance_id }}'
              yellowbrick_io_instance_name: '{{ $labels.yellowbrick_io_instance_name }}'
            annotations:
              summary: "The system has been up less than 600 seconds"
              description: "The system has only been up {{ $value }} seconds."

kind: ConfigMap
metadata:
  name: alert-system-up-less-than-600-seconds
  namespace: monitoring
  labels:
    alert-rule: "true"
EOF
```

```
kubectl -n monitoring apply -f ./alert-600-second-uptime.yaml
```

To remove the alert delete the config map:

```bash
kubectl delete -n monitoring configmap alert-system-up-less-than-600-seconds
```

Diagnosing Problems

In the case of a malformed JSON document, or missing or malformed keys in the document, errors will be posted to the Yellowbrick Operator logs. To inspect the Operator logs, use the following command:

```bash
kubectl logs -l app=yb-operator -n <operator_namespace> -f | grep yb-monitoring-secret
```

An example of an error due to malformed JSON looks something like this:

```txt
2024-01-02T03:04:05Z    ERROR  Secret.Monitoring       Invalid alerting configuration: unable to deserialize the configuration, i
```

In the case of issues sending an alert, errors will be posted to the Alertmanager logs. To inspect the Alertmanager logs, first find the pod name and then retrieve the logs as follows:

```bash
kubectl get pod -l component=alertmanager -n monitoring
kubectl logs <pod_name_from_above_command> -n monitoring -f prometheus-alertmanager
```

An example of errors sending alerts to Slack and Opsgenie respectively look something like this:

```txt
ts=2024-08-22T03:43:18.613Z caller=dispatch.go:353 level=error component=dispatcher msg="Notify for alerts failed" num_alerts=1 er
ts=2024-08-22T03:43:18.692Z caller=dispatch.go:353 level=error component=dispatcher msg="Notify for alerts failed" num_alerts=1 er
```

Disabling Alerting

To completely disable alerting, you can just delete the secret. To do so, use the following command:

```bash
kubectl delete secret yb-monitoring-secret -n monitoring
```

Setting Up AWS Transit Gateway for Yellowbrick Access

Platforms: EE: All cloud platforms

Parent topic: Configuring

Introduction

AWS Transit Gateway (TGW) is a highly scalable networking service that enables communication between multiple Amazon Virtual Private Clouds (VPCs). It simplifies connectivity and routing management across different VPCs, eliminating the need for complex peering configurations. In this guide, we will set up AWS Transit Gateway to enable Tableau access to a Yellowbrick instance hosted in an Amazon Elastic Kubernetes Service (EKS) cluster.

NOTE: This guide uses Tableau as an example service in the Hub VPC. This process, however, is similar regardless of service.

Why Use Transit Gateway?

- **Centralized Connectivity:** Instead of managing multiple peering relationships, Transit Gateway allows a single connection hub, making management easier.
- **Scalability:** Supports thousands of VPCs, simplifying large-scale cloud networking.
- **Improved Security:** Traffic routing and security policies can be centrally managed.
- **Reduced Complexity:** Eliminates the need for complex route table configurations across multiple VPCs.

Prerequisites

Before proceeding, ensure the following requirements are met:

- **Source VPC** (Tableau VPC) contains Tableau deployed in a private subnet.
- **Target VPC** (Yellowbrick VPC) hosts the Yellowbrick instance within EKS across two private subnets.
- **Connectivity Method**: AWS Transit Gateway.
- **Required Ports**:
  - **Inbound to Yellowbrick Instance (Target VPC, EKS Private Subnets)**:
    - 80, 443, 5432, 11111, 11112, 31000, 31001.
  - **Outbound from Tableau (Source VPC, Private Subnet)**:
    - 80, 443, 5432, 11111, 11112, 31000, 31001, 1024-65535.

---

1. Create a Transit Gateway

1. Navigate to **AWS Console** → **VPC Dashboard**.
2. Select **Transit Gateways** → **Create Transit Gateway**.
3. Provide the following details:
   - **Name**: `tableau-to-yellowbrick-tgw`.
   - **Amazon ASN**: Choose or leave as default.
   - **DNS Support**: Enabled.
   - **Auto-Attach VPCs**: Disabled (manual attachment recommended for better control).

4. Click **Create Transit Gateway**.

---

2. Attach VPCs to the Transit Gateway

Attach Source VPC (Tableau VPC)

1. Go to **AWS Console** → **VPC Dashboard**.
2. Select **Transit Gateway Attachments** → **Create Transit Gateway Attachment**.
3. Enter the following details:
   - **Transit Gateway ID**: `tgw-xxxxxxxxxxxxxxxxx`.
   - **VPC ID**: `vpc-xxxxxxxxxxxxxxxxx` (Source VPC).
   - **Subnet IDs**: Select the private subnets where Tableau is deployed.
4. Click **Create Attachment**.

Attach Target VPC (Yellowbrick Instance VPC)

Repeat the same steps for the **Target VPC**, providing:

- **VPC ID**: `vpc-xxxxxxxxxxxxxxxxx` (Target VPC - Yellowbrick Instance).
- **Subnet IDs**: Select the private subnets where Yellowbrick Instance is deployed.

---

3. Update Route Tables

1. Go to **VPC Dashboard** → **Route Tables**.
2. Identify the route table associated with Tableau's private subnet.
3. Click **Routes** → **Edit Routes**.
4. Add a new route:
   - **Destination**: `10.x.x.x/xx` (Target VPC CIDR).
   - **Target**: Transit Gateway (`tgw-xxxxxxxxxxxxxxxxx`).
5. Click **Save Routes**.

Repeat these steps for the Target VPC's route table:

- **Destination**: `10.x.x.x/xx` (Source VPC CIDR).
- **Target**: Transit Gateway (`tgw-xxxxxxxxxxxxxxxxx`).

---

4. Configure Security Groups

1. Navigate to **EC2 Dashboard** → **Security Groups**.
2. Locate the security group attached to the EKS cluster or Yellowbrick Instance.
3. Click **Inbound Rules** → **Edit inbound rules**.
4. Add the required inbound rules.
5. Click **Save Rules**.
6. Edit **Outbound Rules** to allow communication back to Tableau.
7. Click **Save Rules**.

5. Verify Connectivity

1. SSH into the Tableau Server (Source VPC Private Subnet).
2. Test connectivity to Yellowbrick Instance in the Target VPC using `nc` or `telnet`:

```sh
nc -zv 10.x.x.x 5432  # Check PostgreSQL connection
nc -zv 10.x.x.x 80    # Check HTTP connection
nc -zv 10.x.x.x 443   # Check HTTPS connection
```

3. If tests fail, review security group settings, route tables, and DNS resolution.

6. Summary

✅ Transit Gateway Created

✅ VPCs Attached to Transit Gateway

✅ Route Tables Updated in Both VPCs

✅ Security Groups Configured to Allow Required Ports

✅ Connectivity Verified via Network Tests

7. Troubleshooting

If connectivity issues persist, check the following:

1. **Transit Gateway Attachment Status** - Ensure attachments are available in the AWS Console.
2. **Route Table Misconfigurations** - Confirm that both Source and Target VPCs have the correct routes.
3. **Security Group Rules** - Verify that inbound/outbound traffic is allowed for the required ports.
4. **NACL Restrictions** - Ensure that both inbound and outbound rules are correctly configured.
5. **Firewall Rules (Local or External)** - Ensure firewalls are not blocking traffic.
6. **Use AWS Network Analyzer** - To identify connectivity issues between VPCs:
   - Navigate to **AWS Console** → **VPC Dashboard** → **Network Analyzer**.
   - Create a new network path analysis between Source VPC (Tableau) and Target VPC (Yellowbrick Instance in EKS).
   - Validate routes, security groups, and network ACLs to identify any misconfigurations.

By following this guide, you will successfully configure AWS Transit Gateway to enable Tableau access to a Yellowbrick instance in an EKS cluster.

Setting Up AWS VPC Peering for Yellowbrick Access

Introduction

VPC Peering is a networking connection between two Virtual Private Clouds (VPCs) that enables direct communication without requiring a gateway, VPN, or separate network hardware. It allows instances in the peered VPCs to communicate as if they were within the same network, reducing latency and improving security by keeping traffic internal to AWS.

- See VPC Peering Overview. for high level concepts.
- See VPC Peering Tutorial for tutorial and detailed step by step guide.

NOTE: This guide uses Tableau as an example service in the peered VPC. This process, however, is similar regardless of service.

Why Use VPC Peering?

- **Secure Communication**: No need for external network exposure or VPN connections.
- **Low Latency, High Bandwidth**: Direct connection without the performance impact of going over the public internet.
- **Cost-Effective**: No additional bandwidth charges for data transfer within the same region.
- **Simplified Network Architecture**: Facilitates seamless integration between services running in different VPCs.

This guide will walk you through the steps to establish a VPC Peering connection between a Tableau instance in one VPC and a Yellowbrick database in an EKS-hosted environment within another VPC.

Prerequisites

Before starting, ensure the following requirements are met:

- **Source VPC**: Contains Tableau in a private subnet.
- **Target VPC**: Contains Yellowbrick Instance in EKS across two private subnets.
- **Connectivity Method**: VPC Peering.
- **Required Ports to Allow**:
  - **Inbound to Yellowbrick Instance (Target VPC, EKS Private Subnets)**:
    - 80, 443, 5432, 11111, 11112, 31000, 31001
  - **Outbound from Tableau (Source VPC, Private Subnet)**:
    - 80, 443, 5432, 11111, 11112, 31000, 31001, 1024-65535

1. Create a VPC Peering Connection

1. Go to **AWS Console → VPC Dashboard**.
2. Select **Peering Connections → Create Peering Connection**.
3. Enter the following details:

- **Peering connection name**: `tableau-to-yb-peering`
- **Requester VPC (Source - Tableau)**: `vpc-xxxxxxxxxxxxxxxx`
- **Accepter VPC (Target - Yellowbrick Instance in EKS)**: `vpc-xxxxxxxxxxxxxxxx`
- **Account**: Choose **My Account** if both VPCs are in the same AWS account, otherwise, enter the correct AWS account ID.

4. Click **Create Peering Connection**.

Accept the Peering Request

1. Go to **VPC** → **Peering Connections**.
2. Locate the created peering connection.
3. Click **Actions** → **Accept Request**.
4. Enable DNS Resolution for VPC Peering:
   - Go to **VPC Dashboard** → **Peering Connections**.
   - Select the created peering connection.
   - Click **Actions** → **Modify DNS Resolution**.
   - Set **DNS Resolution** to **Enabled** for both Requester and Accepter VPCs.

---

2. Update Route Tables

Since Tableau is in a private subnet and EKS is in two private subnets, update only the private route tables for both VPCs.

Update Source VPC Route Table

1. Go to **VPC Dashboard** → **Route Tables**.
2. Identify the route table associated with Tableau's private subnet.
3. Click **Routes** → **Edit Routes**.
4. Add a new route:
   - **Destination**: `10.x.x.x/xx` (Target VPC CIDR)
   - **Target**: Peering Connection (`pcx-xxxxxxxx`)
5. Click **Save Routes**.

Update Target VPC Route Tables

Since EKS is deployed in two private subnets, update both private route tables.

For each private route table in the Target VPC:

1. Go to **VPC Dashboard** → **Route Tables**.
2. Identify the route tables associated with the two private subnets where EKS is running.
3. Click **Routes** → **Edit Routes**.
4. Add a new route:
   - **Destination**: `10.x.x.x/xx` (Source VPC CIDR)
   - **Target**: Peering Connection (`pcx-xxxxxxxx`)
5. Click **Save Routes**.

---

3. Configure Security Groups

Modify Security Group for Yellowbrick Instance (EKS Private Subnets - Target VPC)

1. Go to **EC2 Dashboard** → **Security Groups**.
2. Find the security group attached to the EKS cluster or Yellowbrick Instance.
3. Click **Inbound Rules** → **Edit inbound rules**.
4. Add the following inbound rules:

Protocol	Port Range	Source CIDR (Tableau Private Subnet - Source VPC)	Description
HTTP	80	`10.x.x.x/xx`	HTTP server port (redirects to 443)
HTTPS	443	`10.x.x.x/xx`	TLS versions 1.1 and 1.2 only
TCP	5432	`10.x.x.x/xx`	Default port for database connections
TCP	11111, 11112	`10.x.x.x/xx`	Control ports for Yellowbrick operations
TCP	31000, 31001	`10.x.x.x/xx`	Yellowbrick UI and API ports

5. Click **Save Rules**.
6. Edit **Outbound Rules**:
   - **Destination**: `10.x.x.x/xx`
   - Allow **TCP 1024-65535** for dynamic ports.
7. Click **Save Rules**.

---

4. Verify Connectivity

1. SSH into the Tableau Server (Source VPC Private Subnet).
2. Test connectivity to Yellowbrick Instance in the Target VPC using `nc` or `telnet`:

```sh
nc -zv 10.x.x.x 5432  # Check PostgreSQL connection
nc -zv 10.x.x.x 80    # Check HTTP connection
nc -zv 10.x.x.x 443   # Check HTTPS connection
```

3. If tests fail, review security group settings, route tables, and DNS resolution.

---

5. Summary

✅ VPC Peering Connection Created

✅ DNS Resolution Enabled

✅ Private Route Tables Updated in Both VPCs

✅ Security Groups Configured to Allow Required Ports

✅ Connectivity Verified via Network Tests

---

6. Troubleshooting

If connectivity issues persist, check the following:

1. **Peering Connection Issues** - Ensure the peering connection status is **Active** in the AWS Console.
2. **Route Table Misconfigurations** - Confirm that both Source and Target VPCs have the correct routes.
3. **Security Group Rules** - Ensure inbound/outbound traffic is allowed for the required ports.

4. **NACL Restrictions** - Verify that both inbound and outbound rules are correctly configured.
5. **Firewall Rules (Local or External)** - Ensure firewalls are not blocking traffic.
6. **Use AWS Network Analyzer**:
   - Go to **AWS Console** → **VPC Dashboard** → **Network Analyzer**.
   - Create a new network path analysis between **Source VPC (Tableau)** and **Target VPC (Yellowbrick Instance in EKS)**.
   - Validate **routes, security groups, and network ACLs** to identify any misconfigurations.

This guide provides a structured approach to setting up VPC Peering for Yellowbrick Access and resolving common issues.

Configuring SSL certificates

Yellowbrick Documentation > How-to Guides > Administer Yellowbrick > Cloud > Configuring > Configuring SSL Certificates

Platforms: EE: All cloud platforms

Parent topic: Configuring

Yellowbrick Manager and Yellowbrick instances come with self-signed certificates. If you want to change these certificates to your own, you can do so with the following instructions.

WARNING

The final step of this process requires restarting the Yellowbrick instance/s, so a short period of scheduled downtime is necessary. This will be improved in a future release.

For illustrative purposes we use AWS Route53 and the LetsEncrypt ClusterIssuer from the cert-manager project. cert-manager also provides for other certificate sources such as Hashicorp Vault, AWS KMS and other cloud providers such as Azure and GCP.

Prerequisites

- DNS Entries (A records or CNAMES) to Service IP addresses for the manager and instance deployments. See Vanity DNS.
- kubectl with kubernetes cluster access with ability add/edit secrets.

Setting up `kubeconfig`

To point to the correct Kubernetes cluster or set your `kubeconfig`:

AWS:

```bash
aws eks --region $region update-kubeconfig --name $eksClusterName
```

EKS may require additional steps to grant your IAM user access to Kubernetes objects. For more information see Kubeconfig Setup on EKS.

Azure:

```bash
az aks get-credentials --resource-group $resourceGroup --name $aksClusterName --admin --overwrite
```

GCP:

```bash
gcloud container clusters get-credentials $clusterName --region=$region
```

Updating the Default Yellowbrick Manager Certificate

Setup

Step 1: Check the certificate before replacing

```bash
openssl s_client -showcerts -connect ${yellowbrickManagerIp}:443 </dev/null
    ## Examples:
    #    openssl s_client -showcerts -connect ***.***.***.168.com:443 </dev/null
    #    openssl s_client -showcerts -connect manager.xyz.dev.yellowbrickcloud.com:443 </dev/null

    ## Example output:
    #   Connecting to ***.***.***.168
    #   CONNECTED(00000005)
    #   depth=0 O=Yellowbrick Manager, CN=Yellowbrick Manager
    #   verify error:num=18:self-signed certificate
    #   verify return:1
    #   depth=0 O=Yellowbrick Manager, CN=Yellowbrick Manager
    #   ....
    #   ....
```

Step 2: Ensure you're pointed to the correct Kubernetes cluster

```bash
kubectl config current-context
```

Step 3: Get the namespace of your Yellowbrick Manager. The namespace will be the one you created during installation. In the below output it will be `yb-ns`.

```bash
kubectl get ns
# Example output:
    # NAME              STATUS   AGE
    # default           Active   22d
    # kube-node-lease   Active   22d
    # kube-public       Active   22d
    # kube-system       Active   22d
    # monitoring        Active   22d
    # yb-ns             Active   22d
```

Step 4: Create the ClusterIssuer configured for your cloud providers DNS. The following example uses a DNS challange against AWS's Route53 The credentials for AWS are provided in the form of an AWS Key and Secret but any method of supported authentication such as roles may be used.

```bash
kubectl apply -f - <<EOF
apiVersion: v1
kind: Secret
metadata:
  name: secret-le-aws
  namespace: kube-system
  annotations:
    kubernetes.io/service-account.name: "default"
type: kubernetes.io/service-account-token
stringData:
  aws_secret_access_key: <redacted>
---
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-contour-cluster-issuer
spec:
  acme:
    email: "platform-ops@yellowbrick.com"
    privateKeySecretRef:
      name: acme-account-key
    server: https://acme-v02.api.letsencrypt.org/directory
    solvers:
      - dns01:
          route53:
            region: us-east-1
```

```
            accessKeyID: <redacted>
            secretAccessKeySecretRef:
              name: secret-le-aws
              key: aws_secret_access_key
EOF
```

```bash
## Take a backup of the existing certificate secret and csr certificate
kubectl -n $NAMESPACE get secret yb-manager-tls -o yaml >> yb-manager-tls-original.yaml
kubectl -n $NAMESPACE get certificate yb-manager-cert -o yaml >> yb-manager-cert-original.yaml
```

Patch the certificate

```bash
kubectl patch certificate yb-manager-cert -n $NAMESPACE --type='merge' -p '{"spec":{"dnsNames":["manager.yb.xip.net"], "commonName
```

or replace it

```bash
kubectl apply -f - <<EOF
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: yb-manager-cert
  namespace: $NAMESPACE
spec:
  secretName: yb-manager-tls
  commonName: manager.yb.xip.net
  dnsNames:
    - manager.yb.xip.net
  issuerRef:
    name: letsencrypt-contour-cluster-issuer
    kind: ClusterIssuer
EOF
```

Since the cluster issuer is changing we must delete the CertificateRequest

```bash
# get the name of the certificaterequest
kubectl get certificaterequests -n $NAMESPACE
    ## Example: kubectl -n $NAMESPACE delete certificaterequest ybinst-jeff-instance-tls-sgnj6


# delete the certificaterequest
kubectl delete certificaterequest yb-manager-cert-n4wfp-xxxx
```

Wait for the certificate to be ready.

```bash
kubectl get certificate -n NAMESPACE -w
```

Restart the yb-manager pod

```bash
kubectl -n $NAMESPACE get pods
kubectl -n $NAMESPACE delete pod yb-manager-xxxxx-xxxxxx
    ## Example: kubectl -n $NAMESPACE delete pod yb-manager-6bd6bfb946-sqmtc
kubectl -n $NAMESPACE get pods
```

Step 5: Check the replaced certificate

```bash
openssl s_client -showcerts -connect ${yellowbrickManagerIp}:443 </dev/null
```

Updating the Default Instance Certificate

This will need to be performed for each instance.

Step 1: Take a backup of the existing instance-tls certificate before replacing it:

```bash
kubectl -n $NAMESPACE get certificate ybinst-${dw-instance-name}-instance-tls -o yaml >> instance-tls-backup.yaml
    # Example: kubectl -n jpk-ns get certificate ybinst-jpk-instance-tls -o yaml >> instance-tls-backup.yaml
```

Step 2: Update the certificate. One for each instance.

```bash
kubectl patch certificate ybinst-${dw-instance-name}-instance-tls -n $NAMESPACE --type='merge' -p '{"spec":{"dnsNames":["${dw-inst
```

or replace it

```bash
kubectl apply -f - <<EOF
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: ybinst-${dw-instance-name}-instance-tls
  namespace: $NAMESPACE
spec:
  secretName: ybinst-${dw-instance-name}-instance-tls
  commonName: ${dw-instance-name}.yb.xip.net
  dnsNames:
    - ${dw-instance-name}.yb.xip.net
  issuerRef:
    name: letsencrypt-contour-cluster-issuer
    kind: ClusterIssuer
  keystores:
    jks:
      create: true
      passwordSecretRef:
        key: jks
        name: ybinst-${dw-instance-name}-tls-secret
    pkcs12:
      create: true
      passwordSecretRef:
        key: pkcs
        name: ybinst-${dw-instance-name}-tls-secret
EOF
```

Since the cluster issuer is changing we must delete the CertificateRequest

```bash
# get the name of the certificaterequest
kubectl get certificaterequests -n $NAMESPACE
    ## Example: kubectl -n $NAMESPACE delete certificaterequest ybinst-jeff-instance-tls-sgnj6

# delete the certificaterequest
kubectl delete  certificate ${dw-instance-name}-instance-tls-xxxx
```

Wait for the certificate to be ready.

```bash
kubectl get certificate -n NAMESPACE -w
```

Step 3: Restart the instance via kubectl

```bash
# Restart each dw instance
kubectl -n $instanceNamespace patch ybinstance/${dw-instance-name} --type=merge -p '{"spec":{"requestedState":"Restart"}}'
```

You can verify that the new SSL certificate is being used

```bash
openssl s_client -starttls postgres -connect ${dw-instance-name}.yb.xip.net:5432 </dev/null
```

Custom Object Storage

Platforms: EE: All cloud platforms

Parent topic: Configuring

Columnar data persisted by the instance is persisted in object storage. By default, the Deployer will automatically create an object storage bucket and configure the instance to use it. However you can opt out of this behaviour by deselecting the **Create initial external storage** option.

If you opt out, you need to use the `CREATE EXTERNAL STORAGE` and `CREATE EXTERNAL LOCATION` commands to configure the object storage location.

You will need to create an object storage bucket if you do not already have one, and note some of its properties or access details to pass into the creation commands. To avoid extra cost and minimise latency it's strongly recommanded to setup this storage in the same cloud provider and region as the one in which the instance has been deployed.

Note that in general, storage capacity is unlimited. If necessary, you can apply disk quotas to limit the size of individual databases, schemas, and tables.

Using an AWS S3 bucket

Step 1: Create a Bucket and IAM User

The following examples will be based on a bucket named `my-yellowbrick-storage-bucket`.

Create a general-purpose S3 bucket in the same region as your deployment, either by using the AWS Management Console or the AWS CLI.

Create a dedicated IAM user for the bucket, and assign it the following two policies: This first policy grants all S3 actions on the S3 bucket and its contents. Here is an example of such policy definition in JSON:

```
{
    "PolicyName": "S3BucketObjectsPolicy",
    "PolicyDocument": {
        "Version": "2012-10-17",
        "Statement": [
            {
                "Effect": "Allow",
                "Action": [
                    "s3:*"
                ],
                "Resource": [
                    "arn:aws:s3:::my-yellowbrick-storage-bucket/*"
                ]
            }
        ]
    }
}
```

This second policy shall grant the `s3:GetBucketLocation` and `s3:ListBucket` actions on the bucket itself. Here is an example of such definition in JSON:

```
{
    "PolicyName": "S3BucketPolicy",
    "PolicyDocument": {
        "Version": "2012-10-17",
        "Statement": [
            {
```

```
                "Effect": "Allow",
                "Action": [
                    "s3:GetBucketLocation",
                    "s3:ListBucket"
                ],
                "Resource": [
                    "arn:aws:s3:::my-yellowbrick-storage-bucket"
                ]
            }
        ]
    }
}
```

Step 2: Create an Access Key and Configure the Instance

Once the user has been created, you will need to create an IAM Access Key associated with it, to be used for an **Application running on an AWS compute service**.

During the creation of the key you will provided with an **Access key** and a **Secret access key**. These two values must be stored carefully, as they will be required for the configuration of the external storage.

Once the key details are known, it's possible to setup the storage in the Yellowbrick Database using SQL:

```
CREATE EXTERNAL STORAGE inst-storage TYPE s3 ENDPOINT 'http://s3.<region>.amazonaws.com' REGION '<region>' IDENTITY '<access key>'
CREATE EXTERNAL LOCATION storage-location path '<bucket name>' EXTERNAL STORAGE inst-storage USAGE PRIMARY DEFAULT;
```

Using an Azure Azure Storage Account

Step 1: Create a Storage Account and Container

You must create a Storage Account for **Blob Storage**, referred to as **Storage V2**. Use the **Locally Redundant Storage (LRS)** and set the performance to standard.

Within this Storage Account you must create a **Storage Container**. It is recommended to keep that container private and never enable public access.

Ste 2: Retrieve Access Keys and Configure the Instance

The Storage Account will provide two **Access Keys**; either one of these keys may be used to configure the storage using SQL:

```
CREATE EXTERNAL STORAGE inst-storage TYPE azure ENDPOINT 'https://<storage account name>.blob.core.windows.net' REGION '<region>'
CREATE EXTERNAL LOCATION storage-location path '<storage container name>' EXTERNAL STORAGE inst-storage USAGE PRIMARY DEFAULT;
```

Google Cloud Storage

Step 1: Create a Storage Bucket and IAM Service Account

You must create a Google Cloud Storage bucket. Then you must create an IAM Service Account that will have necessary permissions to access this bucket. This service account will present itself as an email address, for example `<account name>@<project id>.iam.gserviceaccount.com`

On that service account you must grant at least the following permissions:

```
storage.buckets.delete
storage.buckets.get
storage.objects.create
storage.objects.delete
```

```
storage.objects.get
storage.objects.getIamPolicy
storage.objects.list
storage.objects.update
```

A role called `YBWorkerCloudStorageAccess` role is created during deployment and possesses the necessary permissions. Otherwise you will need to create a custom role, or rely on predefined one that will grant more permissions than necessary.

Step 2: Create an Access Key and Configure the Instance

Create an **Access Key** on the newly created service account. The key will be provided as a JSON document containing various parameters. The `private_key` one must be selected and then encoded in base64 to be used as the credential in the storage configuration:

```
CREATE EXTERNAL STORAGE inst-storage TYPE gs ENDPOINT 'https://storage.googleapis.com' REGION '<region>' IDENTITY '<service accoun
CREATE EXTERNAL LOCATION storage-location path '<bucket name>' EXTERNAL STORAGE inst-storage USAGE PRIMARY DEFAULT;
```

Custom Security Agents

In some cases it is necessary to install customer security agents on a Yellowbrick deployment. Details of supported agents and how to approach running them are outlined here.

CrowdStrike Falcon Kubernetes Protection Agent on EKS

This guide provides recommendations on installing the CrowdStrike Falcon Kubernetes Protection agent on Amazon EKS clusters. The recommended approach is to use Helm, which installs the agent as Kubernetes resources within the cluster.

Installing the agent directly on the EKS nodes themselves is currently not supported, however the capability will be available in a subsequent Yellowbrick release by directly modifying node group launch templates.

Installing via Kubernetes Resources

CrowdStrike Falcon's recommended installation method for Kubernetes environments is to deploy the Falcon sensor as a Kubernetes DaemonSet using Helm. This approach is fully supported and simplifies the management and deployment of agents across your EKS nodes.

We highly recommend following the online documentation to install yourself.

Multi-Instance Configuration

Platforms: EE: All cloud platforms

Parent topic: Configuring

After installation, Yellowbrick Manager supports provisioning and managing additional instances. When working with multi-instance deployments, it can be helpful to automatically configure them in a consistent manner.

Yellowbrick Operator supports automatic execution of a custom initialization script for each instance in order to centralize certain policies, for example making sure auto-suspend is automatically set, or that single sign-on is configured consistently across instances.

The Operator makes sure that the initialization script is run once, when an instance is provisioned. It is deployed as a shell script inside a Kubernetes ConfigMap. The script can run SQL statements or start other provisioning activities.

Kubernetes ConfigMap

The Kubernetes ConfigMap containing customization script logic must be named `yb-operator-custominit-config` and be placed in the namespace that you installed Yellowbrick Operator. A simple example appears below:

```yaml
kind: ConfigMap
metadata:
  name: yb-operator-custominit-config
apiVersion: v1
data:
  inst-setup.sh: |
    echo "Hello, world"
```

Save this content to a file `custom-init-configmap.yaml` and apply it using kubernetes `kubectl`:

```txt
% kubectl -n ${NAMESPACE} apply -f custom-init-configmap.yaml
```

To see the `echo` output for the script, look at the pod logs for the `ybinst-pg` container using `kubectl`.

Example: Auto-Suspend Policy

This example sets up a common policy for new instances that cause them to auto-suspend when not in use for an hour.

```yaml
kind: ConfigMap
metadata:
  name: yb-operator-custominit-config
apiVersion: v1
data:
  inst-setup.sh: |
    # Do the sql initialization
    cat <<EOF | ybsql yellowbrick

    -- Setup instance idle time.
    ALTER SYSTEM SET ybd_instance_idle_time to 3600;
    SELECT pg_reload_conf();
```

```
    EOF
    rc=$?
    if [ "$rc" != "0" ]; then
      echo "Setup failed"
      exit 1
    fi
```

Example: Single Sign-On Policy

When setting up a common identity provider, it is a best-practice to also setup the identity provider EXTERNAL AUTHENTICATION object for new instances so they participate in Single Sign-On. This example runs a SQL statement to configure accordingly:

```yaml
kind: ConfigMap
metadata:
  name: yb-operator-custominit-config
apiVersion: v1
data:
  inst-setup.sh: |
    # Do the sql initialization
    cat <<EOF | ybsql yellowbrick

    -- Add our external authentication customization.
    DROP EXTERNAL AUTHENTICATION IF EXISTS ad;
    CREATE EXTERNAL AUTHENTICATION ad
      issuer 'https://sts.windows.net/{{TENANT_ID}}/'
      user_mapping_claim 'upn'
      grant ('consumer')
      audience ('{{CILENT_ID}}')
      auto_create
      enabled;

    EOF
    rc=$?
    if [ "$rc" != "0" ]; then
      echo "Setup failed"
      exit 1
    fi
```

Note to replace the `TENANT_ID` and `CLIENT_ID` variables if using Microsoft Entra (formerly Azure AD). See Single Sign-On for more information.

Single Sign-On

Platforms: EE: All cloud platforms

Parent topic: Configuring

To enable Single Sign-On (SSO), Yellowbrick can be configured to use an external identity provider (IDP). Such external providers are necessary to support MFA (multi-factor authentication) which should be in use for all but the most simple trial accounts.

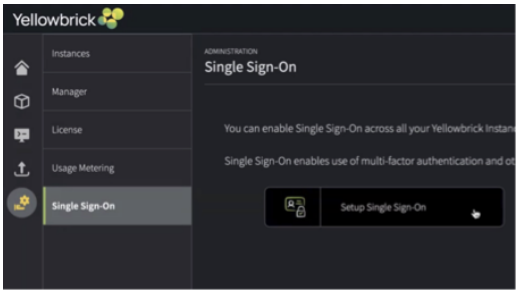
Yellowbrick supports several external authentication providers using the OpenID Connect (based on OAuth 2.0) standard. The currently supported list of providers includes:

- Apple
- Auth0
- Microsoft Entra (formerly Azure Active Directory)
- Microsoft ADFS
- GitHub
- GitLab
- Google
- Keycloak
- Okta
- Salesforce

New providers are being added all the time, so please get in touch if you'd like to see one that isn't on this list.

SSO Configuration Instructions

In order to set up an external IDP, use Yellowbrick Manager. Open Yellowbrick Manager, and navigate to the Single Sign-On page as shown in this screen shot:



Click Setup Single Sign-On to continue and follow the wizard to add a new IDP.

Step 1: Choose Your IDP

Choose the appropriate external identity provider, and give it a name and identifier. Yellowbrick supports multiple IDPs and the Login page will offer users a choice of which one to use. For that reason, each integration has a user friendly name (the Name field) and a unique identifier to register it with the system (the Identifier field).

Step 2: Configure the IDP

To complete this step, you'll need to work with the individual or organization in your company responsible for administration of identity management. Provide them with the Redirect URL shown in this step of the wizard, and ask to to supply you in return the corresponding OpenID Connect parameter values to fill out below.

Note that different providers have different parameters; while all of them have a Client Identifier and Client Secret, others may have various optional fields.

Step 3: Register in the Database

The final step of the wizard provides the SQL necessary to add the provider to the database instance. Log in using the credentials supplied for the Administrator Account during installation to execute the SQL provided.

Altering IDP settings requires the Yellowbrick Manager to restart. This only takes 10-20 seconds, doesn't result in any instance downtime, and should be performed immediately so that the newly registered IDP is operational.

See also CREATE EXTERNAL AUTHENTICATION.

Yellowbrick Custom AMI on AWS

Platforms: EE: All cloud platforms

Parent topic: Configuring

Introduction

Yellowbrick is run on node groups that create EC2 instances from an Amazon Machine Image (AMI). A default installation uses the recommended Amazon EKS optimized Amazon Linux AMI built on Amazon Linux 2023 (AL2023). It is optional to customize your own AMI to be used instead.

When using a custom AMI, please note that Yellowbrick provisions user-data as part of its configuration. To ensure compatibility, any custom AMI that does not use Amazon Linux 2023 must include `nodeadm` as part of its bootstrap process. For more information on using `nodeadm` or building a custom Amazon Linux AMI, see the AWS documentation.

Overview

The configuration of node groups is modeled by the YBNodeGroup Kubernetes Custom Resources. Each YBNodeGroup represents an AWS Nodegroup running on EKS. Updating the AMI used for a YBNodeGroup will create a new version in the launch template of the underlying AWS Nodegroup. Configuring a custom AMI is done by editing the YBNodeGroup Kubernetes Custom Resource and setting the `customImage` field to the target AMI name and owner.

Please note that changing an AMI image will result in a rollout of new nodes in the EKS cluster. The Yellowbrick instance running on these node groups must be suspended before performing an AMI update operation to prevent disruption to the system and its users.

A custom AMI image may be configured for any of the underlying node groups of Yellowbrick:

- yb-bulk
- yb-mgr
- yb-mon
- yb-op
- yb-worker

Once node group configuration is customized, it will remain customized for that specific node group. Users will be responsible for managing future AMI updates.

Customizing AMI

The process outlined in this document will customize the AMI for the `small-v2` compute cluster node group of the Yellowbrick instance `demo`. These steps should be repeated for any additional node groups where customization is desired.

Listing YBNodeGroups

First, list the current YBNodeGroups in the instance namespace. In the case of a default install where the instance name give was `demo`, the instance namespace will be `yb-demo`:

```sh
$ kubectl get ybnodegroup -n yb-demo
NAME                HARDWARE TYPE    IN USE    STATE         ERROR
large-v1            i3en.metal       false     Unused
large-v2            i4i.32xlarge     false     Unused
```

```
small-v1            m5dn.4xlarge    false    Unused
small-v2            i4i.4xlarge     true     Provisioned
yb-bulk-scaled      c5n.9xlarge     false    Unused
yb-bulk-standard    m5.2xlarge      true     Provisioned
yb-compiler-scaled  m5.8xlarge      false    Unused
yb-mgr-scaled       m5d.24xlarge    false    Unused
yb-mgr-standard     m5d.4xlarge     true     Provisioned
yb-mon-scaled       m5.8xlarge      false    Unused
yb-mon-standard     m5.xlarge       true     Provisioned
yb-op-scaled        m5.xlarge       false    Unused
yb-op-standard      t3.large        true     Provisioned
```

Only YBNodeGroups that are currently in use will be provisioned as AWS Nodegroups. YBNodeGroups that are not in use may or may not have an AWS Nodegroup depending on if they have ever been provisioned before. Currently, AWS Nodegroups are only created and updated, they are not deleted.

Describe YBNodeGroup

Inspect the current AMI of a YBNodeGroup:

```sh
$ kubectl describe ybnodegroup -n yb-demo small-v2
Name:         small-v2
Namespace:    yb-demo
Labels:       <none>
Annotations:  <none>
API Version:  cluster.yellowbrick.io/v1
Kind:         YBNodeGroup
Metadata:
  Creation Timestamp:  2024-12-12T06:54:10Z
  Generation:          1
  Resource Version:  4097
  UID:               71a801c9-3368-450a-80c6-6cad4de4d596
Spec:
  Cloud Init Config:   workerNodeConfig
  Custom Image:        amazon:amazon-eks-node-al2023-x86_64-standard-1.30-v20241121
  Hardware Type:       i4i.4xlarge
  Huge Pages:          1073741824
  Inuse:               true
  Network Interfaces:  3
  Node Selector:       yb-worker
Status:
  State:               Provisioned
Events:                <none>
```

In this example, the AMI with name `amazon-eks-node-al2023-x86_64-standard-1.30-v20241121` from the owner `amazon` is being used. This is the AWS recommended EKS optimized AMI for this version of EKS. For stability, AMI versions used by node groups will be fixed based upon the version of Yellowbrick being used. To upgrade the AMI version, either upgrade your version of Yellowbrick or update the YBNodeGroup to a recommended and supported AMI version.

Updating YBNodeGroup

Create a patch to the YBNodeGroup that includes a new `customImage`, referencing the new AMI. A valid value will be in the format "name" or "owner:name". If owner is not specified, "amazon" will be inferred:

```sh
$ kubectl patch ybnodegroup small-v2 \
    -n yb-demo \
    --type='merge' \
    -p '{"spec":{"customImage":"amazon:amazon-eks-node-al2023-x86_64-standard-1.30-v20241205"}}'
```

It is also possible to use `kubectl edit` to modify the custom resource directly:

```sh
$ kubectl edit ybnodegroup small-v2 -n yb-demo
```

The owner value can be an owner alias such as `self`, `amazon`, `aws-backup-vault`, `aws-marketplace`, or the AWS account ID. A wildcard pattern can be used in the `customImage` field such as `amazon-eks-node-al2023-x86_64-standard-1.30-*`. If more than one AMI is returned from this pattern, the most recent will be used.

The AWS CLI can be used to understand the correct name and owner values given a specific AMI ID:

```sh
$ aws ec2 describe-images --image-ids ami-03a81a3c47f5d8d98
{
    "Images": [
        {
            "Architecture": "x86_64",
            "CreationDate": "2024-12-05T18:24:32.000Z",
            "ImageId": "ami-03a81a3c47f5d8d98",
            "ImageLocation": "amazon/amazon-eks-node-al2023-x86_64-standard-1.30-v20241205",
            "ImageType": "machine",
            "Public": true,
            "OwnerId": "602401143452",
            "PlatformDetails": "Linux/UNIX",
            "UsageOperation": "RunInstances",
            "State": "available",
            "BlockDeviceMappings": [
                {
                    "DeviceName": "/dev/xvda",
                    "Ebs": {
                        "DeleteOnTermination": true,
                        "Iops": 3000,
                        "SnapshotId": "snap-0e0a4e2237d64dc25",
                        "VolumeSize": 20,
                        "VolumeType": "gp3",
                        "Throughput": 125,
                        "Encrypted": false
                    }
                }
            ],
            "Description": "EKS-optimized Kubernetes node based on Amazon Linux 2023, (k8s: 1.30.7, containerd: 1.7.*)",
            "EnaSupport": true,
            "Hypervisor": "xen",
            "ImageOwnerAlias": "amazon",
            "Name": "amazon-eks-node-al2023-x86_64-standard-1.30-v20241205",
            "RootDeviceName": "/dev/xvda",
            "RootDeviceType": "ebs",
            "SriovNetSupport": "simple",
            "VirtualizationType": "hvm",
            "BootMode": "uefi-preferred",
            "DeprecationTime": "2026-12-05T18:24:32.000Z",
            "ImdsSupport": "v2.0"
        }
    ]
}
```

Provisioning

An update to the YBNodeGroup AMI will trigger the creation of a new launch template version and update the underlying AWS Nodegroup to use that version. While this is happening, the status of the YBNodeGroup will be `provisioning`. After the updated has been made, the status of the YBNodeGroup be `provisioned`. Any existing EKS nodes running in that node group with the old AMI will be replaced with new nodes running the new AMI. If an error occurred during the update, it will be shown in the `error` field of the YBNodeGroup for that node group.

User Data

A custom AMI may need to be coordinated with custom user data. If your custom AMI also requires custom user data, this is considered an advanced deployment scenario and must be performed directly in the AWS console. The outline of the steps to be taken are:

1. Identify the node group launch template.
2. Create a new version of the launch template that includes the new custom user data and the new custom AMI.
3. Update the node group to use that new launch template version.
4. Update your YBNodeGroup `customImage` to match the AMI as outlined above.

Please note if creating custom user data as a multi-part document, you must not use `==YBBOUNDARY==` markers. Doing so could result in your custom user data being overwritten when upgrading versions of Yellowbrick.

Cloud: Installation

Platforms: EE: All cloud platforms

Parent topic: Cloud

In this section:

Yellowbrick is installed into your own cloud account using a tool called the **Yellowbrick Deployer**. The Deployer is a machine image available across supported public cloud platforms and regions. It contains all Yellowbrick software along with a deployment tool. The deployment tool can be driven via a graphical user interface for simple installations, or via a command line for more complex installations. It's responsible for installs, upgrades and uninstalls.

If you're doing a Yellowbrick PoC or just getting started, we strongly recommend doing a public installation using the Deployer web application since it's by far the easiest way to get started and requires minimal cloud infrastructure administration.

Public Installation using the Web Application

Public installations deploy Yellowbrick into a network with public (internet-facing) access. The Deployer creates necessary cloud infrastructure (such as VPCs, networks, routers, gateways and a Kubernetes cluster) on your behalf. Most of the infrastructure is private but the instance is accessed over the internet, like most SaaS applications.

To secure the deployment, you can add IP address access restrictions and configure multi-factor authentication using single sign-on (SSO). The Deployer web application can export an infrastructure manifest if you're curious to see exactly what will be configured.

- AWS public installation instructions
- Azure public installation instructions
- GCP public installation instructions

We highly recommend setting up SSO with Multi-Factor Authentication (MFA) before loading confidential data or doing business-critical work.

Private Installation using the Web Application

Yellowbrick uniquely supports installation into a completely private network that's inaccessible from the internet. To set up and configure such a deployment, you'll need to have skills in cloud infrastructure administration and networking in order to provision the prerequisites needed before running the Deployer. This is a more involved and intricate process than a public installation.

Private installations are documented here.

Self-Managed Installation

These instructions are for installing Yellowbrick as a self-managed installation. This is considered an advanced scenario and is used when one wants to customize every aspect of the installation process.

Self-managed installation is documented here.

CLI Deployments

For scripting deployments as part of CI/CD pipelines or automating deployments in a headless fashion, the deployment tool can be driven from the CLI to perform both public and private installations. CLI deployment also allows further customization of the installation by allowing things like timeouts to be overridden.

CLI deployments are primarily driven by a JSON configuration file which can be exported by the web application as a starting point.

CLI installations are documented here

CLI Install Instructions

Platforms: EE: All cloud platforms

Parent topic: Installing

In this section:

The Deployer CLI allows headless automation of the Yellowbrick installation process as well as more extensive customization of timeouts and providing additional options. To use the CLI, you'll need to have a working knowledge of cloud administation and be comfortable modifying JSON, working on the console and handling error messages.

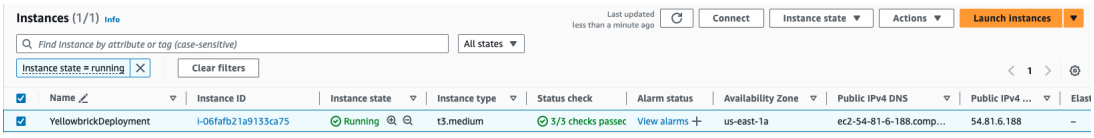
To access the CLI, first to start the Deployer virtual machine by following the install instructions. Then connect via SSH to the instance, using the instructions provided below. Once connected, you can use the CLI in the `/opt/ybd` directory:

```
/opt/ybd/yb-install --help
A CLI tool for managing installation of Yellowbrick Instance

Usage:
  yb-install [command] [flags]
o
...
```

See the CLI reference and other documentation in this section for more information.

AWS Connection Instructions

Use the EC2 portal. Select the Deployer and click on the **Connect** button, then follow the instructions provided by AWS.



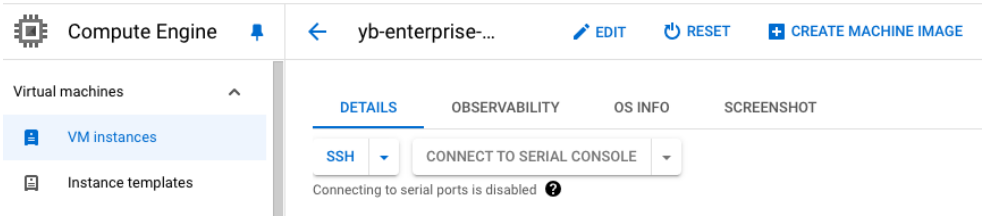
Azure Connection Instructions

You provided a SSH key during the creation of the Deployer, as it is one of its required parameters. Directly use this key to connect to the machine, with the user `azureuser` and the IP address provided in the Deployer outputs:

```
ssh -i <private key> azureuser@<deployer IP>
```

GCP Connection Instructions

Connect to the Deployer buy using the VM Instances portal. Go on the Deployer details and click on the **SSH** button, then follow the instructions provided by GCP.


Compute Engine
yb-enterprise-...
EDIT
RESET
CREATE MACHINE IMAGE
Virtual machines
VM instances
Instance templates
DETAILS
OBSERVABILITY
OS INFO
SCREENSHOT
SSH
CONNECT TO SERIAL CONSOLE
Connecting to serial ports is disabled

Deployer CLI Reference

Platforms: EE: All cloud platforms

Parent topic: CLI Install Instructions

The common syntax is:

```
/opt/ybd/yb-install [command] [flags]
```

Many of the commands consume a JSON Configuration file documented here. The easiest way to obtain a configuration file is to run through the installation web application, fill out all the fields for an installation, then choose to export the configuration file at the end without proceeding with the installation.

Flags

- `--allow/-a a.b.c.d/32` (Optional) When performing an install, uninstall, or interactive operation, allow the supplied CIDR network `a.b.c.d/32` access to service endpoints during the operation.
- `--log-file/-l` (Optional) File path to which this CLI redirects its logs, defaults to `/tmp/install.log`.
- `--verbose/-v` (Optional) Enables Verbose log (`interactive` command is always verbose).
- `--help/-h` Shows the help text
- `--version` Shows the version

Commands

interactive

This launches the web application server UI.

```
Usage

./yb-install interactive [--port=<PORT> --key==<key>]
```

Additional Flags

- `--port/-p` (Optional) Port on which the web server runs. Defaults to `8443`
- `--key/-k` (Optional) Deployer Access Key that protects the installation. By Default, it is **auto generated**, and is logged on the console. See cloud deployer access key.

validate

Validates a given configuration to ensure that it is good for proceeding and shows any errors.

```
Usage

./yb-install validate --config-path=<path-to-configuration-file>
```

Additional Flags

- `--config-path/-c` : (Required) Path to config for installing Yellowbrick Data Warehouse instance

---

install

Initiates the installation of infrastructure and Yellowbrick software. This subcommand will also run the validate/plan subcommand before starting the installation.

Note: At the end of the install, the final configuration used is saved as a Kubernetes Secret in the cluster created during the installation. The secret name is `yb-install-config` and it is present in the same namespace as the Yellowbrick Operator.

```
Usage

./yb-install install --config-path=<path-to-configuration-file>
```

Flags

- `--config-path/-c` : (Required) Path to config for installing Yellowbrick Data Warehouse instance
- `--encryption-key-id/-e` : (Optional) Full ID of the encryption key in the cloud provider KMS
- `--tags/-t` : (Optional) A comma separated list of additional tags to set on cloud resources, example: key1=value1,key2=value2

---

manifest

Display a JSON manifest of the different cloud resources that would be created by the `install` command.

```
Usage

./yb-install manifest --config-path=<path-to-configuration-file>
```

This does **not** create any resource, but only shows what would be created with the current configuration.

Additional Flags

- `--config-path/-c` : (Required) Path to config for installing Yellowbrick Data Warehouse instance

---

upgrade

Performs an Upgrade of the Infrastructure used for running a Yellowbrick Data Warehouse Instance, see Upgrading Platform for more info.

```
Usage

./yb-install upgrade --provider=<cloud_provider> --cluster=<cluster-id> --namespace=<namespace> [--dry-run]
```

Additional Flags

- `--provider/-p` : (Required) Path to file for provider config (see provider model)
- `--cluster/-c` : (Required) Kubernetes cluster name for your existing Installation(AKS, EKS or GKE name).
- `--namespace/-n` : (Required) Kubernetes namespace in which the Yellowbrick software was installed.
- `--dry-run/-d` : (Optional) Only show a dry run plan of what actions will take place in the upgrade(no resources are updated/created).

---

uninstall

Uninstalls a prior installation of Yellowbrick software. This subcommand, unless disabled, invokes a confirmation prompt before intiating the uninstall, which includes typing the name of database instance.

THIS IS A DESTRUCIVE AND IRREVERSIABLE OPERATION AND WILL LEAD TO DATA LOSS

```
Usage

./yb-install uninstall --provider=<cloud_provider> --cluster-id=<cluster-id> --namespace=<namespace> [--skip-prompt]

OR

./yb-install uninstall  --config-path=<path-to-configuration-file> [--skip-prompt]
```

This command has two possible of flag sets/usage.

- Use the first set of flags if the installation, even if partial, was able to reach the point where it created a Kubernetes Cluster.
- Use the second flag only if the installation was partial, and failed even before it could create a Kubernetes Cluster.

Additional Flags

- `--provider/-p` : Path to file for provider config (see provider model)
- `--cluster/-c` : Kubernetes cluster name for your existing Installation(AKS, EKS or GKE name).
- `--namespace/-n` : Kubernetes namespace in which the Yellowbrick software was installed.
- `--config-path` : Path to the configuration file used while installing Yellowbrick Data Warehouse instance. (Fail safe)
- `--skip-prompt/-s` : Boolean, Skips prompt confirmation before uninstall

---

push

Pushes Docker Images and Helm Charts for the current version of Yellowbrick Components. When the cloud provider is AWS, it also creates the respective ECR repos for artifacts.

```
Usage

./yb-install push --provider=<cloud_provider> --registry-id=<registry_id> --cluster-id=<cluster-id> --namespace=<namespace>

OR

./yb-install push  --provider=<cloud_provider> --registry-id=<registry_id>
```

INFO

The default `log-file` destination could be in use by the `yb-install interactive` command, and you may need to override it to a different location.

This command has two possible of flag sets/usage.

- The first set of flags can be used to push the artifacts and register the new version with the yellowbrick operator.
- The second set of flags only push the artifacts, and would be the best choice when doing a self managed install, when the yellowbrick operator is already not installed.

Additional Flags

- `--provider/-p` : Path to file for provider config (see provider model)
- `--cluster/-c` : Kubernetes cluster name for your existing Installation(AKS, EKS or GKE name).
- `--namespace/-n` : Kubernetes namespace in which the Yellowbrick software was installed.

- `--registry-id`: ID associated with registry in the given cloud provider. For AWS, this is treated as a prefix to ECR repos which are created. For azure/gcp, this is the name of the ACR/GCR resource.

get-assets

Outputs a json of assets bundled by the yellowbrick deployer.

```
Usage

./yb-install get-assets
```

This can be useful during a self managed install, and help in finding the version of helm charts/docker images. As an example, the following command can be used to find the helm chart version of `cert-manager`:

```
./yb-install get-assets &> ~/manifest.json # /opt/ybd is readonly
cat ~/manifest.json | jq -r '.containerImages[] | select(.repo == "cert-manager") | .tag'
```

JSON Configuration File

At the top level, the configuration model has the following structure.

```
{
    "provider": {},
    "instance": {},
    "kubernetes": {},
    "network": {},
    "access": {},
    "operator": {},
    "account": {},
    "registry": {},
    "dependencies": {}
}
```

`provider` model

This block of the configuration defines the Cloud provider information.

```
{
    "type": "Required, has to be one of `aws`, `azure` or `gcp`",
    "region": "Required, cloud specific region code",
    "gcpProjectId": "Required if type is `gcp`, the project id where the install will occur"
    "azureSubscriptionId": "Required if type is `azure`. The azure subscription ID",
    "azureResourceGroupName": "Required if type is `azure`. The resource group in which you want to install in",
    "azureLogAnalyticsWorkspaceId": "Optional, if type is `azure`. The id of an existing log analytics workspace."
}
```

`instance` model

This block is some basic info about your instance.

```
{
    "name": "Required, the name of your Instance`",
    "namespace": "Required, the kubernetes namespace in which the instance is created."
    "timeout": "Optional, string representation of the timeout for all instance and manager operations"
}
```

`kubernetes` model

This block of the configuration defines parameters related to the kubernetes cluster which will be created during the install.

```
{
    "name": "Defaults to the `instance.name` property of the config. Not overridable yet.",
    "kmsKeyId": "Optional, Full ID of the encryption key in the cloud provider KMS"
    "timeout": "Optional, string representation of the timeout for all kubernetes operations"
}
```

`network` model

This block of the configuration defines parameters related to the kubernetes cluster which will be created during the install.

```
{
    "cidr": "IPv4 range for the VPC/Network",
    "id": "ID associated with an existing network",
    "subnets": [], # Required, list of subnets, see more info below. AWS requires minimum 2 subnets across two different availabil
    "timeout": "Optional, string representation of the timeout for all network operations"
}
```

`subnet` model

```
{
    "cidr": "IPv4 range for the Subnet, container within the network.cidr configuration",
    "id": "ID associated with an existing subnet",
    "zone": "The availability zone for the given subnet."
}
```

`access` model

This defines the restriction to your database instance. Applied to Yellowbrick Manadger, Instance and Kubernets cluster

```
{
    "type": "Type of access for your environment, one of `public` or `private",
    "restrict": [] # List of Restrictions you want to add, see below
}
```

`restrict` model

```
{
    "type": "Type of restriction, one of `ip` or `cidr`",
    "ip": "IPv4 address if type specified as `ip`",
```

```
    "address": "IPv4 cidr range if type specified as `cidr`"
}
```

`operator` model

```
{
    "namespace": "Required, the kubernetes namespace in which the Yellowbrick Operator/Manager are created."
    "timeout": "Optional, string representation of the timeout for all operator operations"
}
```

`account` model

```
{
    "instanceAccountName": "Username for the default Instance admin user.",
    "instanceAccountPassword": "Password for the default Instance admin user."
}
```

`registry` model

```
{
    "timeout": "Optional, string representation of the timeout for all registry operations"
}
```

`dependencies` model

```
{
    "timeout": "Optional, string representation of the timeout for all third party operations"
}
```

Timeouts

Platforms: EE: All cloud platforms

Parent topic: CLI Install Instructions

The Deployer creates, or updates, different cloud resources. The duration of each operation depends on different factors like the resource type or the cloud provider and its current load in the selected region. Therefore the installation program polls for the completion of its action, regularly checking for progress. To avoid checking forever, different timeouts have been implemented.

The default timeout values have been designed for observed behaviours and should be large enough to allow operations to complete. However in exceptional situations is could be useful to increase those values. Customisation settings have been implemented in the installation program configuration to enable this. It is recommended to change the timeout values only when issues are encountered, and rely on the default settings as much as possible.

Timeout customisation and format

Customisation of the timeouts can be performed through specific fields in the installer configuration file, using a string format representing a duration. A duration string is a sequence of decimal numbers, each with optional fraction and a unit suffix, such as `300ms`, `1.5h` or `2h45m`. Valid time units are `ns`, `us` (or `µs`), `ms`, `s`, `m`, `h`. The location of each override option below is described as a JSON Path that can be altered in the configuration file, for example `network.timeout` corresponds to a JSON document such as:

```
{
    "network": {
        "timeout": "15m"
    }
}
```

List of Timeout Values and JSON Paths

Action	Default Timeout	Override Option
Creation or deletion of a VPC	10m	network.timeout
Creation or deletion of a registry	10m	registry.timeout
Upload of all the container images	1h	registry.timeout
Creation or deletion of a Kubernetes cluster	1h	kubernetes.timeout
Deployment of a third party dependency	5m	dependencies.timeout
Deployment of the Yellowbrick Operator	15m	operator.timeout
Deployment of the Yellowbrick Instance	15m	instance.timeout
Deployment of the Yellowbrick Manager	15m	instance.timeout

Public Install Instructions

Platforms: EE: All cloud platforms

Parent topic: Installing

In this section:

This section contains instructions for installing Yellowbrick into your cloud account using the Yellowbrick Deployer.

AWS Public Installation Instructions

Platforms: EE: AWS cloud

Parent topic: Public Install Instructions

Prerequisites

Running the Deployer on AWS requires permission to create a virtual network, a security group, an EC2 instance and custom IAM roles. Although not mandatory, it is highly recommended to use a dedicated AWS account to simplify permission setup and ensure resource isolation.

The Deployer can be launched into the default VPC (if present) or one of your choice. You'll need to be able to reach the VPC from a web browser to use the Deployer, for example by providing an internet gateway. If you want to use a default VPC and it's been deleted, it can be recreated by running the following command:

```bash
% aws ec2 create-default-vpc
```

It is possible to run the interactive Deployer directly from the AWS Management Console. If you want to use the command line interface, you will need to be logged in with the AWS CLI.

Quota

The following **minimum** quota limits are required to successfully run the Deployer.

Quota name	Value
Running On-Demand Standard (A, C, D, H, I, M, R, T, Z) instances	46

Launching the Deployer

NOTE

By installing Yellowbrick Enterprise Edition software into your Cloud Account, you agree to Yellowbrick's Enterprise Edition EULA.

The Deployer is launched via an AWS CloudFormation template.

One-Click Deploy on AWS

Click the button below to launch Yellowbrick Deployer via the AWS console.

Launch on AWS → Region: US East (N. Virginia) - us-east-1

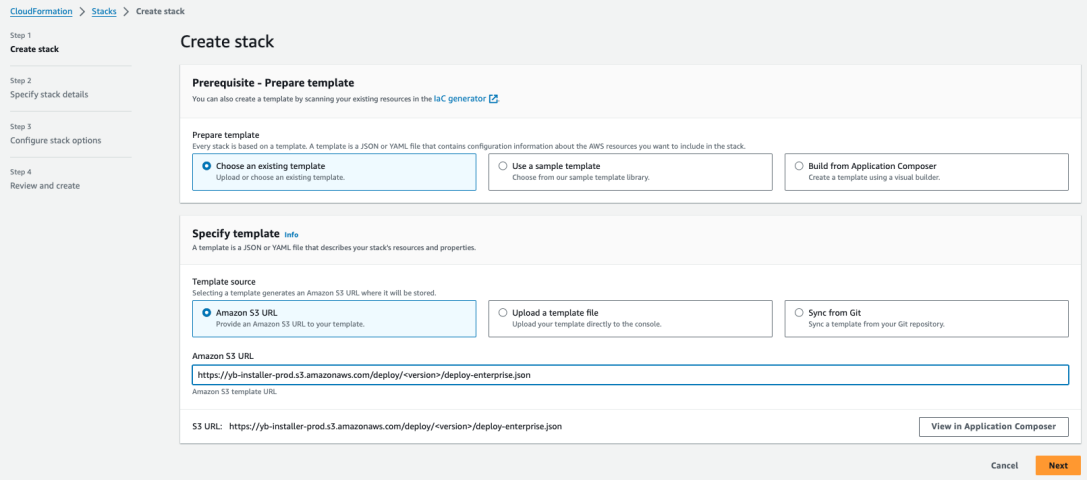
Deploying via an AWS CloudFormation Template

The URL of the template is different for commercial vs. government cloud regions:

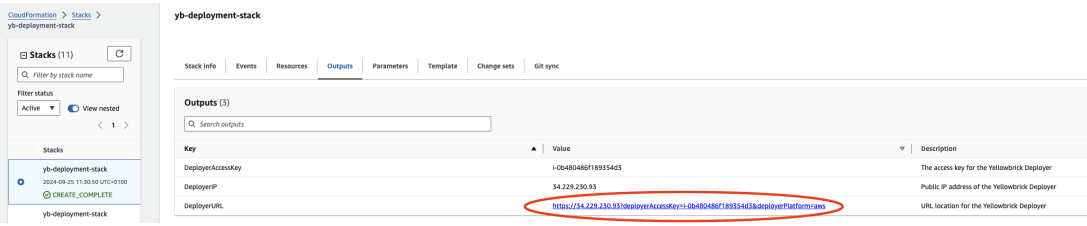
- **Commercial Cloud**: `https://yb-installer-prod.s3.amazonaws.com/deploy/7.3.0-74895.181ea900/deploy-enterprise.json`

- **Government Cloud**: `https://yb-installer-prod.s3.us-gov-west-1.amazonaws.com/deploy/7.3.0-74895.181ea900/deploy-enterprise.json`

You must create a new stack based on the published Deployer template:



Fill out the different parameters and follow the UI until you trigger the stack creation, then wait until it finishes. You will find the URL of the interactive Deployer in the **output** tab:



The SSH keypair for the EC2 instance will be randomly generated. If access to the instance is needed, the private key can be found in AWS Systems Manager Parameter Store under the value `yellowbrick-deployment-keypair`.

Deploying via AWS Command Line Interface with CloudFormation Template

You can launch the deployer in your aws account using the `aws` command line interface. Use the commands below to launch the stack `yb-deployer` and wait for its completion.

```bash
% aws cloudformation create-stack \
  --stack-name yb-deployer \
  --template-url https://yb-installer-prod.s3.amazonaws.com/deploy/7.3.0-74895.181ea900/deploy-enterprise.json \
  --capabilities CAPABILITY_NAMED_IAM \
  --parameters ParameterKey=AccessCIDR,ParameterValue=<access-cidr> ParameterKey=VPC,ParameterValue=<vpc-id> ParameterKey=Subnet,P
% aws cloudformation wait stack-create-complete --stack-name yb-deployer
```

Replace the values for `VPC` and `Subnet` accordingly.

After launching the deployer, the output values can be displayed with this command:

```bash
  % aws cloudformation describe-stacks --stack-name yb-deployer --query "Stacks[0].Outputs"
```

The output will look similar to the following:

```json
[
    {
        "OutputKey": "DeployerURL",
        "OutputValue": "https://52.45.231.38?deployerAccessKey=i-0dfd83899be89af72&deployerPlatform=aws",
        "Description": "URL location for the Yellowbrick Deployer"
    },
    {
        "OutputKey": "DeployerIP",
        "OutputValue": "52.45.231.38",
        "Description": "Public IP address of the Yellowbrick Deployer"
    },
    {
        "OutputKey": "DeployerAccessKey",
        "OutputValue": "i-0dfd83899be89af72",
        "Description": "The access key for the Yellowbrick Deployer"
    }
]
```

You may launch the deployer web user interface with the url given for `deployer`, or connect via ssh with the `deployerIP` with user `ubuntu`:

```bash
% ssh ubuntu@52.45.231.38

ubuntu@i-0dfd83899be89af72:~$ ls -l /opt/ybd/
total 276152
-rwxr-xr-x 1 root       root             538 May  9 12:12 bootstrap.sh
-rw-r--r-- 1 yb-install yb-install       224 May  8 23:31 environment
-rwxr-xr-x 1 yb-install yb-install       127 May  8 23:31 get-access-key
drwxr-xr-x 2 yb-install yb-install      4096 May  8 23:31 helm-charts
drwxr-xr-x 4 yb-install yb-install      4096 May  8 23:29 image
-rwxr-xr-x 1 yb-install yb-install 282751272 May  8 23:29 yb-install
```

Cleanup

To remove the Deployer, navigate to the CloudFormation service in the AWS Management Console, and select the Stack used to install the Deployer. Delete the Stack in the console. This will remove the EC2 instance and associated resources.

Azure Public Installation Instructions

Platforms: EE: Azure cloud

Parent topic: Public Install Instructions

Prerequisites

On Azure, you need a resource group and the ability to manage IAM access for it, as you must have the permissions to create and assign roles within that resource group.

It is recommended to use a dedicated resource group to isolate resources and simplify access management, such as by having owner permissions for that resource group.

You can use the Azure Portal to launch the interactive Deployer. However you must also be logged in to the Azure CLI in order to create the template spec as outlined below. Also, you'll need to make sure you can access the network the deployer is running in from a web browser.

Quota

The following **minimum** Compute quota limits are required to successfully run the Deployer.

Quota name	Value
Standard DDSv5 Family vCPUs	16
Standard Dv5 Family vCPUs	16
Standard LSv3 Family vCPUs	32
Total Regional vCPUs	60

Launching the Deployer

NOTE

By installing Yellowbrick Enterprise Edition software into your Cloud Account, you agree to Yellowbrick's Enterprise Edition EULA.

The Azure Deployer is based on Azure Resource Manager.

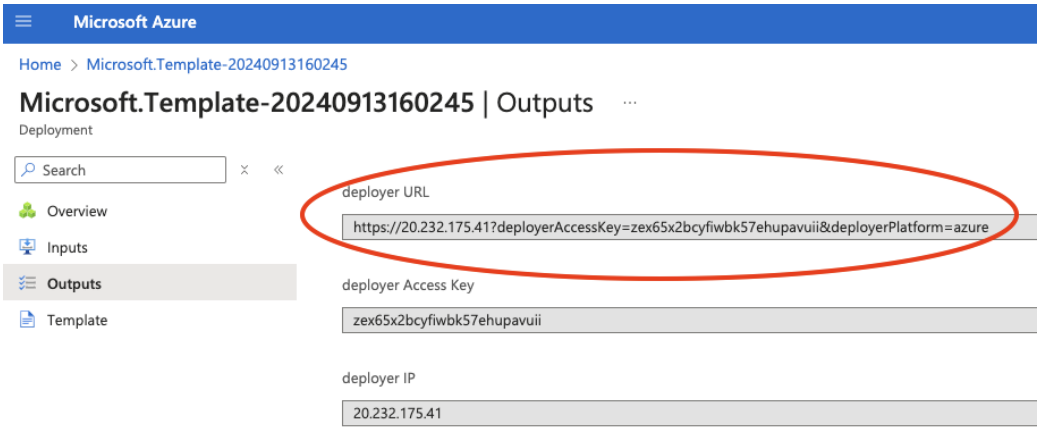
One-Click Deploy on Azure

Click the button below to launch Yellowbrick Deployer via the Azure portal.

Launch on Azure →

Deployer Interactive Installer

Once the deployment is completed, you will find the URL of the interactive Deployer in the **output** panel:



Deploying via Azure Command Line Interface (az cli)

You can launch the deployer in your azure subscription using the `az` command line interface. Locate a public key you will use for ssh access, and launch the command below:

```bash
% az stack group create \
  --resource-group <resource group name> \
  --template-uri https://ybinstallerprod.blob.core.windows.net/7-3-0-74895-181ea900/deployment.json
```

When prompted, provide your ssh public key. The public key must begin with the string `ssh-rsa` or `ssh-ed25519`. You may also pass your public key as an extra parameter this way:

```bash
% az stack group create \
  --resource-group <resource group name> \
  --template-uri https://ybinstallerprod.blob.core.windows.net/7-3-0-74895-181ea900/deployment.json \
  --parameters sshPublicKey='ssh-rsa ....<your-public-key>'
```

When the deployment is finished, a JSON document is presented which will contain outputs you will use to launch the deployer user interface or connect to the ip address via ssh.

The outputs will contain a fragment which looks like this:

```json
  "outputs": {
    "deployer": {
      "type": "String",
      "value": "https://20.1.152.227?deployerAccessKey=xu7zkdfz5k5qazbfizu4l2g3po&deployerPlatform=azure"
    },
    "deployer Access Key": {
      "type": "String",
      "value": "xu7zkdfz5k5qazbfizu4l2g3po"
    },
    "deployer IP": {
      "type": "String",
      "value": "20.1.152.227"
    },
    "vm Name": {
      "type": "String",
      "value": "yb-deployer-ycxydoqcmo3ji"
    }
  },
```

You may launch the deployer web user interface with the url given for `deployer`, or connect via ssh with the `deployer IP` with user `azureuser`:

```bash
% ssh azureuser@20.1.152.227

azureuser@yb-deployer-ycxydoqcmo3ji:~$ ls -l /opt/ybd/
total 276152
-rwxr-xr-x 1 root       root            538 May  9 12:12 bootstrap.sh
-rw-r--r-- 1 yb-install yb-install       224 May  8 23:31 environment
-rwxr-xr-x 1 yb-install yb-install       127 May  8 23:31 get-access-key
drwxr-xr-x 2 yb-install yb-install      4096 May  8 23:31 helm-charts
drwxr-xr-x 4 yb-install yb-install      4096 May  8 23:29 image
-rwxr-xr-x 1 yb-install yb-install 282751272 May  8 23:29 yb-install
```

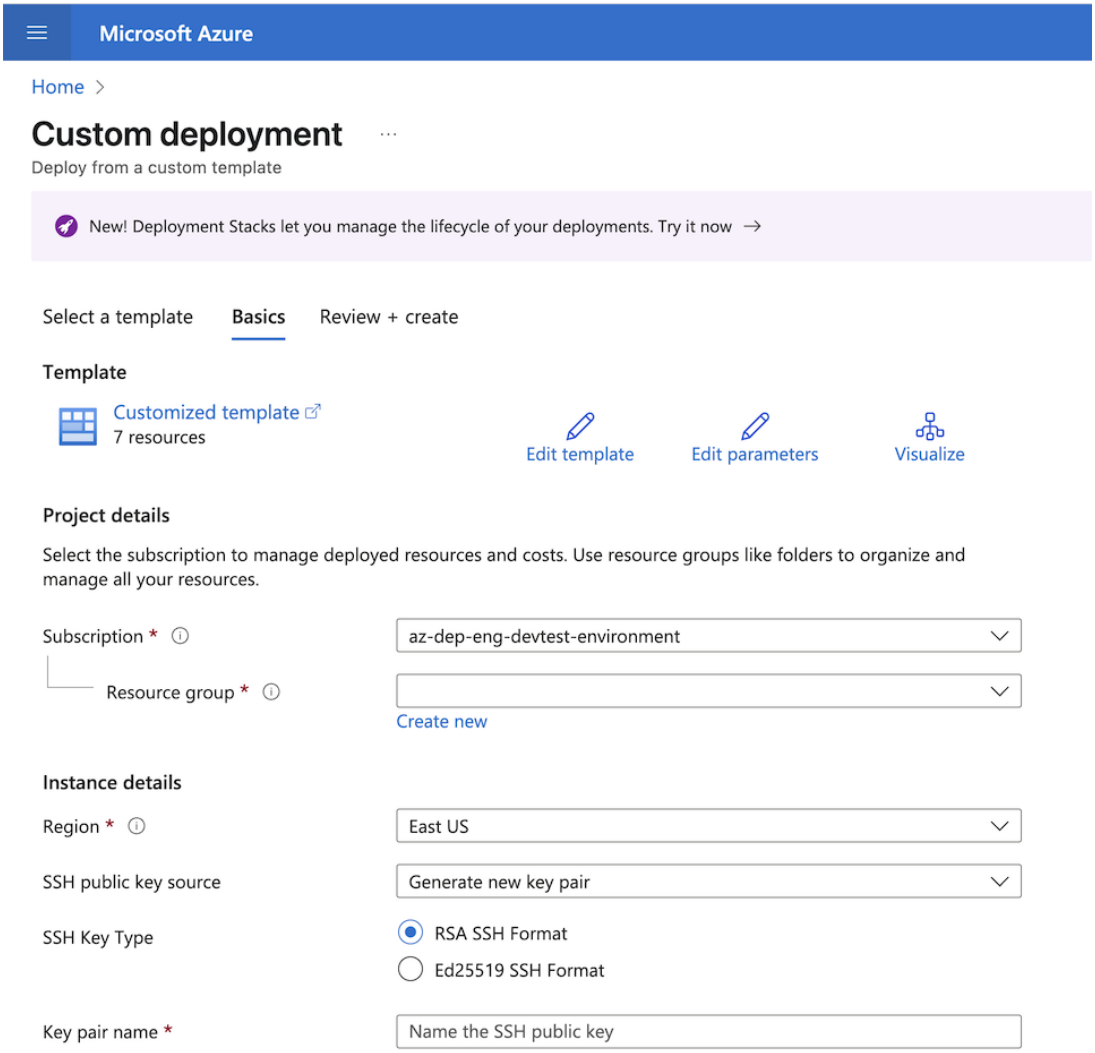
Deploying via an Azure Portal

The deployer template can be downloaded here: Deployer Template

To use this template in the Azure Portal:

1. Download the template
2. Visit Azure Portal Custom Deployment
3. Choose the option "Build your own template in the editor"
4. Load the deployer template you downloaded to your local machine, and press Save

The custom deployment will appear with instructions to choose or create an ssh public key used to launch the deployer.



Cleanup

To remove the Deployer, navigate to the Azure Deployment Stacks service in the Azure portal under the resource group you used to create the Deployer. Select the deployment stack you created and named to install the Deployer and delete the deployment stack. For the "Managed Resources" options shown, choose the option to "Delete resources" vs. "Detach resources", which will remove the Azure VM instance and associated resources for the Deployer.

GCP Public Installation Instructions

Prerequisites

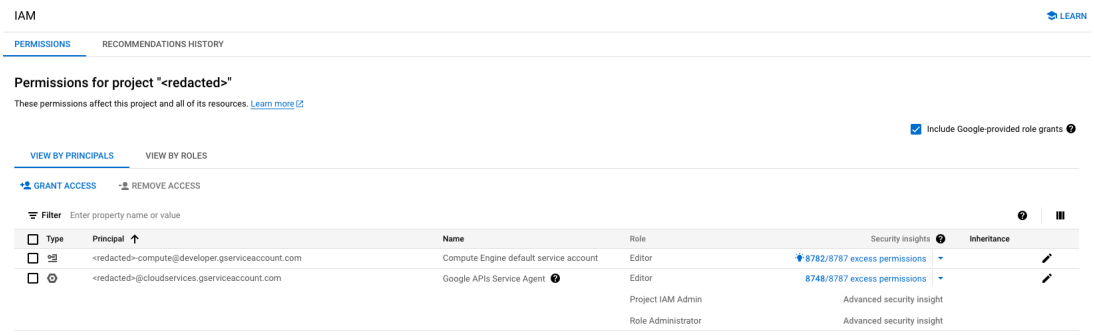
On GCP, you need a project and the permissions to create and assign IAM roles. While not required, it is recommended to use a dedicated project for resource isolation. To run the Deployer you will need to be logged with the `gcloud` CLI. The following APIs must be enabled:

- Google Cloud APIs
- Cloud Resource Manager API
- Artifact Registry API
- Cloud Autoscaling API
- Compute Engine API
- Kubernetes Engine API
- Identity and Access Management (IAM) API
- Cloud Storage API

You'll need to make sure you can access the network the deployer is running in from a web browser. You will need to give the `Project IAM Admin` and `Role Administrator` permissions to the `Google APIs Service Agent` service account on the IAM portal. That service account is the one with the following email format:

```txt
<project-number>@cloudservices.gserviceaccount.com
```

It may not appear by default in the IAM page; the `Include Google-provided role grants` checkbox must be checked to show it:



Quota

The following **minimum** quota limits are required to successfully run the Deployer.

Quota name	Value
C3 CPUs	82

Quota name	Value
Local SSD per VM family (GB), Family C3	7500Gb

Launching the Deployer

NOTE

By installing Yellowbrick Enterprise Edition software into your Cloud Account, you agree to Yellowbrick's Enterprise Edition EULA.

The GCP Deployer does not have any graphical interface as it relies on Google Deployment Manager. However creating the Deployer instance is relatively easy with the use of the gcloud CLI.

First your must retrieve the archive containing all the input files and decompress it:

```bash
% gcloud storage cp gs://yb-installer-prod/7.3.0-74895.181ea900/installer.tar.gz installer.tar.gz

% tar -xzvf installer.tar.gz
```

This should have created 4 files, only the `installer.yaml` will be used as it references the others. Run this command:

```bash
% gcloud deployment-manager deployments create yb-enterprise-deployer --config installer.yaml
```

Once this completes, the output section will contain the URL of the Deployer.

Cleanup

To remove the Deployer you can simply use the Google Deployment Manager with the `delete` operation:

```bash
% gcloud deployment-manager deployments delete yb-enterprise-deployer
```

Public Uninstall Instructions

Platforms: EE: All cloud platforms

Parent topic: Installing

This section contains instructions for uninstalling Yellowbrick from your cloud account using the Yellowbrick Deployer.

To uninstall Yellowbrick:

1. Navigate to the "(2) Deployer" option on the left side menu of the Deployer Web UI and select the "Uninstall" option
2. Select "Next"
3. Choose the cloud provider, cloud region and instance name to uninstall
4. Note that there are Advanced options complete, if you have customized the Kubernetes cluster name or namespace
5. Select "Next". The Deployer will then configure the uninstallation process
6. Finally, select "Uninstall"

The Deployer will display the progress as it uninstalls the Yellowbrick components and infrastructure.

For information on how to uninstall using the command line interface, see the CLI Reference Guide.

Instructions: Private Install

Platforms: EE: All cloud platforms

Parent topic: Installing

In this section:

When the Yellowbrick Deployer is launched from a public network, the infrastructure services it creates (such as networks and the managed Kubernetes service) must themselves be public so that it can reach them. It can't create a completely private network, disconnected from the internet, because then by definition it couldn't route to it.

These "public" installations still make sure that Yellowbrick Kubernetes workers are not routable from public networks, and access to object store is via private endpoints. However, if you need to make sure the platform in its entirety along with all infrastructure (such as gateways, load balancers or the Kubernetes API) are not public routable, you'll need to perform a fully private installation. This is a very common deployment scenario and one of the key reasons for choosing Yellowbrick.

To create a fully private installation, you must first design and configure the network architecture. A typical architecture for private installations follows a **hub-and-spoke** model with separate VPCs, ensuring controlled access and segmentation of resources. This architecture must be set up before installing Yellowbrick. The VPC Peering Concepts guide provides more detail on configuring a hub-and-spoke network.

Once the network is in place, you must build the necessary managed Kubernetes service, along with any private paths for accessing other cloud services such as object storage or cloud services APIs. You can integrate the new network into your corporate network as necessary. The infrastructure you create must be tagged so that the Deployer can discover it.

You then launch the Deployer from within that newly created private network and complete the installation using either the web application or the CLI.

Instructions on how to accomplish this across all public cloud providers, along with links to working Terraform examples, are included in this section.

AWS Private Installation Instructions

Platforms: EE: AWS cloud

Parent topic: Private Install Instructions

Overview

This document provides a comprehensive guide for installing Yellowbrick in a private Amazon Web Services (AWS) environment using a bring-your-own-VPC approach. This guide assumes that you will configure the necessary infrastructure within AWS, including the Virtual Private Cloud (VPC), to meet the requirements of a private deployment that creates no public IP addresses. This reference serves as a baseline but can be customized to fit your specific environment.

NOTE

A private installation requires advanced knowledge of cloud infrastructure and networking. Please contact your internal IT and operations teams as needed, or reach out to Yellowbrick Support for assistance.

Understanding a Private EKS Cluster

A private Amazon Elastic Kubernetes Service (EKS) cluster is configured to restrict access to both the control plane and compute nodes within a Virtual Private Cloud (VPC), with no public internet exposure. The key characteristics of a private EKS cluster include:

- **Private EKS API:** The EKS Kubernetes API is accessed via a private endpoint within the VPC, with no public IP address assigned.
- **Private Compute Nodes:** Compute nodes are deployed in private subnets without public IP addresses, ensuring that they are only accessible within the VPC.
- **VPC Endpoints for AWS Services:** Use of VPC interface and gateway endpoints (VPCE) to securely connect to AWS services like S3, EC2, and ECR, without requiring internet access.
- **Outbound Connectivity:** Outbound traffic can be managed through a NAT Gateway, VPC interface endpoints, or a custom gateway allowing secure access to AWS services and external resources.

For more details, refer to the AWS documentation on private EKS clusters.

Infrastructure Preparation

Creating the VPC Network

Before deploying Yellowbrick, create a VPC network that satisfies the requirements for a private installation. The network should be designed to support private communication between EKS nodes and AWS services while minimizing cross-AZ data transfer costs.

1. **VPC Configuration:**
   - Create a VPC with an appropriate address space. The AWS VPC CNI for EKS will allocate an IP address per Kubernetes pod, and thus requires a large number of IP addresses in the VPC. The VPC should be sufficiently large to account for all subnets, including a primary subnet of recommended size /19 which will host the EKS nodes and EKS pod workloads. Smaller sized subnets can be utilized for installations with smaller Yellowbrick cluster sizes, though it is not recommended to created subnets smaller than /24.

- Configure the VPC to allow access to AWS service APIs. This may include:
  - **Internet Gateway:** Required if you plan to use a NAT Gateway for outbound internet access.
  - **NAT Gateway:** Deploy in a small, public subnet to allow outbound access to AWS services.
  - **Route Tables:** Associate route tables with the subnets to manage traffic routing within the VPC. Configure ACL rules as appropriate.
  - **VPC Endpoints:** Configure AWS VPC Endpoints for internal communication with AWS services. The Yellowbrick installation will always create a VPC Gateway Endpoint to S3 automatically. It is not recommended to use Amazon S3 interface endpoints (PrivateLink) due to additional data transfer costs. VPC interface endpoints to other services do not have the same concern. For more information on a list of AWS services, see Access to AWS Service APIs

2. **Subnet Configuration:**
   - **Private Subnets:** Create two private subnets, each in a different availability zone (AZ). These subnets should not assign public IP addresses and will host the EKS nodes.
   - **Primary Subnet:** An EKS installation requires at least two subnets. The Deployer will place the majority of the EKS nodes in a single primary subnet to minimize cross-AZ data transfer costs. This primary subnet should have a `/19` CIDR block to support the anticipated pod IP address requirements. When limited address space is available, bias the majority of addresses to the primary subnet. You must tag this subnet with `primary: true` to properly be identified by the Deployer.
3. **Tagging:**
   - Apply the additional tag `cluster_yellowbrick_io_owner = yb-install` to all manually created network and EKS infrastructure. This allows the Deployer to identify the correct components during deployment.

For examples of setting up a customized private infrastructure, refer to the Terraform reference architecture.

Access to AWS Service APIs

The Deployer and the Yellowbrick Operator require access to several AWS services. You can choose between two approaches:

1. **Outbound NAT Gateway:**
   - Create a third small public subnet to host a NAT Gateway, allowing outbound access to AWS APIs through the Internet Gateway. This method is simple and effective for most scenarios.
2. **AWS VPC Interface Endpoints:**
   - Use AWS PrivateLink to create Interface Endpoints for the following services:
     - `autoscaling`
     - `ec2`
     - `ecr.api`
     - `ecr.dkr`
     - `eks`
     - `elasticloadbalancing`
     - `logs`
     - `sts`

The Deployer will create a VPC Gateway Endpoint to S3, so you don't need to do that. Note that if using a custom DNS domain, be aware that there is no Interface Endpoint for Route 53, which may necessitate the use of a NAT Gateway for DNS management from within the EKS cluster.

Executing the Deployer

The Deployer will use an EC2 instance within the same VPC where Yellowbrick will be deployed. This instance will require access to the EKS cluster via the private endpoint and will manage the deployment process, including the creation of additional resources, ECR uploads, and Kubernetes workload installations.

Installation Process

> **NOTE**
>
> By installing Yellowbrick Enterprise Edition software into your Cloud Account, you agree to Yellowbrick's Enterprise Edition EULA.

1. Subscribe to the Yellowbrick Data Warehouse Enterprise Edition AMI.
2. Create the base infrastructure as outlined in this deployment guide.

Proceed to follow either the instructions using CloudFormation or launching the the Yellowbrick Deployer custom AMI manually. Once one of those two options is complete, access the Deployer from a web browser to continue with the installation.

Option 1: CloudFormation

1. Launch the Yellowbrick Deployer CloudFormation template.
2. Along with a stack name and basic parameters of the Deployer, the CloudFormation parameters will offer a drop-down choice for a VPC and subnet to launch the Deployer into. It is very important to select the correct VPC and subnet from the previously created infrastructure. This will ensure when the Deployer creates the EKS cluster, it will have has access to EKS cluster's API via the private endpoint during installation.
3. After setting all remaining CloudFormation parameters as appropriate for your installation, proceed to create the stack.
4. Upon completion of the stack creation, locate the Deployer URL on the outputs of the CloudFormation stack. Navigate to this URL to continue with the installation using the Deployer. Please note this instance may not be accessible from the internet, and you may need to perform additional steps to ensure HTTPS access.

Option 2: Custom AMI

1. Deploy an EC2 instance using the AMI in the VPC and subnet from the previously created infrastructure. This will ensure when the Deployer creates the EKS cluster, it will have has access to EKS cluster's API via the private endpoint during installation. Please note this instance may not be accessible from the internet, and you may need to perform additional steps to gain SSH and HTTPS access.
2. **The instance must be launched with an instance profile that assumes an IAM role containing** the policies listed here. This IAM policy uses attribute-based access control (ABAC). Please ensure that the IAM role includes the following tags:
   - `cluster_yellowbrick_io_owner = yb-install`
   - `cluster_yellowbrick_io_creator = yb-install`

   These tags are essential for the proper functioning of access control, enabling the instance to manage resources securely within the Yellowbrick infrastructure.
3. Create an SSH connection to the instance as the `ubuntu` user using the SSH keypair specified during the launch of the AMI.
4. The EC2 instance is configured to automatically start the interactive web UI for the deployment process. Accessing this UI requires an access key that can be retrieved by executing `/opt/ybd/get-access-key` from the remote shell.
5. From a web browser, access the EC2 instance over HTTPS port 443. Use the DNS or IP address of the EC2 instance as the hostname. Web traffic will be encrypted over TLS and a self-signed certificate will be used.
6. When accessing the Yellowbrick Deployer UI, you will need to provide the Deployer access key retrieved from the previous step.

Accessing Yellowbrick Deployer UI

1. With a web browser, access the Deployer by following the instructions given in each previous method.
2. On the "Restrict Access" step, indicate this is a private installation and click Next.

3. On the "Network" step, choose the correct VPC network previously created.
4. Continue with the deployment process as normal. The Deployer will configure the cluster, set up necessary IAM roles, create additional node groups, and deploy the Yellowbrick Operator and related workloads.

Terraform Reference

For a Terraform reference of this infrastructure, please see deployer-contrib.

Conclusion

By following this guide, you can establish a private IP address environment for Yellowbrick within AWS and tailor the infrastructure to your specific requirements.

Azure Private Installation Instructions

Platforms: EE: Azure cloud

Parent topic: Private Install Instructions

Overview

This document provides a comprehensive guide for installing the Yellowbrick Data Warehouse in a private Azure Kubernetes Service (AKS) cluster. Due to the unique requirements of enterprise environments, this guide follows a bring-your-own-Kubernetes approach, allowing you to configure the infrastructure to align with your specific needs. The installation is then customized according to the requirements of the Yellowbrick product. The reference architecture provided serves as a baseline for deployment but can be adapted to fit your enterprise's existing infrastructure and policies.

NOTE

A private installation requires advanced knowledge of cloud infrastructure and networking. Please contact your internal IT and operations teams as needed, or reach out to Yellowbrick Support for assistance.

Understanding a Private AKS Cluster

A private Azure Kubernetes Service (AKS) cluster is an AKS configuration where the API server endpoint is accessible only within a private network, rather than over the public internet. This enhances security by ensuring that the cluster is isolated from public exposure. The key characteristics of a private AKS cluster include:

- **Private Nodes:** Nodes are deployed in a VirtualMachineScaleSet (VMSS) without public IP addresses, ensuring that they are inaccessible from outside the virtual network.
- **Private Endpoint for AKS API:** The AKS API server is accessed via a private endpoint within a virtual network, ensuring that only resources within that network can communicate with it.
- **Private Access to Azure Container Registry (ACR):** The AKS cluster pulls images from ACR using a private endpoint, preventing public access.
- **Customizable Egress Configuration:** Users can define specific outbound connectivity through a user-defined outbound type, typically managed through Azure Firewall or Network Security Groups (NSGs).

For more details, refer to Microsoft's documentation on private AKS clusters.

Infrastructure Preparation

Creating the Base Infrastructure

Before beginning the installation process, you must create the foundational infrastructure within Azure. Below is a suggested example configuration, but you may customize it according to your organization's needs:

1. **Resource Group:**
   - Create a dedicated resource group to logically contain all resources related to the Yellowbrick deployment.
2. **Networking:**
   - **Virtual Network:** Create a /16 virtual network to provide sufficient IP address space.
   - **Subnets:**

- `/22 Subnet for Yellowbrick Deployment:` Ensure this subnet includes a service endpoint for Microsoft.Storage.
- `/26 Subnet for Azure Firewall:` Dedicated for managing network security.
- `/26 Subnet for Azure Firewall Management:` Separate management plane.

3. **Azure Firewall:**

- Deploy an Azure Firewall instance with Layer 7 (application-level) rules to control outbound traffic. Ensure the firewall allows traffic to the required egress endpoints on ports 80 and 443.

4. **Route Table:**

- Associate a route table with the Yellowbrick subnet that directs all outbound traffic through the Azure Firewall.

5. **Azure Container Registry (ACR):**

- Create an ACR instance with a private endpoint. The naming convention should follow the format `${instance_name}${SHA1(resource_group_full_id)}` for consistency across deployments.

6. **Private DNS Zones:**

- Create private DNS zones to resolve the private endpoints for both ACR and AKS within your virtual network. Refer to the Azure private cluster guide for more details on AKS private DNS zones.

7. **Tagging:**

- Apply the additional tag `cluster_yellowbrick_io_owner = yb-install` to all manually created infrastructure. This allows the Deployer to identify the correct components during deployment.

Firewall Egress Rules

To ensure proper communication between the private AKS cluster and Azure services, specific egress rules must be configured. These rules allow the cluster to access required Azure resources without exposing the cluster to the public internet.

The following endpoints must be allowed through your firewall:

- `*.data.mcr.microsoft.com` - Used for pulling container images.
- `*.hcp.[azure_location].azmk8s.io` - For AKS management.
- `acs-mirror.azureedge.net` - Container image distribution.
- `login.microsoftonline.com` - Authentication services.
- `management.azure.com` - Azure Resource Manager.
- `mcr-0001.mcr-msedge.net` and `mcr.microsoft.com` - Additional image repositories.

Refer to the official outbound rules guide for further details.

Deploying the Private AKS Cluster

When deploying the AKS cluster, ensure the following configurations:

1. **Cluster Configuration:**

- **Kubernetes Version:** Must be 1.30 or later.
- **SKU Tier:** Standard.
- **Automatic Upgrade**: Upgrade must be disabled.
- **Node Security Channel Type**: Node security channel type must be disabled.
- **Private Endpoint:** Enable private endpoint and disable the public FQDN for the API server.

- **Identity Management:** Enable local accounts, workload identity, and Azure RBAC.
- **Azure Policy:** Enforce Azure Policy for resource governance.
- **Networking:** Network configuration must use kubenet.

2. **System Node Pool:**

- **Type:** VirtualMachineScaleSets.
- **Autoscaling:** Enabled.
- **VM Size:** Standard_D4ds_v5.
- **Disk Size:** 128 GB.
- **Operating System:** Ubuntu.
- **Networking:** No public IP addresses assigned to nodes.
- **Node Labels:** Ensure the appropriate labels are applied:
  - `cluster.yellowbrick.io/node_type: yb-operator`
  - `cluster.yellowbrick.io/hardware_type: Standard_D4ds_v5`

3. **Managed Identity:**

- The AKS cluster should be configured with a user-provided managed identity, which requires the following roles:
  - **Private DNS Zone Contributor:** Scoped to the private DNS zones created earlier.
  - **Network Contributor:** For the virtual network and route table, allowing the cluster to manage the subnet and routes.

Virtual Machine Deployment

Deploy a virtual machine within the Yellowbrick installation subnet:

1. **VM Source:** Launch the VM using the Yellowbrick-provided image from the Azure Marketplace.
2. **Managed Identity:** Assign the necessary roles to the VM's managed identity according to the `{permissions.json}` file provided by Yellowbrick. This step is crucial to ensure that the VM has the appropriate permissions to manage the installation and interact with the Azure resources.

Installation Process

> **NOTE**
>
> By installing Yellowbrick Enterprise Edition software into your Cloud Account, you agree to Yellowbrick's Enterprise Edition EULA.

1. Subscribe to the Yellowbrick Data Warehouse Enterprise Edition image in the Azure Marketplace.
2. Create the base infrastructure as outlined in this deployment guide.
3. Launch an Azure VM using the Yellowbrick image in the target VNet of the deployment. **Assign the** necessary roles listed here **to the VM's managed identity**. As this VM will not be accessible from the internet, you may need to perform additional steps to ensure SSH and HTTPS access.
4. Create an SSH connection to the VM as the `ybdadmin` user using the SSH key pair specified during the launch.
5. The VM is configured to automatically start the interactive web UI for the deployment process. Retrieve the access key by executing `/opt/ybd/get-access-key` from the remote shell.
6. From a web browser, access the VM over HTTPS on port 443 using the VM's private IP address or DNS name. The connection will be encrypted with a self-signed certificate.
7. Enter the access key from the previous step to proceed with the installation process.

8. During the installation, specify that this is a private installation and provide the name of the AKS cluster previously created. The existence of the EKS cluster will be validated, and the network configuration will be shown. Please verify those values are correct.

9. The Deployer will complete the deployment by configuring the cluster, assigning necessary Azure RBAC roles, creating additional node pools, and deploying the Yellowbrick Operator and related workloads.

Terraform Reference

For a Terraform reference of this infrastructure, please see deployer-contrib.

Conclusion

By following this guide, you can establish a secure environment for the Yellowbrick Data Warehouse within Azure.

GCP Private Installation Instructions

Platforms: EE: GCP cloud

Parent topic: Private Install Instructions

Overview

This document outlines the process for installing Yellowbrick on Google Cloud Platform (GCP) using a private Google Kubernetes Engine (GKE) cluster. GCP offers flexibility for deploying infrastructure with varying levels of privacy and customization. This guide supports a bring-your-own-VPC approach, allowing customers to create custom networking configurations that meet their specific requirements, while still leveraging the Yellowbrick installer to manage the deployment process.

NOTE

A private installation requires advanced knowledge of cloud infrastructure and networking. Please contact your internal IT and operations teams as needed, or reach out to Yellowbrick Support for assistance.

Understanding a Private GKE Cluster

In GCP, a private GKE cluster is a Kubernetes cluster where the control plane (master) and nodes have restricted access, with no public IP addresses assigned to them. The key features of a private GKE cluster include:

- **Private Nodes:** Nodes are deployed without public IP addresses, ensuring that they are only accessible from within the VPC.
- **Private Control Plane (Master):** The GKE control plane is accessed through internal IP addresses within the VPC, with no public endpoint.
- **Private IP Address Google Services Access:** For private communication with Google services such as Google Cloud Storage, private IP address services access can be configured.

For more details, refer to the Google Cloud documentation on private GKE clusters.

Infrastructure Preparation

Creating the VPC Network

Before deploying Yellowbrick, create a VPC network that satisfies the requirements for a private installation. This network will support the private cluster and provide secure communication with Google Cloud services.

1. **VPC Setup:**
   - Create a VPC to accommodate your cluster and future expansion.
2. **Subnet Configuration:**
   - Configure a subnet for GKE nodes within the VPC, ensuring it is large enough to handle pod IP address allocations. A `/22` subnet is recommended to allow for growth, but smaller subnets can be used if you know the specific scale of your installation.
   - Ensure that all nodes reside within this subnet to avoid possible cross-zone communication costs.
   - **Private Services Access:** Enable private services access to connect the subnet to Google-managed services without exposing data to the public internet.
3. **Firewall Rules:**

- Create firewall rules to allow internal communication between GKE nodes and necessary Google Cloud services. For private installations, ensure that no public ingress is allowed, and restrict egress as needed. Yellowbrick does not require egress beyond GKE and Google services.

Executing the Deployer

The Deployer will use a Compute Engine VM instance within the same private subnet where Yellowbrick will be deployed. This instance will create a GKE cluster and requires access to the GKE cluster via private endpoints and to Google Cloud services through private services access. This Deployer will then complete the deployment process, including the creation of any additional resources and the installation of Kubernetes workloads.

Installation Process

> **NOTE**
>
> By installing Yellowbrick Enterprise Edition software into your Cloud Account, you agree to Yellowbrick's Enterprise Edition EULA.

1. Subscribe to the Yellowbrick Data Warehouse Enterprise Edition image in the Google Cloud Marketplace.
2. Create the base infrastructure as outlined in this deployment guide.
3. Launch a Google Compute Engine VM using the Yellowbrick image in the target subnet within the VPC. **Assign the** necessary roles listed here **to the VM's service account**. As this VM will not be accessible from the internet, you may need to perform additional steps to ensure SSH and HTTPS access.
4. Create an SSH connection to the VM as the `ubuntu` user using the SSH key pair specified during the launch.
5. The VM is configured to automatically start the interactive web UI for the deployment process. Accessing this UI requires an access key that can be retrieved by executing `/opt/ybd/get-access-key` from the remote shell.
6. From a web browser, access the VM instance over HTTPS port 443. Use the DNS or IP address of the VM instance as the hostname. Web traffic will be encrypted over TLS and a self-signed certificate will be used.
7. When accessing the Yellowbrick Deployer UI, you will need to provide the Deployer access key retrieved from the previous step.

Accessing Yellowbrick Deployer UI

1. With a web browser, access the Deployer by following the instructions given in each previous method.
2. On the "Restrict Access" step, indicate this is a private installation and click Next.
3. On the "Network" step, choose the correct VPC network previously created.
4. Continue with the deployment process as normal. The Deployer will configure the cluster, set up necessary IAM roles, create additional node pools, and deploy the Yellowbrick Operator and related workloads.

Terraform Reference

For a Terraform reference of this infrastructure, please see deployer-contrib.

Conclusion

By following this guide, you can establish a private IP address environment for Yellowbrick within GCP and tailor the infrastructure to your specific requirements.

Instructions: Self-Managed Install

Platforms: EE: All cloud platforms

Parent topic: Installing

In this section:

> **PREVIEW FEATURE**
>
> This is a preview feature that may have incomplete functionality.
>
> A Self-Managed Install requires extensive end-user customization and advanced knowledge of Kubernetes.

A Yellowbrick self-managed installation is one where the installation of each component is managed by the user. This installation method provides control of every aspect of the installation process. Once installed, the user assumes full lifecycle management of the kubernetes cluster and installed components.

The self-managed installation process involves:

- Using the Yellowbrick Deployer to push assets to your private registry
- Create cloud infrastructure
- Install Helm charts
- Create a Yellowbrick Instance

> **INFO**
>
> Yellowbrick workloads are compatible upto Kuberentes version `1.30`.
>
> Yellowbrick maintained Kubernetes Node AMI for Kubernetes include Kubelet version `1.30`.

Pushing Assets

Installation assets include Helm charts and container images. These assets are pushed into private registries by the user with the Yellowbrick Deployer. Follow the instructions to create the Deployer in your respective environment:

- AWS installation instructions
- Azure installation instructions
- GCP installation instructions

To push assets with the Deployer: - Gain access to the Deployer instance via SSH - Use `/opt/ybd/yb-install push` to copy Helm charts and container images - See the CLI reference for details on the `push` subcommand.

If you choose to not use the Deployer to push assets, you must ensure those components are available to the installation process described below.

Deploying

To quickly get started using Terraform in a self-managed installation, see the Terraform reference architecture.

To cover other deployment methods, please see the following section that lists individual components for each cloud provider environment. The components listed are required deployment dependencies unless stated as optional. Some of these may already be deployed in your environment. Review each to ensure they meet the requirements to perform a successful deployment.

AWS Install Instructions

When installing into an AWS cloud environment, the following components are required:

AWS Add-ons

- AWS Add-on: Amazon VPC CNI plugin for Kubernetes
- AWS Add-on: Amazon EBS CSI Driver
- AWS Add-on: CoreDNS
- AWS Add-on: kube-proxy
- Community Add-on: Kubernetes Metrics Server

Infra

A reference Node Group is required for the Yellowbrick Operator to function, instructions can be found below.

- Infra: Node Group

Helm Charts

The following Helm charts are required. In each section, there may be instructions for installing Helm chart, creating cloud insfrastructure, or both.

Authenticating with ECR

You will need to authenticate helm with ECR to be able to install the helm charts. Replace {account_id} with your AWS account ID and {region} with the AWS region.

```bash
aws ecr get-login-password --region {region} | helm registry login --username AWS --password-stdin {account_id}.dkr.ecr.{region}.a
```

- Helm: cert-manager
- Helm: cluster-autoscaler
- Helm: node-local-dns
- Helm: yb-storageclass
- Helm: yb-operator
- Helm: yb-resources
- Helm: yb-monitoring

Create Yellowbrick Instance

Upon deploying all the required components, the final step is to create a `YBInstance` Custom Resource in Kubernetes. This will instruct the Yellowbrick Operator to proceed to create your first instance.

This example uses `kubectl` to create a `YBInstance` of version `7.2.0-68613.33955ce9`. Refer to YBInstance CRD Spec for more customizations.

```bash
# Setup Initial Admin Password, secret name must be of pattern ybinst-{instance_name}-intial-admin
cat <<EOF >secret.yaml
apiVersion: v1
kind: Secret
metadata:
  name: ybinst-firstinstance-initial-admin
  namespace: yb-firstinstance
type: Opaque
stringData:
  username: myuser
  password: mypassword
EOF

kubectl apply -f secret.yaml

# Create YBInstance
cat <<EOF >instance.yaml
apiVersion: cluster.yellowbrick.io/v1
kind: YBInstance
metadata:
  name: firstinstance
  namespace: yb-firstinstance
spec:
  requestedState: Running
  sharedServicesType: standard
  storageManaged: true
  version: 7.2.0-68613.33955ce9
EOF

kubectl apply -f instance.yml
```

You can then login to Yellowbrick Manager using the credentials as setup in the secret. The URL may be found using the following command:

```bash
kubectl get service -n {namespace} yb-manager-service \
  -o jsonpath='{.status.loadBalancer.ingress[0].hostname}'
```

INFO

In case the credentials are not setup properly, you can run the following command to shell into the database pod and reset the password using `ALTER ROLE/USER` command. If the intial user itself is not set, assume the first user to be `ybdadmin`.

```bash
kubectl exec -it -n {namespace} ybinst-{instance_name}-0 ybinst-pg -- ybsql yellowbrick --command="ALTER ROLE ybdadmin with p
```

Self-Managed: Amazon VPC CNI plugin for Kubernetes

Platforms: EE: All cloud platforms

Parent topic: Self-Managed Install Instructions

Install the Amazon VPC CNI plugin for Kubernetes as an AWS add-on.

What are AWS add-ons?

AWS add-ons are built, validated, and maintained by AWS. These add-ons receive regular security patches and updates through AWS service teams. AWS add-ons are published directly through AWS service endpoints and undergo continuous security scanning. AWS add-ons help you implement essential cluster services or integrate with other AWS services.

For more information on AWS add-ons, see AWS add-ons.

Installing

For details on installing Amazon VPC CNI plugin for Kubernetes add-on, please refer to the documentation

Parameters

When installing the VPC CNI, you will find more efficient usage of IP addresses by using these values:

```json
{
  "env": {
    "MINIMUM_IP_TARGET": "4",
    "WARM_IP_TARGET": "2"
  }
}
```

Self-Managed: Amazon EBS CSI driver

Platforms: EE: All cloud platforms

Parent topic: Self-Managed Install Instructions

Install the Amazon EBS CSI driver as an AWS add-on.

INFO

Please ensure the role being used by this addon has sufficient permissions to provision EBS volumes.

See Helm: yb-storageclass.

What are AWS add-ons?

AWS add-ons are built, validated, and maintained by AWS. These add-ons receive regular security patches and updates through AWS service teams. AWS add-ons are published directly through AWS service endpoints and undergo continuous security scanning. AWS add-ons help you implement essential cluster services or integrate with other AWS services.

For more information on AWS add-ons, see AWS add-ons.

Installing

For details on installing the Amazon EBS CSI driver add-on, please refer to the documentation

Self-Managed: CoreDNS

Platforms: EE: All cloud platforms

Parent topic: Self-Managed Install Instructions

Install CoreDNS as an AWS add-on.

What are AWS add-ons?

AWS add-ons are built, validated, and maintained by AWS. These add-ons receive regular security patches and updates through AWS service teams. AWS add-ons are published directly through AWS service endpoints and undergo continuous security scanning. AWS add-ons help you implement essential cluster services or integrate with other AWS services.

For more information on AWS add-ons, see AWS add-ons.

Installing

For details on installing the CoreDNS AWS add-on, please refer to the documentation

Self-Managed: kube-proxy

Platforms: EE: All cloud platforms

Parent topic: Self-Managed Install Instructions

Install kube-proxy as an AWS add-on.

What are AWS add-ons?

AWS add-ons are built, validated, and maintained by AWS. These add-ons receive regular security patches and updates through AWS service teams. AWS add-ons are published directly through AWS service endpoints and undergo continuous security scanning. AWS add-ons help you implement essential cluster services or integrate with other AWS services.

For more information on AWS add-ons, see AWS add-ons.

Installing

For details on installing kube-proxy add-on, please refer to the documentation

Self-Managed: Kubernetes Metrics Server

Platforms: EE: All cloud platforms

Parent topic: Self-Managed Install Instructions

Install Metrics Server as a Community add-on.

INFO

Metrics Server is used for Horizontal Pod Autoscaling of Yellowbrick Database Workloads.

What are Community add-ons?

Community Add-ons in Amazon EKS are popular open-source Kubernetes add-ons that have been packaged, scanned, and validated for compatibility by AWS. These add-ons provide essential cluster capabilities and can be easily integrated into EKS clusters.

It's important to note that using community add-ons is at the user's discretion, and they should carefully evaluate each add-on's security implications before installation. For the most up-to-date information on available add-ons and their features, users should consult the official AWS documentation.

Installing

For details on installing the Kubernetes Metrics Server add-on, please refer to the documentation

Self-Managed: Node Group

Platforms: EE: All cloud platforms

Parent topic: Self-Managed Install Instructions

Create a base Node group for Yellowbrick Oeprator and Manager to run. The configuration in this node group is then used by the yellowbrick operator to create more node groups dynamically.

When using the commands or values outlined here, please make appropriate substitutions defined as:

Value	Description
{cluster_endpoint}	The https endpoint of the EKS Cluster found on the console
{certificate_authority}	The CA data of the EKS Cluster found on the console
{cluster_name}	The name of the EKS cluster
{region}	The AWS region of the EKS cluster
{ami_id}	The AMI ID to be used for the nodes.
{security_group_ids}	Comma Separated list of security group ids to be attached to the node group.
{subnets}	Space separated list of Subnet Ids for node group to be in.
{node_iam_role_arn}	IAM role to be attached to the nodes in the ndoe group

INFO

1. Security Groups used should ensure connection to the rest of the nodes in the cluster as well as the control plane or it could lead to nodes not joining. Recommendation is to use the same as the rest of the cluster.
2. We recommend that a single subnet is used for the node group to ensure the nodes land in a single availability zone - or this could lead to cross az costs.
3. The Node IAM role used must have the AWS recommended node policy as well as any other policy required to run any daemonsets/workloads on the existing cluster. Additionally, the role *requires* the `AmazonEKS_CNI_Policy` policy for the compute cluster nodes to function.
4. It is recommended to use Amazon Linux 2. Yellowbrick also maintains node AMIs based on Amazon Linux 2, the Name of the AMI is `yb-enterprise-eks-node-1.30-v20250115-20250116232817` owned by account id `732123782549` for commericial aws and `335348567317` for gov cloud. The following command can be used to refer to the AMI ID for your region.

```bash
aws ec2 describe-images \
  --owners 732123782549 \
  --filters "Name=name,Values=yb-enterprise-eks-node-1.30-v20250115-20250116232817" \
  --query "Images[*].ImageId" \
  --region {region}
```

Creating Cloud Infrastructure

Node IAM Role (Optional)

The following steps are required only if you are creating a Node IAM role exclusively for the operator node group and the ones created by the Yellowbrick Operator.

Create the IAM role:

```bash
aws iam create-role \
  --role-name yb-eks-node-{instance-name}-{region} \
  --assume-role-policy-document file://trust-policy.json
```

The trust policy:

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
          "sts:AssumeRole"
      ],
      "Principal": {
          "Service": [
              "ec2.amazonaws.com"
          ]
      }
    }
  ]
}
```

Attach the IAM policy

```bash
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy \
  --role-name yb-eks-node-{instance-name}-{region}
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryPullOnly \
  --role-name yb-eks-node-{instance-name}-{region}
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy \
  --role-name yb-eks-node-{instance-name}-{region}
```

Yellowbrick Operator Node group

Preparing Node User Data

> **INFO**
>
> This User data is specific for Amazon Linux 2 based AMIs. Please adjust the user data if you are using a different distro/AMI.

Create a file, `user-data.yaml` with the following content, and correct parameters.

```yaml
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="==YBBOUNDARY=="

--==YBBOUNDARY==
Content-Type: text/cloud-config; charset="us-ascii"

write_files:
  - path: /root/bootstrap.sh
    permissions: "0700"
```

```
    content: |
      #!/bin/bash
      set -ex

      sysctl net.ipv4.ip_forward=1
      sed 's/net.ipv4.ip_forward=0/net.ipv4.ip_forward=1/g' /etc/sysctl.conf > /etc/sysctl.conf

      /etc/eks/bootstrap.sh \
        --apiserver-endpoint "{cluster_endpoint}" \
        --b64-cluster-ca "{certificate_authority}" \
        "{cluster_name}"

runcmd:
  - |
    set -x
    (
      while [ ! -f /root/bootstrap.sh ]; do
        sleep 1
      done
      if ! /root/bootstrap.sh; then
        shutdown now
      fi
    )
--==YBBOUNDARY==--
```

Then Base64-encode this file if you plan to embed it in the JSON for the launch template:

```bash
USER_DATA_B64=$(cat user-data.yaml | base64 | tr -d '\n')
```

Preparing Launch Template

After the user data is prepared, we proceed to created the launch template for the node group. This is a ***must do***.

INFO

Additional Parameters such as `KeyPair` and `Tags` maybe added to the launch templates.

```bash
aws ec2 create-launch-template \
  --launch-template-name "yb-op-standard-lt" \
  --version-description "default" \
  --region "{region}" \
  --launch-template-data "{
    \"ImageId\": \"{ami_id}\",
    \"InstanceType\": \"t3.large\",
    \"SecurityGroupIds\": {security_group_ids},
    \"MetadataOptions\": {
      \"HttpEndpoint\": \"enabled\",
      \"HttpPutResponseHopLimit\": 2,
      \"HttpTokens\": \"optional\"
    },
    \"UserData\": \"${USER_DATA_B64}\"
  }"
```

Creating Node Group

After the launch template is created, we can proceed to create the group.

INFO

The Node group must be named `yb-op-standard` to ensure yellowbrick operator can inherit parameters while creating additional node groups.

Additional Parameters such as `Tags` maybe added to the node group.

```bash
aws eks create-nodegroup \
  --cluster-name "{cluster_name}" \
  --nodegroup-name "yb-op-standard" \
  --subnets {subnets} \
  --node-role "{node_iam_role_arn}" \
  --scaling-config minSize=1,maxSize=3,desiredSize=1 \
  --labels "cluster.yellowbrick.io/owned=true,cluster.yellowbrick.io/hardware_type=t3.large,cluster.yellowbrick.io/node_type=yb-op
  --taints "key=cluster.yellowbrick.io/owned,value=true,effect=NO_SCHEDULE" \
  --launch-template name=yb-op-standard-lt,version=1 \
  --capacity-type ON_DEMAND \
  --region "{region}"
```

Self-Managed: cert-manager

Platforms: EE: All cloud platforms

Parent topic: Self-Managed Install Instructions

INFO

Yellowbrick uses cert-manager for certificate management of it's components.

Install cert-manager with Helm. Reference ArtifactHub for all possible options.

When using the commands or values outlined here, please make appropriate substitutions defined as:

Value	Description
{cluster-name}	The name of the Kubernetes cluster
{version}	The chart version of `cert-manager`
{image-repo}	The container image repository pushed by the Deployer
{namespace}	The Kubernetes namespace into which you want to install

Helm Chart

Running the Yellowbrick Deployer will push the Helm charts and container images you need into your cloud environment. For instructions on pushing assets using the Deployer, see the documentation.

Chart name: `cert-manager`

The `get-assets` subcommand can be used to find the version of chart name `cert-manager`, see cli reference.

Install Command

See Authenticating with ECR

```bash
helm install cert-manager oci://{image-repo}/cert-manager \
  -n {namespace}  \
  -f values.yaml \
  --version {version}
```

Values

Please note that you may need to adjust the node selector values as appropriate for your installation.

```yaml
nodeSelector: &nodeSelector
  cluster.yellowbrick.io/hardware_type: t3.large
  cluster.yellowbrick.io/node_type: yb-op-standard
```

```
tolerations: &tolerations
  - effect: NoSchedule
    key: cluster.yellowbrick.io/owned
    operator: Equal
    value: "true"

cainjector:
  image:
    repository: {image-repo}/jetstack/cert-manager-cainjector
  nodeSelector: *nodeSelector
  tolerations: *tolerations

image:
  repository: {image-repo}/jetstack/cert-manager-controller

installCRDs: true

nodeSelector: *nodeSelector

securityContext:
  enabled: true
  fsGroup: 1001

startupapicheck:
  image:
    repository: {image-repo}/jetstack/cert-manager-ctl
  nodeSelector: *nodeSelector
  tolerations: *tolerations

tolerations: *tolerations

webhook:
  image:
    repository: {image-repo}/jetstack/cert-manager-webhook
  nodeSelector: *nodeSelector
  tolerations: *tolerations
```

Self-Managed: cluster-autoscaler

Platforms: EE: All cloud platforms

Parent topic: Self-Managed Install Instructions

Install cluster-autoscaler with Helm. Reference ArtifactHub for all possible options.

INFO

Cluster Autoscaler is restricted to only autoscale node groups created by Yellowbrick Operator.

When using the commands or values outlined here, please make appropriate substitutions defined as:

Value	Description
{cluster-name}	The name of the Kubernetes cluster
{cloud-provider}	Your cloud provider: aws, azure, or gce
{version}	The chart version of `cluster-autoscaler`
{image-repo}	The container image repository pushed by the Deployer
{namespace}	The Kubernetes namespace into which you want to install

Helm Chart

Running the Yellowbrick Deployer will push the Helm charts and container images you need into your cloud environment. For instructions on pushing assets using the Deployer, see the documentation.

Chart name: `cluster-autoscaler`

The `get-assets` subcommand can be used to find the version of chart name `cluster-autoscaler`, see cli reference.

Install Command

See Authenticating with ECR

```bash
helm install cluster-autoscaler oci://{image-repo}/cluster-autoscaler \
  -n {namespace}  \
  -f values.yaml \
  --version {version}
```

Values

Please note that you may need to adjust the node selector and toleration values as appropriate for your installation.

```yaml
clusterName: &clusterName { cluster-name }

affinity:
  podAntiAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
            - key: app.kubernetes.io/name
              operator: In
              values:
                - cluster-autoscaler
        namespaceSelector: {}
        topologyKey: kubernetes.io/hostname

autoDiscovery:
  clusterName: *clusterName
  tags:
    - k8s.io/cluster-autoscaler/enabled
    - k8s.io/cluster-autoscaler/{cluster-name}
    - k8s.io/cluster-autoscaler/node-template/label/cluster.yellowbrick.io/owned

cloudProvider: { cloud-provider }

extraArgs:
  ignore-daemonsets-utilization: true
  logtostderr: true
  max-empty-bulk-delete: "10"
  max-graceful-termination-sec: "600"
  max-node-provision-time: 15m
  max-total-unready-percentage: "45"
  new-pod-scale-up-delay: 0s
  ok-total-unready-count: "3"
  scale-down-delay-after-add: 10m
  scale-down-delay-after-delete: 10s
  scale-down-delay-after-failure: 3m
  scale-down-unneeded-time: 10m
  scale-down-unready-time: 10m
  scale-down-utilization-threshold: "0.5"
  scan-interval: 10s
  skip-nodes-with-local-storage: "false"
  skip-nodes-with-system-pods: "false"
  stderrthreshold: info
  v: 4

extraVolumeMounts:
  - mountPath: /etc/ssl/certs/ca-certificates.crt
    name: ssl-certs
    readOnly: true

extraVolumes:
  - hostPath:
      path: /etc/ssl/certs/ca-bundle.crt
    name: ssl-certs

image:
  repository: { image-repo }/autoscaling/cluster-autoscaler
  tag: v1.29.4

podAnnotations:
  cluster-autoscaler.kubernetes.io/safe-to-evict: "false"

podDisruptionBudget: null

nodeSelector:
  cluster.yellowbrick.io/hardware_type: t3.large
  cluster.yellowbrick.io/node_type: yb-op-standard
```

```
tolerations:
  - effect: NoSchedule
    key: cluster.yellowbrick.io/owned
    operator: Equal
    value: "true"
rbac:
  serviceAccount:
    annotations:
      eks.amazonaws.com/role-arn: { role-arn }
```

Creating Cloud Infrastructure

AWS

When installing on AWS, an IRSA service account is used. For details on IRSA, please see the AWS documentation.

Create the IAM role:

bash

```
aws iam create-role \
  --role-name yb-eks-pod-cluster-autoscaler-{instance-name}-{region} \
  --assume-role-policy-document file://trust-policy.json
```

The trust policy:

json

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "{oidc-provider-arn}"
     },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "{oidc-provider}:sub": "system:serviceaccount:{namespace}:cluster-autoscaler-aws-cluster-autoscaler"
       }
     }
   }
  ]
}
```

The IAM policy:

bash

```
aws iam put-role-policy \
  --role-name yb-eks-pod-cluster-autoscaler-{instance-name}-{region} \
  --policy-name diags-upload \
  --policy-document file://iam-policy.json
```

json

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "autoscaling:DescribeAutoScalingGroups",
        "autoscaling:DescribeAutoScalingInstances",
```

```
        "autoscaling:DescribeLaunchConfigurations",
        "autoscaling:DescribeScalingActivities",
        "ec2:DescribeImages",
        "ec2:DescribeInstanceTypes",
        "ec2:DescribeLaunchTemplateVersions",
        "ec2:GetInstanceTypesFromInstanceRequirements",
        "eks:DescribeNodegroup"
      ],
      "Resource": ["*"]
    },
    {
      "Effect": "Allow",
      "Action": [
        "autoscaling:SetDesiredCapacity",
        "autoscaling:TerminateInstanceInAutoScalingGroup"
      ],
      "Resource": ["*"]
    }
  ]
}
```

To the values above, add these values and include the ARN of the AWS IAM role in place of `{role-arn}`:

yaml

```yaml
rbac:
  serviceAccount:
    annotations:
      eks.amazonaws.com/role-arn: { role-arn }
```

Self-Managed: node-local-dns

Platforms: EE: All cloud platforms

Parent topic: Self-Managed Install Instructions

INFO

Node Local DNS is required for the compute clusters to function.

Install node-local-dns with Helm. Reference ArtifactHub for all possible options.

When using the commands or values outlined here, please make appropriate substitutions defined as:

Value	Description
{image-repo}	The container image repository pushed by the Deployer
{version}	The chart version of `node-local-dns`
{namespace}	The Kubernetes namespace into which you want to install

Helm Chart

Running the Yellowbrick Deployer will push the Helm charts and container images you need into your cloud environment. For instructions on pushing assets using the Deployer, see the documentation.

Chart name: `node-local-dns`

The `get-assets` subcommand can be used to find the version of chart name `node-local-dns`, see cli reference.

Install Command

See Authenticating with ECR

```bash
helm install node-local-dns oci://{image-repo}/node-local-dns \
  -n {namespace}  \
  -f values.yaml \
  --version {version}
```

Values

Please note that you may need to adjust the `pillar_dns_server` value to match your Kubernetes service network. Please note that the `pillar_local_dns` value must be `169.254.0.53`.

```yaml
pillar_dns_server: 10.0.0.10
pillar_local_dns: 169.254.0.53
```

```
image:
  repository: { image-repo }/dns/k8s-dns-node-cache
```

Self-Managed: yb-storageclass

Platforms: EE: All cloud platforms

Parent topic: Self-Managed Install Instructions

Install yb-storageclass with Helm. This will create all the Yellowbrick Kubernetes Storage Classes required for creating an Yellowbrick instance.

INFO

Given immutable nature of storageclass, this helm chart only requires to be applied once. A uninstall will explicitly delete the storage classes, and may need to be cleaned up manually.

When using the commands or values outlined here, please make appropriate substitutions defined as:

Value	Description
{cloud-provider}	Your cloud provider: aws, azure, or gcp
{version}	The chart version of `yb-storageclass`

Helm Chart

Running the Yellowbrick Deployer will push the Helm charts and container images you need into your cloud environment. For instructions on pushing assets using the Deployer, see the documentation.

Chart name: `yb-storageclass`

The `get-assets` subcommand can be used to find the version of chart name `yb-storageclass`, see cli reference.

Install Command

See Authenticating with ECR

```bash
helm install yb-storageclass oci://{image-repo}/yb-storageclass \
  -n {namespace}  \
  -f values.yaml \
  --version {version}
```

Values

Please note that you may need to adjust the `allowedTopologies` and `parameters` values as appropriate for your installation.

INFO

This helm chart creates storage classes which can provision `gp3` and `io1` volumes. Without a prefix, the storage class will use names similar to the types on aws(for example `gp3`, `io1-scaled` and `io1-standard`). To avoid a conflict with existing standard names, we recommend that you use a `yb-` prefix, which will prefix the name of the storage class(for example `yb-gp3`)

The storage classes can be locked to a specific zone by using `allowedTopologies` and extra parameters can be applied to all storage classes. See here

```yaml
cloudProvider: {cloud-provider}

storageClass:
  prefix: yb-

  allowedTopologies: []
  parameters: {}
```

INFO

Please note that the `prefix` given here needs to match the `prefix` given for the `yb-resources` Helm chart.

Self-Managed: yb-operator

Platforms: EE: All cloud platforms

Parent topic: Self-Managed Install Instructions

Install yb-operator with Helm. This will install the Yellowbrick Operator and Yellowbrick Manager.

INFO

Yelowbrick Operator creates and manages node groups in the EKS cluster for Yellowbrick Datawarehouse workloads.

If using yellowbrick created storage classes, it is required to install `yb-storageclass` See Helm: yb-storageclass.

When using the commands or values outlined here, please make appropriate substitutions defined as:

Value	Description
{account}	When on AWS, the account ID
{cloud-provider}	Your cloud provider: aws, azure, or gcp
{cluster-name}	The name of the Kubernetes cluster
{version}	The chart version of `yb-operator`
{node-role-arn}	The Role attached to the node group created during Infra: Node Group
{instance-name}	The name of the Yellowbrick instance
{image-repo}	The container image repository pushed by the Deployer
{image-tag}	The corresponding tag of the container image
{namespace}	The Kubernetes namespace into which you want to install
{observability-ns}	The Kubernetes namespace of the observability suite
{observability-storage}	The name of the storage location for the observability, when AWS an S3 bucket name, when Azure a Storage Account name
{oidc-provider-arn}	When on AWS, the OpenID Connect provider ARN
{oidc-provider}	When on AWS, the OpenID Connect provider
{partition}	When on AWS, the partition: aws or aws-gov
{role-arn-operator}	when on AWS, the IAM role ARN of the operator service account
{role-arn-diags}	when on AWS, the IAM role ARN of the diagnostics service account
{role-arn-compute}	when on AWS, the IAM role ARN of the computer cluster service account
{role-arn-manager}	when on AWS, the IAM role ARN of the Yellowbrick Manager service account
{region}	The name of the cloud provider region
{storage-class}	The general purpose storage class name, e.g. AWS: gp3, Azure: standard, GCP: pd-balanced

Helm Chart

Running the Yellowbrick Deployer will push the Helm charts and container images you need into your cloud environment. For instructions on pushing assets using the Deployer, see the documentation.

Chart name: `yb-operator`

The `get-assets` subcommand can be used to find the version of chart name `yb-operator`, see cli reference.

Install Command

See Authenticating with ECR

```bash
helm install yb-operator oci://{image-repo}/yb-operator \
  -n {namespace}  \
  -f values.yaml \
  --version {version}
```

Values

Please note that you may need to adjust the node selector values as appropriate for your installation.

INFO

The `get-assets` subcommand can be used to find the image tag for `yellowbrick/yb-operator`, `yellowbrick/yb-manager` and `yellowbrick/yb-template`, see cli reference.

```yaml
image:
  registry: {image-repo}
  repository: yellowbrick/yb-operator
  tag: {image-version}

nodeSelector: &nodeSelector
  cluster.yellowbrick.io/hardware_type: t3.large
  cluster.yellowbrick.io/node_type: yb-op-standard
tolerations: &tolerations
  - effect: NoSchedule
    key: cluster.yellowbrick.io/owned
    operator: Equal
    value: "true"

config:
  data:
    provider: {cloud-provider}
    region: {region}
    diagsContainer: diags
    registryHost: {image-repo}
    observabilityBucketName: {observability-storage}
    klusterName: {cluster-name}
    monitoringNamespace: {observability-ns}
    additionalTags:
      cluster_yellowbrick_io_creator: yb-install
      cluster_yellowbrick_io_name: {instance-name}
      cluster_yellowbrick_io_owner: yb-install

installCrd: true

serviceAccount:
```

```
  annotations:
    eks.amazonaws.com/role-arn: { role-arn-operator }

diags:
  serviceAccount:
    annotations:
      eks.amazonaws.com/role-arn: { role-arn-diags }

worker:
  serviceAccount:
    annotations:
      eks.amazonaws.com/role-arn: { role-arn-compute }

pvc:
  storageClassName: {storage-class}

certManager:
  create: true
  selfSignedIssuerName: selfsigned

yb-manager:
  enabled: true
  cloudProvider: aws
  containerImage: {image-repo}/yellowbrick/yb-manager:{image-tag}
  instance:
    global: true
  # Using service.annotations disregards loadBalancer.internal and loadBalancer.tags
  service:
    annotations: {}
  serviceAccount:
    annotations:
      eks.amazonaws.com/role-arn: { role-arn-manager }
  loadBalancer:
    internal: false
    tags: ""
  nginxRepository: {image-repo}/nginx
  nodeSelector: *nodeSelector
  tolerations: *tolerations

  storageClassName: {storage-class}

  ybTemplateImage: {image-repo}/yellowbrick/yb-template:{image-tag}
```

If you are using the `yb-storageclass` Helm chart, please use those storage classes as the value for `storageClassName`.

INFO

Please note if you do not want to install Yellowbrick Custom Resource Definitions, add this value:

yaml
```
installCrd: false
```

If you want to use custom service annotations for customizing the LoadBalancer for YB Manager, use the following values:

yaml
```
yb-manager:
  service:
    annotations: {}
    # your custom annotaitons
    # foo: bar
```

Creating Cloud Infrastructure

AWS

When installing on AWS, IRSA service accounts are used. For details on IRSA, please see the AWS documentation.

Operator

To the values above, add this value to include the ARN of the AWS IAM role:

```yaml
serviceAccount:
  annotations:
    eks.amazonaws.com/role-arn: { role-arn-operator }
```

Create the IAM role:

```bash
aws iam create-role \
  --role-name yb-eks-pod-yb-operator-{instance-name}-{region} \
  --assume-role-policy-document file://trust-policy.json
```

The trust policy:

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "{oidc-provider-arn}"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "{oidc-provider}:sub": "system:serviceaccount:{namespace}:yb-operator-sa"
        }
      }
    }
  ]
}
```

Create the IAM policy

```bash
aws iam put-role-policy \
  --role-name yb-eks-pod-yb-operator-{instance-name}-{region} \
  --policy-name operator-policy \
  --policy-document file://iam-policy.json
```

The IAM policy:

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:DeleteObjectVersion",
        "s3:ListBucketVersions",
        "s3:CreateBucket",
        "s3:ListBucket",
```

```
        "s3:GetBucketPolicy",
        "s3:GetEncryptionConfiguration",
        "s3:PutBucketTagging",
        "s3:DeleteObject",
        "s3:DeleteBucket",
        "s3:GetObject",
        "s3:PutBucketPolicy",
        "s3:GetObjectVersion",
        "ec2:DescribePlacementGroups",
        "ec2:DescribeLaunchTemplates",
        "ec2:CreateLaunchTemplate",
        "ec2:CreateLaunchTemplateVersion",
        "ec2:CreatePlacementGroup",
        "ec2:DescribeLaunchTemplateVersions",
        "ec2:CreateTags",
        "ec2:RunInstances",
        "ec2:DescribeImages",
        "ec2:DescribeSubnets",
        "ec2:DescribeVolumes",
        "ec2:DescribeSnapshots",
        "ec2:CreateSnapshot",
        "ec2:CreateVolume",
        "ec2:DescribeSnapshotAttribute",
        "ec2:DescribeVolumeAttribute",
        "ec2:DeleteSnapshot",
        "ec2:DeleteVolume",
        "ec2:ModifyLaunchTemplate",
        "eks:ListNodegroups",
        "eks:TagResource",
        "eks:DescribeCluster",
        "eks:CreateNodegroup",
        "eks:DescribeNodegroup",
        "eks:UpdateNodegroupVersion",
        "ecr:GetDownloadUrlForLayer",
        "ecr:GetAuthorizationToken",
        "ecr:BatchGetImage",
        "autoscaling:CreateOrUpdateTags",
        "iam:GetRole",
        "iam:ListAttachedRolePolicies"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": ["iam:PassRole"],
      "Resource": "arn:{partition}:iam::{account}:role/{node-role-arn}"
    }
  ]
}
```

Diagnostics

S3 bucket

To create an s3 bucket, the following command can be used:

```bash
aws s3api create-bucket \
  --bucket {observability-storage} \
  --region {region} \
  --create-bucket-configuration LocationConstraint={region}
```

The above command is a simple example, and can be modified to include complex configurations.

IAM Role

To the values above, add this value to include the ARN of the AWS IAM role:

```yaml
diags:
  serviceAccount:
    annotations:
      eks.amazonaws.com/role-arn: { role-arn-diags }
```

Create the IAM role:

```bash
aws iam create-role \
  --role-name yb-eks-pod-diags-{instance-name}-{region} \
  --assume-role-policy-document file://trust-policy.json
```

The trust policy:

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "{oidc-provider-arn}"
     },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "{oidc-provider}:sub": "system:serviceaccount:{namespace}:yb-diags-sa"
       }
     }
   }
  ]
}
```

Create the IAM policy

```bash
aws iam put-role-policy \
  --role-name yb-eks-pod-diags-{instance-name}-{region} \
  --policy-name diags-upload \
  --policy-document file://iam-policy.json
```

The IAM policy:

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["s3:PutObject", "s3:GetObject", "s3:DeleteObject"],
      "Resource": "arn:{partition}:s3:::{observability-storage}/*"
    }
  ]
}
```

YB Manager

To the values above, add this value to include the ARN of the AWS IAM role:

```yaml
yb-manager:
  serviceAccount:
    annotations:
      eks.amazonaws.com/role-arn: { role-arn-manager }
```

Create the IAM role:

```bash
aws iam create-role \
  --role-name yb-eks-pod-manager-{instance-name}-{region} \
  --assume-role-policy-document file://trust-policy.json
```

The trust policy:

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "{oidc-provider-arn}"
     },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "{oidc-provider}:sub": "system:serviceaccount:{namespace}:yb-manager-serviceaccount"
       }
     }
   }
  ]
}
```

Create the IAM policy

```bash
aws iam put-role-policy \
  --role-name yb-eks-pod-manager-{instance-name}-{region} \
  --policy-name manager \
  --policy-document file://iam-policy.json
```

The IAM policy:

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
          "s3:GetObject"
      ],
      "Resource": "arn:aws:s3:::{observability-storage}/*"
    },
    {
      "Effect": "Allow",
      "Action": "ec2:DescribeTags",
      "Resource": "*"
    }
  ]
}
```

Compute Cluster

To the values above, add this value to include the ARN of the AWS IAM role:

```yaml
worker:
  serviceAccount:
    annotations:
      eks.amazonaws.com/role-arn: { role-arn-compute }
```

Create the IAM role:

```bash
aws iam create-role \
  --role-name yb-eks-pod-compute-cluster-{instance-name}-{region} \
  --assume-role-policy-document file://trust-policy.json
```

The trust policy:

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "{oidc-provider-arn}"
     },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "{oidc-provider}:sub": "system:serviceaccount:{namespace}:yb-{namespace}-worker-sa"
       }
     }
   }
  ]
}
```

Create the IAM policy

```bash
aws iam put-role-policy \
  --role-name yb-eks-pod-compute-cluster-{instance-name}-{region} \
  --policy-name diags-upload \
  --policy-document file://iam-policy.json
```

The IAM policy:

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["s3:*"],
      "Resource": "arn:aws:s3:::yb-*/*"
    },
    {
      "Effect": "Allow",
      "Action": ["s3:GetBucketLocation", "s3:ListBucket"],
```

```
      "Resource": "arn:aws:s3:::yb-*"
    }
  ]
}
```

INFO

The security of this policy can be further restricted to use `yb-*-{suffix}` as the resource pattern. The `{suffix}` variable is uniquely calculated as the first 6 characters of the SHA1 hash of the string concatenation of the cluster name, account ID, and region.

For example:

bash

```
$ A="cluster-name" B="123456789012" C="us-east-1" && echo -n "${A}${B}${C}" | sha1sum | head -c6
bdc688
```

Self-Managed: yb-resources

Platforms: EE: All cloud platforms

Parent topic: Self-Managed Install Instructions

Install yb-resources with Helm. This will create all the Yellowbrick Kubernetes Custom Resources required for creating an Yellowbrick instance.

INFO

Some of the custom resources created by yb-resource include EKS node groups. At this time, simply uninstalling the yb-resources release does not delete the node groups, and they must be cleaned up manually using aws cli/console.

The Yellowbrick Operator also modifies the YBNodegroup resources to turn on/off the nodegroup, and it recommended to not upgrade the chart unless future versions explicitly require an upgrade.

When using the commands or values outlined here, please make appropriate substitutions defined as:

Value	Description
{cloud-provider}	Your cloud provider: aws, azure, or gcp
{namespace}	The Kubernetes namespace into which you want to install
{version}	The chart version of `yb-resources`

Helm Chart

Running the Yellowbrick Deployer will push the Helm charts and container images you need into your cloud environment. For instructions on pushing assets using the Deployer, see the documentation.

Chart name: `yb-resources`

The `get-assets` subcommand can be used to find the version of chart name `yb-resources`, see cli reference.

Install Command

INFO

1. This chart ***MUST*** be installed after the `yb-operator` is installed as it contains references to `CustomResourceDefinitions` in that chart.
2. It is recommended to use the same namespace as the one used to install `yb-operator` in.

See Authenticating with ECR

```bash
helm install yb-resources oci://{image-repo}/yb-resources \
  -n {namespace}  \
  -f values.yaml \
  --version {version}
```

Values

Please note that you may need to adjust the node selector values as appropriate for your installation.

```yaml
cloudProvider: {cloud-provider}

storageClass:
  prefix: yb-
```

INFO

Please note that the `prefix` given here needs to match the `prefix` given for the `yb-storageclasses` Helm chart.

INFO

Please note that the `selfSignedIssuerName` given here needs to match the issuername while deploying yb-operator helm chart.

Self-Managed: yb-monitoring

Platforms: EE: All cloud platforms

Parent topic: Self-Managed Install Instructions

Install yb-monitoring with Helm. This will deploy workloads enabling monitoring for Yellowbrick Data Warehouse. Typically containts `loki`, `grafana`, `prometheus` and `fluent-bit`.

INFO

If using yellowbrick created storage classes, it is required to install `yb-storageclass` See Helm: yb-storageclass.

The Yellowbrick Operator also modifies the yb-monitoring helm chart for certain features such as changing the node group tier and enabling log retention on s3 buckets, and it is recommended to take this behaviour into account by reusing values when using automated deployments to manage this chart.

When using the commands or values outlined here, please make appropriate substitutions defined as:

Value	Description
{image-repo}	The container image repository pushed by the Deployer
{namespace}	The Kubernetes namespace into which you want to install
{role-arn}	when on AWS, the IAM role ARN of the Fluent bit service account
{version}	The chart version of `loki-stack`
{observability-storage}	The name of the storage location for the observability, when AWS an S3 bucket name, when Azure a Storage Account name. Must be same as what used when deploying yb-operator chart.
{oidc-provider-arn}	When on AWS, the OpenID Connect provider ARN
{oidc-provider}	When on AWS, the OpenID Connect provider
{partition}	When on AWS, the partition: aws or aws-gov
{storageclass}	The general purpose storage class name, e.g. AWS: gp3, Azure: standard, GCP: pd-balanced

Helm Chart

Running the Yellowbrick Deployer will push the Helm charts and container images you need into your cloud environment. For instructions on pushing assets using the Deployer, see the documentation.

Chart name: `loki-stack`

The `get-assets` subcommand can be used to find the version of chart name `loki-stack`, see cli reference.

Install Command

See Authenticating with ECR

```bash
helm install loki oci://{image-repo}/loki-stack \
  -n {namespace}  \
  -f values.yaml \
  --version {version}
```

INFO

Please note that the release name while installing yb-monitoring must be ***loki*** and the namespace must be the same as supplied while installing the yb-operator helm chart. We recommend to use a namespace which is different from the one used for yb-operator.

Values

Please note that the node group for yb-monitoring workloads is managed by the yellowbrick operator, and we recommend not changing the node selectors and tolerations in the values file below.

```yaml
fluent-bit:
  enabled: true
  image:
    repository: {image-repo}/yellowbrickdata/fluent-bit-plugin-loki
    tag: 2.8.8-13
  serviceAccount:
    annotations:
      eks.amazonaws.com/role-arn: { role-arn }
  nodeSelector: &nodeSelector
    cluster.yellowbrick.io/node_type: yb-mon-standard
  serviceAccount:
    annotations:
      eks.amazonaws.com/role-arn: {role-arn}
  tolerations: &tolerations
  - effect: NoSchedule
    key: cluster.yellowbrick.io/owned
    operator: Equal
    value: "true"
grafana:
  deploymentStrategy:
    type: Recreate
  downloadDashboardsImage:
    repository: {image-repo}/curlimages/curl
    tag: 8.11.1
  image:
    repository: {image-repo}/grafana/grafana
    tag: 12.0.0
  initChownData:
    image:
      repository: {image-repo}/library/busybox
      tag: 1.31.1
  persistence:
    storageClassName: yb-gp3
  sidecar:
    image:
      repository: {image-repo}/kiwigrid/k8s-sidecar
      tag: 1.28.0
  nodeSelector: *nodeSelector
  tolerations: *tolerations
ingress:
  enabled: false
loki:
  extraContainers:
  - command:
    - /bin/sh
    - -c
```

```
      - |-
        trap cleanup 15
        cleanup()
        {
            echo "Shutting down the loki pvc monitor"
            exit
        }

        while true; do
            /delete_files_if_low_memory.sh
            sleep 360 &
            PID=$!
            wait $PID
        done;
      env:
      - name: SPACEMONITORING_FOLDER
        value: /data/loki/chunks
      image: {image-repo}/yellowbrickdata/loki-log-trimmer:v5
      name: pvcleanup
      volumeMounts:
      - mountPath: /data
        name: storage
    image:
      repository: {image-repo}/grafana/loki
      tag: 3.5.0
    persistence:
      size: 200Gi
      storageClassName: {storageclass}
    nodeSelector: *nodeSelector
    tolerations: *tolerations
  prometheus:
    alertmanager:
      enabled: false
      image:
        repository: {image-repo}/prometheus/alertmanager
        tag: v0.27.0
      nodeSelector: *nodeSelector
      tolerations: *tolerations
    alertmanagerFiles: {}
    configmapReload:
      alertmanager:
        enabled: true
        image:
          repository: {image-repo}/jimmidyson/configmap-reload
          tag: v0.8.0
      prometheus:
        image:
          repository: {image-repo}/jimmidyson/configmap-reload
          tag: v0.8.0
    kube-state-metrics:
      image:
        repository: {image-repo}/kube-state-metrics/kube-state-metrics
        tag: v2.13.0
      nodeSelector: *nodeSelector
      tolerations: *tolerations
    nodeExporter:
      image:
        repository: {image-repo}/prometheus/node-exporter
        tag: v1.8.0
      nodeSelector: *nodeSelector
      tolerations: *tolerations
    processExporter:
      image:
        repository: {image-repo}/ncabatoff/process-exporter
        tag: sha-7ef0b73
      nodeSelector: *nodeSelector
      tolerations: *tolerations
    pushgateway:
```

```
    image:
      repository: {image-repo}/prom/pushgateway
      tag: v1.9.0
    nodeSelector: *nodeSelector
    tolerations: *tolerations
  server:
    image:
      repository: {image-repo}/prometheus/prometheus
      tag: v2.49.1
    persistentVolume:
      enabled: true
      size: 100Gi
      storageClass: {storageclass}
    nodeSelector: *nodeSelector
    tolerations: *tolerations
    extraInitContainers:
    - image: {image-repo}/library/busybox:1.31.1
      name: prometheus-wal-cleanup
      command:
      - /bin/sh
      - -c
      - if [ $(du -sm /data/wal | cut -f1) -gt 1024 ]; then rm -rf /data/wal/*; fi
      volumeMounts:
      - mountPath: /data
        name: storage-volume
  serverFiles:
    alerting_rules.yml:
      groups:
      - name: Host alerts
        rules:
        - alert: PVCUtilizationHigh
          annotations:
            message: The persistentVolume used by {{ $labels.persistentvolumeclaim
              }} is {{ $value | humanize }}% utilized. Please check and take appropriate
              action.
            summary: PVC utilization on PVC {{ $labels.persistentvolumeclaim }} is
              high
          expr: 100 * sum(kubelet_volume_stats_used_bytes) by(persistentvolumeclaim)
            /sum(kubelet_volume_stats_capacity_bytes) by (persistentvolumeclaim) >
            90
          for: 5m
          labels:
            severity: warning
        - alert: HostOutOfDiskSpace
          annotations:
            description: |-
              Disk is almost full (< 80% left)
                VALUE = {{ $value | humanize }}
                LABELS: {{ $labels }}
            message: |-
              Disk is almost full (< 80% left)
                VALUE = {{ $value | humanize }}%
                Node Name: {{ $labels.node }}
            summary: Host out of disk space (instance {{ $labels.node }})
          expr: 100 - ((node_filesystem_avail_bytes{mountpoint="/"} * 100) / node_filesystem_size_bytes{mountpoint="/"})
            > 80
          for: 1s
          labels:
            severity: warning
```

Creating Cloud Infrastructure

AWS

When installing on AWS, an IRSA service account is used. For details on IRSA, please see the AWS documentation.

Create the IAM role:

```bash
aws iam create-role \
  --role-name yb-eks-pod-fluent-bit-{instance-name}-{region} \
  --assume-role-policy-document file://trust-policy.json
```

The trust policy:

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "{oidc-provider-arn}"
     },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "{oidc-provider}:sub": "system:serviceaccount:{namespace}:yb-{namespace}-worker-sa"
       }
     }
   }
  ]
}
```

Create the IAM policy

```bash
aws iam put-role-policy \
  --role-name yb-eks-pod-fluent-bit-{instance-name}-{region} \
  --policy-name diags-upload \
  --policy-document file://iam-policy.json
```

The IAM policy:

```json
{
  "Version": "2012-10-17",
  "Statement": [
      {
          "Effect": "Allow",
          "Action": [
              "s3:*"
          ],
          "Resource": "arn:aws:s3:::{observability-storage}/*"
      }
  ]
}
```

To the values above, add these values in the `fluent-bit` block and include the ARN of the AWS IAM role in place of `{role-arn}`:

```yaml
fluent-bit:
...
  serviceAccount:
    annotations:
      eks.amazonaws.com/role-arn: { role-arn }
...
```

Permissions

Platforms: EE: All cloud platforms

Parent topic: Installing

In this section:

The role used by the Deployer needs specific permissions granted in order to be able to complete successfully. If you launch the Deployer through the provided installer templates (in Amazon CloudFormation, Microsoft Azure Resource Manager, or GCP Deployment Manager) a role with the appropriate permissions will be created automatically and used to launch the Deployer.

If you want to launch the Deployer programatically, without the preconfigured templates, you'll need to create a role with suitable permissions to run the Deployer. The permissions required for each cloud provider are documented in this section.

Creating a role with these permissions may require help from another person or organization with sufficient privileges to do so.

AWS Installation Permissions

The following permissions are required to run the Deployer on AWS:

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AutoscalingList",
            "Effect": "Allow",
            "Action": [
                "autoscaling:DescribeAutoScalingGroups",
                "autoscaling:DescribeScalingActivities"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AutoscalingWrite",
            "Effect": "Allow",
            "Action": [
                "autoscaling:CreateOrUpdateTags"
            ],
            "Resource": "*"
        },
        {
            "Sid": "EC2List",
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeAccountAttributes",
                "ec2:DescribeAddresses",
                "ec2:DescribeAddressesAttribute",
                "ec2:DescribeAddressTransfers",
                "ec2:DescribeAvailabilityZones",
                "ec2:DescribeCustomerGateways",
                "ec2:DescribeDhcpOptions",
                "ec2:DescribeEgressOnlyInternetGateways",
                "ec2:DescribeHosts",
                "ec2:DescribeImages",
                "ec2:DescribeInstanceConnectEndpoints",
                "ec2:DescribeInstances",
                "ec2:DescribeInstanceStatus",
                "ec2:DescribeInternetGateways",
                "ec2:DescribeKeyPairs",
                "ec2:DescribeLaunchTemplates",
                "ec2:DescribeLaunchTemplateVersions",
                "ec2:DescribeNatGateways",
                "ec2:DescribeNetworkAcls",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribePlacementGroups",
                "ec2:DescribeRegions",
                "ec2:DescribeRouteTables",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSnapshots",
                "ec2:DescribeSubnets",
                "ec2:DescribeTags",
                "ec2:DescribeVolumes",
```

```
                "ec2:DescribeVolumeStatus",
                "ec2:DescribeVpcEndpoints",
                "ec2:DescribeVpcEndpointServiceConfigurations",
                "ec2:DescribeVpcPeeringConnections",
                "ec2:DescribeVpcs",
                "ec2:DescribeVpnConnections",
                "ec2:DescribeVpnGateways"
            ],
            "Resource": "*"
        },
        {
            "Sid": "EC2TagOnWrite",
            "Effect": "Allow",
            "Action": [
                "ec2:CreateTags"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "ec2:CreateAction": [
                        "AllocateAddress",
                        "CreateInternetGateway",
                        "CreateLaunchTemplate",
                        "CreateNatGateway",
                        "CreatePlacementGroup",
                        "CreateRouteTable",
                        "CreateSubnet",
                        "CreateVpc",
                        "CreateVpcEndpoint",
                        "RunInstances"
                    ]
                }
            }
        },
        {
            "Sid": "EC2Write",
            "Effect": "Allow",
            "Action": [
                "ec2:ModifyNetworkInterfaceAttribute"
            ],
            "Resource": "*"
        },
        {
            "Sid": "EC2WriteWithTags",
            "Effect": "Allow",
            "Action": [
                "ec2:AllocateAddress",
                "ec2:CreateInternetGateway",
                "ec2:CreateLaunchTemplate",
                "ec2:CreateNatGateway",
                "ec2:CreatePlacementGroup",
                "ec2:CreateRouteTable",
                "ec2:CreateSubnet",
                "ec2:CreateVpc",
                "ec2:CreateVpcEndpoint",
                "ec2:RunInstances"
            ],
            "Resource": "*",
            "Condition": {
                "Null": {
                    "aws:TagKeys": "false"
                },
                "StringEquals": {
                    "aws:RequestTag/cluster_yellowbrick_io_creator": "${aws:PrincipalTag/cluster_yellowbrick_io_creator}",
                    "aws:RequestTag/cluster_yellowbrick_io_owner": "${aws:PrincipalTag/cluster_yellowbrick_io_owner}"
                }
            }
        },
```

```
        {
            "Sid": "EC2WriteTaggedResource",
            "Effect": "Allow",
            "Action": [
                "ec2:AssociateRouteTable",
                "ec2:AttachInternetGateway",
                "ec2:CreateLaunchTemplateVersion",
                "ec2:CreateNatGateway",
                "ec2:CreateRoute",
                "ec2:CreateRouteTable",
                "ec2:CreateSubnet",
                "ec2:CreateVpcEndpoint",
                "ec2:DeleteInternetGateway",
                "ec2:DeleteLaunchTemplate",
                "ec2:ModifyLaunchTemplate",
                "ec2:DeleteNatGateway",
                "ec2:DeletePlacementGroup",
                "ec2:DeleteRouteTable",
                "ec2:DeleteSubnet",
                "ec2:DeleteVpc",
                "ec2:DeleteVpcEndpoints",
                "ec2:DetachInternetGateway",
                "ec2:DisassociateRouteTable",
                "ec2:ModifyNetworkInterfaceAttribute",
                "ec2:ModifyVpcAttribute",
                "ec2:ReleaseAddress",
                "ec2:RunInstances"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/cluster_yellowbrick_io_creator": "${aws:PrincipalTag/cluster_yellowbrick_io_creator}",
                    "aws:ResourceTag/cluster_yellowbrick_io_owner": "${aws:PrincipalTag/cluster_yellowbrick_io_owner}"
                }
            }
        },
        {
            "Sid": "ECRRead",
            "Effect": "Allow",
            "Action": [
                "ecr:DescribeRepositories"
            ],
            "Resource": "arn:*:ecr:*:*:repository/*"
        },
        {
            "Sid": "ECRReadTaggedResource",
            "Effect": "Allow",
            "Action": [
                "ecr:BatchCheckLayerAvailability",
                "ecr:BatchGetImage",
                "ecr:DescribeRepositories",
                "ecr:GetDownloadUrlForLayer",
                "ecr:ListTagsForResource"
            ],
            "Resource": "arn:*:ecr:*:*:repository/yb-*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/cluster_yellowbrick_io_owner": "${aws:PrincipalTag/cluster_yellowbrick_io_owner}"
                }
            }
        },
        {
            "Sid": "ECRToken",
            "Effect": "Allow",
            "Action": [
                "ecr:GetAuthorizationToken"
            ],
            "Resource": "*"
```

```
        },
        {
            "Sid": "ECRWriteTaggedResource",
            "Effect": "Allow",
            "Action": [
                "ecr:CompleteLayerUpload",
                "ecr:DeleteRepository",
                "ecr:InitiateLayerUpload",
                "ecr:PutImage",
                "ecr:UploadLayerPart"
            ],
            "Resource": "arn:*:ecr:*:*:repository/yb-*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/cluster_yellowbrick_io_owner": "${aws:PrincipalTag/cluster_yellowbrick_io_owner}"
                }
            }
        },
        {
            "Sid": "ECRWriteWithTags",
            "Effect": "Allow",
            "Action": [
                "ecr:CreateRepository",
                "ecr:TagResource"
            ],
            "Resource": "arn:*:ecr:*:*:repository/yb-*",
            "Condition": {
                "Null": {
                    "aws:TagKeys": "false"
                },
                "StringEquals": {
                    "aws:RequestTag/cluster_yellowbrick_io_creator": "${aws:PrincipalTag/cluster_yellowbrick_io_creator}",
                    "aws:RequestTag/cluster_yellowbrick_io_owner": "${aws:PrincipalTag/cluster_yellowbrick_io_owner}"
                }
            }
        },
        {
            "Sid": "EKSList",
            "Effect": "Allow",
            "Action": [
                "eks:DeleteAccessEntry",
                "eks:DescribeAccessEntry",
                "eks:DescribeCluster",
                "eks:ListAssociatedAccessPolicies",
                "eks:ListClusters"
            ],
            "Resource": "*"
        },
        {
            "Sid": "EKSReadTaggedResource",
            "Effect": "Allow",
            "Action": [
                "eks:DescribeAddon",
                "eks:DescribeUpdate",
                "eks:ListNodegroups"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/cluster_yellowbrick_io_owner": "${aws:PrincipalTag/cluster_yellowbrick_io_owner}"
                }
            }
        },
        {
            "Sid": "EKSDescribeNodegroup",
            "Effect": "Allow",
            "Action": [
                "eks:DescribeNodegroup"
```

```
            ],
            "Resource": "*",
            "Condition": {}
        },
        {
            "Sid": "EKSWriteWithNodegroupRole",
            "Effect": "Allow",
            "Action": [
                "iam:GetRole"
            ],
            "Resource": [
                "arn:*:iam::*:role/aws-service-role/eks-nodegroup.amazonaws.com/AWSServiceRoleForAmazonEKSNodegroup"
            ]
        },
        {
            "Sid": "EKSWriteWithNodegroupRun",
            "Effect": "Allow",
            "Action": [
                "ec2:RunInstances"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "EKSWriteTaggedResource",
            "Effect": "Allow",
            "Action": [
                "eks:AssociateAccessPolicy",
                "eks:CreateAccessEntry",
                "eks:CreateAddon",
                "eks:CreateNodegroup",
                "eks:DeleteCluster",
                "eks:DeleteNodegroup",
                "eks:DisassociateAccessPolicy",
                "eks:TagResource",
                "eks:UpdateAddon",
                "eks:UpdateClusterConfig",
                "eks:UpdateClusterVersion",
                "eks:UpdateNodegroupVersion"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/cluster_yellowbrick_io_owner": "${aws:PrincipalTag/cluster_yellowbrick_io_owner}"
                }
            }
        },
        {
            "Sid": "EKSWriteWithTags",
            "Effect": "Allow",
            "Action": [
                "eks:CreateAccessEntry",
                "eks:CreateAddon",
                "eks:CreateCluster",
                "eks:CreateNodegroup",
                "eks:TagResource"
            ],
            "Resource": "*",
            "Condition": {
                "Null": {
                    "aws:TagKeys": "false"
                },
                "StringEquals": {
                    "aws:RequestTag/cluster_yellowbrick_io_creator": "${aws:PrincipalTag/cluster_yellowbrick_io_creator}",
                    "aws:RequestTag/cluster_yellowbrick_io_owner": "${aws:PrincipalTag/cluster_yellowbrick_io_owner}"
                }
            }
```

```
        },
        {
            "Sid": "ELBList",
            "Effect": "Allow",
            "Action": [
                "elasticloadbalancing:DescribeLoadBalancers",
                "elasticloadbalancing:DescribeTargetGroups",
                "elasticloadbalancing:DescribeTags",
                "elasticloadbalancing:DeleteLoadBalancer"
            ],
            "Resource": "*"
        },
        {
            "Sid": "IAMPassRole",
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": "arn:*:iam::*:role/*",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": [
                        "autoscaling.amazonaws.com",
                        "ec2.amazonaws.com",
                        "eks.amazonaws.com"
                    ]
                }
            }
        },
        {
            "Sid": "IAMReadTaggedResource",
            "Effect": "Allow",
            "Action": [
                "iam:GetOpenIDConnectProvider",
                "iam:GetRole",
                "iam:GetRolePolicy",
                "iam:ListAttachedRolePolicies",
                "iam:ListRolePolicies"
            ],
            "Resource": [
                "arn:*:iam::*:oidc-provider/*",
                "arn:*:iam::*:role/*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/cluster_yellowbrick_io_owner": "${aws:PrincipalTag/cluster_yellowbrick_io_owner}"
                }
            }
        },
        {
            "Sid": "IAMWriteTaggedResource",
            "Effect": "Allow",
            "Action": [
                "iam:AttachRolePolicy",
                "iam:CreateRole",
                "iam:DeleteOpenIDConnectProvider",
                "iam:DeleteRole",
                "iam:DeleteRolePolicy",
                "iam:DetachRolePolicy",
                "iam:ListInstanceProfilesForRole",
                "iam:PutRolePolicy",
                "iam:RemoveRoleFromInstanceProfile",
                "iam:TagOpenIDConnectProvider",
                "iam:TagRole",
                "iam:UpdateAssumeRolePolicy"
            ],
            "Resource": "*",
            "Condition": {
```

```
                "StringEquals": {
                    "aws:ResourceTag/cluster_yellowbrick_io_owner": "${aws:PrincipalTag/cluster_yellowbrick_io_owner}"
                }
            }
        },
        {
            "Sid": "IAMWriteWithTags",
            "Effect": "Allow",
            "Action": [
                "iam:CreateOpenIDConnectProvider",
                "iam:CreateRole",
                "iam:TagOpenIDConnectProvider",
                "iam:TagRole"
            ],
            "Resource": "*",
            "Condition": {
                "Null": {
                    "aws:TagKeys": "false"
                },
                "StringEquals": {
                    "aws:RequestTag/cluster_yellowbrick_io_creator": "${aws:PrincipalTag/cluster_yellowbrick_io_creator}",
                    "aws:RequestTag/cluster_yellowbrick_io_owner": "${aws:PrincipalTag/cluster_yellowbrick_io_owner}"
                }
            }
        },
        {
            "Sid": "S3List",
            "Effect": "Allow",
            "Action": [
                "s3:ListAllMyBuckets"
            ],
            "Resource": "*"
        },
        {
            "Sid": "S3Read",
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation",
                "s3:GetBucketObjectLockConfiguration",
                "s3:GetBucketOwnershipControls",
                "s3:GetBucketPolicy",
                "s3:GetBucketVersioning",
                "s3:GetEncryptionConfiguration",
                "s3:ListBucket",
                "s3:ListBucketVersions"
            ],
            "Resource": "arn:*:s3:::yb-*"
        },
        {
            "Sid": "S3Write",
            "Effect": "Allow",
            "Action": [
                "s3:CreateBucket",
                "s3:DeleteBucket",
                "s3:PutBucketPolicy",
                "s3:PutBucketTagging",
                "s3:DeleteObject"
            ],
            "Resource": "arn:*:s3:::yb-*"
        },
        {
            "Sid": "ServiceLinkedRoleRead",
            "Effect": "Allow",
            "Action": [
                "iam:GetRole"
            ],
            "Resource": [
                "*"
```

```
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:AWSServiceName": [
                        "autoscaling.amazonaws.com",
                        "eks-nodegroup.amazonaws.com",
                        "eks.amazonaws.com",
                        "elasticloadbalancing.amazonaws.com"
                    ]
                }
            }
        },
        {
            "Sid": "SSMReadTaggedResource",
            "Effect": "Allow",
            "Action": [
                "ssm:GetParameters"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/cluster_yellowbrick_io_owner": "${aws:PrincipalTag/cluster_yellowbrick_io_owner}"
                }
            }
        },
        {
            "Sid": "STSRead",
            "Effect": "Allow",
            "Action": [
                "sts:GetCallerIdentity"
            ],
            "Resource": "*"
        }
    ]
}
```

Azure Installation Permissions

The following permissions are required to run the Deployer on Azure:

```json
[
    {
        "actions": [
            "Microsoft.Resources/subscriptions/resourcegroups/read",
            "Microsoft.Network/virtualNetworks/read",
            "Microsoft.Network/virtualNetworks/write",
            "Microsoft.Network/virtualNetworks/delete",
            "Microsoft.Network/virtualNetworks/subnets/read",
            "Microsoft.Network/virtualNetworks/subnets/join/action",
            "Microsoft.ContainerRegistry/registries/read",
            "Microsoft.ContainerRegistry/registries/write",
            "Microsoft.ContainerRegistry/registries/delete",
            "Microsoft.ContainerRegistry/registries/generateCredentials/action",
            "Microsoft.ContainerRegistry/registries/pull/read",
            "Microsoft.ContainerRegistry/registries/push/write",
            "Microsoft.ManagedIdentity/userAssignedIdentities/write",
            "Microsoft.ManagedIdentity/userAssignedIdentities/read",
            "Microsoft.ManagedIdentity/userAssignedIdentities/delete",
            "Microsoft.ContainerService/managedClusters/read",
            "Microsoft.ContainerService/managedClusters/write",
            "Microsoft.ContainerService/managedClusters/delete",
            "Microsoft.ContainerService/managedClusters/listClusterAdminCredential/action",
            "Microsoft.ContainerService/managedClusters/agentPools/read",
            "Microsoft.ContainerService/managedClusters/agentPools/write",
            "Microsoft.ContainerService/managedClusters/agentPools/delete",
            "Microsoft.Authorization/roleAssignments/read",
            "Microsoft.Authorization/roleAssignments/write",
            "Microsoft.Authorization/roleAssignments/delete",
            "Microsoft.Authorization/roleDefinitions/read",
            "Microsoft.Authorization/roleDefinitions/write",
            "Microsoft.Authorization/roleDefinitions/delete",
            "Microsoft.ManagedIdentity/userAssignedIdentities/federatedIdentityCredentials/read",
            "Microsoft.ManagedIdentity/userAssignedIdentities/federatedIdentityCredentials/write",
            "Microsoft.ManagedIdentity/userAssignedIdentities/federatedIdentityCredentials/delete",
            "Microsoft.Storage/storageAccounts/read",
            "Microsoft.Storage/storageAccounts/write",
            "Microsoft.Storage/storageAccounts/delete",
            "Microsoft.Storage/storageAccounts/blobServices/containers/read",
            "Microsoft.Storage/storageAccounts/blobServices/containers/write",
            "Microsoft.OperationalInsights/workspaces/sharedkeys/read"
        ]
    }
]
```

GCP Installation Permissions

The following permissions are required to run the Deployer on GCP:

```json
[
    "artifactregistry.repositories.create",
    "artifactregistry.repositories.delete",
    "artifactregistry.repositories.deleteArtifacts",
    "artifactregistry.repositories.downloadArtifacts",
    "artifactregistry.repositories.get",
    "artifactregistry.repositories.list",
    "artifactregistry.repositories.update",
    "artifactregistry.repositories.uploadArtifacts",
    "compute.networks.access",
    "compute.networks.create",
    "compute.networks.delete",
    "compute.networks.get",
    "compute.networks.list",
    "compute.networks.update",
    "compute.networks.updatePolicy",
    "compute.networks.use",
    "compute.routers.create",
    "compute.routers.get",
    "compute.routers.delete",
    "compute.routers.list",
    "compute.subnetworks.create",
    "compute.subnetworks.delete",
    "compute.subnetworks.get",
    "compute.subnetworks.list",
    "compute.subnetworks.update",
    "compute.subnetworks.use",
    "compute.firewalls.list",
    "compute.firewalls.delete",
    "compute.zones.list",
    "container.apiServices.create",
    "container.apiServices.delete",
    "container.apiServices.get",
    "container.apiServices.getStatus",
    "container.apiServices.list",
    "container.apiServices.update",
    "container.apiServices.updateStatus",
    "container.bindings.create",
    "container.bindings.delete",
    "container.bindings.get",
    "container.bindings.list",
    "container.bindings.update",
    "container.clusterRoleBindings.create",
    "container.clusterRoleBindings.delete",
    "container.clusterRoleBindings.get",
    "container.clusterRoleBindings.list",
    "container.clusterRoleBindings.update",
    "container.clusterRoles.bind",
    "container.clusterRoles.create",
    "container.clusterRoles.delete",
    "container.clusterRoles.escalate",
    "container.clusterRoles.get",
```

```
        "container.clusterRoles.list",
        "container.clusterRoles.update",
        "container.clusters.connect",
        "container.clusters.create",
        "container.clusters.delete",
        "container.clusters.get",
        "container.clusters.getCredentials",
        "container.clusters.list",
        "container.clusters.update",
        "container.configMaps.create",
        "container.configMaps.delete",
        "container.configMaps.get",
        "container.configMaps.list",
        "container.configMaps.update",
        "container.customResourceDefinitions.create",
        "container.customResourceDefinitions.delete",
        "container.customResourceDefinitions.get",
        "container.customResourceDefinitions.getStatus",
        "container.customResourceDefinitions.list",
        "container.customResourceDefinitions.update",
        "container.customResourceDefinitions.updateStatus",
        "container.daemonSets.create",
        "container.daemonSets.delete",
        "container.daemonSets.get",
        "container.daemonSets.getStatus",
        "container.daemonSets.list",
        "container.daemonSets.update",
        "container.daemonSets.updateStatus",
        "container.deployments.create",
        "container.deployments.delete",
        "container.deployments.get",
        "container.deployments.getScale",
        "container.deployments.getStatus",
        "container.deployments.list",
        "container.deployments.update",
        "container.endpoints.create",
        "container.endpoints.delete",
        "container.endpoints.get",
        "container.endpoints.list",
        "container.endpoints.update",
        "container.events.get",
        "container.events.list",
        "container.ingresses.create",
        "container.ingresses.delete",
        "container.ingresses.get",
        "container.ingresses.getStatus",
        "container.ingresses.list",
        "container.ingresses.update",
        "container.ingresses.updateStatus",
        "container.jobs.create",
        "container.jobs.delete",
        "container.jobs.get",
        "container.jobs.list",
        "container.jobs.update",
        "container.mutatingWebhookConfigurations.create",
        "container.mutatingWebhookConfigurations.delete",
        "container.mutatingWebhookConfigurations.get",
        "container.mutatingWebhookConfigurations.list",
        "container.mutatingWebhookConfigurations.update",
        "container.namespaces.create",
        "container.namespaces.delete",
        "container.namespaces.finalize",
        "container.namespaces.get",
        "container.namespaces.list",
        "container.namespaces.update",
        "container.networkPolicies.create",
        "container.networkPolicies.delete",
        "container.networkPolicies.get",
```

```
    "container.networkPolicies.list",
    "container.networkPolicies.update",
    "container.nodes.create",
    "container.nodes.delete",
    "container.nodes.get",
    "container.nodes.getStatus",
    "container.nodes.list",
    "container.nodes.update",
    "container.operations.get",
    "container.operations.list",
    "container.persistentVolumeClaims.create",
    "container.persistentVolumeClaims.delete",
    "container.persistentVolumeClaims.get",
    "container.persistentVolumeClaims.list",
    "container.persistentVolumeClaims.update",
    "container.persistentVolumes.create",
    "container.persistentVolumes.delete",
    "container.persistentVolumes.get",
    "container.persistentVolumes.list",
    "container.persistentVolumes.update",
    "container.podDisruptionBudgets.create",
    "container.podDisruptionBudgets.delete",
    "container.podDisruptionBudgets.get",
    "container.podDisruptionBudgets.list",
    "container.podDisruptionBudgets.update",
    "container.podSecurityPolicies.create",
    "container.podSecurityPolicies.delete",
    "container.podSecurityPolicies.get",
    "container.podSecurityPolicies.list",
    "container.podSecurityPolicies.update",
    "container.podSecurityPolicies.use",
    "container.podTemplates.create",
    "container.podTemplates.delete",
    "container.podTemplates.get",
    "container.podTemplates.list",
    "container.podTemplates.update",
    "container.pods.attach",
    "container.pods.create",
    "container.pods.delete",
    "container.pods.evict",
    "container.pods.exec",
    "container.pods.get",
    "container.pods.getLogs",
    "container.pods.getStatus",
    "container.pods.initialize",
    "container.pods.list",
    "container.pods.portForward",
    "container.pods.update",
    "container.replicaSets.create",
    "container.replicaSets.delete",
    "container.replicaSets.get",
    "container.replicaSets.getScale",
    "container.replicaSets.list",
    "container.replicaSets.update",
    "container.replicaSets.updateScale",
    "container.roleBindings.create",
    "container.roleBindings.delete",
    "container.roleBindings.get",
    "container.roleBindings.list",
    "container.roleBindings.update",
    "container.roles.bind",
    "container.roles.create",
    "container.roles.delete",
    "container.roles.escalate",
    "container.roles.get",
    "container.roles.list",
    "container.roles.update",
    "container.secrets.create",
```

```
      "container.secrets.delete",
      "container.secrets.get",
      "container.secrets.list",
      "container.secrets.update",
      "container.serviceAccounts.create",
      "container.serviceAccounts.createToken",
      "container.serviceAccounts.delete",
      "container.serviceAccounts.get",
      "container.serviceAccounts.list",
      "container.serviceAccounts.update",
      "container.services.create",
      "container.services.delete",
      "container.services.get",
      "container.services.list",
      "container.services.update",
      "container.statefulSets.create",
      "container.statefulSets.delete",
      "container.statefulSets.get",
      "container.statefulSets.getStatus",
      "container.statefulSets.list",
      "container.statefulSets.update",
      "container.storageClasses.create",
      "container.storageClasses.delete",
      "container.storageClasses.get",
      "container.storageClasses.list",
      "container.storageClasses.update",
      "container.thirdPartyObjects.create",
      "container.thirdPartyObjects.delete",
      "container.thirdPartyObjects.get",
      "container.thirdPartyObjects.list",
      "container.thirdPartyObjects.update",
      "container.validatingWebhookConfigurations.create",
      "container.validatingWebhookConfigurations.delete",
      "container.validatingWebhookConfigurations.get",
      "container.validatingWebhookConfigurations.list",
      "container.validatingWebhookConfigurations.update",
      "container.volumeAttachments.create",
      "container.volumeAttachments.delete",
      "container.volumeAttachments.get",
      "container.volumeAttachments.list",
      "container.volumeAttachments.update",
      "iam.roles.create",
      "iam.roles.delete",
      "iam.roles.get",
      "iam.roles.list",
      "iam.roles.update",
      "iam.roles.undelete",
      "iam.serviceAccounts.actAs",
      "iam.serviceAccounts.create",
      "iam.serviceAccounts.delete",
      "iam.serviceAccounts.get",
      "iam.serviceAccounts.getAccessToken",
      "iam.serviceAccounts.getIamPolicy",
      "iam.serviceAccounts.list",
      "iam.serviceAccounts.setIamPolicy",
      "iam.serviceAccounts.update",
      "resourcemanager.projects.get",
      "resourcemanager.projects.getIamPolicy",
      "resourcemanager.projects.setIamPolicy",
      "storage.buckets.create",
      "storage.buckets.delete",
      "storage.buckets.get",
      "storage.buckets.list",
      "storage.buckets.update",
      "storage.objects.create",
      "storage.objects.delete",
      "storage.objects.get",
      "storage.objects.getIamPolicy",
```

```
    "storage.objects.list",
    "storage.objects.update",
    "cloudkms.keyRings.getIamPolicy",
    "cloudkms.keyRings.setIamPolicy"
]
```

Access Key

Platforms: EE: All cloud platforms

Parent topic: Installing

When using the Deployer launched from a cloud provider template (AWS, Azure, GCP), a key called the `Deployer Access Key` is required to proceed. Its role is to protect the hosted web application from arbitrary users. If the Deployer VM is suspended, restarted or destroyed, or launched from the CLI, this access key is no longer valid. The Deployer will not function if this key is not entered correctly on the welcome page.

When launching from the cloud provider, the deployer access key is provided in the following locations:

Cloud Provider	Location
AWS	in the CloudFormation output variables
Azure	in the Deployment outputs section
GCP	in the console printed from the Deployer VM, similar to below

```
|
|  Please visit the URL https://<ipaddress>:<port> and provide the Deployer Access Key 8d9afd724bf89c111ff69efbfe499fa6
|
```

Similarly, if launched from the cloud provider without the deployment template provided by Yellowbrick, the access key is found in the Deployer VM logs. To directly obtain the Access Key key from the Deployer VM:

- SCP file `/tmp/install.log` or
- SSH to the launched deployer VM and start `journalctl -u yb-install.service`

Deployer Web UI Notes

Provisioning cloud infrastructure can very occasionally fail in unexpected ways. If this happens to you, or you'd like help or advice, please contact us at deployer-support@yellowbrick.com or, if you're an existing customer, file a support ticket.

If you can, please attach a debug log, which you can download by selecting *Actions → Download Debug Log* at the bottom right of the Deployer window.

The remainder of this document comprises notes on specific fields and screens in the Deployer.

TIP

If you start an install, and hit cancel, it will not remove partially created infrastructure. Run an *Uninstall* to do so instead.

Welcome

When you start the deployer, an access key should have been prepopulated and you can just keep going. The access key is a way to make sure random internet users don't stumble upon your deployer and mess with it.

If you suspend and resume the deployer you may need to repopulate the access key. See here for instructions on how to do so.

Deployer

On the Deployer page, you can choose to Install, Upgrade or Uninstall Yellowbrick. If you're part way through the process, a *Continue* option will also be available.

Install

Start an installation. Much of the installation process is gathering and validating information. At the end of the process, before the installation is actually performed, there's an option available to download an installation manifest or JSON configuration file without actually doing anything. The installation manifest shows, in a human readable or printable form, all of the cloud infrastructure components (such as VPCs, role names, service account names, node groups, etc) that would be created by an installation; it's ideal for showing other departments in larger organizations. The JSON configuration is the starting point for invoking a command line installation. If you quit the deployer without installing, you can always come back and pick up where you left off.

Continue

This operation is only visible if you started down the installation path but didn't complete the operation.

Upgrade

The Upgrade operation's primary function is to push a new version of the Yellowbrick database software to the artifact registry. It also triggers an immediate update and restart of the Yellowbrick Operator and Yellowbrick Manager. None of these operations result in downtime for Yellowbrick instances or users of the database. The database itself can be upgraded to the new version at a later date, using Yellowbrick Manager by navigating to *Instance Management → Actions → Upgrade*.

On rare occasions, running an upgrade will also upgrade cloud infrastructure, such as Kubernetes, to a newer version. These upgrades *will* require scheduled downtime. In such cases, the Deployer will warn you in red on the screen and with extra prompts.

Uninstall

This operation uninstalls a previously installed instance, Yellowbrick Operator and Yellowbrick Manager. If, for whatever reason, a previous installation failed, you'll need to Uninstall here and then try again.

Information

This page is available for Upgrade and Uninstall operations only. For Upgrades and Uninstalls, you'll need to re-enter the details of the cloud provider and other associated fields so that the Deployer can locate the Yellowbrick deployment. If during installation you customized the Kubernetes cluster name or namespace name, you'll also need to re-enter them here.

Provider

Instance name

This is the most important field; it's used by default for the Kubernetes cluster name, namespace name, along with many other infrastructure objects. You can override some of these names in the *Advanced* section, but if you do so, remember you'll also need to re-enter them during upgrades. The field is 20 alphanumeric characters long.

Provider

Choose which cloud provider you want to install into.

Region

The regions in the dropdown list have been formally qualified by Yellowbrick for production use. If you'd like to try one that isn't on the list, feel free to type it in but please let us know by emailing deployer-support@yellowbrick.com.

Various cloud-specific fields

For other cloud providers, you may need to enter other deployment information here, such as a project ID or subscription etc.

Advanced → Kubernetes Cluster/Namespace Name

You can provide names different from the instance name if you like.

Advanced → Block Storage Custom Encryption Key ID

Yellowbrick block storage volumes (also known as EBS volumes) are encrypted by default using cloud provider managed keys. Using this option lets you supply your own custom key. Note that if the key is deleted, the data may be irreversibly lost. We recommend not using this option unless you have sufficient experience with cloud KMS, key versions, and key rotation processes. Each cloud provider uses a different format for the key ID, and only the corresponding format for that provider will function correctly.

For AWS, use the full ARN of the key in the KMS, for example `arn:aws:kms:us-west-2:012345678901:key/deadbeef-dead-beef-dead-beefdeadbeef`.

For Azure, use the resource ID of the Disk Encryption Set, including the subscription ID and resource group name, for example `/subscriptions/deadbeef-dead-beef-dead-beefdeadbeef/resourceGroups/resource-group-name/providers/Microsoft.Compute/diskEncryptionSets/key-name`.

For GCP, the format is the full ID of the key in the KMS, for example `projects/0123456789/locations/us-east1/keyRings/key-ring-name/cryptoKeys/key-name`.

Advanced → Custom Tags/Labels

See resource tagging and costing.

Restrict Access

A default *public* installation of Yellowbrick sets the Kubernetes API, listener pod (PostgreSQL port 5432 along with the Yellowbrick tools data and service ports) and Yellowbrick Manager to be publicly routable. Note that despite being routable, the Kubernetes API is protected by client certificates tied to logins (normally with MFA), and the other ports can be

protected by single sign-on. Even in these public installations, all of the workers and paths Kubernetes workers are not routable from public networks, and access to object store is via private endpoints. So public installs are still highly secure.

By setting up network access restrictions, you can add allow lists for access to the public ports and the Kubernetes API. Be really careful when changing these to not block access from the machine the deployer is running on or the machine you want to access the installed product from!

Yellowbrick Manager allows changing all of the allow lists after installation, except for the Kubernetes API which has to be changed via the cloud provider's management console. Unless you are particularly paranoid, there is no reason to restrict access at this point in the installation.

If you choose the private routing option, you'll need to follow the instructions for private installation first.

Network

This is where you specify the network the instance runs in. Our guidance is to make this network as large as possible; we recommend starting with a `/16` network. You also need to nominate a first subnet to install into (or, for AWS, three subnets, two for autoscaler HA and a third for the NAT gateway for AWS API access).

Make the subnet as large as possible too. Kubernetes consumes a large number of IP addresses (one per pod, not one per instance). The larger the subnet, the more compute cluster nodes can be accommodated in future. The GUI will estimate roughly how many nodes you have room for as you modify the subnet.

> **WARNING**
>
> These networks **can't be resized after installation**, so be sure to plan for several years' future growth.

Initial Account

The instance has an initial administrator account. By default we auto-generate a strong password which you can download, view or copy to the clipboard. Or you can enter your own credentials here, if you like.

Storage

Columnar data in Yellowbrick is persisted on object storage. By default we create storage buckets as part of the installation. You can choose to make your own object storage bucket afterwards by following the custom object storage documentation.

Install

This is the last page in the installation sequence that shows you all actions the Deployer is planning to undertake. You can review the infrastructure manifest here or export a JSON configuration file by pressing the *Actions* button at the bottom right of the window.

Resource Tagging and Costing

Platforms: EE: All cloud platforms

Parent topic: Installing

The Deployer adds ownership tags on all of the cloud resources it creates so it can track its actions. This is also convenient for cost accounting.

The default tags are as follows:

Key	Value
cluster_yellowbrick_io_creator	yb-install
cluster_yellowbrick_io_owner	yb-install
cluster_yellowbrick_io_instance	`<instance-name>`

If a cloud resource does not have these tags, or if the values are unexpected, it will not be destroyed during the uninstallation process.

Custom tags

Additional tags can be added to the cloud resources to be created by using the `--tags/-t` flag in the install command or in the web user interface. These tags will not be checked at uninstallation time.

Please be aware that each cloud provider may have specific constraints regarding the format of tags.

On Google Cloud Platform, labels are used to store the ownership and custom tags, as they are more suited for tracking and reporting costs.

Cloud: Kubernetes Guides

Platforms: EE: All cloud platforms

Parent topic: Cloud

In this section:

Yellowbrick Enterprise Edition is built on standard Kubernetes constructs, such as stateful sets, deployments, services and load balancers. As such, standard Kubernetes tooling works to inspect, manage and monitor the resources and infrastructure that a Yellowbrick instance runs on.

Yellowbrick Enterprise Edition includes a custom Operator which controls the lifecycle of Yellowbrick instances. This Operator utilizes Custom Resource Definitions (CRDs) to manage instances and their companion resources. See Yellowbrick Operator for more information.

Kubernetes Configuration (`KUBECONFIG`)

How you configure Kubernetes for command line and other API clients depends on which cloud your Yellowbrick instance is deployed on:

- AWS Elastic Kubernetes Service
- Microsoft Azure Kubernetes Service
- Google Kubernetes Engine

Kubernetes Recommendations for Yellowbrick

Generally, it is NOT recommended to scale resources manually utilizing `kubectl` and related tooling. So while it is possible to use `kubectl scale` to spin down an instance pod, this will not gracefully shutdown companion services and such. Furthermore, direct modifications to service definitions, such as adding annotations and what/not for cloud-provider specific features, is not supported.

It is recommended to use Yellowbrick Manager for suspending, resuming, changing, upgrading, and tuning Yellowbrick instances. If you desire to script such functionality, please see the Kubectl Administration for how to use kubectl and related tooling to accomplish these tasks.

Kubeconfig Setup on AKS

Platforms: EE: All cloud platforms

Parent topic: Kubernetes Guides

Obtaining a Kubernetes Configuration for AKS

Given that you know:

- the name of the resource group you installed to `$rg`
- the name of the Kubernetes cluster you installed `$cluster`
- the region you installed to `$region`

You can use the `az` command line tool to export a Kubernetes config file:

```bash
az aks get-credentials --name $cluster --file ~/.kube/aks/$cluster-$region.config --resource-group $rg --admin
```

Setup for Azure Console

You may see the following note in your Azure console for the AKS cluster of your Yellowbrick instance.

INFO

The client '{your-account-name}' with object id '{identifier}' does not have authorization to perform action 'Microsoft.Resources/subscriptions/resourceGroups/read' over scope '/subscriptions/{subscription}/resourceGroups/{resourcegroup}' or the scope is invalid. If access was recently granted, please refresh your credentials.

To resolve this, configure your role to add a Role Assignment of `Azure Kubernetes Service Cluster Admin Role` under `Access control (IAM)` in the Azure console for the AKS cluster.

Kubeconfig Setup on EKS

Platforms: EE: All cloud platforms

Parent topic: Kubernetes Guides

Grant Access to EKS

To access the EKS Kubernetes API, you must grant access via Access Entries. See the AWS documentation for details on this topic.

AWS Console

Configure your IAM role under the `Access` tab in the AWS console for the EKS cluster by adding your role to the `IAM access entries` section with an access policy that is appropriate for your requirements. For example, to manage all resources in the cluster, configure the `AmazonEKSClusterAdminPolicy` at the `Cluster` access scope.

AWS CLI

First, identity your IAM principal ARN (role or user). If you don't know whether you are logged in as a user or assuming a role, this can be found by using `aws sts get-caller-identity`. If your identity ARN includes `assumed-role`, you are assuming a role, otherwise if it includes `user` then you are a user.

This is an example of a user:

```bash
{
    "UserId": "AIDAQPFEP7X2BWRP3L3FO",
    "Account": "012345678901",
    "Arn": "arn:aws:iam::012345678901:user/example-user-name"
}                                       ^---------------^
                                                | This is your user name
```

This is an example of an assumed role:

```bash
{
    "UserId": "AROAQPFEP7Y1JNOB3EYME:example@yellowbrick.com",
    "Account": "012345678901",
    "Arn": "arn:aws:sts::012345678901:assumed-role/example-role-name/example@yellowbrick.com"
}                                               ^---------------^
                                                        | This is your role name
```

When you know your user or role name, you can query the full ARN.

To list an IAM user ARN:

```bash
aws iam get-user \
    --user-name "$user_name" \
    --query "User.Arn" \
    --output text
```

To list an IAM role ARN:

```bash
aws iam get-role \
    --role-name "$role_name" \
    --query "Role.Arn" \
    --output text
```

Now that you know your princiapal ARN, create an EKS Access Entry and grant it a desired access policy. To grant the user access to all namespaces in the cluster, give it the `AmazonEKSClusterAdminPolicy` access policy:

```bash
aws eks create-access-entry \
    --cluster-name "$cluster" \
    --principal-arn "$principal_arn"

aws eks associate-access-policy \
    --cluster-name "$cluster" \
    --principal-arn "$principal_arn" \
    --policy-arn "arn:aws:eks::aws:cluster-access-policy/AmazonEKSClusterAdminPolicy" \
    --access-scope "type=cluster"
```

Please note that `AmazonEKSClusterAdminPolicy` is an AWS-managed policy. If you are running on a different partition such as GovCloud, you will need to adjust your ARNs to have the proper partition. If you are unsure about the correct policy ARN, you can see the proper value with:

```bash
aws eks list-access-policies \
    --output table
```

To see more information on available access policies, see the AWS associate access policies with access entries documentation.

Obtaining a Kubernetes Configuration for EKS

Given that you know:

- the name of the Kubernetes cluster you installed `$cluster`
- the region you installed to `$region`

You can use the `aws` command line tool to export a Kubernetes config file:

```bash
aws eks update-kubeconfig --region $region --name $cluster --kubeconfig ~/.kube/aws/$cluster-$region.config --alias aws-$cluster-$
```

Setup for AWS Console

You may see the following note in your AWS console for the EKS cluster of your Yellowbrick instance.

> **INFO**
>
> **Your current IAM principal doesn't have access to Kubernetes objects on this cluster.**
>
> This may be due to the current user or role not having Kubernetes RBAC permissions to describe cluster resources or not having an entry in the cluster's auth config map.

To resolve this, follow the instructions provided above in Grant Access to EKS. Ensure that the Access Policy granted provides access to the namespace you are attempting to access.

Kubeconfig Setup on GKE

Platforms: EE: All cloud platforms

Parent topic: Kubernetes Guides

Obtaining a Kubernetes Configuration for GKE

Given that you know:

- the name of the Kubernetes cluster you installed `$cluster`
- the region you installed to `$region`

You can use the `gcloud` command line tool to export a Kubernetes config file:

```bash
KUBECONFIG=~/.kube/gke/$cluster-$region.config gcloud container clusters get-credentials $cluster --zone $region
```

Kubectl Administration

Platforms: EE: All cloud platforms

Parent topic: Kubernetes Guides

This administration guide lists `kubectl` commands that can be used to manage Yellowbrick instances. All of these commands use the YBInstance CRD which is installed and managed by the Yellowbrick Operator.

Given that you know:

- The namespace where your instance is installed `$instanceNamespace`
- The name of your instance `$instanceName`

the following activities can be performed:

Suspend an Instance

```
kubectl -n $instanceNamespace patch ybinstance/$instanceName --type=merge -p '{"spec":{"requestedState":"Suspended"}}'
```

Resume an Instance

```
kubectl -n $instanceNamespace patch ybinstance/$instanceName --type=merge -p '{"spec":{"requestedState":"Running"}}'
```

Restart an Instance

```
kubectl -n $instanceNamespace patch ybinstance/$instanceName --type=merge -p '{"spec":{"requestedState":"Restart"}}'
```

Upgrade an Instance (or downgrade)

```
kubectl -n $instanceNamespace patch ybinstance/$instanceName --type=merge -p '{"spec":{"version":"{{ VERSION }}"}}'
```

Change an Instance to Scaled Shared Services

```
kubectl -n $instanceNamespace patch ybinstance/$instanceName --type=merge -p '{"spec":{"sharedServicesType":"scaled"}}'
```

Change an Instance to Standard Shared Services

```
kubectl -n $instanceNamespace patch ybinstance/$instanceName --type=merge -p '{"spec":{"sharedServicesType":"standard"}}'
```

Kubernetes LDAP Configuration

Platforms: EE: All cloud platforms

Parent topic: Kubernetes Guides

This document explains the LDAP configuration options for integrating external LDAP authentication and synchronization with a Yellowbrick instance. The configuration is managed via a ConfigMap and a Secret, both of which need to be properly set up in the corresponding Kubernetes namespace for the instance.

LDAP Configuration Overview

The LDAP integration for Yellowbrick supports two main features:

- **LDAP Authentication**: Allows users to authenticate via an external LDAP server.
- **LDAP Synchronization**: Synchronizes users and groups from the external LDAP server into Yellowbrick's internal database.

These two features are configured using the following Kubernetes resources:

1. **ConfigMap**: Stores general LDAP settings.
2. **Secret**: Stores sensitive LDAP credentials.

Both resources must be deployed in the same namespace as the Yellowbrick instance and follow the naming convention (given you know the name of the Yellowbrick instance as `$instanceName`):

- ConfigMap: `ybinst-${instanceName}-ldap-config`
- Secret: `ybinst-${instanceName}-ldap-secret`

LDAP ConfigMap Configuration

The ConfigMap includes various options to enable and customize the LDAP setup. Here's a breakdown of the main fields:

Common Configuration

- **`name`**: Always set to `"default"`, unless multiple configurations are needed (not covered by default).
- **`ldapType`**: Specifies the LDAP authentication type.
  - Options: `"none"`, `"bind"`, or `"search+bind"`
  - `"none"`: Disables LDAP authentication.
  - `"bind"`: Uses simple bind for authentication.
  - `"search+bind"`: Searches for the user, then binds.
- **`ldapServers`**: The hostname of the LDAP server.
  - Example: `"ldap.example.com"`
- **`ldapPort`**: Port number for the LDAP connection (leave empty for default).
- **`ldapConnectionType`**: Specifies the connection security type.

- Options: `"ssl"`, `"tls"`, or `"unsecured"`

Bind Authentication Configuration (for `ldapType` = `"bind"`)

- **`ldapSuffix`**: LDAP domain components, e.g., `"dc=example,dc=com"`
- **`ldapPrefix`**: (Optional) Prefix used for the bind DN.

Search+Bind Authentication Configuration (for `ldapType` = `"search+bind"`)

- **`ldapBindDN`**: Distinguished Name used for binding to the LDAP server (admin user).
  - Example: `"cn=admin,dc=example,dc=com"`
- **`ldapBaseDN`**: Base DN where searches for users will start.
- **`ldapSearchAttribute`**: The LDAP attribute used to search for the user.
  - Default: `"uid"`
- **`ldapSearchFilter`**: Filter for user search queries.
  - Default: `"(objectClass=inetOrgPerson)"`

LDAP Synchronization Settings

These settings control the synchronization of users and groups from LDAP:

- **`ldapSyncEnabled`**: Enables or disables LDAP synchronization.
  - Options: `"true"` or `"false"`
- **`ldapSyncInterval`**: Time interval (in seconds) for running sync jobs.
  - Default: `"3600"` (1 hour)
- **`ldapSyncTrace`**: Enables verbose logging for synchronization.
  - Options: `"true"` or `"false"`
- **`ldapSyncLowercase`**: Converts LDAP usernames to lowercase during synchronization.
  - Options: `"true"` or `"false"`
- **`ldapSyncDrop`**: Removes users from Yellowbrick if they no longer exist in LDAP.
  - Options: `"true"` or `"false"`

LDAP Synchronization Connection Settings

- **`ldapSyncServer`**: LDAP server address for synchronization.
  - Example: `"ldapsync.example.com"`
- **`ldapSyncPort`**: Port number for LDAP synchronization (leave empty for default).
- **`ldapSyncBindDN`**: DN used for binding to the LDAP sync server.
  - Example: `"cn=sync,dc=example,dc=com"`

- `ldapSyncSecure`: Security setting for the synchronization connection.
  - Options: `"ssl"`, `"tls"`, or `"unsecured"`

User and Group Synchronization Filters

- `ldapSyncUsersFilter`: Filter used to find users in LDAP.
  - Default: `"(objectClass=person)"`
- `ldapSyncUsersBaseDN`: Base DN for searching users.
  - Example: `"ou=Users,dc=example,dc=com"`
- `ldapSyncUsersAttribute`: The attribute used to identify users.
  - Default: `"cn"`
- `ldapSyncGroupsFiltering`: Enables or disables group synchronization.
  - Options: `"true"` or `"false"`
- `ldapSyncGroupsFilter`: Filter used to find groups in LDAP.
  - Default: `"(objectClass=groupOfNames)"`
- `ldapSyncGroupsBaseDN`: Base DN for searching groups.
  - Example: `"ou=Groups,dc=example,dc=com"`
- `ldapSyncGroupsAttribute`: The attribute used to identify groups.
  - Default: `"cn"`

LDAP Secret Configuration

Sensitive information such as passwords for LDAP bind and sync users must be stored securely in a Kubernetes Secret. The Secret is named: `ybinst--ldap-secret`

Required Fields

- `ldapBindPassword`: Base64-encoded password for the LDAP bind DN.
- `ldapSyncBindPassword`: Base64-encoded password for the LDAP sync bind DN.

Example Configuration

Here's a sample YAML manifest for both the ConfigMap and Secret resources:

ConfigMap Example

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: ybinst-example-ldap-config
  namespace: your-namespace
data:
```

```
  name: "default"
  ldapType: "search+bind"
  ldapServers: "ldap.example.com"
  ldapConnectionType: "ssl"
  ldapBindDN: "cn=admin,dc=example,dc=com"
  ldapBaseDN: "dc=example,dc=com"
  ldapSearchAttribute: "uid"
  ldapSearchFilter: "(objectClass=inetOrgPerson)"
  ldapSyncEnabled: "true"
  ldapSyncServer: "ldapsync.example.com"
  ldapSyncBindDN: "cn=sync,dc=example,dc=com"
  ldapSyncUsersFilter: "(objectClass=person)"
  ldapSyncGroupsFilter: "(objectClass=groupOfNames)"
```

Secret Example

yaml

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: ybinst-example-ldap-secret
  namespace: your-namespace
type: Opaque
data:
  ldapBindPassword: "encoded-bindpassword"
  ldapSyncBindPassword: "encoded-syncpassword"
```

Yellowbrick Operator

Platforms: EE: All cloud platforms

Parent topic: Kubernetes Guides

Yellowbrick Operator provides lifecycle for Yellowbrick instances, including managing their creation, destruction, suspend/resume operations, upgrades and configuration. The operator monitors changes to Custom Resource Definitions (CRDs) by provisioning and managing Kubernetes resources that support Yellowbrick instances.

Custom Resource Definitions

The following describes each of the custom resource definitions managed by Yellowbrick Operator.

Name	CRD	Description
YBInstance	ybinstances.cluster.yellowbrick.io	The Yellowbrick instance specification and status
YBInstanceTask	ybinstancetasks.cluster.yellowbrick.io	Tasks that perform lifecycle operations on Yellowbrick instances
YBVersion	ybversions.cluster.yellowbrick.io	Version uploaded by Yellowbrick Deployer, containing registry location and semver
YBNodeGroup	ybnodegroups.cluster.yellowbrick.io	The hardware node group used by Kubernetes in a cloud provider to configure how to acquire and configure compute infrastructure
YBHwInstanceType	ybhwinstancetypes.cluster.yellowbrick.io	The hardware instance types supported by Yellowbrick compute clusters and their configuration
YBSharedService	ybsharedservices.cluster.yellowbrick.io	The parameters for scaling between different Yellowbrick instance shared service configurations
YBExternalInstance	ybexternalinstances.cluster.yellowbrick.io	Registration of an unmanaged Yellowbrick instance to be manageable by Yellowbrick Manager
YBStorageProvisionRequest	ybstorageprovisionrequests.cluster.yellowbrick.io	Describes a storage provision request made for an object storage location on behalf of a Yellowbrick instance

Usage

Refer to Kubectl Administration to see some common commands you can use to manage Yellowbrick instances.

Use Cases

Listing Versions

```
% kubectl get -A ybversion

NAME                  AGE
7.0.1-58367.66d9e03e   2d
7.0.1-58423.ec33cf36   4d22h
7.0.1-58503.7c5c6d83   2d4h
7.0.1-58583.7dae3b48   24h
```

Creating an Instance

Using the following:

- the target namespace for the instance `$instanceNamespace`
- the name of the instance `$instanceName`
- the version of the instance `$version` (see listing versions above)

```
% kubectl -n $instanceNamespace apply -f - <<-EOF
apiVersion: cluster.yellowbrick.io/v1
kind: YBInstance
metadata:
  name: $instanceName
spec:
  version: $version
  sharedServicesType: standard
  storageManaged: true
EOF
```

Adjust the `.spec` parameters according to your requirements.

Destroying an Instance

Normally, a custom resource can be removed with `kubectl -n [namespace] delete [type] [name]`; however Yellowbrick Operator prevents accidental deletion of an entire Yellowbrick instance through a finalizer and custom annotation which must be provided on the instance to proceed with deleting the object storage and catalog / rowstore volumes for the instance.

In order to annotate the Yellowbrick instance for deletion, use the following:

- the target namespace for the instance `$instanceNamespace`
- the name of the instance `$instanceName`

```
% id=$(kubectl -n $instanceNamespace get ybinstance $instanceName -o jsonpath='{.metadata.uid}' 2>/dev/null)
% kubectl -n $instanceNamespace patch ybinstance $instanceName -p "{\"metadata\":{\"annotations\":{\"yellowbrick.io/delete-confirm
```

This prepares the instance for deletion, which is then done with this command:

```
% kubectl -n $instanceNamespace delete ybinstance $instanceName
```

Obtain the Details for an Instance

Given:

- the namespace for the instance `$instanceNamespace`
- the name of the instance `$instanceName`

```
% kubectl get -n $instanceNamespace ybinstance $instanceName -o yaml
```

▶ **Example Output**

Listing the Tasks of an Instance

Given:

- the namespace for the instance `$instanceNamespace`
- the name of the instance `$instanceName`

```
% k get -n $instanceNamespace get ybinstancetask |grep $instanceName

NAME                          INSTANCE NAMESPACE     INSTANCE NAME  OPERATION  STATE      AGE    COMPLETED             RESULT
kw-test-10-create-230f60b3    yellowbrick-operator   kw-test-10     Create     Completed  145d   2024-04-18T17:42:12Z  Success
kw-test-10-delete-230f60b3    yellowbrick-operator   kw-test-10     Delete     Completed  145d   2024-04-18T17:45:28Z  Success
kw-test-10-resume-crgph       yellowbrick-operator   kw-test-10     Resume     Completed  145d   2024-04-18T17:43:52Z  Success
```

CRD Reference

YBInstance

▶ **Custom Resource Definition**

YBInstanceTask

▶ **Custom Resource Definition**

YBVersion

▶ **Custom Resource Definition**

Observability Overview

Platforms: EE: All cloud platforms

Parent topic: Cloud

In this section:

This section provides a comprehensive reference for Prometheus metrics and alerting rules used to monitor system health, performance, and reliability across cloud platform components.

Metrics

The Metrics Documentation lists all Prometheus metrics emitted by various components. Each metric entry includes its type, collection frequency, labels, and a description. This is useful for:

- Building dashboards
- Analyzing component behavior
- Understanding what instrumentation is available

Alerts

The Alerts Documentation describes all alert rules configured in our Prometheus setup. Alerts are grouped by component and include severity levels, trigger conditions, and human-readable descriptions.

Use this to:

- Understand why an alert fired
- Debug active incidents
- Tune alert sensitivity or thresholds

Threshold Reference

Some alerts reference templated threshold values from our Helm charts. These are documented separately in the Threshold Reference page.

Observability Alerts

Yellowbrick Documentation > How-to Guides > Administer Yellowbrick > Cloud > Observability > Observability Alerts

Platforms: EE: All cloud platforms

Parent topic: Observability

In this section:

- Always Firing Alerts
- Backup Alerts
- Catalog Row Store Alerts
- Cluster Alerts
- Cluster Manager Alerts
- Compute Node Alerts
- Host Alerts
- Instance Alerts
- LDAP And SSL Alerts
- Session Alerts
- Storage Alerts
- Workload Manager Alerts
- Yellowbrick Row Store Alerts

This section lists all Prometheus alerting rules defined across Yellowbrick components.

Each alert definition includes severity, triggering condition, and description to help operators respond effectively to system issues.

For information on enabling and configuring alerting please see Configuring Alerting.

Note: When multiple alerts share the same name but have different severity levels (e.g., CRITICAL, MAJOR, MINOR), only the highest-severity active alert will be shown at a time. Lower-severity variants are suppressed to avoid noise.

The following pages provide detailed breakdowns of alerts by component area:

- Always Firing Alerts
- Backup Alerts
- Catalog Row Store Alerts
- Cluster Alerts
- Cluster Manager Alerts
- Compute Node Alerts
- Host Alerts
- Instance Alerts
- Ldap And Ssl Alerts
- Session Alerts
- Storage Alerts
- Workload Manager Alerts
- Yellowbrick Row Store Alerts

Always Firing Alerts

This page documents alerts that are expected to **always be firing** under normal circumstances. These alerts serve as a heartbeat to verify that the alerting pipeline is functioning correctly.

Purpose

The `Watchdog` alert is intentionally designed to fire continuously. If it stops firing, this indicates a potential issue with the Prometheus alerting system itself — such as broken rule evaluation or a disconnected alertmanager.

To prevent unnecessary noise, this alert is configured to route to a **null receiver** in Alertmanager, meaning it will **not generate notifications**.

Alert Definition

Alert	Severity	Trigger After	Description	Threshold Ref
`Watchdog`	NONE	0m	This alert should always be firing. If it stops, the alert pipeline may be broken.	

Backup Alerts

Platforms: EE: All cloud platforms

Parent topic: Observability Alerts

Backup alerts are triggered when a backup chain has not been updated within an acceptable timeframe. This condition can prevent Yellowbrick from performing critical database maintenance operations like garbage collection and VACUUM.

Purpose

The primary purpose of this alert is to ensure that backup chains are not stale for extended periods. In Yellowbrick, backups act as a constraint on the Table Delete Horizon — a mechanism that prevents garbage collection of table versions that might still be needed for backup purposes.

When a backup chain becomes too old, it can indefinitely stall garbage collector (GC) and VACUUM processes, leading to storage bloat and degraded performance.

The threshold for this alert is configurable and documented here.

Alert Definition

Alert	Severity	Trigger After	Description	Threshold Ref
`Backup Chain Age`	CRITICAL	-	At least one backup chain is older than [old_chain_threshold_days] day(s).	`old_chain_threshold_days`

Catalog Row Store Alerts

Platforms: EE: All cloud platforms

Parent topic: Observability Alerts

These alerts track usage of the internal catalog row store, which maintains metadata necessary for core query planning and schema management. The row store has hard limits, and when it nears or exceeds these thresholds, system behavior is progressively degraded — from warnings to read-only mode and, eventually, full query block.

Purpose

These alerts notify operators when:

- The catalog row store is nearing its **read-only threshold**.
- The store has crossed into **read-only mode**.
- The system reaches the **blocking threshold**, at which point new queries are fully disabled.

These alerts follow a tiered severity model and are crucial for maintaining system availability. Administrators can use the referenced thresholds to tune alerting behavior.

Alert Definitions

Alert	Severity	Trigger After	Description	Threshold Ref
`Row Store Status`	CRITICAL	-	The database catalog row store is full ([current_value]). All queries have been disabled until usage falls below [blocking_threshold], and the database will remain read-only until it drops below [read_only_threshold].	
`Row Store Status`	MAJOR	-	The database catalog row store is nearly full ([current_value]), and the database has been set to read-only. Normal operations will resume when usage drops below [read_only_threshold]. Query execution will be blocked entirely if it reaches [blocking_threshold].	
`Row Store Status`	MINOR	-	The database catalog row store is nearing capacity: [current_value] used out of [read_only_threshold]. If usage exceeds [read_only_threshold], the database will switch to read-only mode.	

Cluster Alerts

Platforms: EE: All cloud platforms

Parent topic: Observability Alerts

Cluster alerts notify about the operational health of the Yellowbrick cluster as a whole. These alerts help detect and respond to states where the system is either partially or entirely degraded, or explicitly suspended.

Purpose

These alerts are triggered based on cluster-level availability and control plane signals. They serve to identify when:

- The cluster is **offline**, such as when too many compute nodes are unreachable.
- The cluster is **degraded**, with some compute nodes unavailable but not enough to cause a full outage.
- The cluster is in a **suspended** state, which is expected during administrative operations.

Values such as `[missing_compute_nodes]` and `[reason]` are dynamic and inserted at runtime to explain the nature of the issue.

Alert Definitions

Alert	Severity	Trigger After	Description	Threshold Ref
`Cluster State`	CRITICAL	2m	The cluster is offline: [reason].	
`Cluster State`	CRITICAL	2m	The cluster is offline: too many compute nodes ([missing_compute_nodes]) are unavailable.	
`Cluster State`	MAJOR	5m	The cluster is degraded: [reason].	
`Cluster State`	MAJOR	5m	The cluster is degraded: missing compute nodes ([missing_compute_nodes]).	
`Cluster State`	INFO	0m	The cluster was suspended by an administrator.	

Cluster Manager Alerts

Platforms: EE: All cloud platforms

Parent topic: Observability Alerts

Cluster Manager alerts highlight failures in the orchestration and configuration layer of the Yellowbrick platform. These alerts typically require operator attention, as they may signal deeper issues with node orchestration or persistent storage integrity.

Purpose

These alerts are generated when:

- A **cluster reconfiguration** fails, potentially leaving the cluster in an inconsistent or unusable state.
- A **filesystem check** detects corruption or failure, especially in critical back-half operations.

All alerts in this category are high severity and should be investigated immediately. Descriptions include dynamic value `[reason]` to assist in diagnosis. For persistent issues, it is recommended to contact Yellowbrick support.

Alert Definitions

Alert	Severity	Trigger After	Description	Threshold Ref
`Cluster Reconfiguration Failure`	CRITICAL	-	A cluster reconfiguration failed with offline reason '[reason]'. Contact customer support if the problem persists.	
`Filesystem Check Failure`	CRITICAL	-	A back-half filesystem check failed. Contact customer support if the problem persists.	
`Filesystem Check Failure`	CRITICAL	-	A cluster filesystem check failed. Contact customer support if the problem persists.	

Compute Node Alerts

Platforms: EE: All cloud platforms

Parent topic: Observability Alerts

Compute node alerts monitor the health and stability of individual compute node pods in the Yellowbrick cluster. These alerts help detect unexpected crashes, normal exits, and diagnostic events such as minidump generation.

Purpose

These alerts are useful for identifying:

- **Unexpected failures**, such as crashes or minidumps, which often warrant immediate investigation.
- **Normal exits**, which can be useful for tracking lifecycle operations (e.g. restarts or rolling upgrades), even if they don't require alerting action.

Minidump alerts provide early signals of crashes that may generate diagnostic data. In contrast, info-level alerts like `Compute Node Exit` with exit code `0` are logged for observability but not intended to trigger notification policies or automated response actions.

Alert Definitions

Alert	Severity	Trigger After	Description	Threshold Ref
`Compute Node Exited`	VARIES	-	A compute node has exited. Severity varies by exit code: INFO for graceful exit (code 0), CRITICAL for hardware or software crashes. Reason is included in the alert description at runtime.	
`Compute Node Minidump`	CRITICAL	-	Minidump detected on the compute node pod.	

Host Alerts

Platforms: EE: All cloud platforms

Parent topic: Observability Alerts

Host alerts monitor the health of PVCs (Persistent Volume Claims) and the disks of the Kubernetes nodes in the cluster.

Purpose

These alerts are useful for identifying **low disk space** on the underlying Kubernetes nodes or PVCs used in the cluster.

Alert Definitions

Alert	Severity	Trigger After	Description	Threshold Ref
`Host Out of Disk Space`	WARNING	-	Disk is almost full (< 80% left) on the kubernetes node.	
`PVC Utilization High`	WARNING	-	The persistentVolume is 90% utilized. Please check and take appropriate action.	

Instance Alerts

Platforms: EE: All cloud platforms

Parent topic: Observability Alerts

Instance alerts monitor the health and behavior of Yellowbrick database frontend components. These alerts capture both normal and abnormal exit conditions, allowing administrators to distinguish between expected pod terminations and critical failures.

Purpose

These alerts help identify:

- **Unexpected terminations** of the database frontend, which may trigger automated recovery actions.
- **Crash events**, where a minidump is generated and attached for support analysis.
- **Intentional or benign exits**, such as during controlled shutdowns or reconfigurations.

Critical alerts signal potential failures requiring action, while informational alerts are surfaced for visibility and correlation.

Alert Definitions

Alert	Severity	Trigger After	Description	Threshold Ref
`Database Frontend Exit`	CRITICAL	-	Database frontend exited. Restarting for recovery.	
`Database Frontend Exit`	INFO	-	Database frontend exit requested.	
`Database Frontend Exit`	INFO	-	Database frontend pod exited.	
`Database Frontend Minidump`	CRITICAL	-	Database frontend crashed. Minidump file is included in the alert.	

LDAP and SSL Alerts

Platforms: EE: All cloud platforms

Parent topic: Observability Alerts

LDAP and SSL alerts track issues related to authentication and secure communication across Yellowbrick components.

Purpose

These alerts help operators detect and act on:

- **LDAP synchronization failures**, such as misconfigured or unreachable directories.
- **Missing LDAP configuration**, which may affect external user authentication.
- **SSL certificate issues**, including expiration or upcoming expiry of certificates used for secure communication.

The goal is to proactively surface misconfigurations or expirations before they cause login failures or service degradation.

Alert Definitions

Alert	Severity	Trigger After	Description	Threshold Ref
`LDAP Sync Status`	MAJOR	-	LDAP synchronization has failed: [reason].	
`LDAP Sync Status`	INFO	5m	LDAP synchronization is not configured.	
`SSL Certificate Expiry`	CRITICAL	-	SSL certificate '[certificate_name]' expired.	
`SSL Certificate Expiry`	MAJOR	-	SSL certificate '[certificate_name]' expires on [expiration_date].	`ssl_cert_expiry_threshold_days`

Session Alerts

Platforms: EE: All cloud platforms

Parent topic: Observability Alerts

Session alerts monitor the number of active user connections to the Yellowbrick database system. These alerts help detect potential overload situations where the system is nearing or exceeding its configured connection capacity.

Purpose

A high number of concurrent sessions can impact performance and prevent new users from connecting. These alerts are triggered based on thresholds defined for `max_user_connections` and allow for early detection of stress before it causes failures.

Yellowbrick triggers these alerts at two severity levels:

- **MAJOR**: Usage exceeds 85% of the limit.
- **CRITICAL**: Usage exceeds 95% of the limit.

Both thresholds require sustained overuse (5 minutes) to reduce false positives. The actual connection count at alert time is embedded in `[current_value]`.

Thresholds are configurable and documented on the thresholds page.

Alert Definitions

Alert	Severity	Trigger After	Description	Threshold Ref
`High Database Connections`	CRITICAL	5m	The number of user connections has exceeded 95% of the configured threshold for at least 5 minutes. Current: [current_value].	`max_user_connections`
`High Database Connections`	MAJOR	5m	The number of user connections has exceeded 85% of the configured threshold for at least 5 minutes. Current: [current_value].	`max_user_connections`

Storage Alerts

Platforms: EE: All cloud platforms

Parent topic: Observability Alerts

Storage alerts monitor the health and usage of disk resources in Yellowbrick.

Purpose

Yellowbrick systems rely on stable, high-performance storage subsystems. These alerts help detect:

- **Disk space overuse**: Key mount points like `/mnt/rowstore` and `/mnt/ybdata` are monitored to avoid errors due to full disks.

Alert Definitions

Alert	Severity	Trigger After	Description	Threshold Ref
`High Disk Usage Rowstore`	CRITICAL	1m	Disk usage on /mnt/rowstore has remained above 90% for over 1 minute.	
`High Disk Usage YBdata`	CRITICAL	1m	Disk usage on /mnt/ybdata has remained above 90% for over 1 minute.	

Workload Manager Alerts

Platforms: EE: All cloud platforms

Parent topic: Observability Alerts

Workload Manager (WLM) alerts are triggered when queries exceed resource limits or violate workload management rules. These alerts help ensure system performance, fairness, and stability across competing workloads.

Purpose

WLM policies are defined to protect system resources and prioritize workloads. When a query breaches such a rule (e.g., CPU usage, memory consumption, execution time), an alert is raised.

The severity of the alert reflects the action taken:

- **INFO/WARNING** for soft limits (e.g., user notifications)
- **ERROR/CRITICAL** for enforced rules (e.g., query termination)

Each alert typically includes:

- Query ID
- Rule name
- Impacted database or user

These alerts are a key indicator of workload contention, misconfiguration, or the need to adjust WLM rules.

Alert Definitions

Alert	Severity	Trigger After	Description	Threshold Ref
`Query Alert`	VARIES	-	This alert is raised when a query violates a WLM rule. The severity reflects the rule impact (e.g., warning or error). Details include the rule name, query ID, and database.	

Yellowbrick Row Store (YRS) Alerts

Platforms: EE: All cloud platforms

Parent topic: Observability Alerts

These alerts monitor the internal state of the Yellowbrick row store system. They also provide early warnings and critical alerts when the number of row store commit records, data files, deleted records, or unused files exceeds configured thresholds.

Purpose

These alerts help operators detect and respond to:

- **Excessive commit records**.
- **Rising numbers of data or unused files**.
- **Accumulation of deleted records**.

Each alert is grouped by severity and is linked to a configurable threshold variable. These thresholds can be adjusted at runtime and override any appliance-level GUCs used in previous Yellowbrick deployments.

Alert Definitions

Alert	Severity	Trigger After	Description	Threshold Ref
`High Yellowbrick Row Store (YRS) Commit Records Count`	CRITICAL	1m	Number of Yellowbrick row store commit records has exceeded [yrs_commit_records_count_critical_threshold] for more than a minute. Current: [current_value].	`yrs_commit_records_count_critical_threshold`
`High Yellowbrick Row Store (YRS) Commit Records Count`	MAJOR	1m	Number of Yellowbrick row store commit records has exceeded [yrs_commit_records_count_major_threshold] for more than a minute. Current: [current_value].	`yrs_commit_records_count_major_threshold`
`High Yellowbrick Row Store (YRS) Commit Records Count`	MINOR	1m	Number of Yellowbrick row store commit records has exceeded [yrs_commit_records_count_minor_threshold] for more than a minute. Current: [current_value].	`yrs_commit_records_count_minor_threshold`

Alert	Severity	Trigger After	Description	Threshold Ref
`High Yellowbrick Row Store (YRS) Data Files Count`	CRITICAL	1m	Number of Yellowbrick row store data files has exceeded [yrs_data_files_count_critical_threshold] for more than a minute. Current: [current_value].	`yrs_data_files_count_critical_threshold`
`High Yellowbrick Row Store (YRS) Data Files Count`	MAJOR	1m	Number of Yellowbrick row store data files has exceeded [yrs_data_files_count_major_threshold] for more than a minute. Current: [current_value].	`yrs_data_files_count_major_threshold`
`High Yellowbrick Row Store (YRS) Data Files Count`	MINOR	1m	Number of Yellowbrick row store data files has exceeded [yrs_data_files_count_minor_threshold] for more than a minute. Current: [current_value].	`yrs_data_files_count_minor_threshold`
`High Yellowbrick Row Store (YRS) Delete Records Count`	CRITICAL	1m	Number of Yellowbrick row store delete records has exceeded [yrs_delete_records_count_critical_threshold] for more than a minute. Current: [current_value].	`yrs_delete_records_count_critical_threshold`
`High Yellowbrick Row Store (YRS) Delete Records Count`	MAJOR	1m	Number of Yellowbrick row store delete records has exceeded [yrs_delete_records_count_major_threshold] for more than a minute. Current: [current_value].	`yrs_delete_records_count_major_threshold`
`High Yellowbrick Row Store (YRS) Delete Records Count`	MINOR	1m	Number of Yellowbrick row store delete records has exceeded [yrs_delete_records_count_minor_threshold] for more than a minute. Current: [current_value].	`yrs_delete_records_count_minor_threshold`
`High Yellowbrick Row Store (YRS) Unused Files Count`	CRITICAL	1m	Number of Yellowbrick row store unused files has exceeded [yrs_unused_files_count_critical_threshold] for more than a minute. Current: [current_value].	`yrs_unused_files_count_critical_threshold`
`High Yellowbrick Row Store (YRS) Unused Files Count`	MAJOR	1m	Number of Yellowbrick row store unused files has exceeded [yrs_unused_files_count_major_threshold] for more than a minute. Current: [current_value].	`yrs_unused_files_count_major_threshold`

Alert	Severity	Trigger After	Description	Threshold Ref
`High Yellowbrick Row Store (YRS) Unused Files Count`	MINOR	1m	Number of Yellowbrick row store unused files has exceeded [yrs_unused_files_count_minor_threshold] for more than a minute. Current: [current_value].	`yrs_unused_files_count_minor_threshold`
`Row Store Status`	MAJOR	-	The Yellowbrick row store is full, user inserts will be slower.	

Observability Metrics

Platforms: EE: All cloud platforms

Parent topic: Observability

In this section:

This section lists all Prometheus metrics exposed by Yellowbrick components.

The following pages provide detailed breakdowns of each metric by component area:

- Compiler Service Metrics
- Compute Cluster Metrics
- Compute Node Metrics
- Database Metrics
- Disk Caches Metrics
- Disk Exporter Metrics
- JVM Metrics
- LDAP and SSL Metrics
- Locks Metrics
- Process Exporter Metrics
- Replication and Backup Metrics
- Row Store Metrics
- Sessions Metrics
- System Metrics
- Transactions Metrics
- Vacuum Metrics
- Workload Manager Metrics

Compiler Service Metrics

Platforms: EE: All cloud platforms

Parent topic: Observability Metrics

This page documents Prometheus metrics emitted by the **Compiler Service**, which is responsible for handling compile-time requests across the Yellowbrick platform.

Purpose

These metrics are used to track:

- The volume of compile requests processed (`requests_total`)
- Error rates (`request_error_total`)
- The number of active compiler service instances (`service_count`)

They are primarily used to monitor service availability and error trends, and to support alerting and long-term performance analysis.

Metrics

Name	Type	Freq	Labels	Description
`yb_lime_compiler_request_error_total`	counter	1m	-	Total number of compile requests ending in error
`yb_lime_compiler_requests_total`	counter	1m	-	Total number of compile requests
`yb_lime_compiler_service_count`	gauge	1m	-	Number of compiler services connected

Compute Cluster Metrics

Platforms: EE: All cloud platforms

Parent topic: Observability Metrics

This page documents Prometheus metrics emitted by the **Compute Cluster** component, which manages distributed query execution across compute nodes in a Yellowbrick deployment.

Purpose

These metrics are used to monitor the **health, performance, and resource usage** of compute clusters. They provide visibility into:

- Query activity and throughput
- Memory and CPU utilization
- Query compilation and execution times
- Cluster degradation or compute node loss

They are critical for performance tuning, detecting resource contention, and ensuring cluster reliability at scale.

Metrics

Name	Type	Freq	Labels	Description
`yb_lime_active_queries`	gauge	10s	cluster, pool	Number of active queries
`yb_lime_cluster_degraded_status`	gauge	10s	cluster, status, reason	Cluster degraded status
`yb_lime_cluster_missing_workers`	gauge	10s	cluster	Number of missing workers in each cluster
`yb_lime_cluster_state`	gauge	10s	cluster, state, reason	Cluster state
`yb_lime_queries_completed_backend_total`	counter	10s	cluster, pool, state	Total number of queries completed via backend
`yb_lime_queries_completed_total`	counter	10s	cluster, state	Total number of queries completed
`yb_lime_queries_submitted_total`	counter	10s	cluster	Total number of queries submitted
`yb_lime_query_bytes_network`	histogram	10s	cluster	Bytes network
`yb_lime_query_bytes_read`	histogram	10s	cluster	Bytes read
`yb_lime_query_bytes_read_spill`	histogram	10s	cluster	Bytes read spill
`yb_lime_query_bytes_written`	histogram	10s	cluster	Bytes written
`yb_lime_query_bytes_written_spill`	histogram	10s	cluster	Bytes written spill

Name	Type	Freq	Labels	Description
`yb_lime_query_cache_efficiency`	histogram	10s	cluster	Query cache efficiency
`yb_lime_query_compile_time`	histogram	10s	cluster	Query compile time in seconds
`yb_lime_query_cpu_usage`	gauge	10s	cluster, pool	Duration weighted average of longest worker CPU query usage as percentage of allocated CPU
`yb_lime_query_lock_time`	histogram	10s	cluster	Query lock time in seconds
`yb_lime_query_memory_granted`	gauge	10s	cluster, pool	Total memory granted to queries in bytes
`yb_lime_query_memory_used`	gauge	10s	cluster, pool	Total memory used by queries in bytes
`yb_lime_query_run_time`	histogram	10s	cluster	Query run time in seconds
`yb_lime_query_total_time`	histogram	10s	cluster	Query total time in seconds
`yb_lime_query_wait_time`	histogram	10s	cluster	Query wait time in seconds

Compute Node Metrics

Platforms: EE: All cloud platforms

Parent topic: Observability Metrics

This page documents Prometheus metrics emitted by **Compute Nodes**, which are the processes that execute distributed query fragments and manage data movement in Yellowbrick's architecture.

Purpose

These metrics provide detailed visibility into the health, stability, and behavior of individual compute nodes. They are used to:

- Monitor compute nodes uptime and crash patterns
- Diagnose issues with heartbeat timing and object store I/O
- Detect exit codes and termination reasons (e.g., out-of-memory, unrecoverable signals)
- Track YRD traffic, loader cache usage, and time synchronization skew

These insights are vital for debugging compute nodes instability, verifying cluster coordination, and building high-reliability monitoring dashboards.

Metrics

Name	Type	Freq	Labels	Description
`yb_heartbeat_recv_ms_total`	counter	10s	index, instance_uuid, cluster, worker_logical_id, worker_uuid	Milliseconds from when a Heartbeat was sent until response
`yb_heartbeat_send_ms_total`	counter	10s	index, instance_uuid, cluster, worker_logical_id, worker_uuid	Milliseconds from when a Heartbeat was scheduled until sent
`yb_lime_heartbeat_elapsed_time_percent`	histogram	10s	cluster, worker_id	Elapsed time in percent of the heartbeat timeout
`yb_lime_heartbeat_error_total`	counter	10s	cluster, worker_id	Total number of heartbeat errors
`yb_lime_loader_cache_available_size`	gauge	10s	cluster	Estimated minimum available loader cache space across all workers in the compute cluster
`yb_obj_store_recv`	gauge	10s	index, instance_uuid, cluster, worker_logical_id, worker_uuid	Object store HTTP bytes received
`yb_obj_store_recv_fail_total`	counter	10s	index, instance_uuid, cluster, worker_logical_id, worker_uuid	Object store socket failed recv calls

Name	Type	Freq	Labels	Description
`yb_obj_store_send`	gauge	10s	index, instance_uuid, cluster, worker_logical_id, worker_uuid	Object store HTTP bytes sent
`yb_obj_store_send_fail_total`	counter	10s	index, instance_uuid, cluster, worker_logical_id, worker_uuid	Object store socket failed send calls
`yb_timeout_queue_late_ms_total`	counter	10s	index, instance_uuid, cluster, worker_logical_id, worker_uuid	Milliseconds the Timeout Queue was late
`yb_tsc_skew1000_percent_total`	counter	10s	index, instance_uuid, cluster, worker_logical_id, worker_uuid	1000x Percent Skew between TSC and Timespec
`yb_worker_exit_code_cluster_reset`	counter	10s	instance_uuid, cluster, worker_logical_id, worker_uuid	exit due to CLUSTER_RESET
`yb_worker_exit_code_configure_not_quiesced`	counter	10s	instance_uuid, cluster, worker_logical_id, worker_uuid	exit due to CONFIGURE_NOT_QUIESCED
`yb_worker_exit_code_general_error`	counter	10s	instance_uuid, cluster, worker_logical_id, worker_uuid	exit due to GENERAL_ERROR
`yb_worker_exit_code_ib_connection_down`	counter	10s	instance_uuid, cluster, worker_logical_id, worker_uuid	exit due to IB_CONNECTION_DOWN
`yb_worker_exit_code_minidump_exception`	counter	10s	instance_uuid, cluster, worker_logical_id, worker_uuid	exit due to MINIDUMP_EXCEPTION
`yb_worker_exit_code_minidump_repeated`	counter	10s	instance_uuid, cluster, worker_logical_id, worker_uuid	exit due to MINIDUMP_REPEATED
`yb_worker_exit_code_numa_oom`	counter	10s	instance_uuid, cluster, worker_logical_id, worker_uuid	exit due to NUMA_OUT_OF_MEMORY
`yb_worker_exit_code_other_reason`	counter	10s	instance_uuid, cluster, worker_logical_id, worker_uuid	exit due to OTHER_REASON
`yb_worker_exit_code_other_signal`	counter	10s	instance_uuid, cluster, worker_logical_id, worker_uuid	exit due to some other signal
`yb_worker_exit_code_recopy_worker`	counter	10s	instance_uuid, cluster, worker_logical_id, worker_uuid	exit due to RECOPY_WORKER

Name	Type	Freq	Labels	Description
`yb_worker_exit_code_sigabrt`	counter	10s	instance_uuid, cluster, worker_logical_id, worker_uuid	exit due to signal SIGABRT
`yb_worker_exit_code_sigalrm`	counter	10s	instance_uuid, cluster, worker_logical_id, worker_uuid	exit due to signal SIGALRM
`yb_worker_exit_code_sigbus`	counter	10s	instance_uuid, cluster, worker_logical_id, worker_uuid	exit due to signal SIGBUS
`yb_worker_exit_code_sigfpe`	counter	10s	instance_uuid, cluster, worker_logical_id, worker_uuid	exit due to signal SIGFPE
`yb_worker_exit_code_sighup`	counter	10s	instance_uuid, cluster, worker_logical_id, worker_uuid	exit due to signal SIGHUP
`yb_worker_exit_code_sigill`	counter	10s	instance_uuid, cluster, worker_logical_id, worker_uuid	exit due to signal SIGILL
`yb_worker_exit_code_sigint`	counter	10s	instance_uuid, cluster, worker_logical_id, worker_uuid	exit due to signal SIGINT
`yb_worker_exit_code_sigkill`	counter	10s	instance_uuid, cluster, worker_logical_id, worker_uuid	exit due to signal SIGKILL
`yb_worker_exit_code_sigpipe`	counter	10s	instance_uuid, cluster, worker_logical_id, worker_uuid	exit due to signal SIGPIPE
`yb_worker_exit_code_sigsegv`	counter	10s	instance_uuid, cluster, worker_logical_id, worker_uuid	exit due to signal SIGSEGV (segmentation fault)
`yb_worker_exit_code_sigtrap`	counter	10s	instance_uuid, cluster, worker_logical_id, worker_uuid	exit due to signal SIGTRAP
`yb_worker_exit_code_success`	counter	10s	instance_uuid, cluster, worker_logical_id, worker_uuid	exit due to SUCCESS
`yb_worker_exit_code_unknown`	counter	10s	instance_uuid, cluster, worker_logical_id, worker_uuid	exit due to unknown
`yb_worker_exit_code_ybd_assert`	counter	10s	instance_uuid, cluster, worker_logical_id, worker_uuid	exit due to YBD_ASSERT

Name	Type	Freq	Labels	Description
`yb_worker_last_exit_code`	gauge	10s	instance_uuid, cluster, worker_logical_id, worker_uuid	The last exit code or -1
`yb_worker_uptime_sec`	counter	10s	instance_uuid, cluster, worker_logical_id, worker_uuid	Worker uptime in seconds
`yb_yrd_re_tx_bytes_total`	counter	10s	index, instance_uuid, cluster, worker_logical_id, worker_uuid	YRD Bytes re-transmitted
`yb_yrd_tx_bytes_total`	counter	10s	index, instance_uuid, cluster, worker_logical_id, worker_uuid	YRD Bytes transmitted

Database Metrics

Platforms: EE: All cloud platforms

Parent topic: Observability Metrics

This page documents Prometheus metrics related to **user databases** in the Yellowbrick system. These metrics reflect internal catalog states, transaction aging, and user data sizing across distributed storage.

Purpose

These metrics are primarily used to:

- Monitor database size and catalog footprint
- Detect potential transaction ID wraparound risks
- Track compression and table count trends across databases

They are valuable for capacity planning, transaction maintenance (e.g. `VACUUM`), and debugging database-level anomalies or storage bloat.

Metrics

Name	Type	Freq	Labels	Description
`yb_database_age_frozen_transaction_id`	gauge	1h	database_oid	Age of the oldest transaction ID that has not been frozen in the database
`yb_database_age_minimum_multi_transaction_id`	gauge	1h	database_oid	Age of the oldest multi-transaction ID that has been replaced with a transaction ID in the database
`yb_database_catalog_data_bytes`	gauge	1h	database_oid	Catalog and metadata size of the database
`yb_database_table_count`	gauge	1h	database_oid	Number of user tables in each database
`yb_database_user_data_compressed_bytes`	gauge	1h	database_oid	Compressed user data size for the database (worker nodes)

Disk Cache Metrics

Platforms: EE: All cloud platforms

Parent topic: Observability Metrics

This page documents Prometheus metrics related to **disk-based caches** within the Yellowbrick platform, which help accelerate query execution by reducing I/O latency for frequently accessed data.

Purpose

These metrics provide visibility into the behavior and efficiency of various internal caches used for temporary storage on disk. They can be used to:

- Monitor cache size and usage patterns
- Track hit/miss ratios for performance analysis
- Identify areas under- or over-utilizing cache space
- Inform tuning of caching policies for different subsystems

Each cache is identified by its area label, allowing for targeted troubleshooting and monitoring. The area label distinguishes between the types of internal caches. It can have one of the following values:

- ast – The Abstract Syntax Tree (AST) cache
- source – The source code cache
- object – The compiled object code cache

This labeling makes it easy to track the size and health of each cache independently in Prometheus and Grafana dashboards.

Metrics

Name	Type	Freq	Labels	Description
`yb_lime_cache_entries`	gauge	1m	area	Number of entries in the corresponding cache
`yb_lime_cache_hits`	gauge	1m	area	Hit count of the corresponding cache
`yb_lime_cache_misses`	gauge	1m	area	Miss count of the corresponding cache
`yb_lime_cache_size`	gauge	1m	area	Size in bytes of the corresponding cache

Disk Exporter Metrics

Platforms: EE: All cloud platforms

Parent topic: Observability Metrics

This page documents Prometheus metrics collected by the **Disk Exporter**, which reports on local disk usage across mounted volumes and Yellowbrick-specific data directories.

Purpose

These metrics are used to monitor disk space consumption across the system, including:

- Overall disk usage per mounted path
- Specific usage of level-1 storage areas (e.g., **rowstore**, staging files)

They are essential for capacity planning, proactive alerting, and detecting abnormal disk growth due to unflushed data or log buildup.

Metrics

Name	Type	Freq	Labels	Description
`node_disk_usage_bytes`	gauge	10s	path	Disk usage in bytes per mount
`node_disk_usage_level_1_bytes`	gauge	10s	path	Disk usage of level-1 files (e.g., rowstore)

JVM Metrics

Platforms: EE: All cloud platforms

Parent topic: Observability Metrics

This page documents Prometheus metrics exposed by the **JVM instrumentation module** of the Prometheus Java Client Library inside of Yellowbrick components.

Purpose

JVM metrics are essential for understanding the behavior and health of long-running Java processes. They cover a wide range of subsystems:

- Memory usage (heap, non-heap, and memory pools)
- Garbage collection frequency and duration
- JIT compilation time
- Thread activity and contention
- Class loading dynamics
- Process-level system statistics (CPU usage, open file descriptors)
- Native memory tracking

These metrics help detect memory leaks, GC pressure, thread exhaustion, and performance degradation across critical components.

Metrics

Name	Type	Freq	Labels	Description
`Buffer Pool Metrics`	group	10s	-	JVM buffer pool metrics
`Class Loading Metrics`	group	10s	-	JVM class loading stats
`Compilation Metrics`	group	10s	-	JVM JIT compilation metrics
`Garbage Collector Metrics`	group	10s	-	Garbage collector performance
`Memory Metrics`	group	10s	-	JVM heap/non-heap memory usage
`Memory Pool Allocation Metrics`	group	10s	-	JVM memory pool allocation metrics
`Native Memory Metrics`	group	10s	-	JVM native memory tracking
`Process Metrics`	group	10s	-	Process-level stats like CPU, memory, open fds, start time
`Runtime Metrics`	group	10s	-	JVM runtime and system properties
`Threads Metrics`	group	10s	-	JVM thread-related metrics

LDAP and SSL Metrics

Platforms: EE: All cloud platforms

Parent topic: Observability Metrics

This page documents Prometheus metrics related to **LDAP synchronization** and **SSL certificate monitoring** for Yellowbrick platform components.

Purpose

These metrics are essential for tracking the status of two critical subsystems:

- **LDAP Sync:** Ensures that external user/group identities from LDAP are synchronized regularly and successfully. Metrics expose both sync state and failure reasons.
- **SSL Certificate Expiry:** Helps monitor expiration times of SSL certificates (e.g., for internal APIs or customer access endpoints) to prevent outages or security warnings.

These metrics are commonly used for setting up alerts to detect expired or soon-to-expire certificates and to troubleshoot authentication issues caused by sync failures.

Metrics

Name	Type	Freq	Labels	Description
`yb_ldap_sync_status`	gauge	10s	status, reason	LDAP synchronization status as labels: INITIALIZING, NOT_CONFIGURED, SUCCEEDED, FAILED; 1 if active, else 0.
`yb_ssl_cert_expiry_seconds`	gauge	1d	certificate_name	SSL certificate expiration timestamp in epoch seconds

Locks Metrics

Platforms: EE: All cloud platforms

Parent topic: Observability Metrics

This page documents Prometheus metrics related to **database-level locking** in Yellowbrick. These metrics provide insight into lock contention and concurrency patterns within transactional workloads.

Purpose

The lock metrics help monitor how many locks are currently held across databases, and what types of locks (e.g., shared, exclusive) are in use. This is useful for:

- Detecting lock contention or long-running transactions
- Analyzing workload concurrency
- Debugging DDL/DML blocking behavior
- Enabling better tuning decisions around transaction isolation and access patterns

Lock metrics are especially important for scenarios where concurrent modifications may introduce contention.

Metrics

Name	Type	Freq	Labels	Description
`yb_database_locks`	gauge	1m	database_oid, mode	Number of locks held by database with oid 'database_oid', per lock mode

Process Exporter Metrics

Platforms: EE: All cloud platforms

Parent topic: Observability Metrics

This page documents Prometheus metrics collected from the **Process Exporter**, which reports resource usage statistics for grouped processes running on Yellowbrick nodes.

Purpose

These metrics track system-level resource consumption by named process groups. They are used to:

- Monitor CPU and memory utilization across core services
- Track open file descriptors, thread counts, and process sprawl
- Analyze I/O throughput (read/write bytes)
- Identify resource leaks, runaway processes, or unexpected restarts

The metrics are grouped using label `groupname`, allowing for fine-grained service-level observability.

Metrics

Name	Type	Freq	Labels	Description
`namedprocess_namegroup_cpu_seconds_total`	counter	10s	groupname	CPU seconds used by process group
`namedprocess_namegroup_memory_bytes`	gauge	10s	groupname	Memory used by process group
`namedprocess_namegroup_num_procs`	gauge	10s	groupname	Number of processes in group
`namedprocess_namegroup_num_threads`	gauge	10s	groupname	Threads used by process group
`namedprocess_namegroup_open_filedesc`	gauge	10s	groupname	Open file descriptors
`namedprocess_namegroup_read_bytes_total`	counter	10s	groupname	Bytes read by process group
`namedprocess_namegroup_write_bytes_total`	counter	10s	groupname	Bytes written by process group

Replication and Backup Metrics

Platforms: EE: All cloud platforms

Parent topic: Observability Metrics

This page documents Prometheus metrics related to **data replication, backup, restore, load, and unload** workflows in Yellowbrick. These operations are essential for durability, disaster recovery, and data movement across environments.

Purpose

These metrics allow you to track:

- **Replication behavior**: Duration, cycle counts, errors, retries, throughput per replica
- **Operations**: Backup, restore, load, and unload counts and durations
- **Operational health**: Success/failure breakdowns and replica state transitions

They are critical for monitoring long-running data operations, detecting replication lag or errors, validating backup freshness, and ensuring recovery readiness.

Metrics

Name	Type	Freq	Labels	Description
`yb_lime_backup_chain_age`	histogram	5m	-	Age in days of each backup
`yb_lime_backups_duration`	histogram	5m	-	Duration in seconds of active backups
`yb_lime_backups_total`	counter	5m	status	Number of backups in success/error/active states
`yb_lime_loads_duration`	histogram	5m	-	Duration in seconds of active loads
`yb_lime_loads_total`	counter	5m	status	Number of bulk loads in success/error/active states
`yb_lime_replica_cycles_total`	counter	5m	replica_id	Number of replication cycles completed (both success and errors included) for the corresponding replica
`yb_lime_replica_elapsed_seconds_total`	counter	5m	replica_id	Number of seconds spent actively replicating for the corresponding replica (cumulative over all replication cycles)
`yb_lime_replica_errored_cycles_total`	counter	5m	replica_id	Number of replication cycles that ended with error for the corresponding replica
`yb_lime_replica_retries_total`	counter	5m	replica_id	Number of retries for the corresponding replica (cumulative over all replication cycles)
`yb_lime_replica_sent_bytes_total`	counter	5m	replica_id	Number of bytes sent for the corresponding replica (cumulative over all replication cycles)
`yb_lime_replica_states`	gauge	5m	state	Number of replicas in each state
`yb_lime_replica_written_bytes_total`	counter	5m	replica_id	Number of bytes written for the corresponding replica (cumulative over all replication cycles)

Name	Type	Freq	Labels	Description
`yb_lime_restores_duration`	histogram	5m	-	Duration in seconds of active restores
`yb_lime_restores_total`	counter	5m	status	Number of restores in success/error/active states
`yb_lime_unloads_duration`	histogram	5m	-	Duration in seconds of active unloads
`yb_lime_unloads_total`	counter	5m	status	Number of bulk unloads in success/error/active states

Row Store Metrics

Platforms: EE: All cloud platforms

Parent topic: Observability Metrics

This page documents Prometheus metrics related to the **row store** system in Yellowbrick.

Purpose

These metrics help track the internal state, storage usage, and operational status of the row store layers, including:

- File counts for different Yellowbrick row store components (data, commit, delete, unused)
- Accumulated unflushed data and total bytes stored
- System health and status transitions
- Row store table counts and utilization thresholds

They are useful for debugging catalog performance issues, tracking file churn, and monitoring for cleanup/backlog risks in the transactional metadata layer.

Metrics

Name	Type	Freq	Labels	Description
`yb_catalog_rowstore_bytes_used`	gauge	15m	ro_threshold_bytes, block_threshold_bytes	Total number of bytes used in database catalog row store
`yb_rowstore_status`	gauge	15m	status	Row store status as labels: NORMAL, WARNING, READ_ONLY, YRS_FULL, FULL, or ABNORMAL; 1 if active, else 0.
`yb_yrs_rowstore_active_flushes`	gauge	15m	-	Number of Yellowbrick row store tables with unflushed bytes
`yb_yrs_rowstore_commit_file_count`	gauge	15m	-	Total number of Yellowbrick row store commit files
`yb_yrs_rowstore_data_file_count`	gauge	15m	-	Total number of Yellowbrick row store data files
`yb_yrs_rowstore_delete_file_count`	gauge	15m	-	Total number of Yellowbrick row store delete files
`yb_yrs_rowstore_table_count`	gauge	15m	-	Number of row Yellowbrick store tables
`yb_yrs_rowstore_unused_file_count`	gauge	15m	-	Total number of Yellowbrick row store unused files

Sessions Metrics

Platforms: EE: All cloud platforms

Parent topic: Observability Metrics

This page documents Prometheus metrics related to **database session activity** in Yellowbrick. These metrics provide insight into the number of active and idle sessions, as well as session-level memory consumption over time.

Purpose

Session metrics help operators and developers:

- Monitor how many sessions are active or idle across the system
- Identify long-idle sessions that may need to be closed (e.g. for resource hygiene)
- Analyze memory usage trends across sessions, including per-cluster patterns
- Tune connection pool behavior, session limits, or auto-idle disconnect logic

These metrics are critical for capacity planning, managing concurrency, and detecting orphaned client connections or resource exhaustion caused by excessive memory use.

Metrics

Name	Type	Freq	Labels	Description
`yb_idle_session_count`	gauge	30s	session_type	Number of idle database sessions
`yb_idle_session_time_seconds_bucket`	counter	30s	le, session_type	Cumulative count of sessions idle for <= bucket thresholds
`yb_idle_session_time_seconds_count`	counter	30s	session_type	Total number of idle sessions (for histogram)
`yb_idle_session_time_seconds_sum`	gauge	30s	session_type	Total sum of idle session times
`yb_peak_session_memory_bytes_bucket`	counter	1m	le, session_type, cluster	Cumulative count of sessions by peak memory usage (bytes)
`yb_peak_session_memory_bytes_count`	counter	1m	session_type, cluster	Total number of sessions contributing to peak memory usage
`yb_peak_session_memory_bytes_sum`	gauge	1m	session_type, cluster	Total peak memory usage across all sessions (in bytes)
`yb_session_count`	gauge	30s	session_type, cluster	Number of open database sessions

System Metrics

Platforms: EE: All cloud platforms

Parent topic: Observability Metrics

This page documents Prometheus metrics that reflect the **overall uptime and availability status** of key Yellowbrick system components, including the Lime service and the platform as a whole.

Purpose

System-level metrics are essential for tracking service health, restart behavior, and uptime baselines. Specifically, they help:

- Monitor whether the system is currently marked as “up” or “down”
- Track the total uptime duration of services (e.g., Lime)
- Detect unexpected restarts or prolonged downtimes

Metrics

Name	Type	Freq	Labels	Description
`yb_lime_uptime_total`	counter	10s	-	Uptime of lime service in seconds
`yb_system_is_up`	gauge	1m	-	Is the system up and running
`yb_system_uptime_seconds`	gauge	1m	-	YB system uptime in seconds

Transactions Metrics

Platforms: EE: All cloud platforms

Parent topic: Observability Metrics

This page documents Prometheus metrics related to **active database transactions** in Yellowbrick, offering insight into transactional concurrency and long-running transaction behavior.

Purpose

These metrics are useful for:

- Tracking the number of currently open transactions in the system
- Detecting long-running transactions that could block vacuum, DDL, or create risk of wraparound
- Supporting transaction-level diagnostics during query or session analysis
- Informing tuning of idle timeout or transaction isolation strategies

Metrics

Name	Type	Freq	Labels	Description
`yb_active_transactions`	gauge	1m	-	Current number of active transactions
`yb_active_transactions_oldest_timestamp_seconds`	gauge	1m	-	The current maximum transaction age in seconds

Vacuum Metrics

Platforms: EE: All cloud platforms

Parent topic: Observability Metrics

This page documents Prometheus metrics related to **vacuum operations** in Yellowbrick — both autovacuum and manual vacuum — which help manage table bloat, reclaim dead tuples, and prevent transaction ID wraparound.

Purpose

Vacuum metrics are critical for monitoring the health and efficiency of background maintenance processes. They help:

- Track how often tables are vacuumed across databases
- Distinguish between automatic and manual vacuums
- Detect stale or unvacuumed tables (e.g. through long “time since last vacuum”)
- Measure the duration and frequency of vacuum operations

These metrics support **bloat management**, **query performance tuning**, and **alerting** for vacuum starvation or regressions in autovacuum behavior.

Metrics

Name	Type	Freq	Labels	Description
`yb_active_autovacuum_count`	gauge	1m	-	Number of currently active autovacuum processes
`yb_active_manual_vacuum_count`	gauge	1m	-	Number of currently active manual vacuum processes
`yb_time_since_last_vacuum_seconds_bucket`	counter	1m	le, database_oid	Cumulative count of tables since last vacuum (manual or auto), grouped by database
`yb_time_since_last_vacuum_seconds_count`	counter	1m	database_oid	Total number of vacuumed tables per database
`yb_time_since_last_vacuum_seconds_sum`	gauge	1m	database_oid	Total time since last vacuum across all tables per database
`yb_vacuum_time_seconds_bucket`	counter	1m	le	Cumulative count of vacuum operations running ≤ each duration bucket
`yb_vacuum_time_seconds_count`	counter	1m	-	Total number of currently running vacuum operations
`yb_vacuum_time_seconds_sum`	gauge	1m	-	Total execution time of all currently running vacuum operations

Workload Manager Metrics

Platforms: EE: All cloud platforms

Parent topic: Observability Metrics

This page documents Prometheus metrics related to the Workload Manager (WLM), which governs resource allocation and query concurrency across pools in Yellowbrick.

Purpose

The Workload Manager ensures fair and efficient use of system resources across multiple query workloads. This metric allows operators to:

- Track how many queries are currently running or waiting in each resource pool
- Identify pool-level contention or saturation
- Detect imbalances between configured concurrency limits and actual usage
- Optimize pool configurations based on query patterns

This metric is especially useful for SREs and platform admins monitoring SLA compliance or debugging queue buildup.

Metrics

Name	Type	Freq	Labels	Description
`yb_lime_resourcepool`	gauge	10s	cluster, pool, state	Query counts by running/waiting state

Alert Thresholds

Platforms: EE: All cloud platforms

Parent topic: Observability

This page documents all configurable thresholds used by Prometheus alerts in the Yellowbrick platform. These values control when an alert is triggered based on the current system state.

Note These values take precedence over any appliance-level GUCs with the same name. For cloud deployments, only the Prometheus threshold is evaluated.

Threshold List

Threshold Name	Description	Used In Alert(s)	Default Value
`max_user_connections`	Maximum allowed user connections before raising session load alerts.	`High Database Connections`	`2000`
`yrs_data_files_count_critical_threshold`	Yellowbrick row store: critical threshold for data file count.	`High Yellowbrick Row Store (YRS) Data Files Count`	`-1` (disabled)
`yrs_data_files_count_major_threshold`	Yellowbrick row store: major threshold for data file count.	`High Yellowbrick Row Store (YRS) Data Files Count`	`-1` (disabled)
`yrs_data_files_count_minor_threshold`	Yellowbrick row store: minor threshold for data file count.	`High Yellowbrick Row Store (YRS) Data Files Count`	`-1` (disabled)
`yrs_commit_records_count_critical_threshold`	Yellowbrick row store: critical threshold for commit record count.	`High Yellowbrick Row Store (YRS) Commit Records Count`	`-1` (disabled)
`yrs_commit_records_count_major_threshold`	Yellowbrick row store: major threshold for commit record count.	`High Yellowbrick Row Store (YRS) Commit Records Count`	`-1` (disabled)
`yrs_commit_records_count_minor_threshold`	Yellowbrick row store: minor threshold for commit record count.	`High Yellowbrick Row Store (YRS) Commit Records Count`	`-1` (disabled)
`yrs_delete_records_count_critical_threshold`	Yellowbrick row store: critical threshold for delete record count.	`High Yellowbrick Row Store (YRS) Delete Records Count`	`-1` (disabled)
`yrs_delete_records_count_major_threshold`	Yellowbrick row store: major threshold for delete record count.	`High Yellowbrick Row Store (YRS) Delete Records Count`	`-1` (disabled)

Threshold Name	Description	Used In Alert(s)	Default Value
`yrs_delete_records_count_minor_threshold`	Yellowbrick row store: minor threshold for delete record count.	`High Yellowbrick Row Store (YRS) Delete Records Count`	`-1` (disabled)
`yrs_unused_files_count_critical_threshold`	Yellowbrick row store: critical threshold for unused file count.	`High Yellowbrick Row Store (YRS) Unused Files Count`	`-1` (disabled)
`yrs_unused_files_count_major_threshold`	Yellowbrick row store: major threshold for unused file count.	`High Yellowbrick Row Store (YRS) Unused Files Count`	`-1` (disabled)
`yrs_unused_files_count_minor_threshold`	Yellowbrick row store: minor threshold for unused file count.	`High Yellowbrick Row Store (YRS) Unused Files Count`	`-1` (disabled)
`old_chain_threshold_days`	Maximum age of a backup chain in days before triggering an alert. Rounded to the closest matching bucket (e.g., 1, 5, 10, 15, 20, 25, 30, 60).	`Backup Chain Age`	`30`
`ssl_cert_expiry_threshold_days`	Number of days left before expiration before raising SSL certificate expiration alert.	`SSL Certificate Expiry`	`30`

How to Modify Thresholds

Please contact Yellowbrick Technical Support team if you need to update any of these default values.

How to Suspend Instances

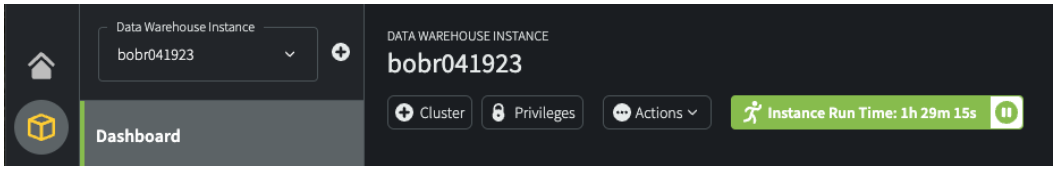
Yellowbrick Documentation > How-to Guides > Administer Yellowbrick > Cloud > Suspend an Instance

Platforms: EE: All cloud platforms

Parent topic: Cloud

To manage cost for cloud-based hardware resources, it is a best practice to suspend a data warehouse instance when it is not in use for some period of time. Suspending an instance implies the suspension of all of its compute clusters, regardless of the suspension and timeout policy set for those clusters and their current state. How long you want to allow an unused instance to remain up and running is your choice, provided that you monitor the cost. In addition to considering the cost, remember that it takes a little while to resume an instance once it has been suspended.

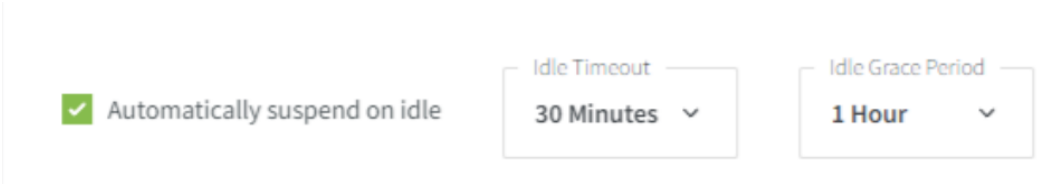
On the Yellowbrick Manager Dashboard, you can easily see how long each instance has been up and running:



You can click the pause button next to the run time to suspend an instance. The run time is not cumulative; it restarts from 0 when you suspend and resume an instance.

Important: When you create an instance, the default behavior is that an idle instance is not automatically suspended. After creating an instance, you can modify this behavior by selecting **Actions** > **Change** for the instance. On the **Change Instance** screen, select **Automatically suspend on idle**, then enter an **Idle Timeout** value and an **Idle Grace Period** value.

The **Idle Timeout** value is subject to an additional grace period. For example, if you set **Idle Timeout** to 30 minutes and **Idle Grace Period** to 1 hour, the instance will not automatically suspend until an idle time of 90 minutes has passed. The idle timeout is not enforced until the grace period has elapsed.



An instance is considered idle when no user-submitted operations (create and alter cluster operations, backend queries, bulk loads, backups, and so on) are running on clusters that belong to the instance. Front-end database queries and system work that runs in the background do not apply to the idle timeout thresholds.

A cluster that is running but idle does not prevent an instance from auto-suspending. The instance suspends its clusters when it auto-suspends.

Note that you cannot change the auto-suspend behavior of an instance if the instance is currently suspended.

Upgrading Cloud Platform and Infrastructure

Yellowbrick Documentation > How-to Guides > Administer Yellowbrick > Cloud > Upgrading Platform

Platforms: EE: All cloud platforms

Parent topic: Cloud

Yellowbrick Deployer let's you upgrade the platform and cloud infrastructure powering the Yellowbrick Data Warehouse Instance. A typical Infrastructure upgrade could include:

- **Kubernetes Infrastructure**: Any control plane (AKS/EKS/GKE) version and subsquent node upgrades.
- **Infrastructure**: Other infrastructure components such as IAM roles/policies/permissions on the respective cloud platform.
- **Components**: Components including Yellowbrick Operator, Manager, Monitoring or any other kubernetes components used for running Yellowbrick.
- **Software Version**: A new version of Yellowbrick Data Warehouse Instance.

INFO

The Infrastructure Upgrade process is a ***Zero Downtime Event*** for the Database instance, and any database functions won't be affected. During the upgrade, only Yellowbrick Manager and Operator functions may be unavailable.

Running the Deployer

To start the ugpgrade process, follow the instructions to create the Deployer in your respective environment:

- AWS installation instructions
- Azure installation instructions
- GCP installation instructions

After running the deployer, you can navigate to the deployer UI to start the Upgrade process.

If you want to use the CLI to upgrade, see the CLI reference for more information.

Database Administration

Platforms: All platforms

Parent topic: Administer Yellowbrick

In this section:

This section introduces the *database* administration tasks that you may have to perform routinely or periodically to create and maintain physical Yellowbrick databases and their database objects.

For information about bulk loading tables, see Bulk Loading Tables.

Backup and Restore

Platforms: All platforms

Parent topic: Databases

In this section:

This section explains how to back up and restore Yellowbrick databases by using Java-based tools that you call from a client system (`ybbackup` and `ybrestore`). You can also manage backups with the `ybbackupctl` tool.

Overview

Platforms: All platforms

Parent topic: Backup & Restore

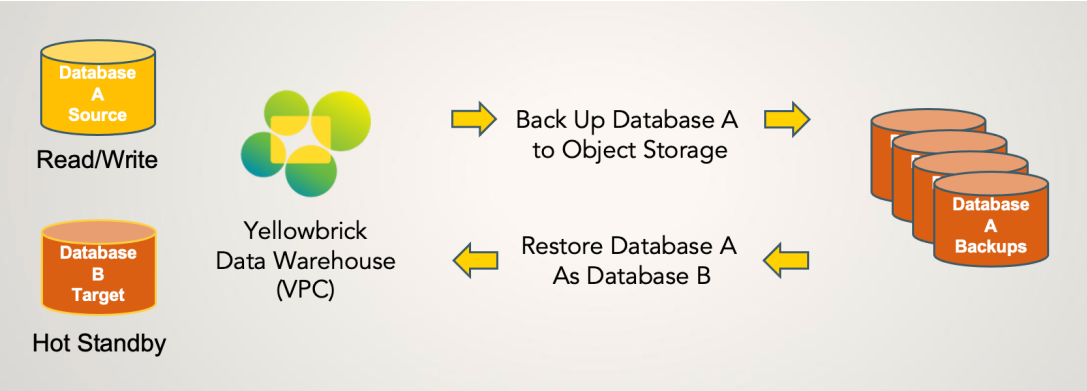
In this section:

This section introduces concepts, terms, and behavior that you need to be familiar with before attempting to back up and restore Yellowbrick databases.

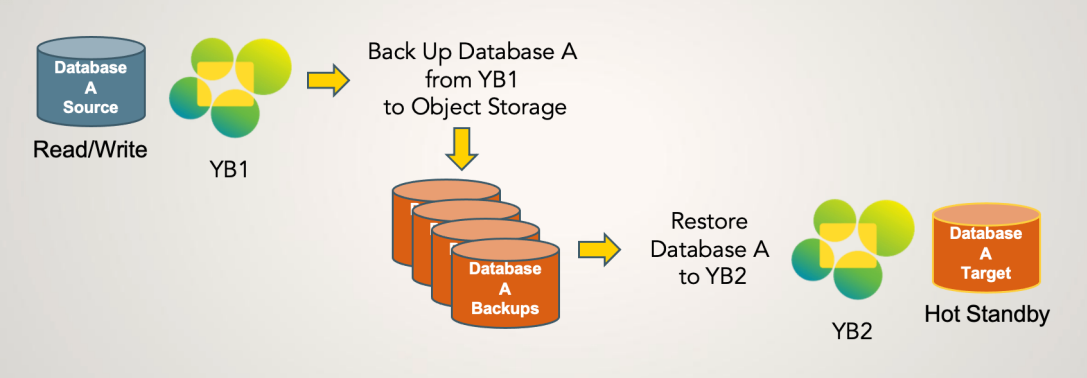
Use Cases

The backup and restore feature primarily supports disaster recovery (DR) scenarios: cases where a Yellowbrick system is irreparably damaged or lost, or a specific database on a system is compromised. You can resolve these problems by restoring databases from cold storage to the same cluster or a dedicated DR system. To keep a DR system in sync with minimal overhead, you can use incremental restore operations, which apply changes to databases after they have been fully restored. "Cold storage" may be cloud-based object storage, local file systems, or mounted file systems that are accessible to the client.

You can back up and restore a database to the same data warehouse instance where the original database was created (or to another instance on the same VPC):



Alternatively, you can restore a database to a completely separate Yellowbrick instance on a separate VPC. In this example, the backup is taken on YB1, then restored to a new database on YB2.



To facilitate database access on different systems, options are available to restore with or without shared security objects and attributes: users, roles, role memberships, and grants. You can also take backups that *exclude* specific schemas and their objects. See Scope of Backup Operations.

Another use case is the ability to make a copy of a database. You may want to copy a database under a different name, then use the new database as a separate development environment: for example, to test a specific workload or diagnose a problem without disrupting the source database.

Online Backups, Hot-Standby Restores

Yellowbrick supports *online backups*. Backup operations do not interfere with user queries or other database activities. Note that changes being introduced by running transactions that have not yet committed are not captured in a concurrently running backup. As soon as a transaction that changes the database commits, a subsequently started backup picks up those changes.

When you do an initial full restore of a database, the restored database is placed in "hot-standby" mode, allowing read queries and subsequent incremental restores but blocking other write operations. See HOT_STANDBY and READONLY Modes.

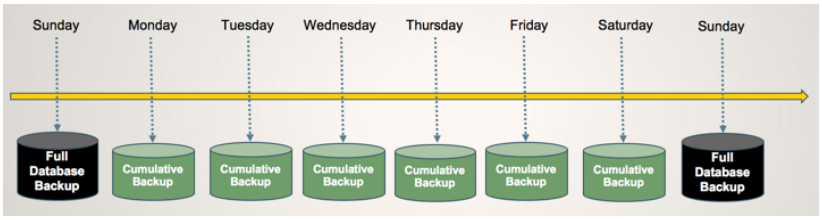
Backup Types

Yellowbrick supports backups at three different levels:

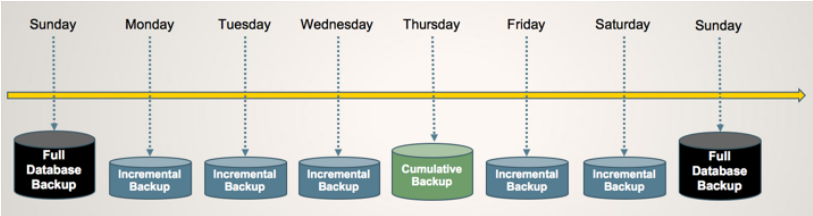
- **Full**: backs up the entire database from scratch.
- **Cumulative**: backs up all changes to the database since the last full or cumulative backup.
- **Incremental**: backs up all changes to the database since the last full, cumulative, or incremental backup.

Important: Yellowbrick recommends two specific strategies for scheduling a sequence of backups:

- Full, followed by a series of cumulatives (no incrementals), and so on. For example:



- Full, followed by a series of incrementals, then a cumulative, then another series of incrementals, and so on. For example:



Note: See Backup Strategies for important details about restoring databases when incremental and cumulative backups are mixed.

The timing of backups in both cases depends on the size of the databases in question and how long it takes to do larger cumulative or full backups. For example, in the second case a cumulative backup might occur every weekend rather than during the week. In both cases shown here, a full backup might be done only once a month or once every three months.

If you are using incremental backups, you can delete the physical backup files periodically, sometime after the cumulative backup is taken. You can use `ybbackupctl` commands to delete backups. In the same way, you can delete cumulative backups sometime after a full backup is taken.

Although other combinations of backups may work, as a best practice Yellowbrick recommends these two strategies.

Scope of Backup Operations

A single `ybbackup` command backs up a single named database. Within a full backup, you can exclude one or more schemas and all of the objects they contain. You cannot exclude tables or other objects independently.

A database backup includes the following items (or changes to the following items):

- All schemas (except where one or more schemas are explicitly excluded from full backups)
- Tables and views (temporary tables are not backed up)
- Data in tables
- Stored procedures
- Sequences
- System catalog
- Users and roles (may be excluded when the database is restored)
- Ownership and permissions on database objects, including granted privileges and role memberships (may be excluded when the database is restored)

Note: Other shared (system-level) artifacts are not backed up:

- WLM profiles
- External storage, location, and format objects
- Table statistics (they are recalculated and re-created when a database is restored)
- Encryption keys for SQL access, as created with the `CREATE KEY` command

All backup files are saved in a proprietary format, and the data is not encrypted. Users cannot access these files directly, and restore operations work only when the target is a Yellowbrick cluster. If you need to encrypt backup data at rest, implement encryption within your storage system.

Backup Chains

A *backup chain* defines a sequence of backups that are stored in the same *backup location* and belong to the same, single database. A backup location is a path to a target directory on a storage system. A chain of backups in a specific location is also known as a *backup set* or a *backup bundle*.

A full backup occurs by default the first time you back up a database under a new backup chain. When you take a new full backup of a given database in a given location, you break the existing backup chain and start a new one.

To reclaim space, it is recommended that you drop backup chains with a SQL command when they are no longer needed:

- If you are running daily full backups, drop the backup chain as soon as each full backup completes. (Alternatively, use the same backup chain name every day; when each full backup is taken, the existing chain will be overwritten.)
- If you are using incremental backups, drop the backup chain as soon as you are certain that no further backups will be required for that chain.

You can use the ybbackupctl tool to delete physical backups from the file system.

Backup Points or Snapshots

A backup point or snapshot represents a specific point in time for a database, with metadata that defines the last transaction that committed before the snapshot was created. This point in time serves as a reliable record of the database state at that time: which tables and views were present, how many rows they contained, and so on. When you back up a database with the `ybbackup` tool, a snapshot record is implicitly created in the system and is visible in the `sys.backup_snapshot` view.

Scope of Restore Operations

You can restore a database (all objects) from a backup. You can also restore security-related objects: roles, users, and ACLs. See ybrestore Options.

WLM Resources for Backup and Restore Operations

Backup and restore operations are assigned by default to the `large` pool in the `Default` profile. In some cases, this pool provides adequate resources for intensive write operations. However, you can create your own pools and WLM rules to run these operations if the `Default` profile cannot manage them efficiently.

You may need to use resource pools that have more memory or a higher maximum concurrency; 16GB per compute node is a recommended minimum amount of memory, regardless of the Yellowbrick platform where your application is running. If you do not set up your WLM profile and resource pools to allocate a 16GB minimum, backup and restore operations may run out of memory. This recommendation applies to regular backup and restore operations and database replication.

Backup versioning

Every *backup bundle* has a *bundle version* associated with it, specifying its internal format. It's used as a way of defining what set of capabilities it supports. Similarly, all the client tools that operate on backup bundles (`ybbackup` , `ybrestore` and `ybbackuptctl`) have an associated *bundle version*, which represents the latest set of capabilities they support.

Yellowbrick always ensures backwards compatibility of backup bundles. This means that newer client tools should always be able to open any old backup bundle and interpret its associated data. Backup bundles are automatically upgraded if the tools detect the version in the bundle doesn't match their default set of capabilities. This is the default mode of operation: backup tools are compatible with earlier versions.

However, once a bundle gets upgraded, older client tools may lose the ability to operate on them. It's possible to change this behaviour by overriding the compatibility mode but this is highly discouraged. Please download a new version of the tools when that happens or contact Yellowbrick's customer support.

Backing up System Databases

Yellowbrick discourages backing up system databases such as `yb_metering` and `yellowbrick` . Restoring these databases is not guaranteed to work across legacy releases, as their unique identifiers (`datuid` in `pg_database`) may change in future versions.

Backup and Restore Clients

The backup and restore clients are part of the `ybtools` package. Some of the parameters and options are the same as those used by other tools (such as the logging and connectivity options). See Client Tools and Drivers.

These tools connect to the database via JDBC as Java client applications. After you have downloaded and installed the `ybtools`, you can run the `ybbackup`, `ybrestore`, and `ybbackupctl` commands:

- `ybbackup`: for running full, cumulative, and incremental backups
- `ybrestore`: for running full and incremental restores
- `ybbackupctl`: for listing, copying, deleting, and purging backups

Note: You must run `ybbackup` and `ybrestore` commands as a database superuser. Regular users cannot run these commands. The `ybbackupctl` command does not require a database connection, so superuser access is not required.

Getting Online Help

Use one of the following options to return online help text for `ybbackup`, `ybrestore`, or `ybbackupctl`:

- `-?`
- `--help`
- `--help-advanced`

The following example returns help for the `ybbackup` command:

```
myuser@myybd:~/usr/bin$ ./ybbackup -?
Yellowbrick Backup and Restore version 6.1.3-20221102160444
Backup one database and associated metadata on a Yellowbrick appliance to the specified bundle

Common Options:
  -?,  --help
     Display this help message
  --version
     Shows the version number
...
```

Setting Up a Database Connection

To run backup and restore operations, you have to start a database session on the server where the target database resides. This session requires connection information that you can provide either as `ybbackup` or `ybrestore` options on the command line or as current values for environment variables.

Note: These connection parameters are not required for `ybbackupctl` commands. These commands do not require a database connection.

Command-Line Options	Environment Variable	Description	Example

Command-Line Options	Environment Variable	Description	Example

`-h` or `--host`	`YBHOST`	Destination server host name. Default: `localhost`	`-h test.ybsystem.io`
			
			`export YBHOST=test.ybsystem.io`

`-p` or `--port`	`YBPORT`	Destination server port number. Default: `5432`	`--port 5433`
			
			`export YBPORT=5433`

`-U` or `--username`	`YBUSER`	Database login username. No default.	`-U bobr`
			
			`export YBUSER=bobr`
			
			Note: You must run `ybbackup` and `ybrestore` commands as a database superuser. Regular users cannot run these commands. You can run the `ybbackupctl` command as any database user because this command does not require a connection to a database.

`-W` or `--password`	`YBPASSWORD`	Interactive prompt for the database user's password. No default.	`--password`
			
			`export YBPASSWORD=********`

Backup and Restore Glossary

backup bundle
: All of the artifacts that are stored in a physical backup of a single database.

backup chain
: A logical name for a sequence of backups that are chained together. Typically, one database has one chain.

backup point
: A logical name for a point in time when a backup was taken, similar to a "snapshot."

backup set
: All of the artifacts that are stored in a physical backup of a single database. Also known as a *backup bundle*.

backup snapshot
: A logical name for a point in time when a backup was taken. Also known as a *backup point*.

cumulative backup
: A backup that consists of all changes to the database since the last full or cumulative backup. (Also known as a *differential backup*.)

full backup
: A backup of the entire database.

hot-standby database
: A database that is primed for incremental restore operations but also available for reads (queries).

incremental backup
: A backup that consists of all changes to the database since the last backup of any kind.

incremental restore
: A restore operation that uses one or more incremental or cumulative backups as its restore point. Restore operations work from a restore sequence that starts with a full backup. An incremental restore applies a subsequent set of changes to the restored database. A restored database must remain in `HOT_STANDBY` mode to be eligible for additional incremental restore operations.

read-only database
: A database that has been locked down for read operations only.

replication
: Creation and maintenance of database replicas for disaster recovery, typically on remote systems.

restore point
: A specific point in time that a database is restored to, given a sequence of backups.

restore sequence
: A sequence of restore operations, as needed to replay changes saved in a series of backups.

snapshot d: A point in time for a database that captures its exact state as of its last committed transaction. Snapshots provide valid backup and restore points.

Backup Strategies

This section presents some examples that highlight how backups can be scheduled and combined to create an effective strategy.

Cumulative and incremental backups have some fundamental differences. A cumulative backup contains and subsumes changes that were captured by previous incremental backups. This means that a restore operation does not need the incremental backups when the database is restored to its state after a cumulative backup. The restore can skip the incremental backups and only has to "play back" the latest full backup and cumulative backup.

Cumulative and full backups both reset the "horizon" (or starting point) for subsequent backup operations. An incremental backup that follows a cumulative or full backup only picks up changes made since that backup. An incremental backup does not subsume any changes from any previous backups and does not reset the horizon for incremental restore operations. When restoring the database to its state after one or more incremental backups, restore operations cannot skip over those backups. They must be "played back" in order.

Important: A side-effect of these differences in backup types is that you cannot restore a database from a cumulative backup after you have already restored that database from an incremental backup. If you plan to restore a database periodically, as part of a routine schedule, you should restore it from the last cumulative or full backup. For example, if you plan to do proactive daily restores of a database, take *daily cumulative backups* of that database. Restoring from an incremental backup is recommended only in case of a catastrophic failure, or if your strategy involves full and incremental backups only (no cumulative backups).

Some examples of restore paths follow in this section.

Restore Paths

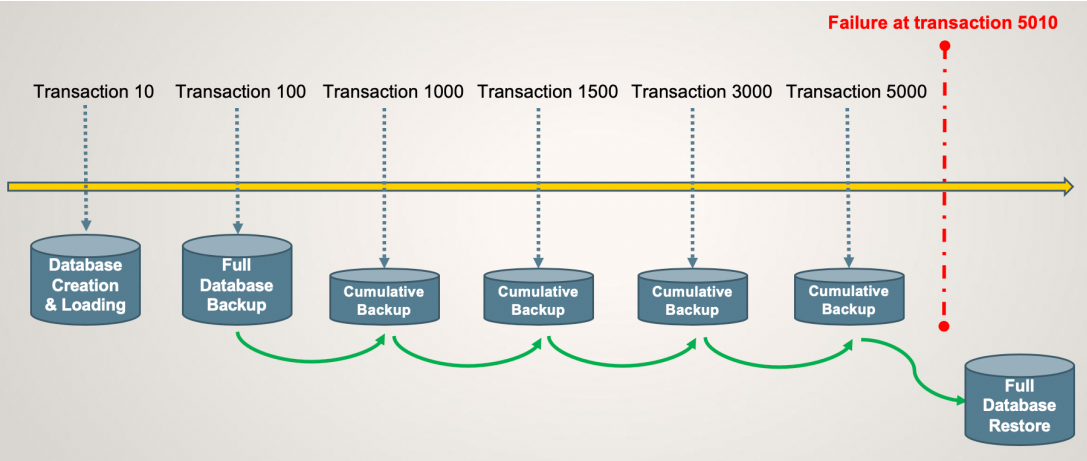
Assume that a database has the following backups:

1. A full backup taken on Monday (F1)
2. An incremental backup taken on Tuesday (I2)
3. An incremental backup taken on Wednesday (I3)
4. A cumulative backup taken on Thursday (C4)

These backups are all valid "snapshots" of the database at a point in time. Each snapshot is a valid "restore point," but note the restore behavior when incremental and cumulative backups occur in a sequence.

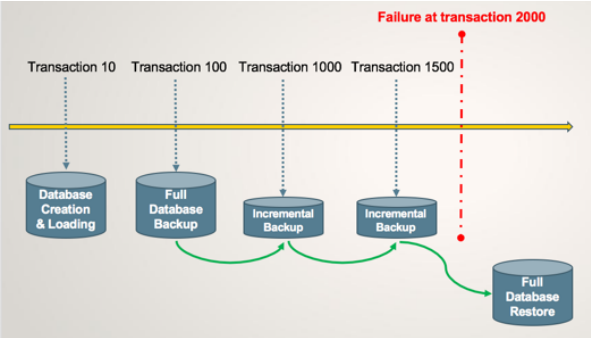
- A restore to F1 replays F1.
- A restore to I2 replays F1 and I2. *A subsequent attempt to restore to C4 will fail.*
- A restore to I3 replays F1, I2, and I3. *A subsequent attempt to restore to C4 will fail.*
- A restore to C4 replays F1 and C4 only.

In the following example, the restore path has multiple steps because a failure occurs after a full backup and a series of cumulative backups. Any writes that occurred between transactions `5000` and `5010` would be lost, but the database could be fully recovered to the latest backup point, using the original full backup and the last four cumulative backups. All of the cumulative backups need to be applied as part of the restore sequence because they contain discrete sets of changes (assuming that continuous changes were made during that time frame). Note that this is a single restore operation from the user's point of view. You do not have to run multiple `ybrestore` commands to pick up each backup in the restore path.

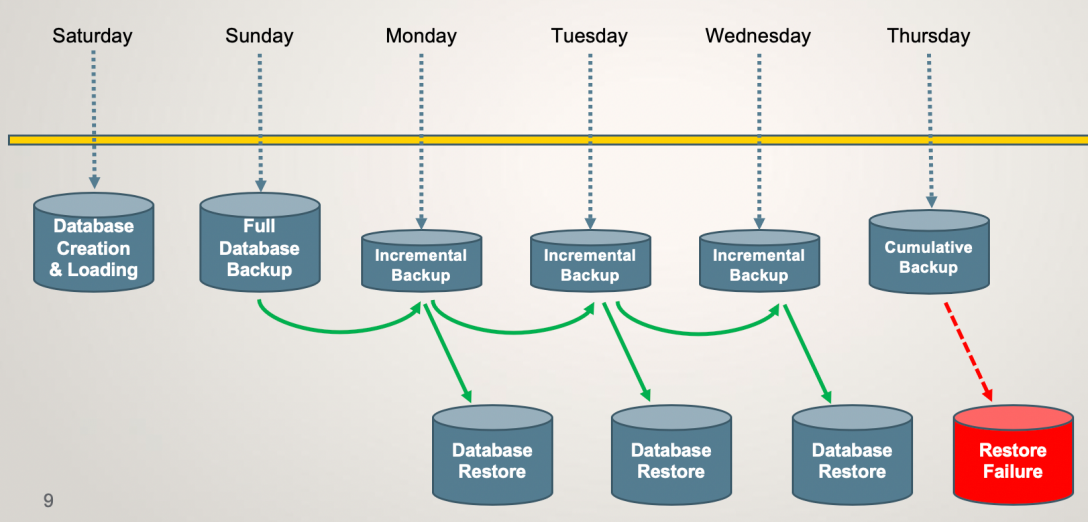


Because only full and cumulative backups are used in this case, the database can be restored from any one of the cumulative backups without compromising subsequent backups in the chain. For example, you could restore to the backup at transaction `1500`, then later restore to the backup at transaction `5000` if you needed to do that.

The following diagram shows a similar restore path when a failure occurs after a full backup and two incremental backups. Any writes that occurred between transactions 1500 and 2000 would be lost, but the database could be fully recovered to the latest backup point, using the original full backup and two subsequent incremental backups. Again, from the user's point of view, this is a single restore operation.



In the third example, a database is being incrementally restored every day. This works fine until a cumulative backup is taken after restores from incremental backups have already occurred. These restores break the restore path for the cumulative backup.



The solution in this case is one of the following:

- Take only full and cumulative backups; do not use incremental backups.
- Proactively restore only from full or cumulative backups. Restore from an incremental backup only as an emergency measure.

Choosing a Backup Strategy

In choosing your backup schedule and strategy, keep the following factors in mind:

- Backups take place online so you do not need to schedule backups for periods when users are offline; however, backups require system resources that may detract from the resources available to other workloads.
- How much and how often does the data in your databases actually change?
- How much data can you afford to lose if an unexpected system failure occurs? (Or how long will it take to reload the lost data?)
- Incremental backups take less time and resources to run than cumulative backups.
- A mixture of incremental and cumulative backups may not serve your purpose for periodic restores. As noted earlier, having restored a database from an incremental backup, you will not be able to restore it from a subsequent cumulative backup.
- You can delete a series of prior incremental backups once you have taken a cumulative backup. (Or you can just move the incrementals to another system for audit purposes.)
- A series of cumulative backups may result in a waste of storage space, depending on the extent of the changes that they pick up.

Restore Operations

Platforms: All platforms

Parent topic: Overview

The first restore of a database starts by replaying a full backup and any subsequent cumulative or incremental backups that are required to reach the specified restore point. Incremental restores provide an efficient way to continuously update a restored database with incremental changes (based on cumulative or incremental backups). This method is useful for maintaining DR systems and keeping them in sync with source databases. This approach to database recovery is a simpler alternative to setting up database replication. See Running Incremental Restores.

Restore operations provide some flexibility in terms of the selected *restore point* you want to restore. A restore point is a reference to a particular backup that you want to restore from. You can restore from any point in a backup chain, and you can restore one database while other users continue to access other databases on the same system.

You can specify the restore point for each database in several different ways:

- Use known keywords to specify the `latest` or `oldest` backup
- Specify the backup name (the "snapshot")
- Specify a point in time that matches the timestamp for a listed snapshot

Based on the restore point you choose, the system automatically determines the restore path (the sequence of full, cumulative, and incremental backups) that need to be restored, one by one.

Each restore in the sequence is committed separately. If a specific restore fails to commit, the database is restored to the point of the previous successful commit.

Note: During a restore operation, the database that is being restored is accessible to the `yellowbrick` superuser. It is strongly recommended that you do not log in to the database until the restore is complete.

ybbackup Commands

Platforms: All platforms

Parent topic: Backup & Restore

In this section:

Each `ybbackup` command defines:

- Database connection options
- A target database (database name)
- The backup type: full, cumulative, or incremental
- The backup name (which must be unique across all backup chains; do not reuse backup names)
- The backup chain
- Other options, such as logging parameters
- The target location(s) where the backup will be stored (URIs)

You must run the `ybbackup` tool as a superuser.

Syntax Summary

```
./ybbackup
--dbname database_name
--full | --cumu | --inc
--name backup_name
--chain backup_chain
[ database_connection_options ]
[ other_options ]
[ storage_locations ]
```

For details about the options and their syntax, see ybbackup Options.

To look at your backup history, use the `ybbackupctl --list` command or query the sys.log_backup view.

Storage Locations for Backups

Two main types of storage locations are supported:

- Local or locally mounted file systems (relative paths are allowed)
- Object storage:
- AWS S3 (and S3-compatible storage). For example: `s3://bucket/backup/`
- Microsoft Azure Blob. For example: `azure://container/backup/`
- Microsoft Azure Data Lake. For example: `azdl2://container/backup/`

Storage locations are not prefaced with an option name and may appear anywhere in the command line. For clarity, a good practice is to specify storage locations at the end of the command. For example:

```
$ ybbackup -d premdb --name Oct4full --full --chain October2019
/home/yb100/premdb_daily_backups
```

This example specifies two directories:

```
$ ybbackup -d premdb --name Oct4full --full --chain October2019
/home/yb100/premdb_daily_backups/friday1
/home/yb100/premdb_daily_backups/friday2
```

The same behavior and requirements apply when you specify source file directories in `ybbackupctl` commands.

Multiple backup chains and databases may be backed up to the same storage location. Backups that belong to different chains and databases are stored in distinct sets of files. Only the metadata files are common.

ybbackup Examples

This section contains some simple examples of backup commands, using different options.

Full backup of a database

This example backs up a database for the first time. Here is a breakdown of the `ybbackup` command options used in this example:

Command Syntax	Explanation
`ybbackup`	Call the `ybbackup` tool on the client.
`-W`	Prompt for the user's password.
`--username yb100`	Run as user `yb100`.
`-d premdb`	Back up the `premdb` database.
`--full`	Take a full backup.
`--name Oct23Full`	Name the backup `Oct23Full`.
`--chain October2019`	Name the backup chain `October2019`.

`/home/yb100/premdb_daily_backups`	Store the backup files in this local directory (no option name required, just the path). An S3 storage path would look like this:
	
	`s3://ybbobr/premdb_bar`
	
	An Azure Blob storage path would look like this:
	
	`azure://premdb_bar`

Note: The host is not set explicitly in this case. It is set with the `YBHOST` environment variable. For example:

```
$ export YBHOST=ybdemo.devue1.myawsvpc.com
```

Here is the command line and the full output of the backup operation:

```
~$ ybbackup -W --username yb100 -d premdb --full --name Oct23Full --chain October2019 /home/yb100/premdb_daily_backups
Password for user yb100:
15:16:16.003 [ INFO] ABOUT CLIENT:
```

```
    app.cli_args         = "-W" "--username" "yb100" "-d" "premdb" "--full" "--name" "Oct23Full" "--chain" "October2019" "/home/yb1
    app.name_and_version = "Yellowbrick Backup and Restore version 4.0.0-1464"
    java.home            = "/usr/lib/jvm/java-8-oracle/jre"
    java.version         = "1.8.0_101"
    jvm.memory           = "512.00 MB (max=4.00 GB)"
    jvm.name_and_version = "Java HotSpot(TM) 64-Bit Server VM (build 25.101-b13, mixed mode)"
    jvm.options          = "-Xms512m, -Xmx4g, -XX:+UseG1GC, -Dapp.name=ybbackup2, -Dapp.pid=31652, -Dapp.repo=/opt/ybtools/lib, -Da
    jvm.vendor           = "Oracle Corporation"
    os.name_and_version  = "Linux 4.4.0-31-generic (amd64)"

15:16:16.004 [ INFO] Running backup using output locations: [file:///home/yb100/premdb_daily_backups]
15:16:16.670 [ INFO] Collecting metadata for object premdb
15:16:16.756 [ INFO] Cluster UUID: 0620603c-91fb-43b0-a5b7-2bcd2ba2cc84
15:16:16.756 [ INFO] DB      UUID: 210233bb-9477-23cd-1eae-2cacffeae2be
15:16:16.756 [ INFO] Encoding    : 16
15:16:16.756 [ INFO] CType       : C
15:16:16.756 [ INFO] Collation   : C
15:16:16.756 [ INFO] Metadata collection for object premdb complete
ybbackup_meta | 100.0% |         Read: 7.26 KiB / 7.26 KiB         |     Write: 7.29 KiB     |  0.00 B/s   |   2019-10-23 15:16:18

ybbackup_data | 100.0% |       Read: 950.68 MiB / 950.68 MiB       |    Write: 950.96 MiB    |  4.07 MB/s  |   2019-10-23 15:17:14
```

Incremental backup of the same database after loading new rows

The only changes to the previous command are the backup type (`--inc`) and the backup name (`--name Oct23Inc1`):

```
$ ybbackup -W --username yb100 -d premdb --inc --name Oct23Inc1 --chain October2019 /home/yb100/premdb_daily_backups
Password for user yb100:
15:58:33.395 [ INFO] ABOUT CLIENT:
...
15:58:33.396 [ INFO] Running backup using output locations: [file:///home/yb100/premdb_daily_backups/]
15:58:35.548 [ INFO] Collecting metadata for object premdb
15:58:35.602 [ INFO] Cluster UUID: 0620603c-91fb-43b0-a5b7-2bcd2ba2cc84
15:58:35.602 [ INFO] DB      UUID: 210233bb-9477-23cd-1eae-2cacffeae2be
15:58:35.602 [ INFO] Encoding    : 16
15:58:35.603 [ INFO] CType       : C
15:58:35.603 [ INFO] Collation   : C
15:58:35.603 [ INFO] Metadata collection for object premdb complete
ybbackup_meta | 100.0% |         Read: 1.29 KiB / 1.29 KiB         |     Write: 1.31 KiB     |  0.00 B/s   |   2019-10-23 15:58:36

ybbackup_data | 100.0% |       Read: 951.68 MiB / 951.68 MiB       |    Write: 951.97 MiB    |  4.23 MB/s  |   2019-10-23 15:59:29
```

Backup with multiple storage locations

This example takes a single cumulative backup of a database and spreads the backup files across multiple directories:

```
$ ybbackup -W --username yb100 -d worldcupdb --cumu --name worldcupOct23Cumu --chain wcdb
/home/ybbackups/wcdb_daily_backups/bk1
/home/ybbackups/wcdb_daily_backups/bk2
/home/ybbackups/wcdb_daily_backups/bk3
/home/ybbackups/wcdb_daily_backups/bk4
```

Note that the result of this type of command is *one backup* stored across multiple locations. To restore from this backup, make sure you name the first backup directory in the `ybrestore` command. This directory contains the metadata files (`metaDB.*`) for the backup.

Full backup that excludes a schema

This command takes a full backup of the specified database but excludes one of its schemas, using the `--exclude` option.

```
$ ybbackup -W --username yb100 -d worldcupdb --full --name worldcupOct24Full
--exclude uefa --chain wcdb /home/ybbackups/wcdb_daily_backups
```

Full backup to object storage

The following command takes a full backup and writes the files to an AWS S3 bucket. Note the "object store" options and the path to S3 where the backup files will be stored. This example also uses the `--temp-workdir` option.

```
% ./ybbackup --full --name Feb17Full --chain Feb2023
--object-store-region us-west-2 --object-store-identity ***************** --object-store-credential ****************************
--temp-workdir /Users/brumsby/tmp_backups
s3://ybbobr/premdb_bar
14:50:14.474 [ INFO] ABOUT CLIENT:
...
14:50:14.475 [ INFO] Running backup using output locations: [s3://ybbobr/premdb_bar]
...
14:50:16.316 [ INFO] Loaded file handler for s3://
14:50:16.317 [ INFO]
Configuration (S3 client):
   endpoint           : ********
   region             : us-west-2
   identity/credential: ********
14:50:17.833 [ INFO] Collecting metadata for object premdb
...
14:51:40.972 [ INFO] Done
```

ybbackup Options

This section contains detailed descriptions of the `ybbackup` command-line options.

See also:

- Common Options in ybtools
- Object Storage Options
- ybrestore Options

Note the following points about the format of these options and their values:

- Options are listed in alphabetical order for quick reference.
- Option names are shown in lowercase; *they are case-sensitive*.
- Specific valid option values (such as `true` and `false`) are shown in lowercase. Variables for option values, such as `STRING`, are shown in uppercase. Option values *are not case-sensitive*.
- The requirements for quoting option strings vary by client platform. Values are shown without quotes, but quotes are sometimes required. For example, if you specify the `#` character in a Linux shell, it must be enclosed by single or double quotes.

Note: Storage locations for backup files are not prefaced with an option name. See Storage Locations for Backups.

To see all of these options in the online help, run the `ybbackup --help` and `ybbackup --help-advanced` commands.

-c STRING, --chain STRING

Specify the name of the backup chain. You can use any meaningful name that defines the contents of a particular backup set. If you do not specify this option, the chain is named `default`.

Multiple chains and databases may be backed up to the same storage location. Backups that belong to different chains and databases are stored in distinct sets of files.

--compatibility-mode BACKWARD | FORWARD | ALL | NONE

Allow/forbid running commands on backup bundles with newer/older bundle versions than the client tools bundle version.

- `BACKWARD` allows operating on backup bundles with bundle versions same as or older than the client tools bundle version.
- `FORWARD` allows operating on backup bundles with bundle versions same as or newer than the client tools bundle version.
- `ALL` allows operating on all backup bundles.
- `NONE` allows operating on backup bundles with the same bundle version as the client tools bundle version.

By default `BACKWARD` is chosen.

-C, --cumu

Use this option to take a cumulative backup.

-d, --dbname

Name the database to back up. You must specify one database only.

--desc STRING

Include a description of the backup. The maximum length is 256 characters.

--exclude SCHEMA

Specify the names of schemas that you want to exclude from a full backup. Database objects that belong to the specified schemas will not be backed up. This option may only be used for full backups; however, all backups in a given chain must respect the exclude list as defined for the first full backup. You cannot define an exclude list in the first backup, then change it in a subsequent full backup. The system depends on a consistent scope for all the backups in a single backup chain.

You can use a comma-separated list or a wildcard character to specify multiple schemas. Do not include any blanks (space characters) in the list; they will be treated as part of the schema name. For example:

```
--exclude public,dev,new
--exclude dev*
--exclude yb*
```

-f, --full

Use this option to take a full backup.

-h, --host

Specify the database server host. Alternatively, you can set this value with the `YBHOST` environment variable. Defaults to `localhost`.

-I, --inc

Use this option to take an incremental backup.

--integrity-check NONE | FAST | STRICT

Specify integrity checking (checksum validation) of backup files after they have been written to storage, if required. Integrity checking has an impact on backup performance but also reduces the likelihood that subsequent restore operations will fail because backup files are corrupted.

`NONE` (the default): no integrity checking; no impact on backup performance.

`FAST`: file-level integrity checking; reduces performance by a small percentage.

`STRICT`: stricter integrity checking, including packet placements inside each file; reduces performance by a greater amount.

--name

Specify a unique name (across all backup chains) for this backup snapshot. Do not try to reuse the same snapshot name within different backup chains.

The name must be no greater than 64 characters. Directory paths are not allowed. For example:

```
--name sept6_2019_premdb_weekly_cumu
```

The backup name must contain only the following valid ASCII characters:

- A-Z
- a-z
- 0-9
- _

Mixed-case names are allowed, and the backup storage file system must support Linux-style mixed-case file names.

The `--name` value in the `ybbackup` command corresponds to the `snapshot_name` in the output for `ybbackupctl` commands.

--retry-failed-backups, --no-retry-failed-backups

Enable retrying up to `--retry-per-table-max` times if a table fails to back up during the session. This option is only applicable to YBD instances that also support retrying.

--retry-per-table-max NUMBER

The maximum number of retry attempts for a single table before the program terminates with an error.

--temp-workdir STRING

Use the specified location as a temporary working directory for backup bundle processing. By default, the current working directory is used. Backup and restore operations that write or read from object storage require a writable local directory during processing. When processing is finished, this directory is empty.

ybbackupctl Commands

Platforms: All platforms

Parent topic: Backup & Restore

In this section:

The `ybbackupctl` tool runs commands that help you manage backups on the client system:

- List the contents of a backup set and identify the chain name, the type and name of each backup in the set, the number of tables in the backup, its timestamp, and its size on disk. This information is useful for checking backup history and for identifying restore points when a database needs to be restored.
- Copy a backup to another physical location.
- Delete a backup from its physical location because it is no longer needed.
- Purge obsolete `FULL` backups or `INVALID` (partial) backups in order to clean up the storage system and reclaim space.
- Fix paths to backups that have been moved since the backup was taken, or need to be resolved from a different mount point.
- Return help text for the tool itself.

Syntax Summary

The `ybbackupctl` syntax takes one of the following "control commands," followed by certain options. The required options vary by command.

```
ybbackupctl
--copy-to |
--delete |
--fix-bundle-paths |
--help |
--list |
--purge
[ options ]
```

For more details about the complete syntax, see ybbackupctl Options.

You do not need to connect to the database to run these commands, so do not try to set the user and password options that are required for `ybbackup` and `ybrestore` tools.

Any user can run the `ybbackupctl` commands.

An alternative way to look at your backup history is to query the sys.log_backup view.

ybbackupctl Examples

The following examples show how to use the `ybbackupctl` tool to list, delete, purge, or copy backups.

List Backups with the --verbose option

```
$ ybbackupctl --list /home/yb100/premdb_daily_backups --verbose
16:02:32.348 [ INFO] ABOUT CLIENT:
   app.cli_args         = "--list" "/home/yb100/premdb_daily_backups" "--verbose"
   app.name_and_version = "Yellowbrick Backup and Restore version 4.0.0-1385"
   java.home            = "/usr/lib/jvm/java-8-oracle/jre"
   java.version         = "1.8.0_101"
   jvm.memory           = "512.00 MB (max=4.00 GB)"
   jvm.name_and_version = "Java HotSpot(TM) 64-Bit Server VM (build 25.101-b13, mixed mode)"
   jvm.options          = "-Xms512m, -Xmx4g, -XX:+UseG1GC, -Dapp.name=ybbackupctl, -Dapp.pid=12991, -Dapp.repo=/opt/ybtools/lib, -
   jvm.vendor           = "Oracle Corporation"
   os.name_and_version  = "Linux 4.4.0-31-generic (amd64)"

16:02:32.349 [ INFO] Running backup control using input locations: [file:/home/yb100/premdb_daily_backups/]
Database: premdb (057af4c6-b817-4181-0b51-0f7e65df3df0)
Chain: October2019
   Tables                                                  size    deleted rows   inserted rows
   (6 total)                                               (GB)         (total)         (total)
     public.hometeam                                       0.00               0              50
     public.awayteam                                       0.00               0              50
     public.season                                         0.00               0              25
     public.match                                          0.04               0         2203136
     public.team                                           0.00               0              50
     public.newmatchstats                                  0.24               0        24785280

Restore points:

       type                            snapshot_name            backup_timestamp   size (GB)
       FULL                            October15Full 2019-10-15T22:16:23.108024Z        0.00
 INCREMENTAL                            October15Inc1 2019-10-15T22:20:42.999975Z        0.01
 INCREMENTAL                            October15Inc2 2019-10-15T22:55:10.787697Z        0.23
 INCREMENTAL                            October15Inc3 2019-10-15T23:02:15.944328Z        0.04
```

Remove Backups

You can use the `ybbackupctl --delete` command to remove specific backups from the system. For example:

```
$ ybbackupctl --delete --point Feb16Inc --uuid 'fed0f424-94c8-3945-b3cc-b7e926db03e8' --chain February2020 /home/yb100/premdb_dail
21:43:16.743 [ INFO] ABOUT CLIENT:
   app.cli_args         = "--delete" "--point" "Feb16Inc" "--uuid" "fed0f424-94c8-3945-b3cc-b7e926db03e8" "--chain" "February2020"
   app.name_and_version = "Yellowbrick Backup and Restore version 4.0.0-21024"
   java.home            = "/usr/lib/jvm/java-8-oracle/jre"
   java.version         = "1.8.0_101"
   jvm.memory           = "512.00 MB (max=8.00 GB)"
   jvm.name_and_version = "Java HotSpot(TM) 64-Bit Server VM (build 25.101-b13, mixed mode)"
   jvm.options          = "-Xms512m, -Xmx8g, -XX:+UseG1GC, -Dapp.name=ybbackupctl, -Dapp.pid=17155, -Dapp.repo=/opt/ybtools/lib, -
```

```
   jvm.vendor           = "Oracle Corporation"
   os.name_and_version  = "Linux 4.4.0-31-generic (amd64)"

21:43:16.744 [ INFO] Running backup control using input locations: [file:///home/yb100/premdb_daily_backups/]
21:43:17.989 [ INFO] Removing backup:[Feb16Inc] from database:[premdb/fed0f424-94c8-3945-b3cc-b7e926db03e8] chain:[February2020] a
```

Purge Backups

You can purge obsolete `FULL` backups and `INVALID` backups. The following example finds two full backups of the same database and purges the older one:

```
$ ybbackupctl --purge FULL /home/yb100/premdb_daily_backups --chain February2020 --uuid fed0f424-94c8-3945-b3cc-b7e926db03e8
21:47:59.726 [ INFO] ABOUT CLIENT:
...
21:47:59.728 [ INFO] Running backup control using input locations: [file:///home/yb100/premdb_daily_backups/]
21:48:01.347 [ INFO] Removing backup [Feb14Full/FULL] from backup chain: [February2020] taken from: [premdb/fed0f424-94c8-3945-b3c
```

An invalid backup is a partial backup that failed at some point in the `ybbackup` operation. In this case, an invalid backup was found and deleted:

```
$ ybbackupctl --purge INVALID /home/yb100/premdb_backups --chain Sept2019 --uuid 0bee7ef7-0736-6938-5f33-2b9b419c8d30
17:47:31.597 [ INFO] ABOUT CLIENT:
   app.cli_args         = "--purge" "INVALID" "/home/yb100/premdb_backups" "--chain" "Sept2019" "--uuid" "0bee7ef7-0736-6938-5f33-
   app.name_and_version = "Yellowbrick Backup and Restore version 4.0.0-913"
...
Removing backup [Sept9_2019_IncAgain/INCREMENTAL] from backup chain: [Sept2019] taken from: [0bee7ef7-0736-6938-5f33-2b9b419c8d30/
```

Copy a Backup to Another Location

In this example the backup is copied from the *current working directory* (which is `premdb_daily_backups`) to a directory called `CopyPremdbFeb18Full`:

```
yb100@yb100:~/premdb_daily_backups$ ybbackupctl --copy-to /home/yb100/CopyPremdbFeb18Full --point Feb18Full --chain February2020 -
20:59:47.391 [ INFO] ABOUT CLIENT:
   app.cli_args         = "--copy-to" "/home/yb100/CopyPremdbFeb18Full" "--point" "Feb18Full" "--chain" "February2020" "--uuid" "f
   app.name_and_version = "Yellowbrick Backup and Restore version 4.0.0-21024"
   java.home            = "/usr/lib/jvm/java-8-oracle/jre"
   java.version         = "1.8.0_101"
   jvm.memory           = "512.00 MB (max=8.00 GB)"
   jvm.name_and_version = "Java HotSpot(TM) 64-Bit Server VM (build 25.101-b13, mixed mode)"
   jvm.options          = "-Xms512m, -Xmx8g, -XX:+UseG1GC, -Dapp.name=ybbackupctl, -Dapp.pid=3195, -Dapp.repo=/opt/ybtools/lib, -D
   jvm.vendor           = "Oracle Corporation"
   os.name_and_version  = "Linux 4.4.0-31-generic (amd64)"

20:59:47.392 [ INFO] Running backup control using input locations: [file:/home/yb100/premdb_daily_backups/]
20:59:56.666 [ INFO] Successfully copied backup [snapshot_name=Feb18Full]
yb100@yb100:~/premdb_daily_backups$ cd /home/yb100/CopyPremdbFeb18Full
yb100@yb100:~/CopyPremdbFeb18Full$ ls
05  0f  21  26  35  38  53  58  5d  74  87  95  9d  a8  b7  d5  f9  metaDB.mv.db  metaDB.mv.db.backup.Tue  metaDB.sql  metaDB.trac
yb100@yb100:~/CopyPremdbFeb18Full$
```

The following equivalent `ybbackupctl copy-to` command explicitly specifies both the destination directory (in the same place as before) and the input directory (at the end of the command). That is, this version of the command does not rely on the location of the user's current working directory.

```
$ ybbackupctl --copy-to /home/yb100/CopyPremdbFeb18Full --point Feb18Full --chain February2020 --uuid 'fed0f424-94c8-3945-b3cc-b7e
20:59:47.391 [ INFO] ABOUT CLIENT:
...
```

Copy a Backup to Multiple Locations

You can specify the `--copy-to` option multiple times if you want to copy the same backup to multiple locations:

```
yb100@yb100:~/premdb_daily_backups$ ybbackupctl
--copy-to /home/yb100/CopyPremdbFeb18Full
--copy-to /home/yb100/CopyPremdbFeb18Full2
--copy-to /home/yb100/CopyPremdbFeb18Full3
--point Feb18Full --chain February2020 --uuid 'fed0f424-94c8-3945-b3cc-b7e926db03e8'
```

List Backups in Object Storage

The following example lists backups that are in S3 object storage.

```
% ./ybbackupctl --list s3://ybbobr/premdb_bar --verbose
--object-store-region us-west-2 --object-store-identity ****************** --object-store-credential **************************
--temp-workdir /Users/brumsby/tmp_backups
14:20:48.517 [ INFO] ABOUT CLIENT:
   app.cli_args         = --list --verbose --temp-workdir /Users/brumsby/tmp_backups --object-store-region us-west-2 --object-stor
...
14:20:48.519 [ INFO] Running backup control using input locations: [s3://ybbobr/premdb_bar]
14:20:49.051 [ INFO] Loaded file handler for s3://
14:20:49.053 [ INFO]
Configuration (S3 client):
   endpoint           : ********
   region             : us-west-2
   identity/credential: ********
Database: premdb (001ac67a-5861-e9e2-cf9b-e3ddd66f0b7a)
Chain: Feb2023
   Tables                                                  size    deleted rows   inserted rows
   (6 total)                                               (GB)         (total)         (total)
     premdb.premdb.awayteam                                0.00               0             100
     premdb.premdb.hometeam                                0.00               0             100
     premdb.premdb.match                                   0.00               0           17212
     premdb.premdb.newmatchstats                           0.02               0         1549080
     premdb.premdb.season                                  0.00               0              50
     premdb.premdb.team                                    0.00               0             100

Restore points:

       type                              snapshot_name            backup_timestamp   size (GB)
       FULL                                  Feb17Full 2023-02-17T22:50:18.433882Z        0.01
       FULL                                  Feb20Full 2023-02-21T03:37:30.527405Z        0.01
```

Copy Backups from Object Storage

This example copies backups to local storage from the location shown in the previous example. Use the UUID for the database, as shown in the list output. The copy-to destination location is specified first; the S3 source is specified at the end of the command:

```
brumsby@Bob-Rumsby-sMac bin % ./ybbackupctl --copy-to /Users/brumsby/premdb_backups --point latest --chain Feb2023 --uuid '001ac67
--object-store-region us-west-2 --object-store-identity ****************** --object-store-credential **************************
--temp-workdir /Users/brumsby/tmp_backups s3://ybbobr/premdb_bar
14:47:54.549 [ INFO] ABOUT CLIENT:
...
14:47:54.551 [ INFO] Running backup control using input locations: [s3://ybbobr/premdb_bar]
14:47:55.235 [ INFO] Loaded file handler for s3://
14:47:55.236 [ INFO]
Configuration (S3 client):
```

```
   endpoint           : ********
   region             : us-west-2
   identity/credential: ********
14:48:06.838 [ INFO] Successfully copied backup [snapshot_name=Feb20Full]
```

Fix the Path to a Backup After Moving It

In this example, backup files have been moved from their original (or expected) location. The `ybbackupctl` command fixes the path to the backup bundle by using the `--fix-bundle-paths` option. Subsequent backup and restore operations will be able to use the updated path.

In this case, the original location of some backups was renamed to: `/home/yb100/premdb_february2020_backups`. The command checks the original location and the renamed location to verify that each file exists at the new location.

```
$ ybbackupctl -X /home/yb100/premdb_february2020_backups
22:45:47.248 [ INFO] ABOUT CLIENT:
   app.cli_args         = "-X" "/home/yb100/premdb_february2020_backups"
   app.name_and_version = "Yellowbrick Backup and Restore version 4.0.0-21176"
   java.home            = "/usr/lib/jvm/java-8-oracle/jre"
   java.version         = "1.8.0_101"
   jvm.memory           = "512.00 MB (max=8.00 GB)"
   jvm.name_and_version = "Java HotSpot(TM) 64-Bit Server VM (build 25.101-b13, mixed mode)"
   jvm.options          = "-Xms512m, -Xmx8g, -XX:+UseG1GC, -Dapp.name=ybbackupctl, -Dapp.pid=18236, -Dapp.repo=/opt/ybtools/lib, -
   jvm.vendor           = "Oracle Corporation"
   os.name_and_version  = "Linux 4.4.0-31-generic (amd64)"

22:45:47.249 [ INFO] Running backup control using input locations: [file:///home/yb100/premdb_february2020_backups/]
22:45:48.546 [ INFO] Found bundle. Locations stored in the bundle=[file:///home/yb100/premdb_daily_backups] : [UUID=e80517b2-a944-
22:45:48.875 [ INFO] Verifying files for [Database=premdb] [Backup chain=February2020] [Snapshot name=Feb20Full] [Backup type=FULL
22:45:48.875 [ INFO] Verifying backup file: [relative path=/0f/be/5d/2f/bc/0fbe5d2f-bcc2-4ab0-a804-83d739f81ff6] [file size=5811]
22:45:48.876 [ WARN] [File=/0f/be/5d/2f/bc/0fbe5d2f-bcc2-4ab0-a804-83d739f81ff6] was not found at the expected [location=file:///h
22:45:48.888 [ WARN] [File=/0f/be/5d/2f/bc/0fbe5d2f-bcc2-4ab0-a804-83d739f81ff6] was found at the new [location=file:///home/yb100
22:45:48.896 [ INFO] Verifying backup file: [relative path=/4d/4d/af/05/a8/4d4daf05-a837-47a9-8228-9d6aa8702014] [file size=283557
22:45:48.896 [ WARN] [File=/4d/4d/af/05/a8/4d4daf05-a837-47a9-8228-9d6aa8702014] was not found at the expected [location=file:///h
22:45:48.901 [ WARN] [File=/4d/4d/af/05/a8/4d4daf05-a837-47a9-8228-9d6aa8702014] was found at the new [location=file:///home/yb100
22:45:48.903 [ INFO] Verifying backup file: [relative path=/66/cb/96/f9/aa/66cb96f9-aaf1-4ee4-b6fb-204ab1a8b85a] [file size=3603]
...
```

Now the `ybbackupctl --list` command (for example) will recognize the new location:

```
$ ybbackupctl --list /home/yb100/premdb_february2020_backups
22:46:46.582 [ INFO] ABOUT CLIENT:
...
22:46:46.583 [ INFO] Running backup control using input locations: [file:///home/yb100/premdb_february2020_backups/]
Database: premdb (e80517b2-a944-b0ba-1213-3c4187b96942)
Chain: February2020
    6  Tables
...
```

You can now run more backups to the new location and restore databases from backups in the new location.

If the command finds that backup files exist in the expected location, their paths are checked but not fixed. For example, you will see messages like this:

```
...
22:41:32.904 [ INFO] Verifying files for [Database=premdb] [Backup chain=February2020] [Snapshot name=Feb20Full] [Backup type=FULL
22:41:32.905 [ INFO] Verifying backup file: [relative path=/78/fc/a3/8b/d2/78fca38b-d2e3-4e86-81cf-2bb10a800f45] [file size=5811]
22:41:32.905 [ INFO] [File=/78/fc/a3/8b/d2/78fca38b-d2e3-4e86-81cf-2bb10a800f45] was found at the expected location and it passes
...
```

ybbackupctl Options

Only one "control command" may be used each time you run the `ybbackupctl` tool. The control commands are as follows:

- `--copy-to`
- `--delete`
- `--fix-bundle-paths`
- `--help`
- `--list`
- `--purge`

If no source location is specified in a `ybbackupctl` command, the location is assumed to be the current working directory. To specify a source location, append the directory path (or paths) to the end of the command. Do not specify an option name for these paths.

See also Storage Locations for Backups.

-c STRING, --chain STRING

The backup chain to use. Default: `default`.

--compatibility-mode BACKWARD | FORWARD | ALL | NONE

Allow/forbid running commands on backup bundles with newer/older bundle versions than the client tools bundle version.

- `BACKWARD` allows operating on backup bundles with bundle versions same as or older than the client tools bundle version.
- `FORWARD` allows operating on backup bundles with bundle versions same as or newer than the client tools bundle version.
- `ALL` allows operating on all backup bundles.
- `NONE` allows operating on backup bundles with the same bundle version as the client tools bundle version.

By default `BACKWARD` is chosen.

-C STRING, --copy-to STRING

Copy a single backup to another location. The `--copy-to` command requires one or more destination (target) storage locations, which should be specified directly after `--copy-to`. The storage you are copying to does not have to exist when you run the `ybbackupctl` command.

You also need to specify the following options:

- `--point` to name the backup
- `--chain` (if not using the `default` chain)
- `--uuid` to identify the database

Specify the input (source) storage location of the backup at the end of the command, or change to the directory where the backup resides. If you do not specify an input location, the current working directory is assumed to be the source of the backup that you want to copy. See ybbackupctl Examples.

Only completed backups can be copied. You cannot copy a backup that is in progress.

Note: When you copy backups with this command, the metadata for the backup files is included. (You do not need to fix the paths with the `-X` option as well.)

-D, --delete

Delete a single backup, as specified with the `--point` argument. This command also requires the `--uuid` option to identify the database. If the backup chain is not named `default`, specify the `--chain` option as well.

The `--delete` option can be used to delete *any backup entry* in a backup chain; in turn, the chain associated with the deleted entry cannot be used to restore a database. (The --purge option is more limited in its scope.)

-X, --fix-bundle-paths STRING

Modify paths to source locations for backup files. This option is useful when backup files have been moved from one storage system to another, or when different clients have different mount point names for the same storage. The path you specify for this option is checked for a valid backup bundle that matches the database, backup chain, and backup name. If the check is successful, the metadata for that backup bundle is updated, making it possible for subsequent backups and restores to run, using the modified path.

--ignore-failed-delete

Ignore failed file deletion in -D, --delete / --purge operations. Throw a warning and continue execution instead of terminating the whole operations.

--integrity-check NONE | FAST | STRICT

Specify integrity checking (checksum validation) of new backup data files that are written when the `--copy-to` command is used. Integrity checking has an impact on performance but also reduces the likelihood that subsequent restore operations will fail because backup files are corrupted.

`NONE` (the default): no integrity checking; no impact on performance.

`FAST`: file-level integrity checking; reduces performance by a small percentage.

`STRICT`: stricter integrity checking, including packet placements inside each file; reduces performance by a greater amount.

-L, --list

List the contents of a backup set. Use the `--verbose` option to return more information about the tables and other objects in the backup. You must specify the location of the backups you want to list.

Only completed backups can be listed. You cannot list backups that are in progress.

--point STRING, -a STRING

Identify the backup you want to manage in one of three ways:

- `latest` or `oldest` backup in the chain
- Backup snapshot name (the value for `--name` when the `ybbackup` command was run). For example: `PremDBFullBackup20190906` or `DB1_Sept_06`
- Backup timestamp in `YYMMDD HH:MM:DD` format. For example: `190906 12:00:00`

--purge FULL | INVALID | INVALID_TABLES_ALL | INVALID_TABLES_AT, -p FULL | INVALID | INVALID_TABLES_ALL | INVALID_TABLES_AT

Purge either obsolete/full backups from the backup set, or all invalid tables. A full backup becomes obsolete when a new full backup is taken, which breaks the backup chain. Backups are invalid if they do not complete successfully (because of cancellation or a network issue, for example). Tables can be INVALID because a transfer failed partway and the corresponding files have not been deleted. You must specify `FULL`, `INVALID`, `INVALID_TABLES_ALL`, `INVALID_TABLES_AT` for the purge command. You also need to specify the following options:

- `--chain` (if not using the `default` chain)
- `--uuid` to identify the database
- `--point` if using `INVALID_TABLES_AT`

See also -D, --delete.

--temp-workdir STRING

Use the specified location as a temporary working directory for backup bundle processing. By default, the current working directory is used. `ybbackupctl` operations that write or read from object storage require a writable local directory during processing. When processing is finished, this directory is empty.

--uuid STRING, -U STRING

Identify the database by specifying its full UUID string. This option is required for `--purge`, `--delete`, and `--copy-to` operations. You can copy and paste the UUID string from the results of the `ybbackupctl --list` command.

--verbose, --no-verbose

Specify --verbose to get more detailed output. The default is `--no-verbose`.

ybrestore Commands

Platforms: All platforms

Parent topic: Backup & Restore

In this section:

Each `ybrestore` command defines:

- Database connection options
- The target database
- The backup chain and directory to restore from
- The restore point, which can be expressed in several different ways
- Other options, such as logging parameters and whether to restore users and roles

You must run the `ybrestore` tool as a superuser.

Only completed backups can be restored. You cannot restore from a backup that is in progress.

Syntax Summary

```
./ybrestore
--chain backup_chain
--point restore_point
[ database_connection_options ]
[ other_options ]
[ storage_locations ]
```

For details about all of the options and their syntax, see ybrestore Options.

To look at your restore history, query the sys.log_restore view.

Ownership of Restored Databases and Permissions on Objects

Users, roles, grants on objects, and ACLs are system-wide objects and attributes; therefore, when you back up and restore a database to a separate system, you may or may not want to re-create them. The backup and restore commands provide an option that helps you maintain the security model that you have put in place for each cluster. For example, you may want to a restore a database to a system that has a different set of users from those on the source system. In this case, re-creating the users and roles that exist on the source system would not be practical.

If you are using LDAP authentication and synchronization, you can also set a restore option that requires the restored database to honor this configuration.

Restoring Users and Roles

When you restore a database, you can restore *all* or *none* of the following security-related objects and artifacts for that database:

- Users (not including superuser privileges)
- Roles and membership in roles
- ACLs
- Ownership of database objects
- Privileges granted to objects

Grants and ACLs on system views (`sys` schema) are not restored.

When making the choice to use `--security-mode all` or `--security-mode none`, keep the following considerations in mind:

- Users and roles are system-wide objects. You do not create them within a single database, and they are often used across databases.
- The target system for the restored database may not have or need the same user and role setup as the source system.
- If you use `--security-mode none`, the restore operation creates a "backup" user on the target system. This user becomes the default owner of the restored database and its objects. The user name is constructed as follows:

```
backup_<source_cluster_uuid>_<userid>
```

For example:

```
backup_eb1469fb-8278-4bdb-b4a0-1ec43735178f_16388
```

The backup user is created with virtually no access to the system and the user's connection limit is set to zero. You can run `ALTER USER` and `GRANT` commands to modify the name, attributes, and privileges of this user.

- When you use `--security-mode all`, superusers on the source system will be re-created as non-superusers on the target system. If necessary, you can run an `ALTER USER` command to give the `SUPERUSER` privilege back to the account.

Running Incremental Restores

This section lays out the steps and requirements for making use of incremental restore operations. Incremental restores reduce the time and resources required to maintain an up-to-date copy of a database that was originally created from a full restore.

Follow these steps:

1. Back up the source database with at least one full backup.
2. Restore the database with the same name on a separate DR cluster or with a different name on the same source cluster. Do not set `--hot-standby off` in the `ybrestore` command.
3. Leave the restored database in `HOT_STANDBY` state, which is the default state for a restore operation. Do not set the restored database to `READONLY` or set `HOT_STANDBY` to `OFF`.
4. Continue backing up the source database on a regular schedule, using cumulative or incremental backups. Keep track of the backup history by using the `ybbackupctl --list` command or running queries on the `sys.log_backup` view.
5. Periodically run incremental restores on the restored database. Make sure you specify the correct settings for the following options:

- `--point`: equivalent to the backup name in the `ybbackup` command
- `--chain`: the backup chain for the source database

The `ybrestore` command checks the restore point and only applies changes as needed.

ybrestore Examples

This section contains some simple examples of restore commands, using different options.

Restore an existing database under a new name

This example restores a database to the same system under a different name. Here is a breakdown of the `ybrestore` command options used in this example:

Command Syntax	Explanation
`ybrestore`	Call the `ybrestore` tool on the client.
`-W`	Prompt for the user's password.
`--username yb100`	Run as user `yb100` (must be a superuser).
`-d premdb_restored`	Restore the database under the name `premdb_restored`. (You cannot use the name of the source database in the command unless it has been dropped from the target system or does not exist on the target system.)
`--point October15Inc3`	Identify the restore point, which may be a valid backup name for the database you want to restore: `October15Inc3` in this case. (You can also specify `latest`, `oldest`, or a timestamp; see ybrestore Options.
`--chain October2019`	Name the backup chain `October2019`.

`/home/brumsby/premdb_daily_backups`	Identify the location of the backup files, as specified when the `ybbackup` command was run. If the backup used multiple directories, make sure you name the first backup directory in the `ybrestore` command. This directory contains the metadata files (`metaDB.*`) for the backup. (You do not need to list the other directories.)An S3 storage path would look like this:
	
	`s3://ybbobr/premdb_bar`
	
	An Azure Blob storage path would look like this:
	
	`azure://premdb_bar`

Tip: You can run the `ybbackupctl --list` command to verify backup information before running a restore command. See ybbackupctl Examples.

Here is the command line and the full output of the restore operation, which uses a sequence of four backups:

```
$ ybrestore -W --username yb100 -d premdb_restored --point October15Inc3 --chain October2019 /home/brumsby/premdb_daily_backups
Password for user yb100:
17:44:56.066 [ INFO] ABOUT CLIENT:
   app.cli_args         = "-W" "--username" "yb100" "-d" "premdb_restored" "--point" "October15Inc3" "--chain" "October2019" "/hom
```

```
   app.name_and_version = "Yellowbrick Backup and Restore version 4.0.0-1385"
   java.home            = "/usr/lib/jvm/java-8-oracle/jre"
   java.version         = "1.8.0_101"
   jvm.memory           = "512.00 MB (max=4.00 GB)"
   jvm.name_and_version = "Java HotSpot(TM) 64-Bit Server VM (build 25.101-b13, mixed mode)"
   jvm.options          = "-Xms512m, -Xmx4g, -XX:+UseG1GC, -Dapp.name=ybrestore2, -Dapp.pid=4240, -Dapp.repo=/opt/ybtools/lib, -Da
   jvm.vendor           = "Oracle Corporation"
   os.name_and_version  = "Linux 4.4.0-31-generic (amd64)"

17:44:56.067 [ INFO] Running restore using input locations: [file:/home/brumsby/premdb_daily_backups/]
17:44:57.407 [ INFO] RestoreSession: Found candidate for restore : October15Inc3
17:44:57.585 [ INFO] Calculated restore sequence:
17:44:57.585 [ INFO]    1) October15Full
17:44:57.586 [ INFO]    2) October15Inc1
17:44:57.586 [ INFO]    3) October15Inc2
17:44:57.586 [ INFO]    4) October15Inc3
17:44:57.947 [ INFO] Starting restore of 'October15Full' [15468]
ybrestore_meta | 100.0% |        Read: 21.00 KiB / 21.00 KiB        |      Write: 21509       |  0.00 B/s   |   2019-10-15 17:44:5

ybrestore_data | 100.0% |       Read: 284.73 KiB / 284.73 KiB       |    Write: 284.59 KiB    | 28.10 KB/s  |   2019-10-15 17:45:1
17:45:14.012 [ INFO] Validating restored objects
17:45:14.012 [ INFO] Committing
17:45:14.025 [ INFO] Completed restore of 'October15Full'

17:45:14.089 [ INFO] Starting restore of 'October15Inc1' [15767]
ybrestore_meta | 100.0% |         Read: 6.56 KiB / 6.56 KiB         |       Write: 6713       |  9.53 KB/s  |   2019-10-15 17:45:1

ybrestore_data | 100.0% |         Read: 7.98 MiB / 7.98 MiB         |     Write: 7.97 MiB     |  1.58 MB/s  |   2019-10-15 17:45:1
17:45:17.991 [ INFO] Validating restored objects
17:45:17.991 [ INFO] Committing
17:45:18.001 [ INFO] Completed restore of 'October15Inc1'

17:45:18.064 [ INFO] Starting restore of 'October15Inc2' [15828]
ybrestore_meta | 100.0% |         Read: 1.95 KiB / 1.95 KiB         |       Write: 2000       |  5.28 KB/s  |   2019-10-15 17:45:1

ybrestore_data | 100.0% |       Read: 225.88 MiB / 225.88 MiB       |    Write: 225.82 MiB    | 32.21 MB/s  |   2019-10-15 17:45:2
17:45:24.575 [ INFO] Validating restored objects
17:45:24.575 [ INFO] Committing
17:45:24.602 [ INFO] Completed restore of 'October15Inc2'

17:45:24.664 [ INFO] Starting restore of 'October15Inc3' [15888]
ybrestore_meta | 100.0% |         Read: 1.94 KiB / 1.94 KiB         |       Write: 1991       |  5.40 KB/s  |   2019-10-15 17:45:2

ybrestore_data | 100.0% |        Read: 34.64 MiB / 34.64 MiB        |    Write: 34.64 MiB     |  8.64 MB/s  |   2019-10-15 17:45:2
17:45:27.665 [ INFO] Validating restored objects
17:45:27.665 [ INFO] Committing
17:45:27.678 [ INFO] Completed restore of 'October15Inc3'

17:45:27.690 [ INFO] Done
```

Restore a database with users, roles, grants, and ACLs

The following command restores a database with `--security-mode all`. Therefore, security-related objects and privileges are preserved.

```
$ ybrestore -W --username yb100 -d premdb_restored --point Dec18Full --chain December2019
--security-mode all /home/brumsby/premdb_daily_backups
Password for user yb100:
...
```

For details, see Restoring Users and Roles.

Attempt an incremental restore that is not needed

This example attempts an incremental restore, but the system detects that the target database is already in sync with the changes in the specified backup point.

```
$ ybrestore -W --username yb100 -d premdb_restored --all-security --point October15Inc3 --chain October2019 /home/brumsby/premdb_d
Password for user yb100:
16:49:11.963 [ INFO] ABOUT CLIENT:
...
16:49:11.965 [ INFO] Running restore using input locations: [file:/home/brumsby/premdb_daily_backups/]
16:49:13.338 [ INFO] RestoreSession: Found candidate for restore : October15Inc3
16:49:13.498 [ INFO] RestoreSession - Last Backup Snapshot = October15Inc3
16:49:13.498 [ INFO] RestoreSession: Nothing to restore as desired snapshot 'October15Inc3' is already applied to YBD
16:49:13.498 [ INFO] System is up to date
```

Restore a database and disable HOT_STANDBY mode

Note the use of the `--hot-standby off` option in this example. When the restore completes, the database is not kept in `HOT_STANDBY` mode. Therefore, it cannot be used for incremental restores.

```
$ ybrestore -W --username yb100 -d premdb_restored --point Oct23Full --chain October2019
/home/brumsby/premdb_daily_backups --hot-standby off
Password for user yb100:
...
17:32:27.836 [ INFO] Running restore using input locations: [file:///home/brumsby/premdb_daily_backups/]
17:32:29.637 [ INFO] RestoreSession: Found candidate for restore : Oct23Full
17:32:29.786 [ INFO] Calculated restore sequence:
17:32:29.787 [ INFO]    1) Oct23Full
17:32:30.123 [ INFO] Starting restore of 'Oct23Full' [10880]
...
17:33:05.370 [ INFO] Completed restore of 'Oct23Full'

17:33:05.371 [ INFO] Disabling hot standby mode
17:33:05.374 [ INFO] Disabled hot standby mode
17:33:05.391 [ INFO] Done
```

Attempt an incremental restore to a database that is not in hot_standby mode

This example attempts to run an incremental restore on a database that has been taken out of `HOT_STANDBY` mode. Therefore, the restore fails.

```
$ ybrestore -W --username yb100 -d premdb_restored --point October16Inc4 --chain October2019 /home/brumsby/premdb_daily_backups
Password for user yb100:
18:45:18.752 [ INFO] ABOUT CLIENT:
...
18:45:18.754 [ INFO] Running restore using input locations: [file:/home/brumsby/premdb_daily_backups/]
18:45:20.198 [ INFO] RestoreSession: Found candidate for restore : October16Inc4
18:45:20.347 [ INFO] RestoreSession - Last Backup Snapshot = October16Inc3
18:45:20.347 [ INFO] Calculated restore sequence:
18:45:20.347 [ INFO]    1) October16Inc4
18:45:20.444 [ERROR] Failed: Error starting restore session  CAUSED BY: Target database premdb_restored isn't a hot standby
```

Restore from object storage

Restore a database from object storage, using the appropriate credentials:

```
% ./ybrestore -d premdb_restored --point latest --chain Feb2023
--object-store-region us-west-2 --object-store-identity ****************** --object-store-credential ************************
--temp-workdir /Users/brumsby/tmp_backups
s3://ybbobr/premdb_bar
15:26:35.864 [ INFO] ABOUT CLIENT:
...
15:26:35.866 [ INFO] Running restore using input locations: [s3://ybbobr/premdb_bar]
...
15:26:37.582 [ INFO] Loaded file handler for s3://
15:26:37.583 [ INFO]
Configuration (S3 client):
   endpoint           : ********
   region             : us-west-2
   identity/credential: ********
15:26:40.867 [ INFO] RestoreSession: Found candidate for restore : Feb17Full
15:26:41.560 [ INFO] Calculated restore sequence:
15:26:41.560 [ INFO]    1) Feb17Full
15:26:42.189 [ INFO] Starting restore of 'Feb17Full' [4295109834]
ybrestore_meta | 100.0% |        Read: 12.65 KiB / 12.65 KiB        |      Write: 12953       |  0.00 B/s   |   2023-02-17 15:26:4

ybrestore_data | 100.0% |         Read: 8.39 MiB / 8.39 MiB         |     Write: 8.39 MiB     | 899.00 B/s  |   2023-02-17 15:27:5
15:27:56.639 [ INFO] Validating restored objects
15:27:56.639 [ INFO] Committing
15:27:56.902 [ INFO] Completed restore of 'Feb17Full'

15:27:57.477 [ INFO] Done
```

Validate restore locations (in object storage)

The following restore fails because `--validate-locations-exist` is specified, but the restore location cannot be found:

```
% ./ybrestore -d premdb_restored_feb21_all --point latest --chain Feb2023
--object-store-region us-west-2 --object-store-identity ***************** --object-store-credential **********************
--security-mode all --temp-workdir /Users/brumsby/tmp_backups --validate-locations-exist s3://ybbobr/premdb_data
12:57:47.379 [ INFO] ABOUT CLIENT:
...
Configuration (S3 client):
   endpoint           : ********
   region             : us-west-2
   identity/credential: ********
12:57:53.512 [ERROR] Failed: --validate-locations-exist is enabled and invalid backup locations found.; Failed to access one or mo
```

Restore to a new location after moving a backup

If you backed up a database to a single location and then moved it to another location, you can restore it from the new location by specifying the `--use-relative-location` option in the `ybrestore` command.

This example demonstrates a case where the original backup was taken as follows:

```
% ./ybbackup -d premdb --full --name apr3full --chain apr2023 /Users/bkadmin/premdb_apr_backups
```

Sometime later, the backup directory was moved:

```
% mv premdb_apr_backups premdb_full_backups
```

The restore operation now looks like this. Note the path to the new location and the `INFO` message that specifies how the new path overrides the original path:

```
% ./ybrestore -d premdb_restored --point latest --chain apr2023 --temp-workdir /Users/bkadmin/tmpbk --security-mode all
--use-relative-location /Users/bkadmin/premdb_full_backups
...
11:46:11.158 [ INFO] Running restore using input locations: [file:///Users/bkadmin/premdb_full_backups/]
...
11:46:13.502 [ INFO] Overridden restore input location from: [file:///Users/bkadmin/premdb_apr_backups/] to: [file:///Users/bkadmi
11:46:13.781 [ INFO] RestoreSession: Found candidate for restore : apr3full
...
11:46:45.153 [ INFO] Completed restore of 'apr3full'
```

ybrestore Options

This section contains detailed descriptions of the `ybrestore` options. Many of these options work the same for `ybbackup` and `ybbackupctl`. See ybbackup Options. See also Common Options in ybtools.

To see all of these options in the online help, run the `ybrestore --help` and `ybrestore --help-advanced` commands.

-c STRING, --chain STRING

Name the backup chain to use for the restore. See ybbackup Options for information about backup chains.

--compatibility-mode BACKWARD | FORWARD | ALL | NONE

Allow/forbid running commands on backup bundles with newer/older bundle versions than the client tools bundle version.

- `BACKWARD` allows operating on backup bundles with bundle versions same as or older than the client tools bundle version.
- `FORWARD` allows operating on backup bundles with bundle versions same as or newer than the client tools bundle version.
- `ALL` allows operating on all backup bundles.
- `NONE` allows operating on backup bundles with the same bundle version as the client tools bundle version.

By default `BACKWARD` is chosen.

-d, --dbname STRING

Name the database to restore. The named database must not already exist on the target system.

-h, --host

The database server host (the target for the restore operation, which may or may not be the same system where the backup was taken). Alternatively, you can set this value with the `YBHOST` environment variable. Defaults to `localhost`.

--hot-standby [ON | OFF]

Put a restored database in `HOT_STANDBY` mode (`ON`) or take it out of `HOT_STANDBY` mode (`OFF`) when the restore operation is complete. The default is `ON`. If multiple restore operations are restored in a sequence, this setting is not applied until the last restore is complete.

See HOT_STANDBY and READONLY Modes.

-a, --point STRING

Specify the restore point in one of three ways:

- `latest` or `oldest` backup in the chain
- Backup snapshot name (the value for `--name` when the `ybbackup` command was run). For example: `PremDBFullBackup20190906` or `DB1_Sept_06`
- Backup timestamp in `YYMMDD HH:MM:DD` format. For example: `190906 12:00:00`

--preserve-ldap

Preserve LDAP settings in the restored database. This means that the restored system will recognize and preserve LDAP authentication and synchronization settings that were configured for the source system. The default is not to preserve these LDAP settings.

--security-mode ALL | NONE

Restore all security-related objects, including users, roles, ACLs, and privileges granted on objects. The default is `NONE` (restore none of these). *Superuser privileges are not restored in either case.*

--temp-workdir STRING

Use the specified location as a temporary working directory for backup bundle processing. By default, the current working directory is used. Backup and restore operations that write or read from object storage require a writable local directory during processing. When processing is finished, this directory is empty.

--use-relative-location STRING

Specify a location that is used to override the base paths for all the backup files that will be restored. This option is useful when you have moved backups from local storage to object storage or vice versa, or from one local file system to another. When you come to restore those files, they can be easily found at the updated location. You do not need to use the `ybbackupctl --fix-bundle-path` option before running the restore (which could be a time-consuming operation).

This option has two limitations:

- It only works if the backup was stored in a single location. All data files are assumed to be in the same folder, relative to the metadata (`metaDB.mv.db`).
- If you want to run another backup on a folder after moving it, you do have to use the `ybbackupctl –fix-bundle-path` option.

--validate-locations-exist, --no-validate-locations-exist

Prevent a restore operation from proceeding if any of the specified backup locations are invalid. By default, this check does not occur (`--no-validate-locations-exist`).

Decrypting Columns in a Restored Database

If you have tables with encrypted columns in a database that you restore to a database on another system, you need to create the encryption key on the target system in order to decrypt the encrypted data in those columns. This section walks through an example of this procedure. Note that if you are restoring a database on the same system where the source database was backed up, you can reference the existing encryption key, and this procedure is not necessary.

Assume that you have followed the steps under Encrypting Sensitive Data. Your source system has an encryption key called `yb100key` with the secret `a1b2c3d4e5f0`. `ALL` privileges on the key are granted to role `yb100`. You have created some tables in the `premdb` database (`public` schema) and some of the table columns are encrypted. Now you load the database and take a full backup, in anticipation of restoring the database on another system.

To restore the database and enable decryption of the encrypted data in the restored tables, follow these steps:

1. Log in to the restored version of the database and set the schema.
2. Create an identical encryption key in the restored database, *using the same schema, the same key name, and the same secret*:

```
premdb=# create key yb100key secret 'a1b2c3d4e5f0';
CREATE KEY
premdb=# select * from sys.key;
 key_id  |   name   | schema_id | owner_id |         creation_time
---------+----------+-----------+----------+-------------------------------
 1712473 | yb100key |      2200 |    16007 | 2021-11-03 16:30:23.206582-07
(1 row)
```

3. Grant `ALL` privileges (or fewer privileges as appropriate) to the users and roles on the target system who need access to encrypted data.

```
premdb=# grant all on key yb100key to yb100;
GRANT
```

4. Also make sure that the same users and roles have `SELECT` privilege on the target tables that contain encrypted data. For example:

```
premdb=# grant select on encrypted_names to yb100;
GRANT
```

5. Log in to the restored database using one of the accounts with those grants.

```
premdb=# \c premdb yb100
Password for user yb100:
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)
You are now connected to database "premdb" as user "yb100".
```

6. Test the user's ability to decrypt columns by running some queries with the `DECRYPT_KS` function.

```
premdb=> select decrypt_ks(enc,yb100key) from encrypted_names;
 decrypt_ks
------------
 Black Cats
```

```
 Swans
 Robins
...
```

HOT_STANDBY and READONLY Modes

A target database for restore operations or replication must be either created in `HOT_STANDBY` mode or set to `HOT_STANDBY` mode. This mode preserves the database in a state that allows it to accept incremental restores but not other writes. If `HOT_STANDBY` mode is turned off, the database no longer accepts incremental restores.

You can prepare an *empty* database to be a target for restore operations by creating it as follows:

```
premdb=# create database premdb_restored hot_standby on;
CREATE DATABASE
```

Provided that a database remains empty, you can alter it to a `HOT_STANDBY` database:

```
premdb=# alter database newpremdb set hot_standby on;
ALTER DATABASE
premdb=# \c newpremdb
You are now connected to database "newpremdb" as user "yb100".
newpremdb=# \d
No relations found.
newpremdb=# create table a(b int);
ERROR:  cannot execute CREATE TABLE on a hot standby database
```

When you restore a database under a new name, and a database with that name does not exist on the target system, the restore automatically creates the new database in `HOT_STANDBY` mode. You can use the `--hot-standby OFF` option in the `ybrestore` command if you want to override the default behavior and take the database out of `HOT_STANDBY` mode when the restore operation is complete.

A `HOT_STANDBY` database is available for subsequent incremental restores and replication cycles but not for other write operations. The database is also available for read operations (it can be queried). When `HOT_STANDBY` mode is turned off (with `ybrestore --hot-standby OFF` or an `ALTER DATABASE` command), the database defaults to the behavior of a regular database. A `HOT_STANDBY` database cannot be dropped:

```
yellowbrick=# drop database premdb_restored;
ERROR:  Database 'premdb_restored' is a hot standby
```

A `READONLY` database is similar to a `HOT_STANDBY` database but blocks restore operations in addition to other writes. A `READONLY` database can be the source for backup and restore operations but not the target. You cannot create a database in `READONLY` mode, but you can alter a database to put it into `READONLY` mode. You should only change a database to this state when you want to lock it down for some reason. A `READONLY` database continues to be useful for read query access but is frozen in time.

After you have put a `HOT_STANDBY` database into `HOT_STANDBY OFF` or `READONLY` state, you cannot alter it back to `HOT_STANDBY` state. For example:

```
premdb=# alter database premdb_restored set hot_standby on;
ERROR:  A hot standby database must be empty when initialized
```

A single database can be in both states at once: `HOT_STANDBY` and `READONLY`, in which case the overall restrictions on the database are greater.

Operations Available in Each Mode

The following table summarizes the attributes of each type of database:

Operation	HOT_STANDBY State	READONLY State	Both States
DML: INSERT, DELETE, UPDATE, TRUNCATE	No	No	No
DDL: CREATE, ALTER, DROP objects)	No	No*	No
Create temporary objects	Yes	Yes	Yes
GRANT, REVOKE on objects	Yes	No	No
GRANT, REVOKE on roles, users	Yes	Yes	Yes
NEXTVAL function and ALTER SEQUENCE commands	No	No	No
Statistics generation	Yes	No	No
Backups	No	Yes	No
Full restores	Yes (initial restore only)	No	No
Incremental restores	Yes	No	No
SELECT queries	Yes	Yes	Yes
Replication reads	No	Yes	No
Replication writes	Yes	No	No

*When the current database is `READONLY`, `CREATE DATABASE` and `DROP DATABASE` commands are allowed.

Monitoring Backup and Restore with SMC

Platforms: EE: All appliance platforms

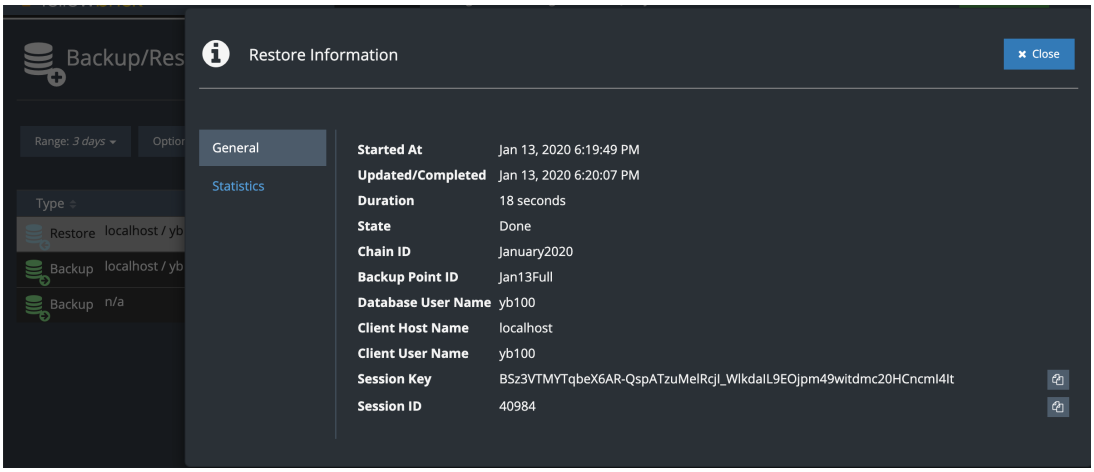
Parent topic: Backup & Restore

You can monitor backup and restore operations by running system view queries or by looking at the reports in the SMC.

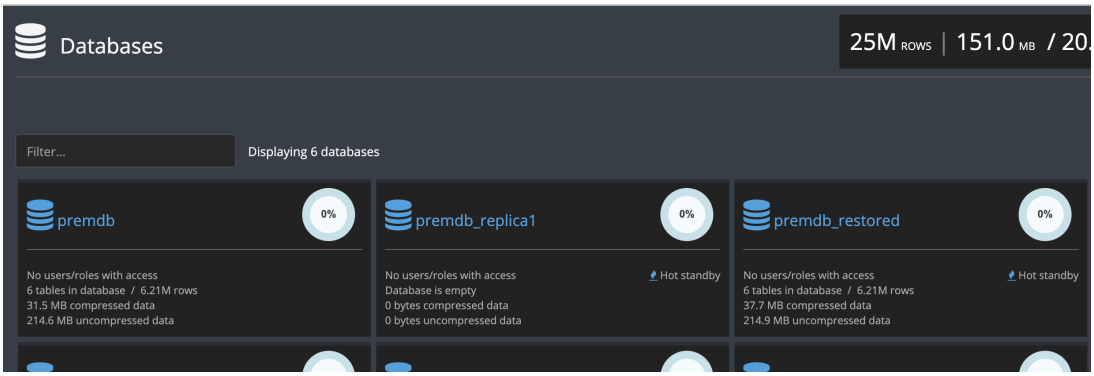
SMC Monitoring

Go to **Monitor** > **Backup/Restore Processes** in the SMC.

You can filter backup and restore events and look at the details for individual operations. For example:



The Databases view in the SMC also shows which databases are in `HOT_STANDBY` mode, either because they were restored or because they are active replicas. For example:



System Views

Write queries against the following system views:

- sys.backup (monitor backups in progress)
- sys.log_backup (view backup history)
- sys.restore (monitor restores in progress)
- sys.log_restore (view restore history)
- sys.backup_chain (list backup chains)
- sys.backup_snapshot (list backup snapshots)

See the view descriptions for examples.

If you are taking backups with `ybbackup` *and* using database replication, the `sys.backup_chain` and `sys.backup_snapshot` views contain information about chains and snapshots that apply to both types of operations.

Database Replication

Platforms: All platforms

Parent topic: Databases

In this section:

Asynchronous database replication is the process of maintaining a replica of a Yellowbrick database on a remote Yellowbrick instance. Replication supports the following use cases:

- Disaster recovery (DR) and high availability (HA): A replica is virtually a live backup, preserving an exact copy of a database on a separate system, ready for use in case of a failure or loss of data on the source system. The primary and secondary (source and target) systems can reverse roles as needed, providing failover and failback support.
- Automated backup and restore: Replication may be used simply as a way to automate backup and restore operations without the need for intermediate storage.
- One-time data migration: replication is a convenient way to move data, such as when you are migrating a database from a small Yellowbrick system to a larger system.
- Adding reporting horsepower: for example, replication supports two instances synchronized with the same data, both of which can provide reporting capability.

Replication occurs at the database level, based on transactions that have been committed to a source database. A database replica is maintained by "backup snapshots" of the source database in the source system. These backups are taken frequently and automatically by a service that runs in the background. The changes are automatically streamed to the database replica in the target system at regular intervals.

The following sections explain how to set up replication, create database replicas, and start the replication of specific databases.

Managing Replication

Platforms: All platforms

Parent topic: Database Replication

In this section:

This section covers some tasks that you may need to perform to make changes to your replication setup.

Dropping a Replica

This section explains how to completely stop replication for a specific replica by dropping it. You may need to run this procedure if you make a mistake with your replication setup and you want to start again, or if you run into a problem that requires troubleshooting.

!Dropping a **reverse** replica leads to the removal of the backup chain, which will cause the entire replication to start from the beginning. There is currently no mechanism for dropping a reverse replica without impacting the forward replica.

1. On the source system, pause replication for a specific replica. You can do this with a SQL command. For example:

```
premdb=# alter database premdb alter replica premdb_replica pause;
PAUSE REPLICA
```

2. On the source system, drop the replica. For example:

```
premdb=# alter database premdb drop replica premdb_replica;
DROP REPLICA
```

3. On the target system, take the target database out of `HOT_STANDBY` mode. For example:

```
yellowbrick=# alter database premdb_replicated set hot_standby off;
WARNING:  This database will no longer accept restore and replication operations
ALTER DATABASE
```

4. On the target system, start using the target database as a regular active database (reads and writes) or drop the database. For example:

```
yellowbrick=# drop database premdb_replicated;
DROP DATABASE
```

Note: Before dropping the database, you may want to take a full backup.

Now you are in a position to set up replication again, using the same source database. Note that the remote server and any other replicas are still intact.

Pausing and Resuming Replication

Platforms: All platforms

Parent topic: Managing Replication

You can use ALTER DATABASE ALTER REPLICA `PAUSE` and `RESUME` commands to stop and restart replication for a specific replica. A replica that has been started can be paused at any time and for any length of time. Pausing a replica stops replication for that specific replica only; replication for other replicas continues.

Important: When replication is paused, the system cancels the in-process replication cycle, if there is one, and rolls back the changes. When replication is resumed, replication restarts, and all of the changes since the last completed replication cycle are sent to the replica database. *The initial replication cycle for a large database may take a very long time. Do not pause the replica during this cycle, or any subsequent cycle that you expect to run for a long time (given the flow of incremental writes).*

The following formula provides a rough estimate of the amount of time required for a replication cycle to complete.

```
Total time (in seconds) = (compressed data size MB) / MIN(3200, Max Network Bandwidth) + (2 * number of tables)
```

This formula calculates the total time as the compressed data size (in megabytes) divided by the maximum network bandwidth. Two seconds of compile time per replicated table are added to the total, but this part of the formula varies much depending on whether the plan has been cached and the width of the tables. Wider tables generate plans that take longer to compile. Concurrency on the source and target systems may also affect the overall replication performance.

Note: The maximum network bandwidth is the maximum throughput of a given system in MB/s (megabytes per second, not Mb/s). The `MIN()` calculation is factored into the formula because bandwidth greater than 3.2 GB/s has not been observed on a 40 Gigabit link.

If much write activity occurs on the source database, a paused replica database may fall far behind the source database. In turn, when you resume replication, it may take a long time for all of the pending updates to be replicated.

You can view the current state of a replica by querying the `sys.replica` view. For example:

```
premdb=# select name, status, alias from sys.replica where alias='premdb_hot';
     name      | status |   alias
---------------+--------+-----------
 premdb_replica | PAUSED | premdb_hot
(1 row)
```

PAUSE

The following command puts a replica in PAUSED state:

```
premdb=# alter database premdb alter replica premdb_replica1 pause;
PAUSE REPLICA
```

Pausing a replica may be useful in the following situations:

- When you know that no updates are occurring on the source database, and you want to suspend replication cycles until they are necessary. Pausing replication will reduce resource consumption on the system.
- When you want to modify the replica before resuming replication.

- When you are going to perform a failover.

RESUME

The following command restarts replication for the specified replica. Replication for that replica returns to the `RUNNING` state, as shown in the results of the following `sys.replica` query:

```
premdb=# alter database premdb alter replica premdb_replica1 resume;
RESUME REPLICA
premdb=# SELECT * from sys.replica where name = 'premdb_replica1';
-[ RECORD 1 ]------------------+------------------
replica_id                     | 16418
database_id                    | 16391
remote_server_id               | 16417
backup_chain_id                | premdb_replica1
name                           | premdb_replica1
last_replication_time          | [NULL]
status                         | RUNNING
frequency                      | 10
alias                          | premdb_replica1_db
...
```

The replica database must be left in `HOT_STANDBY` mode when replication is paused and your intention is to resume at some point. Replication cannot be resumed on a database that is not in `HOT_STANDBY` mode. If you have taken a replica database out of `HOT_STANDBY` mode and you want to resume replication on it, see Rolling Back a Replicated Database.

See Replication Failover and Failback.

Replication Failover and Failback

Platforms: All platforms

Parent topic: Managing Replication

This section contains procedures for failing over and failing back (in the context of using database replication). Yellowbrick recommends that you use the same Yellowbrick database versions on the primary and secondary systems if your replication use case implies a potential need for failover/failback operations.

The example in this section is based on the following configuration of data warehouse instances, databases, and replicas:

Data warehouse instance	Remote Server	Source Database	Replica	Replica Database	Reverse Replica
ybwest (primary)	ybwest_svr	premdb	premdb_replica	premdb_replicated	N/A
ybeast (secondary)	ybeast_svr	premdb_replicated (after failover)	N/A	premdb (after failover)	reverse_premdb_replica

Set Up Replication on the Primary System

Assume that replication has been started in the usual way, replicating data from the primary to the secondary instance. The primary instance is `ybwest` and the secondary is `ybeast`. The `ybwest` instance has a database called `premdb` that replicates data to a database called `premdb_replicated` on `ybeast`.

The following commands were run on the primary instance, `ybwest`:

```
yellowbrick=# create remote server ybwest with (host 'ybeast');
CREATE REMOTE SERVER

yellowbrick=# alter database premdb add replica premdb_replica to ybwest
with (alias premdb_replicated, frequency 60);
ADD REPLICA

yellowbrick=# alter database premdb alter replica premdb_replica start;
START REPLICA
```

Replication is running, and incremental changes to the source `premdb` database are being streamed to the replica database, `premdb_replicated`, on the secondary instance, `ybeast`.

Prepare for a Possible Failover Event on the Secondary System

To be ready in case of a problem with the primary system, you need to create a reverse replica on the secondary system. When the time comes to fail over, you will use this replica to reverse the direction of replication. Follow these steps:

1. Make sure SSL trust is fully configured in both directions between the two systems. See Configuring SSL Trust.
2. Create a remote server on the secondary system. For example:

```
yellowbrick=# create remote server ybeast with(host 'ybwest');
CREATE REMOTE SERVER
```

3. Log in to a database on `ybeast` and create a reverse replica. For example:

```
yellowbrick=# alter database premdb_replicated
add replica reverse_premdb_replica to ybeast
with(alias premdb, reverse_replica premdb_replica, frequency 60);
ADD REPLICA
```

Note that this command refers to the following objects:

- Database being altered: `premdb_replicated`, which is the replica database that was created when replication was first set up (the target for `premdb`).
- New replica being added: `reverse_premdb_replica` (a new and unique replica name)
- Remote server created on the secondary system: `ybeast`
- Database `ALIAS`: `premdb`, which is the name of the original source database
- `REVERSE_REPLICA`: `premdb_replica`, which is the name of the original replica for `premdb`

Note: Allow some time for forward replication to start before trying to create a reverse replica. The best approach is to wait for one replication cycle to complete.

!After creating a reverse replica, it's not possible to drop this replica without restarting replication from the beginning.

Fail Over: Promote the Secondary System

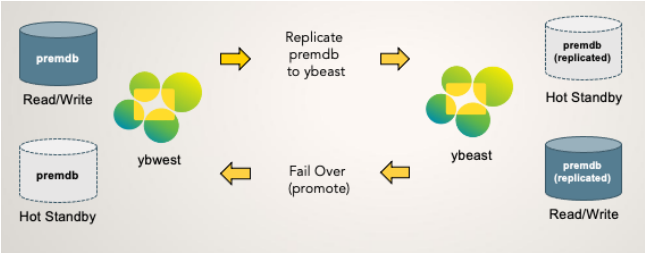
Assume that the `ybwest` instance cannot function as the primary system for some reason, and that replication from `ybwest` to `ybeast` needs to be paused or has stopped running. You can now reverse the roles of the two systems and fail over to the `ybeast` instance on the secondary system.

To fail over, run the following `PROMOTE` command, using the reverse replica you created in preparation for this failover event:

```
yellowbrick=# alter database premdb_replicated alter replica reverse_premdb_replica promote;
PROMOTE REPLICA
```

To summarize, this command takes the `premdb_replicated` database out of `HOT_STANDBY` mode and allows you to start using this database for read and write operations. When the `ybwest` instance comes back online, replication will proceed in the reverse direction so that the `premdb` database receives changes from `premdb_replicated` and can "catch up." For more details about what happens in the background when you run the `PROMOTE` command, see Fail Over With or Without Data Loss.

The `PROMOTE` command starts replication in the reverse direction; you do not have to run a `START` command.



Fail Over With or Without Data Loss

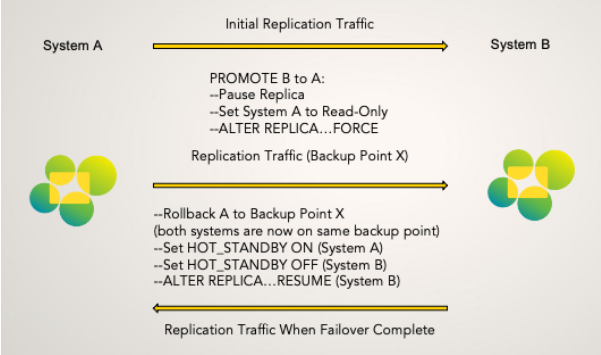
The `PROMOTE` command operates in two modes:

```
ALTER DATABASE name ALTER REPLICA name PROMOTE
or
ALTER DATABASE name ALTER REPLICA name PROMOTE WITH DATA LOSS
```

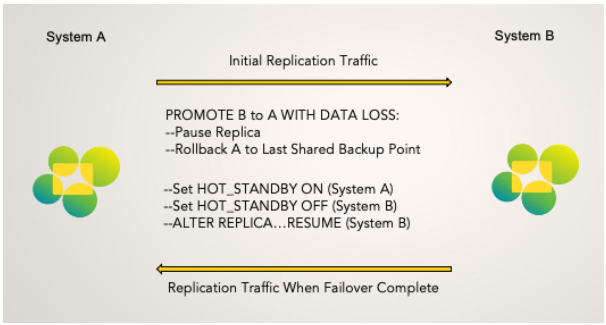
If you use the `PROMOTE` option (without data loss, as described in the previous section), the command tries to force replication from the primary to the secondary and rolls back the source database to a known good rollback point for both databases. If the database on the primary system cannot be reached when the `PROMOTE` command is run, the system returns an error. In this case, you can run the command again, using the `PROMOTE WITH DATA LOSS` option.

When you use the `PROMOTE WITH DATA LOSS` option, the command does not try to force replication from the primary to the secondary. Whatever differences in data existed between the two databases at the time of running the command will exist when replication is reversed. Data written to the primary may be lost because it was not replicated.

The following diagrams illustrate what happens in the background when you run the `PROMOTE` command. The first diagram shows the first case (*without* data loss):



The second diagram shows what happens when `PROMOTE WITH DATA LOSS` is used.



The main difference is that replication is not forced when the secondary system is promoted, meaning that data that was waiting to be replicated may be lost. The primary system is rolled back to the last backup point that the two systems shared.

Consider the following example, which describes a failover scenario where data committed to the primary system has not yet been replicated, and the primary system is not responsive (System A = primary, System B = secondary):

1. System A replicates successfully to B for several cycles.
2. Tables `t1` through `t100` are in sync on both systems. Tables `t1` through `t99` are fully loaded, but `t100` is empty on both systems.
3. During loading of table `t100`, System A becomes unresponsive and requires maintenance. 50 million rows were committed to `t100` before the failure.
4. A failover to System B is initiated. `PROMOTE...WITH DATA LOSS` is run; there can be no attempt to replicate the last set of changes from A to B; 50 million rows in `t100` are effectively lost.
5. A full load of 100 million rows into `t100` is started and completed on B.
6. System A is repaired and brought back online.

7. Replication starts in the reverse direction from B to A. All 100 million rows in `t100` must be streamed to B from A.

Consider the same scenario without data loss: in this case System A is responsive after the failure occurs, so a replication cycle is feasible as part of the failover operation:

1. Replication is forced, and both systems have 50 million rows in `t100` when B is promoted.
2. The remaining 50 million rows are loaded into `t100` on System B.
3. System A is repaired and brought back online.
4. Replication starts in the reverse direction from B to A. 50 million rows in `t100` are replicated.

When You Are Ready, Fail Back

When you are ready, you can revert to the original replication setup, from `ybwest` to `ybeast`. All you need to do is run a `PROMOTE` command that fails back to the primary system.

On the primary system, `ybwest`, run the following command from a database other than `premdb`:

```
yellowbrick=# alter database premdb alter replica premdb_replica promote;
PROMOTE REPLICA
```

READONLY Mode for Source Databases On Restart

A source database for replication typically accepts all read and write transactions. However, if the system goes down or has to be restarted, source databases are temporarily placed in `READONLY` mode. After the restart, there will be a short delay before the source database accepts all transactions and replication cycles can resume. The same behavior occurs after an upgrade.

This behavior protects the two systems in the replication topology while several validation checks are performed. The system verifies that the database has a replica in `RUNNING` state and that the target system can receive replication traffic. When these conditions are met, the source database is taken out of `READONLY` mode and replication can resume.

You can query the sys.database view or use the `ybsql \l` command to check the current state of your databases. The `sys.database` view has a `readonly_reason` column that indicates why the database was placed in `READONLY` mode.

Rolling Back a Replicated Database

Platforms: All platforms

Parent topic: Managing Replication

A rollback is a procedure that reverts a target database to a previous known good snapshot.

The most common use case for rolling back a database is as part of a failover procedure when database replication is in use. A rollback of this kind occurs *automatically* as part of an `ALTER DATABASE...ALTER REPLICA...PROMOTE` command, as explained in Replication Failover and Failback.

You can roll back a target database that is being used for replication manually if you have taken a target database out of `HOT_STANDBY` mode for some reason, such as to run some tests that require writes. After you have run these tests, the target database and source database will be out of sync, and you will not be able to resume replication. Resuming replication in this case requires you to roll back the database to a known good snapshot (a backup snapshot created by the replication process). Rolling back a database also puts the database back into `HOT_STANDBY` mode, which is required before replication can resume.

See also ROLLBACK DATABASE TO SNAPSHOT.

The following procedure contains the steps for rolling back a *replicated database* by using an explicit `ROLLBACK DATABASE` command:

1. On the source system, pause replication:

```
yellowbrick=# alter database premdb alter replica premdb_replica pause;
PAUSE REPLICA
```

2. On the target system, take the target database out of `HOT_STANDBY` mode.

```
yellowbrick=# alter database premdb_replicated set hot_standby off;
WARNING:  This database will no longer accept restore and replication operations
ALTER DATABASE
```

3. On the target system, do some testing on the target database, including some write operations that update or delete from the database. For example:

```
premdb_replicated=# delete from newmatchstats where seasonid=21;
DELETE 34200
```

4. On the source system, determine the rollback point that you need to use in the `ROLLBACK DATABASE` command. Both of the following methods will return the oldest rollback point that will work for the specified replica. *Run these commands from the database that owns the replica.*

```
premdb=# select sys.oldest_rollback_point_id_in_replica('premdb_replica');
 oldest_rollback_point_id_in_replica
-------------------------------------
 premdb_replica_20_02_10_17_56_08
(1 row)
```

```
premdb=# select * from sys.backup_chain;
 database_id |   chain_name   | policy |      oldest_backup_point_id      |     oldest_rollback_point_id     |         created_tim
-------------+----------------+--------+----------------------------------+----------------------------------+--------------------
```

```
       16393 | premdb_replica | {}     | premdb_replica_20_02_10_17_56_08 | premdb_replica_20_02_10_17_56_08 | 2020-02-10 13:31:35
(1 row)
```

5. On the target system, roll back the replica database:

```
premdb_replicated=# rollback database to snapshot 'premdb_replica_20_02_10_17_56_08' hot_standby;
ROLLBACK DATABASE TO SNAPSHOT
```

6. On the source system, resume replication.

```
yellowbrick=# alter database premdb alter replica premdb_replica resume;
RESUME REPLICA
```

7. Verify that the two databases are in sync. For example, in this case they have been rolled back to the point before the `DELETE` command was run:

Target system:

```
premdb_replicated=# select count(*) from premdb_replicated.public.newmatchstats where seasonid=21;
 count
-------
 34200
(1 row)
```

Source system:

```
premdb=# select count(*) from premdb.public.newmatchstats where seasonid=21;
 count
-------
 34200
(1 row)
```

Setting Up Replication

Platforms: All platforms

Parent topic: Database Replication

In this section:

This section explains how to set up replication between two data warehouse instances on separate source and target systems. A system in this context could be either a data warehouse instance in a Yellowbrick cloud deployment or an on-premises appliance. The specific commands that you need to run are listed alphabetically as part of the main SQL Commands section.

To set up replication between two data warehouse instances:

1. Make sure that the instances have network connectivity and that SSL trust is configured. See Configuring SSL Trust.
2. On the source system, create a remote replication server by running the `CREATE REMOTE SERVER` command. This server object must point to a compatible Yellowbrick "host." This host is the target system, which contains target databases that receive replicated data from the source databases on the source system.

For example:

```
premdb=# create remote server yb_repl_svr
with (host 'yb007', sql_port 5432);
CREATE REMOTE SERVER
```

3. Optionally, on the remote system, create an empty "hot-standby" database. The source database will replicate data to the hot-standby database on the remote system.

This step is optional because the `ADD REPLICA` command, later in this procedure, will create the database for you.

For example:

```
yellowbrick=# create database premdb_hot with hot_standby on;
CREATE DATABASE
```

You can also alter an existing empty database to be a hot-standby database by using an `ALTER DATABASE` command.

4. Optionally, seed the hot-standby database by restoring it from a full backup of the source database, using a standard `ybrestore` command. See Seeding a Replica.
5. On the source system, create a "replica" that is associated with the source database. Identify the target database and set the replication frequency, which defines how often the replication cycle will run when it is started. In this example, the target database is named `premdb_hot`, and replication is set to run every 60 seconds.

For example:

```
premdb=# alter database premdb
add replica premdb_replica to yb_repl_svr
with (alias premdb_hot, frequency 60);
ALTER DATABASE
```

Note: If the `ALIAS` database does not exist, the system creates it as a hot-standby on the target system. If you do not include the `ALIAS` option, the system automatically creates a hot-standby target database on the target system with the same name as the source database.

See the ALTER DATABASE ADD REPLICA command for details about other options you can set for the replica.

6. On the source system, start replication by naming the source database and its replica.

For example:

```
premdb=# alter database premdb
alter replica premdb_replica start;
ALTER DATABASE
```

By default, updates to the source database will be replicated to the target at the specified time interval (60 seconds in this example). If the target database was not seeded with a full restore, the first replication cycle may take a long time. After the initial cycle, smaller, faster incremental restores are streamed to the target database.

See ALTER DATABASE ALTER REPLICA for more details about this command.

7. Verify that replication has started running successfully and monitor replication cycles by running queries against the sys.replica_status `and` sys.log_replica_status views.

Configuring SSL Trust

An SSL trust configuration is required to establish a replication channel between two Yellowbrick systems. SSL trust protects information going over the network and ensures that the two systems communicate only with each other (and not an intermediary). The following configuration task is a prerequisite to using the Yellowbrick database replication feature.

When two Yellowbrick systems initialize communication, the connection requires an "SSL handshake" in both directions. This handshake requires a one-time configuration procedure that you complete by using SHOW SSL and IMPORT SSL commands. These commands import and validate SSL certificates that identify the systems in question and authorize communication to proceed.

In an SSL certificate, the common name (CN) is used to identify the host, using either its fully qualified domain name (FQDN) or a wildcard. For example, an FQDN might be `yb007.bbc.jamesbond.com` and a corresponding wildcard would be `*.bbc.jamesbond.com`

In addition to configuring SSL trust, make sure that the CREATE REMOTE SERVER command specifies the host name of the target system correctly; otherwise, replication operations will not be able to proceed.

To configure SSL trust, follow these steps:

1. Establish client (source system) trust of service (target system) in one of two ways:

- Install an SSL certificate, using a well-known Certificate Authority (CA).
- Import the self-signed certificate into the client truststore configuration. The recommended procedure is to install an SSL certificate that is signed by a commercial or custom Certificate Authority (CA). This configuration requires you to contact the appropriate administrators in your organization to obtain an SSL certificate, then change both the HTTPS Yellowbrick Manager listener and PostgreSQL listener to use the certificate for all SSL communications. When these steps are complete, trust is usually established between the client and service.

If the SSL certificate cannot be obtained in the recommended way, follow these steps:

1. **On the remote system**, log in to a database and run the `SHOW SSL SYSTEM` command, which displays the public certificate that the target system has configured for system connections over HTTPS for Yellowbrick Manager and the database (port 5432). Send the results of this command to a file with a `.pem` extension. For example:

```
$ ybsql -d yellowbrick -qAtc "SHOW SSL SYSTEM" > repl_tgt.pem
-----BEGIN CERTIFICATE-----
MIIF2DCCA8CgAwIBAgIJAJ55ViM5MbIpMA0GCSqGSIb3DQEBCwUAMIGCMQswCQYD
VQQGEwJVUzELMAkGA1UECAwCQ0ExHzAdBgNVBAoMFlllbGxvd2JyaWNrIERhdGEs
IEluYy4xLTArBgNVBAsMJDQ1M0M4M0JBLTBCMEMtNDAyMS1BMjcxLTE5NzdDRkJC
MTlGQjEWMBQGA1UEAwwNeWJpbml0X2Rldl9jYTAeFw0yMjAzMTUyMzM5NThaFw0y
NDA2MTcyMzM5NThaMIGLMQswCQYDVQQGEwJVUzELMAkGA1UECAwCQ0ExHzAdBgNV
BAoMFlllbGxvd2JyaWNrIERhdGEsIEluYy4xLTArBgNVBAsMJGI1ODUzNjMyLTFk
ZTgtNDNlYi1hZDUwLWRiYjk2NDU3ZjVhYTEfMB0GA1UEAwwWYnJ1bXNieS55ZWxs
b3dicmljay5pbzCCAiIwDQYJKoZIhvcNAQEBBQADggIPADCCAgoCggIBAOCHqNjj
oNZI73WC4WE4UpO9GlMkhcN+jcPdsAfYiqRTwHGEGEKzumJHFT3/Vbx/pMPx9z5M
BNf7vK7Q26TfjLhQyXeScNYV0uGe/zOgntORVwVGh6xCXBw+q+oww+gld/T/QuE1
ffxVi4XGEBuIL6SU88r7wsCkRUY6u2qJvYBp5SCSsGIrGpNboq51GNb+azak65cB
i+/I3K0WbFn2vhqxJN4NEDzPo5nJ6qJd6Pi0B5kizJV79jdOHyY979bWQ2lVfc+y
z+xUgG3SDrldNH6gexLz/Q7n3poy52C+PRkB3+j/45AE8qozRAhJrbFW09D8DBPD
EyEL0Sh0boBnYWMO4Y0ZWudpSZ1x0tnY4SFiuz2qDT01Pw8H6Tk9nX25Ddh4TL9a
L28KiUnpYznCEGCCTocpY+rSGYkGEMIjutySpzwE3Riqa3hi+NngZfrkLVAaWGh8
TX2CQNlhTFaCf+++hOFiYEtymFietelZ45RVhdT5cxW6kAVkYIWTWp1M0bblE7Ou
1p20WcCLRYA7anE1QSUqH7b+yjUsAtihlZdW3CK6iDornDj4X3njbeoazyy5MbM+
hZ++2pi7HiRDhfMpJ+vGe5qIi2N5tg7UMGoke94QUqphjrws+sd4afGUNx9aSjO/
ZW7DG9J0abNDRP2nMyFn8mzWLzIHO1eVi5CxAgMBAAGjRjBEMAkGA1UdEwQCMAAw
```

```
CwYDVR0PBAQDAgXgMCoGA1UdEQQjMCGCFmJydW1zYnkueWVsbG93YnJpY2suaW+C
B2JydW1zYnkwDQYJKoZIhvcNAQELBQADggIBAB2bpJg+GqDgxUZVx30x6MY9dJjJ
h0ZqDLAAnCscnKl4FMsC7KjJwPliywG3lblrKRCNRoxSIjC0pBv2Ez8iZWADInFz
0pP6Tbl1cWV/lG2dc0FDBPHCAFfUcJkov+lE1fNyn78Cks1+rrNxrwH4nLH2pFOo
yR+s/zy+iveGSUhDIZJm8xw6WLDYZVtgG/6D6qvWIsqADjfVfhBe4OXUC2W6xs0e
Xwd2DIhcCVRPChGKeAEgLzega6zsl10fZjlWRkYvtzdDdr3D6FiHYIIhupKJO5vP
duzzrIvwCOSfVDENTp9x8Q8n9wZEPWfZxaHseplBn4qJhBMIx+UypCjPnH5SkcWS
Gzr/2iBU2ixllQwEsF5m+mLGpVtTDwt8g9rZ44yrVfBDyARWWqhwSkJBiezile7N
q2qd1xsCDYSjcMLDO/kwmEqc5w0P4wvzJXZWes5vN8DXM/enq0QbZYHH7nIGQKbM
4ldKkAo2uzaBb8u0Uf5vYkE8nHoGtdD4XM+jJkSUySfPy8MKrnCZ4IPCidZQrxY6
8ADKd5oMw13Tk6iJtafP0kOBoAfBXXfr9SVPX+pll5+7sww5NIc4Vr3maq8KLmtj
PumO0QZy6oIQWc4P09FzpH159U2O+LHxISGHg+HtcsxDkWQrnSO5o1dXAztsZGne
bFgThLLjWX89ahBe
-----END CERTIFICATE-----
```

Note: After saving the contents of a certificate to a file, you can validate its contents by running an `openssl` command. For example:

```
% openssl x509 -in repl_tgt.pem -text -noout
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 10443218342509856973 (0x90edc3c08124c8cd)
...
```

2. **On the source system**, log in to a database and import the `SYSTEM` configuration from the target system by pasting the results of the SHOW SSL SYSTEM command (or the contents of the `.pem` file) into the IMPORT SSL TRUST command. For example:

```
yellowbrick=# import ssl trust from '-----BEGIN CERTIFICATE-----
yellowbrick'# MIIF2DCCA8CgAwIBAgIJAJ55ViM5MbIpMA0GCSqGSIb3DQEBCwUAMIGCMQswCQYD
yellowbrick'# VQQGEwJVUzELMAkGA1UECAwCQ0ExHzAdBgNVBAoMFlllbGxvd2JyaWNrIERhdGEs
yellowbrick'# IEluYy4xLTArBgNVBAsMJDQ1M0M4M0JBLTBCMEMtNDAyMS1BMjcxLTE5NzdDRkJC
yellowbrick'# MTlGQjEWMBQGA1UEAwwNeWJpbml0X2Rldl9jYTAeFw0yMjAzMTUyMzM5NThaFw0y
yellowbrick'# NDA2MTcyMzM5NThaMIGLMQswCQYDVQQGEwJVUzELMAkGA1UECAwCQ0ExHzAdBgNV
yellowbrick'# BAoMFlllbGxvd2JyaWNrIERhdGEsIEluYy4xLTArBgNVBAsMJGI1ODUzNjMyLTFk
yellowbrick'# ZTgtNDNlYi1hZDUwLWRiYjk2NDU3ZjVhYTEfMB0GA1UEAwwYnJ1bXNieS55ZWxs
yellowbrick'# b3dicmljay5pbzCCAiIwDQYJKoZIhvcNAQEBBQADggIPADCCAgoCggIBAOCHqNjj
yellowbrick'# oNZI73WC4WE4UpO9GlMkhcN+jcPdsAfYiqRTwHGEGEKzumJHFT3/Vbx/pMPx9z5M
yellowbrick'# BNf7vK7Q26TfjLhQyXeScNYV0uGe/zOgntORVwVGh6xCXBw+q+oww+gld/T/QuE1
yellowbrick'# ffxVi4XGEBuIL6SU88r7wsCkRUY6u2qJvYBp5SCSsGIrGpNboq51GNb+azak65cB
yellowbrick'# i+/I3K0WbFn2vhqxJN4NEDzPo5nJ6qJd6Pi0B5kizJV79jdOHyY979bWQ2lVfc+y
yellowbrick'# z+xUgG3SDrldNH6gexLz/Q7n3poy52C+PRkB3+j/45AE8qozRAhJrbFW09D8DBPD
yellowbrick'# EyEL0Sh0boBnYWMO4Y0ZWudpSZ1x0tnY4SFiuz2qDT01Pw8H6Tk9nX25Ddh4TL9a
yellowbrick'# L28KiUnpYznCEGCCTocpY+rSGYkGEMIjutySpzwE3Riqa3hi+NngZfrkLVAaWGh8
yellowbrick'# TX2CQNlhTFaCf+++hOFiYEtymFietelZ45RVhdT5cxW6kAVkYIWTWp1M0bblE7Ou
yellowbrick'# 1p20WcCLRYA7anE1QSUqH7b+yjUsAtihlZdW3CK6iDornDj4X3njbeoazyy5MbM+
yellowbrick'# hZ++2pi7HiRDhfMpJ+vGe5qIi2N5tg7UMGoke94QUqphjrws+sd4afGUNx9aSjO/
yellowbrick'# ZW7DG9J0abNDRP2nMyFn8mzWLzIHO1eVi5CxAgMBAAGjRjBEMAkGA1UdEwQCMAAw
yellowbrick'# CwYDVR0PBAQDAgXgMCoGA1UdEQQjMCGCFmJydW1zYnkueWVsbG93YnJpY2suaW+C
yellowbrick'# B2JydW1zYnkwDQYJKoZIhvcNAQELBQADggIBAB2bpJg+GqDgxUZVx30x6MY9dJjJ
yellowbrick'# h0ZqDLAAnCscnKl4FMsC7KjJwPliywG3lblrKRCNRoxSIjC0pBv2Ez8iZWADInFz
yellowbrick'# 0pP6Tbl1cWV/lG2dc0FDBPHCAFfUcJkov+lE1fNyn78Cks1+rrNxrwH4nLH2pFOo
yellowbrick'# yR+s/zy+iveGSUhDIZJm8xw6WLDYZVtgG/6D6qvWIsqADjfVfhBe4OXUC2W6xs0e
yellowbrick'# Xwd2DIhcCVRPChGKeAEgLzega6zsl10fZjlWRkYvtzdDdr3D6FiHYIIhupKJO5vP
yellowbrick'# duzzrIvwCOSfVDENTp9x8Q8n9wZEPWfZxaHseplBn4qJhBMIx+UypCjPnH5SkcWS
yellowbrick'# Gzr/2iBU2ixllQwEsF5m+mLGpVtTDwt8g9rZ44yrVfBDyARWWqhwSkJBiezile7N
yellowbrick'# q2qd1xsCDYSjcMLDO/kwmEqc5w0P4wvzJXZWes5vN8DXM/enq0QbZYHH7nIGQKbM
yellowbrick'# 4ldKkAo2uzaBb8u0Uf5vYkE8nHoGtdD4XM+jJkSUySfPy8MKrnCZ4IPCidZQrxY6
yellowbrick'# 8ADKd5oMw13Tk6iJtafP0kOBoAfBXXfr9SVPX+pll5+7sww5NIc4Vr3maq8KLmtj
yellowbrick'# PumO0QZy6oIQWc4P09FzpH159U2O+LHxISGHg+HtcsxDkWQrnSO5o1dXAztsZGne
yellowbrick'# bFgThLLjWX89ahBe
yellowbrick'# -----END CERTIFICATE-----';
IMPORT SSL TRUST
```

3. Validate the import by running the SHOW SSL TRUST command. For example:

```
yellowbrick=# show ssl trust;
    hash    |                                                  details                                                  |
------------+-----------------------------------------------------------------------------------------------------------+
 140a8eed.0 | notBefore=Mar 15 23:39:58 2022 GMT                                                                        +|
            | notAfter=Jun 17 23:39:58 2024 GMT                                                                         +|
            | issuer= /C=US/ST=CA/O=Yellowbrick Data, Inc./OU=453C83BA-0B0C-4021-A271-1977CFBB19FB/CN=ybinit_dev_ca    +|
            | subject= /C=US/ST=CA/O=Yellowbrick Data, Inc./OU=b5853632-1de8-43eb-ad50-dbb96457f5aa/CN=brumsby.yellowbrick.io+|
            | serial=9E7956233931B229                                                                                   +|
            |                                                                                                            |
            |                                                                                                            |
            |                                                                                                            |
            |                                                                                                            |
...
```

4. Establish service (target system) trust of client (source system). This step exports the client's identity certificate and imports it into the service's truststore.

For replication purposes, each Yellowbrick system is configured to accept SSL connections from clients that are connected via a trusted client certificate that contains the CN `sys_ybd_system`. This system account is restricted to replication activities, and cannot be used to log in for any other purpose. Each Yellowbrick system is preconfigured with a unique identity (UUID) that is saved in SSL signing certificates as both a custom Certificate Authority (CA) and a signed certificate from the CA. To establish trust from the client to service, you must *import* the public certificate from the CA of the client to the truststore of the service.

1. **On the source system**, log in to a database and export the SSL "CA" configuration, using the SHOW SSL CA command. Send the results of this command to a file with a `.pem` extension. For example:

```
$ ybsql -d yellowbrick -qAtc "SHOW SSL CA" > repl_src_ca.pem
-----BEGIN CERTIFICATE-----
MIIF3jCCA8agAwIBAgIJAJQ3GHVRmUgUMA0GCSqGSIb3DQEBCwUAMH8xCzAJBgNV
BAYTAlVTMQswCQYDVQQIDAJDQTEfMB0GA1UECgwWWWVsbG93YnJpY2sgRGF0YSwg
SW5jLjEtMCsGA1UECwwkZjg2OTc0NjUtM2RmNi00OTZmLWI3NDctZDg5ZTRiZDZh
YjA1MRMwEQYDVQQDDApzeXNfeWJkX2NhMB4XDTIyMDMxNTIzNDEwMVoXDTI0MDYx
NzIzNDEwMVowfzELMAkGA1UEBhMCVVMxCzAJBgNVBAgMAkNBMR8wHQYDVQQKDBZZ
ZWxsb3dicmljayBEYXRhLCBJbmMuMS0wKwYDVQQLDCRmODY5NzQ2NS0zZGY2LTQ5
NmYtYjc0Ny1kODllNGJkNmFiMDUxEzARBgNVBAMMCnN5c195YmRfY2EwggIiMA0G
CSqGSIb3DQEBAQUAA4ICDwAwggIKAoICAQDkJKT20vwwF9wgah6wjpCyAcly9JB1
tsxczP5h0EMGNewOU+J4Du0oPT6CbrpolZWcvd4quVacozKcrYT2cuQWYZP39Gi+
5nx/ks8IRmRs1I+lx5fpXK6HL5u18WHNymNelrpgv24J0Llm4dyVwnQoaMUYJ0nj
H5uNI4hbZMRW/Q9cwhMKURh6m1/fBPycPjnRrt4NKdkMgmNiGQL5zwScm0hYesge
D+Fp9F5JlnK7Cc5vMK/hRAN0Q/VjUbA2d47nd5gI98mbIE2yqgBcOc/te3Baz6EV
apd+zjVB+KQzdTncyQirin/Rlv3YPvE0qiQ2zmgEQRbHfbWCQTXd/RgBIomVLpna
q6psPG5GW7KTwzcG+aHkla+JMiBYqV3DlLyK+ZhO2d1U0HyebmukHgZEjL1cG0iH
wnkA0iEYo443K8zEEPBQ1xVN3Z5ePeEfJK+du3MOpDMSOQIT1nzW4KM5ZWs+DW+I
ROeqXa17JECLDiWDlb1DzgX+8K6FKdiA0wW2e48PGbkriLamIVg+8NFnF+qBUZmQ
f8gtoSawwASaBtZr/nJ2wg7MayxfcLeCIl62ybEYyDHBHmRZbq1+W8+36ktDmJtG
/5HngvB1MSEwcVAfQlRT4G2SemrEheCsrlnkBb18v3T7qLckuXS0pXRVUCvOeebE
dsCEna58lI8b4wIDAQABo10wWzAMBgNVHRMEBTADAQH/MAsGA1UdDwQEAwIB5jAd
BgNVHQ4EFgQUQe6uiCyLW3s6oNemuEf3488F0p4wHwYDVR0jBBgwFoAUQe6uiCyL
W3s6oNemuEf3488F0p4wDQYJKoZIhvcNAQELBQADggIBAAQlpb1aa94KhY1gh+uG
GzSYGstn6KrR60aXSXY+9yjzdViZEo2T/m62g1jk1AjxIjdLlgOXnZ0+TGcX+chU
gaNOQ6FN+mBRRjj9eDsS3+qqPDsgZGZHKbC+U2VpxBahbtz+zJitgqKFhnS5JaIm
clcEGk8gbZJPXUQnCJ1suqSs/r9s0j4EW1KuKFmZE3qnNb0OD8IN02LuqT6U+Wxu
MeaJQjG4BpXAcgneWlSqp/u98SOJZPuaCKAEUceAK/n/eYRDS5BCbZq6i16elm4M
0YYoNwduduNvrVCd6sb4bFdvPf+dRyQaQkjuOERCosuozkQAGhMUkEfqZyVEsP2Y
m3NVQPlttpJa6rndBQWvztXmNij5qAIWJdcmn+/qd+/r6I9b5SS/KrbIhpqxdXS8
3ZoCIhn7hwrzsOx0BFCx9Tz5TpqCv4igtuhjwqe3uIBJI2ha9aKJ3j3Ey9yzZ/4M
L6KzedQZ5DViaOFn3UkSklT021QUklVz7+uDX5UW4FFzzJsfdILG5JAcaJmzWS1N
lcUhx7WyL1wYYUSmiAAJLX2MXBA0tG8i3rhmcLE7ng81/Mnb758PVQZjLDNNl6RA
xhe0QXPTUkeoyZLYqcTOdUOKFRJVuFYE4PoLL5Kaq6zghAE81ireG3sUkVMzkzN+
nRXZBL5f8nMUNrrhapO9KuX2
-----END CERTIFICATE-----
```

Note: This certificate contains a non-existent system CN called `sys_ybd_ca`. This is symbolic and is only used for signing purposes; a database account need not be created for this CN.

2. **On the target system**, log in to a database and run the IMPORT SSL TRUST command to import trust of the client CA certificate. Paste the CA certificate into the command. For example:

```
yellowbrick=# IMPORT SSL TRUST FROM '-----BEGIN CERTIFICATE-----
...
-----END CERTIFICATE-----';
IMPORT SSL TRUST
```

Establishing Trust for New SSL Certificates

Whenever you have to replace the SSL certificates on source and target systems used for replication, you need to re-establish trust between those systems. If you don't do this, replication will no longer work.

Follow these steps to establish trust using the new certificates:

1. Pause replication. (See Pausing and Resuming Replication.)
2. Remove the old trust information from the source and target systems by using the REVOKE SSL TRUST command.
3. Install new SSL certificates.
4. Establish trust for the new certificates by following the procedure in the previous section.
5. Resume replication. (See Pausing and Resuming Replication.)

Monitoring Replication

Platforms: All platforms

Parent topic: Setting Up Replication

You can monitor replication by writing system view queries. You can also monitor replication from a WLM perspective.

System Views

Write queries against the following system views on the source system:

- sys.replica (list database replicas)
- sys.replica_status (show current state of replication activity)
- sys.log_replica_status (show the history of replication activity)
- sys.remote_server (list remote servers available for replication)
- sys.backup_snapshot (list backup snapshots)
- sys.backup_chain (list backup chains)

If you are using database replication *and* taking backups with `ybbackup`, the `sys.backup_chain` and `sys.backup_snapshot` views contain information about chains and snapshots that apply to both types of operations.

Backup and restore operations that are run for the purposes of database replication use the following application name: `replication service`. You can constrain rows in the `sys.query`, `sys.log_query`, `sys.session`, and `sys.log_session` views by using the following condition:

```
where application_name='replication service'
```

See the view descriptions for examples.

WLM Monitoring for Replication Operations

On the source and target systems, look for operations run by the `replication service` application and the `sys_ybd_replicator` system user. For example, on a source system, you will see that a resource pool in the active WLM profile is managing replication tasks that read and send data from a specific backup snapshot to the target system.

Note: For loopback replication, make sure your active resource pool has a maximum/minimum concurrency setting of at least `2/2`.

If you are not sure which resource pool is running replication operations, run a query against the `sys.log_query` view. For example:

```
premdb=# select distinct pool_id from sys.log_query
where application_name='replication service' and database_name='premdb';
     pool_id
--------------------
 replication: large
(1 row)
```

Seeding a Replica

This section explains how to seed a new replica by restoring the target database from a full backup of the source database, then starting replication from that backup point. In some cases, this approach may be preferable to starting replication on an empty database.

When to Seed a Replica

Seeding a replica is a matter of choice. Consider the performance of the link between the two systems. If the initial replication cycle is going to be time-consuming given the bandwidth requirements, it is probably more effective to back up the database to a NAS system, then ship the NAS to the other data center and restore from that backup.

Note the following recommendations.

- Do not seed the replica:
- If both systems are empty or mostly empty.
- When the source system contains a small amount of data and the link between the systems provides good performance.
- Seed the replica when:
- The source system contains a large amount of data but the average amount of change in the data is fairly small.
- The source system has been running backup and restore for some time and you have one or more other clusters available for replication purposes.

If you are going to seed the replica, in some cases you may want to take a new full backup of the source database. Seeding the replica from a new full backup may be more efficient than seeding from a long sequence of backup points that have to be replayed in order.

Procedure for Seeding a Replica

To seed a replica from a backup of the source database, follow these steps:

1. Take a full backup of the source database on the primary system. For example:

```
$ ybbackup -W --username yb100 -d premdb --full --name Feb13Full --chain February2020 /home/yb100/premdb_daily_backups
```

2. Create the target database automatically in `HOT_STANDBY` mode by adding a replica to the source database. For example:

```
premdb=# alter database premdb add replica premdb_replica to yblocal
with (alias premdb_replicated, frequency 60, security_mode 'all');
ADD REPLICA
```

Do not start the replica yet.

3. Restore the source database to the secondary system, using a backup point and chain name that correspond to the values you specified in the `ybbackup` command. For example:

```
$ ybrestore -W --username yb100 -d premdb_replicated --point Feb13Full --chain February2020 /home/yb100/premdb_daily_backups --sec
```

Important: Also note the following requirements:

- The `-d` database name must correspond to the database that will be the target for replication (as specified with the `ALIAS` option in the `ADD REPLICA` command).
- *The security mode for the restore must match the security mode for the replica.* In this example, `--security-mode all` was specified in the `ybrestore` command, which matches the setting specified for the replica. If the security modes do not match, subsequent attempts to replicate will fail with an error.
- Any schemas excluded from the backup chain used to seed the replica must match those excluded from the replica. For example, you cannot exclude `schema1` from the backups used for the restore, then exclude `schema2` when you create the replica.
- Make sure that the restore runs against the secondary system (for example, by setting `YBHOST`).
- Make sure that the secondary system has access to the backup directory. (If the backup files need to be moved, you may need to use the ybbackupctl command with the `--fix-bundle-paths` option.)

4. Get the chain name for the backup you restored from the `sys.backup_chain` view. This chain name must refer to the backup (or sequence of backups) that you restored to the secondary system. *Run this query from the source database on the primary system*. For example:

```
premdb=# select * from sys.backup_chain;
 database_id |  chain_name  | policy | oldest_backup_point_id | oldest_rollback_point_id |         created_time
-------------+--------------+--------+------------------------+--------------------------+-------------------------------
       20309 | February2020 | {}     | Feb13Full              | [NULL]                   | 2020-02-13 18:41:25.927609-08
(1 row)
```

5. Start replication from the source system by using the `START WITH CHAIN` syntax. For example:

```
premdb=# alter database premdb alter replica premdb_replica
start with chain 'February2020';
START REPLICA
```

6. Monitor replication progress by querying the system views. Check that the initial replication cycles only replicate changes made *after* the target database was restored.

Setting Up Loopback Replication

Platforms: All platforms

Parent topic: Setting Up Replication

Loopback replication is the process of replicating to a local database, a database with an alias that resides on the same system as the source database.

Although only one system is being used for replication, you still need to configure SSL trust.

Follow these steps to set up loopback replication:

1. Configure SSL trust. See Configuring SSL Trust.

2. Create a remote server but specify `localhost` in the command. For example:

```
premdb=# create remote server yb100local with (host 'localhost', nohostnamecheck true);
CREATE REMOTE SERVER
```

Query the `sys.remote_server` system view to check the definition of the new remote server:

```
premdb=# select * from sys.remote_server;
 remote_server_id |       name        |          host          | port | control_port | no_hostname_check
------------------+-------------------+------------------------+------+--------------+-------------------
            16419 | yb100local        | localhost              | 5432 |       [NULL] | t
(1 row)
```

If you are using the default self-signed certificate from the cluster or changing the properties of an existing remote server, see CREATE REMOTE SERVER.

3. Create a replica and use the `ALIAS` option to name the target database on the local system. If this database already exists, it must be empty and in `HOT_STANDBY` mode. If it does not exist, the `ADD REPLICA` command creates it. For example:

```
premdb=# alter database premdb add replica local_premdb_replica to yb100local
with(alias premdb_replica1, frequency 60);
ADD REPLICA
```

In this case, the replica database was created and placed in `HOT_STANDBY` mode:

```
premdb=# select name, is_hot_standby from sys.database where name = 'premdb_replica1';
      name       | is_hot_standby
-----------------+---------------
 premdb_replica1 | t
(1 row)
```

See the ALTER DATABASE ADD REPLICA command for details about other options you can set for the replica.

4. Start the local replica:

```
premdb=# alter database premdb alter replica local_premdb_replica start;
START REPLICA
```

Check the `sys.replica` system view to make sure replication is running:

```
premdb=# select name, status, alias from sys.replica;
         name         | status  |      alias
----------------------+---------+-----------------
 local_premdb_replica | RUNNING | premdb_replica1
(1 row)
```

5. Verify that replication has started running successfully and monitor replication cycles by running queries against the sys.replica_status `and` sys.log_replica_status views.

Overview

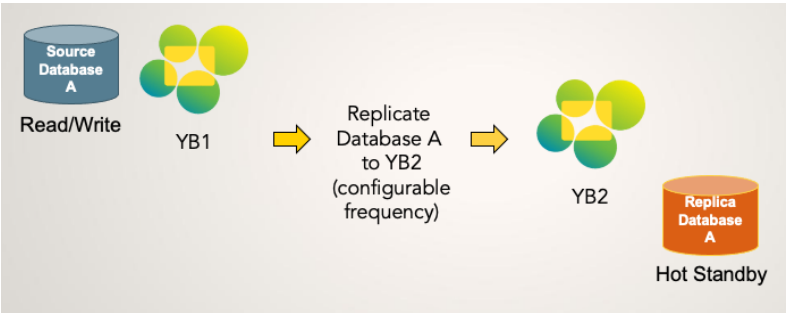
This section introduces concepts, terms, and behavior that you need to be familiar with before you start replicating Yellowbrick databases.

System Requirements

Note the following requirements for setting up and using database replication:

- In most cases, you will replicate from a source system to a physically separate target system. However, you can also set up "loopback replication" into an aliased database on the same system where the source database resides. In this context, a "system" may be an on-premises appliance or a cloud data warehouse instance.
- You can replicate any number of databases from one system to another.
- You can only create and maintain 1 replica per database.
- Network connectivity must be established and maintained between the source and target systems (with SSL trust configured).
- Chained replication is not supported. For example, you can replicate database `A` on instance `YB1` to database `B` on instance `YB2`, but you cannot in turn replicate database `B` on instance `YB2` to database `C` on instance `YB3`.
- Source and target systems need not be running on the same database software and hardware. For example, you can replicate a database from a Version 5.2.x on-premises appliance to a 6.2.x cloud data warehouse instance, or from a Version 6.1.x data warehouse instance to a 6.2.x instance.

For example, in this diagram database `A` on data warehouse instance `YB1` is replicated to database `A` on instance `YB2`. (These instances could be part of the same cloud deployment or could reside in separate deployments.)



What Is Replicated and When

A database replica belongs to a single database. Therefore, a replica maintains a copy of all of the database objects that the source database contains, including tables, views, sequences, and stored procedures. You can either include all schemas or exclude one or more schemas (this feature is also available for regular backup and restore operations).

In addition to database-specific objects, by default replication maintains users, roles, and their privileges and grants. However, superusers are not replicated.

Replication is asynchronous; there is a short delay between a transaction commit in the source database and the moment when changes become visible in the target database. Replication occurs at regular intervals. You can set the interval initially when you create the replica, then modify it later if needed.

Replication Moving Parts

In order for replication to run smoothly, you need to make sure that the two systems are communicating securely, and you have to create some special database objects. This section is a quick summary of the "moving parts" that make replication possible. For details about the setup process, see Setting Up Replication.

SSL trust

A prerequisite for database replication over two systems is SSL trust, which must be established in both directions, based on imported SSL certificates that identify the hosts with fully qualified domain names (FQDNs).

Source and target databases

Replication depends on traffic between two physical databases, a source and a target. The source accepts reads and writes of all kinds and is fully active in that sense.

A target database for replication must be in `HOT_STANDBY` mode, which allows incremental writes via replication but prevents other write operations. Note that a target database is available for reads (queries) and creation of temporary database objects.

Remote servers

A remote server is a logical object that defines the relationship between a source system, where the remote server is defined, and the remote (or target) host that contains target databases for replication. A single remote server can be used to set up multiple replicas.

Replicas

A replica is a logical object that unites a source database, a remote server, and a target database. You can define only one replica per source database. A replica also sets the frequency for replication updates and a few other optional attributes. A *reverse replica* is an object that enables replication in the reverse direction, which is typically needed for failover and failback.

Replication cycles

A replication cycle is an event that occurs at the frequency specified by a replica. Replication can be set to run very frequently (such as every 60 seconds) or infrequently (such as only twice a day). Each cycle goes through a series of states until it is finished and the target system is fully updated. Transactional consistency is maintained between the source and target databases.

Replication and Backup/Restore

Database replication and database backup and restore are closely related operations. Replication makes use of backups and incremental restores to do its work; replication is a kind of automated backup and restore function. The main differences are that replication requires no intermediate storage and provides a means of failing over to a secondary system. Replicated data flows straight from one database to another without landing on disk anywhere. Replication is a good alternative to backup and restore for large systems when the space and cost requirements of NAS systems are prohibitive.

The synchronization of source and target databases depends on snapshots, which identify the state of a database at a given point in time. Each snapshot defines a set of changes that is replicated from the source to the target, providing a rollback capability if the two databases become out of sync.

A new database that is created by replication (or by a restore operation) cannot make use of previous backup chains and requires a new full backup of its own as soon as replication is started. You cannot take incremental backups against an existing backup chain for a replicated database. A new full backup, which implies a new backup chain, must be taken, then incremental backups can be taken against that new chain. (The backups in the original pre-replication backup chain may still be used for restore operations to other databases, but not for incremental backups of the new replicated database.) See also Backup Chains.

Replication Management and Monitoring

When replication is in progress, you can monitor replication cycles by querying system views. For monitoring purposes, information about replication cycles and current progress is visible on the source system, not the target system. Newly replicated data is visible on the target system only when a cycle is complete.

You can use `ALTER DATABASE` commands to modify the state or attributes of a replica:

- Start, pause, or resume replication
- Reverse the order of replication; promote the secondary system to be the primary
- Modify the frequency
- Rename a replica

You can take a target database out of `HOT_STANDBY` mode, but it will no longer accept replication updates; replication cannot be resumed. See HOT_STANDBY and READONLY Modes and Rolling Back a Replicated Database.

Replication Performance Benchmark

This document provides detailed replication benchmarks obtained using Yellowbrick version `7.2.0-69679`. All measurements were performed with two 6-blade Andromeda second generation appliances connected by a 10 Gbps network link. A peak replication rate of 0.981 GB/s was recorded from the primary and secondary for a single replica, which represents 86% of the available network capacity.

Benchmark Setup

Environment

Parameter	Value
Source Appliance	Andromeda (6 blades, v7.2.0-69679)
Destination Appliance	Andromeda (6 blades, v7.2.0-69679)
Network Bandwidth	9.42 Gbps (iperf 30 s, 8 streams)
Memory per Compute Node	512 GB
WLM Profile	Default
Database Encoding	LATIN9
Total Rows	24,576,024,576
Compressed size	630.8 GB
Uncompressed size	2.2 TB
Compression ratio	3.5×
Average row size	27.56 bytes

Table Definition

```sql
CREATE TABLE public.addition_1 (
  worker_id   UUID     NOT NULL,
  worker_lid  SMALLINT NOT NULL,
  node_id     INTEGER  NOT NULL,
  thread_id   SMALLINT NOT NULL,
  range       BIGINT   NOT NULL,
  range_min   BIGINT   NOT NULL,
  range_max   BIGINT   NOT NULL,
  row_number  BIGINT   NOT NULL,
  random_seed BIGINT   NOT NULL,
  random      BIGINT   NOT NULL
)
DISTRIBUTE RANDOM;
```

Query to Populate Table

The following query was used to populate the test table.

```sql
CREATE TABLE public.addition_1 AS SELECT * FROM sys.rowgenerator;

INSERT INTO public.addition_1 SELECT * from public.addition_1;
-- Repeat until the table reaches a size that saturates replication bandwidth,
-- minimizing overhead from startup or I/O latency.
-- In our tests we used ~ 630GB but we could reach network saturation at earlier sizes
-- such as 100GB.
```

Baseline: Replication Without Additional Workload

Each replica was created using the following command:

```sql
ALTER DATABASE "<database_name>" ADD REPLICA "<replica_name>"
TO "<remote_server>"
WITH (
  frequency 99999,
  alias '<replica_db_name>'
);
```

Once created, the replica was started with:

```sql
ALTER DATABASE "<database_name>" ALTER REPLICA "<replica_name>" START;
```

This resulted in the following throughput and timings on the benchmarked system:

Metric	Value
Bytes sent	881,083,886,988
Elapsed time	14 minutes 35.96 seconds (875,959 ms)
Bandwidth	8,046.80 MiB/s (1,005 MB/s)

The single replication workload used 86% of the measured network bandwidth:

Metric	Value
Measured iperf bandwidth	9.42 Gbps
Replication throughput	8.05 Gbps
Utilization of capacity	85.5%

Scalability: Two Concurrent Replication Workloads

Further tests were run with 2 and 4 replicas operating in parallel. The replication process was forced to complete one full cycle in each case.

Replica Count	Bytes Sent	Elapsed Time	Bandwidth (MiB/s)	Bandwidth (MB/s)	Replica Name
1	881,107,588,913	14 min 35.96 sec	8,046.80	1,005	repl_test_bandwidth_a

Replica Count	Bytes Sent	Elapsed Time	Bandwidth (MiB/s)	Bandwidth (MB/s)	Replica Name
2	881,112,026,799	26 min 6.15 sec	4,502.62	562	repl_test_bandwidth_b
2	881,107,588,913	26 min 7.52 sec	4,496.83	562	repl_test_bandwidth_c
4	880,736,450,364	55 min 24.51 sec	2,119.38	264	repl_test_bandwidth_d
4	880,642,224,292	55 min 25.92 sec	2,118.25	264	repl_test_bandwidth_e
4	881,295,115,690	55 min 26.06 sec	2,119.73	264	repl_test_bandwidth_f
4	881,774,587,795	55 min 27.19 sec	2,120.17	265	repl_test_bandwidth_g

Parallel replication tasks were started with the following commands, in the case of 4 replicas:

```sql
-- The following 4 operations are asynchronous
alter database "repl_parallel_1" alter replica "repl_test_bandwidth_d" start;
alter database "repl_parallel_2" alter replica "repl_test_bandwidth_e" start;
alter database "repl_parallel_3" alter replica "repl_test_bandwidth_f" start;
alter database "repl_parallel_4" alter replica "repl_test_bandwidth_g" start;
```

Observations

- Single Replica: Achieved 1,005 MB/s (≈8.05 Gbps), which is 86% of the measured 9.42 Gbps network bandwidth. This indicates good single-stream efficiency and a likely underutilization of the CPUs.
- Two Replicas: Each replica achieved 562 MB/s, summing to 1,124 MB/s (≈ 9.0 Gbps). This is within 4.5% of total capacity, indicating full network utilization, with minor inefficiencies from concurrency overhead, and full CPU utilization.
- Four Replicas: Each replica achieved 264–265 MB/s, totaling 1,056 MB/s (≈ 8.45 Gbps). This shows the network remained saturated, but additional parallelism did not increase total throughput, suggesting a network bottleneck — not a CPU or disk I/O bottleneck.

Delete & Update Operation Performance

Delete and update operations are more expensive than a simple append operation. This is because additional metadata must be sent from the source system to the destination. This can range between 4 bytes (with compression) to ≈130 bytes per deleted / updated row (`rowupdated` , `rowunique` , `rowtxid`).

The performance of the delete operation is highly influenced by how much memory a query executed on the destination system can claim from the applied WLM profile. We recommend allowing at last 25 GB of memory for these types of queries.

Those queries are generally of the form:

```sql
DELETE WITH NO TABLEDELETE FROM public.test_delete
WHERE
    rowtxid != 73370
 AND rowunique % 4 = 1
       AND rowunique IN(
           SELECT rowunique
           FROM public.test_delete
           WHERE rowtxid = 73370 AND rowunique % 4 = 1
               AND rowupdated = 't'
           UNION ALL SELECT rowunique
           FROM public.yb_deletes_17962
           WHERE
               rowtxid = 73370 AND rowunique % 4 = 1
)
```

The compute cluster that the data is being replicated to will partition the query execution into *N* buckets depending on the available memory set by the WLM profile (in the example above, the query was executed in a batch of 4 — `rowunique % 4`).

For example, the following replication cycle included the deletion of all 24,576,024,576 records, and resulted in the following benchmarks:

Metric	Value
sent_bytes	101,214,543,909
elapsed_ms	189,063
elapsed_time	3m 9.06s
bandwidth_mib/s	4,282.79
bandwidth_MB/s	535

In the above table, `sent_bytes` represents the size of the metadata sent from the source to the destination during replication.

The time spent performing the delete operations can be determined by adding up the delete queries:

```sql
yellowbrick=# select sum(total_ms) from sys.log_query where query_text ilike '%yb_deletes_17962%' and type = 'delete';
-----------
 90054.536
(1 row)
```

It follows that the rate of deletion on the replica was ≈ 272 million rows per second:

Metric	Value
Rows Deleted	24,576,024,576
Elapsed time	1 minute 30.05 seconds (90,055 ms)
Rows Deleted/s	272 million

Fixed Overhead Per Table

There is a fixed cost for replicating a single table regardless of the replication operations performed. To demonstrate this, create a database with 1,000 tables each containing a single row, and then start a replication process:

```sql
-- Connected to the DB to be replicated. The simple SQL block simulated 1000 tables
-- that need to be replicated as they contain data.

DO $$
DECLARE
  i INT;
BEGIN
  FOR i IN 1..1000 LOOP
    EXECUTE format('CREATE TABLE public.t%s (x INTEGER);', i);
    EXECUTE format('INSERT INTO public.t%s VALUES (1);', i);
  END LOOP;
END;
$$;

alter database "repl_overhead_per_table" add replica "repl_overhead_per_table_repl" to "remote_server" with (frequency 60, alias r
alter database "repl_overhead_per_table" alter replica "repl_overhead_per_table_repl" start;
```

The results at the end of the single replication cycle were:

Metric	Value
Total bytes sent	243,000
Elapsed time	4 minutes 26.39 seconds
Average overhead/table	≈266 ms

The implication of this ≈266 ms per table replicated overhead is that high table counts with frequent small updates may degrade overall replication performance. This overhead imposes a fundamental limit on how many tables can be replicated over a given period of time:

Time Budget (seconds)	Maximum Number of Replicated Tables
5	18
60	225
300 (5 minutes)	1,126
1,800 (30 minutes)	6,757

The recommendation here is to group operations when possible. Use temporary tables that are automatically excluded from replication for staging operations. Use replication's exclude schema option to exclude entire schemas that are constantly modified that shouldn't be replicated.

Formula to Determine Table Count and Replicated Data Limits

Given the fixed per-table replication overhead and the throughput rates measured, use the following constraint to help plan your replication setup:

Constraint: (maxTables × 0.266) + (maxData / 0.981) ≤ 3600

Where:

- maxTables — number of tables replicated
- maxData — size of data replicated (in GB)
- 0.266 s — fixed cost per table
- 0.981 GB/s — effective data replication rate

This formula leads to the following example constraints:

Time Budget (seconds)	Max Tables	Max Data (GB)
300 (5 min)	1	297.3
300	1,000	35.3
300	500	164.5
600 (10 min)	2,000	70.7
1,800 (30 min)	6,000	885.5
3,600 (1 hour)	5,800	2,000

Replication Glossary

Some of the following terms apply to both replication use cases and regular backup and restore use cases. See also Backup and Restore.

backup chain
: An object that chains together all the snapshots (or backup points) for a specific replica. The default chain name for a replica is the same as the replica name.

DR system
: Disaster recovery system: usually a separate storage system (NAS) or a secondary Yellowbrick cluster. Used as a target system for backup and restore operations or replication.

hot standby database
: A database on a target system that receives replicated data streamed from a database on a source system.

local, primary, or source system
: A Yellowbrick cloud data warehouse instance or an on-premises appliance that contains source databases for replication.

loopback replication
: The process of replicating a database *locally* to a database that is given a different name.

readonly database
: A database that only allows read operations and backups.

remote, secondary, or target system
: A Yellowbrick cloud data warehouse instance that receives replicated data from source databases.

remote server
: An object defined on a source system that defines the target system for replication.

replica
: An object that defines the relationship between a source database, a target database, and a target system.

replica database
: The target database for a specific replica.

reverse replica
: A replica created during failover/failback to reverse replication from the secondary to the primary system.

snapshot
: A logical object that captures the state of a database at a specific point in time. Also known as a backup snapshot or backup point.

source database
: A physical database that replicates its data to a target database.

target database
: A physical database that receives data from a source database via replication.

Replication SQL Commands

Platforms: All platforms

Parent topic: Database Replication

This section contains a summary of the SQL commands that pertain to replication tasks. Full syntax descriptions of these commands are in the main SQL Commands section.

CREATE, ALTER, DROP REMOTE SERVER

The following commands create, alter, and drop remote servers. To set up replication, you must create a remote server object in the source system that identifies the host name of the target system.

```
CREATE [OR REPLACE] REMOTE SERVER name
WITH (HOST remote_address, SQL_PORT remote_port)

ALTER REMOTE SERVER name
WITH (HOST remote_address, SQL_PORT remote_port)

DROP REMOTE SERVER [IF EXISTS] name
```

For more details, see:

- CREATE REMOTE SERVER
- ALTER REMOTE SERVER
- DROP REMOTE SERVER

ALTER DATABASE ADD REPLICA

Use the following command to create a new replica object, which connects a source database to a remote server and database.

```
ALTER DATABASE local_database_name
ADD REPLICA replica_name TO remote_server
WITH (ALIAS remote_database_name, FREQUENCY seconds)
```

For more details, see ALTER DATABASE ADD REPLICA.

ALTER DATABASE DROP REPLICA

Use the following command to drop a replica object that is no longer in use for replication.

```
ALTER DATABASE database_name DROP REPLICA replica_name
```

For more details, see ALTER DATABASE DROP REPLICA.

ALTER DATABASE ALTER REPLICA

Use the following command to alter an existing replica:

```
ALTER DATABASE local_database_name
ALTER REPLICA replica_name
START [ { NEW | EXISTING } chain ] |
PAUSE |
RESUME |
FORCE |
PROMOTE [ WITH DATA LOSS ]
WITH (FREQUENCY seconds)
```

For more details, see ALTER DATABASE ALTER REPLICA.

CREATE DATABASE

This command creates an empty database in `HOT_STANDBY` mode so it can be used as the target database for replication.

```
CREATE DATABASE name WITH (HOT_STANDBY ON)
```

For more details, see CREATE DATABASE.

ALTER DATABASE SET HOT_STANDBY OFF

Use the following command if you need to take a replica database out of `HOT_STANDBY` mode:

```
ALTER DATABASE premdb_replica1_db SET HOT_STANDBY OFF;
```

ROLLBACK DATABASE Command

The following command is used as part of a failover procedure to roll back a database to a previous snapshot. You may also need to run it manually in some situations.

```
ROLLBACK DATABASE TO SNAPSHOT 'rollback_point' HOT_STANDBY
```

SSL Trust Commands

The following commands support SSL trust configuration for clusters used to replicate databases:

- IMPORT SSL TRUST
- SHOW SSL TRUST
- SHOW SSL SYSTEM
- SHOW SSL CA
- SHOW SSL IDENTITY
- REVOKE SSL TRUST

Encrypting Sensitive Data

Yellowbrick Documentation > How-to Guides > Administer Yellowbrick > Databases > Encrypting Sensitive Data

Platforms: All platforms

Parent topic: Databases

In this section:

- Deprecated Encryption Functions
- Encryption Key Rotation
- SQL Encryption Examples

You can use the `ENCRYPT_KS` and `DECRYPT_KS` SQL functions to protect access to sensitive character data, such as Personally Identifiable Information (PII). Encryption functions scramble the input data they receive, rendering it unreadable to any user or role who does not have both the privileges and the access key required to decrypt it. Authorized users can run the encryption functions against any valid SQL expression that evaluates to a character string.

You can also create tables with encrypted columns; in this case, those columns are encrypted automatically when data is loaded or inserted. You do not need to call the `ENCRYPT_KS` function explicitly.

The Yellowbrick implementation of the `ENCRYPT_KS` and `DECRYPT_KS` functions contains code under the OpenSSL license.

Secrets Manager Configuration

The `ENCRYPT_KS` and `DECRYPT_KS` functions are backed by an integrated secrets manager (or key store). The key store functions differently on appliance and cloud platforms.

Appliance platforms

Before you can use the SQL encryption functions, you have to unlock the keystore file by running the following `ybcli` command on the manager node:

```
YBCLI(12545) (PRIMARY - yb100-mgr0)> config keystore sql enable

Are you sure you want to configure keystore for SQL integration?
Response (yes/no): yes
...
```

This command sets up access to the vault. The system looks up keys in the vault when they are specified as arguments to encryption functions in SQL queries.

Other platforms

On other platforms, which use Kubernetes, the database will use Kubernetes secrets and no configuration should be necessary. Check that the secrets manager is correctly configured per the below instructions.

Check the configuration

The `ybd_secrets_manager` parameter defines access to the secrets manager. For example:

```
premdb=> show ybd_secrets_manager;
 ybd_secrets_manager
---------------------
 k8ss
(1 row)
```

SQL Configuration

1. Grant `CREATE` privilege per schema to users and roles who will be able to create, delete, and describe keys created in that schema. (This is the same type of privilege that you would grant to users who are going to create tables and other objects.) For example:

```
premdb=# grant create on schema public to yb100;
GRANT
```

2. Create keys with the `CREATE KEY` SQL command, using a valid hexadecimal string to define a "secret." Keys belong to the database and schema where you create them. For example:

```
premdb=# CREATE KEY yb100key SECRET 'a1b2c3d4e5f0';
CREATE KEY
```

This command creates a key for the public schema in the `premdb` database. The secret for the key is saved in the encryption keystore. (Authorized users pass in the key name, not its associated secret, when they run encryption functions.) Administrators can query the `sys.key` view to return a list of keys that have been created per database and schema. For example:

```
premdb=# select * from sys.key;
 key_id |   name   | schema_id | owner_id |         creation_time
--------+----------+-----------+----------+-------------------------------
  16480 | yb100key |      2200 |    16007 | 2019-11-07 15:26:30.577896-08
(1 row)
```

3. Grant `ENCRYPT` and/or `DECRYPT` access on keys to specific users and roles. For example:

```
premdb=# grant all on key yb100key to yb100;
GRANT
```

This example grants `ENCRYPT` and `DECRYPT` privileges to members of the role named `yb100`.

4. Test the functions by encrypting and then decrypting a column from a table:

```
premdb=# create table encrypted_names as select encrypt_ks(nickname, yb100key) enc from team;
SELECT 50
premdb=# select decrypt_ks(enc,yb100key) from encrypted_names;
 decrypt_ks
------------
 Gunners
 Villains
 Tykes
 Blues
 Rovers
...
```

For more details about the commands used in these steps, see:

- GRANT
- CREATE KEY, DROP KEY, DESCRIBE KEY
- sys.key

Calling the Functions

The `ENCRYPT_KS` and `DECRYPT_KS` functions take the following arguments:

- An input expression (required), which must evaluate to a character string. For example, this could be a column name, the concatenation of values from two columns, or the result of another function.
- The name of the key (required). Users specify keys to ensure that the data is encrypted and decrypted consistently every time the same source data is accessed. Keys are mapped to secrets stored in the vault.
- One of three standard AES algorithms, all of which use Output Feedback Mode (OFB):
- AES with a 128-bit key (the default)
- AES with a 192-bit key
- AES with a 256-bit key
- An optional "initialization vector" provides an additional level of obfuscation. This vector provides an efficient way to re-scramble the encrypted data without having to rebuild the overall character mapping that is generated when a new key is used. Instead of using a different key for each function call, you can use the same key and adjust the vector. Taken together, the key and vector parameters function like a login credential with two levels of privacy and security.

Using `ybunload`, you can unload data with a query that contains the `ENCRYPT_KS` function. You can reload the data as is with `ybload`, then decrypt it with a query or `CTAS` statement that uses the `DECRYPT_KS` function. (You cannot load and decrypt data in one step with `ybload`.)

Creating Tables with Encrypted Columns

When you create a table with encrypted columns, data inserted or loaded into those columns is automatically encrypted using the `ENCRYPT_KS` function. In turn, encrypted data is automatically decrypted when it is selected by users who have `DECRYPT` privilege on the key (and `SELECT` privilege on the table). Users writing queries against encrypted columns do not have to specify the `DECRYPT_KS` function. Users who do not have `DECRYPT` privilege on the key see the stored encrypted values.

The CREATE TABLE statement supports the `ENCRYPTED` column constraint for `VARCHAR` columns only. For example, the following table is defined with three encrypted `VARCHAR` columns: `weekly_wages`, `dob` (date of birth), and `cob` (country of birth):

```
premdb=# create table player(
playerid bigint not null,
teamid smallint not null,
seasonid smallint not null,
firstname varchar(30),
lastname varchar(30),
position char(1),
dob varchar(100) encrypted with(COLUMN_ENCRYPTION_KEY=playerkey,ENCRYPTION_TYPE=RANDOMIZED,ALGORITHM='AES_256_OFB'),
weekly_wages varchar(100) encrypted with(COLUMN_ENCRYPTION_KEY=playerkey,ENCRYPTION_TYPE=RANDOMIZED,ALGORITHM='AES_256_OFB'),
avg_mins_per_match double precision,
matches_played real,
cob varchar(100) encrypted with(COLUMN_ENCRYPTION_KEY=playerkey,ENCRYPTION_TYPE=RANDOMIZED,ALGORITHM='AES_256_OFB'));
CREATE TABLE
premdb=# \d player
                                                                    Table "public.player"
      Column       |          Type          |                                                                  Modifiers
-------------------+------------------------+-------------------------------------------------------------------------------------------
 playerid          | bigint                 | not null
 teamid            | smallint               | not null
 seasonid          | smallint               | not null
 firstname         | character varying(30)  |
 lastname          | character varying(30)  |
 position          | character(1)           |
 dob               | character varying(100) | encrypted with (column_encryption_key = premdb.public.playerkey, encryption_type =
 weekly_wages      | character varying(100) | encrypted with (column_encryption_key = premdb.public.playerkey, encryption_type =
 avg_mins_per_match | double precision       |
 matches_played    | real                   |
 cob               | character varying(100) | encrypted with (column_encryption_key = premdb.public.playerkey, encryption_type =
```

```
Distribution: Hash (playerid)
```

This table is then loaded from a CSV file with rows like these:

```
15,23,27,Virgil,Van Dijk,D,07-08-1991,180000,87.6345,37.4,Netherlands
16,24,27,David,Silva,M,01-08-1986,250000,79.6723,30.5,Spain
17,22,27,Jamie,Vardy,F,01-11-1987,80000,77.1986,31.6,England
...
```

User `bobr` belongs to a role with `SELECT` privilege on the table, but this role does not have `DECRYPT` privilege on the key. Therefore, user `bobr` sees rows like these:

```
premdb=> select * from player where playerid between 15 and 17;
-[ RECORD 1 ]------+------------------------------------------------------------
playerid           | 15
teamid             | 23
seasonid           | 27
firstname          | Virgil
lastname           | Van Dijk
position           | D
dob                | CVdbnbTmmeYaaA4SZfOV590jszv8Ooj09zGHsIlB7Q9WhjS2ywNMrg9JVA==
weekly_wages       | CYJzBAQSvlufeMDyWtH2Yr5TA7Ao+ulFpP3eY2SwBSVpNb/DlQjn
avg_mins_per_match | 87.6345
matches_played     | 37.4
cob                | CYz7BPoHAjpNJF0E+O8K/gU8yhRj7oFgcCKvb37lG/LlSep/9M2UbFHHECo=
-[ RECORD 2 ]------+------------------------------------------------------------
playerid           | 16
teamid             | 24
seasonid           | 27
firstname          | David
lastname           | Silva
position           | M
dob                | CUdWko/n2n6SbaaVb39Ck6RYwPH9+intp+oP0p7Uq5M+mMIPQa6iV+wMzA==
weekly_wages       | CXWWk/rZxpNJM5LtBfW32nx+eoQ/0X9NexX236RNAC3N4533++rD
avg_mins_per_match | 79.6723
matches_played     | 30.5
cob                | CXjnwR9g7+iFgQi8IfBsZRYpmj4+sgkmlvSHrZFdCnZmfG+q4YI=
-[ RECORD 3 ]------+------------------------------------------------------------
playerid           | 17
teamid             | 22
seasonid           | 27
firstname          | Jamie
lastname           | Vardy
position           | F
dob                | CeVFkbgdYq2ZOjW7s0E/RQSWlN8W+TsvwYVu8W5EDDPRuRFJS5Mwu7/QyQ==
weekly_wages       | CRMmOw16X8foeWOsq0N+x7kTNXTTFsHo5aXXeH6yzhwmeWu1XMI=
avg_mins_per_match | 77.1986
matches_played     | 31.6
cob                | Cf9r4Q60m/5FtYWSa8jwu3164z0SoL8UDxuz0p3FPQaRQmTjU7oL7w==
```

If user `bobr` is granted `DECRYPT` privilege on the key (either directly or via a role in which he has membership), all of the column values are visible in plain text:

```
premdb=# grant decrypt on key playerkey to bobr;
GRANT
premdb=# \c premdb bobr
You are now connected to database "premdb" as user "bobr".
premdb=> select * from player where playerid between 15 and 17;
 playerid | teamid | seasonid | firstname | lastname | position |    dob     | weekly_wages | avg_mins_per_match | matches_played
----------+--------+----------+-----------+----------+----------+------------+--------------+--------------------+---------------
       15 |     23 |       27 | Virgil    | Van Dijk | D        | 07-08-1991 | 180000       |            87.6345 |          37.4 |
       16 |     24 |       27 | David     | Silva    | M        | 01-08-1986 | 250000       |            79.6723 |          30.5 |
```

```
        17 |     22 |       27 | Jamie     | Vardy    | F        | 01-11-1987 | 80000       |           77.1986 |           31.6 |
(3 rows)
```

For more details about creating tables with encrypted columns, see CREATE TABLE. For details about the specific algorithms used to encrypt data, see Encryption and Decryption Algorithms.

Deprecated Encryption Functions

The `ENCRYPT` and `DECRYPT` functions are deprecated. Yellowbrick recommends the use of `ENCRYPT_KS` and `DECRYPT_KS`; see Encrypting Sensitive Data.

Note: The deprecated encryption functions are not backed by a keystore on the manager node. You must take steps to secure the raw values that are used for keys and vectors. When the `ENCRYPT` and `DECRYPT` functions are executed, the syntax you use is logged to `~/ybd/postgresql/build/db/log/pg.log`. Key values may also be compromised when ODBC or JDBC applications are used to pass these functions to the database or when log files are sent back to Yellowbrick for investigation by Customer Support.

Here is a simple example of the `ENCRYPT` function. The `nickname` column in the `team` table is encrypted in the results of this query. The MD5 function is used to hash the key value as a hexadecimal string:

```
premdb=# select teamid, name,
encrypt(nickname,md5('We are such stuff as dreams are made on')),
city, stadium, capacity
from team
where city in('Birmingham','London')
order by city;
 teamid |         name         |    encrypt     |    city    |     stadium      | capacity
--------+----------------------+----------------+------------+------------------+----------
      2 | Aston Villa          | RuCsrql{7K2    | Birmingham | Villa Park       |    42785
      4 | Birmingham City      | FiSmyyE        | Birmingham | St. Andrew's     |    30016
     43 | West Bromwich Albion | FOyq{KlzQ0     | Birmingham | The Hawthorns    |    27000
      1 | Arsenal              | A8jstaVuQ0     | London     | Emirates Stadium |    60260
     12 | Charlton Athletic    | CCCqmyF{Q0     | London     | The Valley       |    27111
     13 | Chelsea              | T8isgKF|7CpX21 | London     | Stamford Bridge  |    41663
     15 | Crystal Palace       | 8OyqraFu       | London     | Selhurst Park    |    26255
     18 | Fulham               | EWCmjqFzCGYX   | London     | Craven Cottage   |    25700
     32 | Queens Park Rangers  | 5WysfyE        | London     | Loftus Road      |    18439
     41 | Tottenham Hotspur    | USTmhyE        | London     | White Hart Lane  |    36284
     44 | West Ham United      | 5OSsqaVuQ0     | London     | Upton Park       |    35016
     46 | Wimbledon            | 9Wisg0         | London     | Selhurst Park    |    26255
(12 rows)
```

Only two parameters are supplied for the function in this example: the input column name (`nickname`) and the key, which must be a hexadecimal string (in this case, a user-defined character string with `MD5` applied to it). The default algorithm is used, and no initialization vector is specified. Every time this query is run with these exact parameters, the `nickname` values are encrypted with the same results. If these query results were stored in a table, the encrypted nickname values could be returned by using the `DECRYPT` function with the same parameters. For example:

```
select decrypt(encrypted_nickname_column,md5('We are such stuff as dreams are made on')) as nickname
from encrypted_team_table
...
```

Note: This example presents the actual key value in plain text for demonstration purposes. In a production system, you would need to "hide" the key itself. For example, you could select a key value that is stored in another table, where the stored value is already a hash of the actual key, not the original character string. For example, you might have a table called `userkeys`, which stores database usernames and keys (hexadecimal strings):

```
premdb=> select * from userkeys;
 whoami |               key
--------+----------------------------------
 bobr   | 4765123b652644f84007485d110fa29f
(1 row)
...
premdb=> select encrypt(nickname,(select key from userkeys where whoami=current_user)) from team order by teamid;
      encrypt
--------------------
 CyftgWky|C2f10
 V8zzguku5qYmwD8
 T8jpjKkwi0
 BuyvqKkwi0
 R0VtqKkviC2
...
```

Encryption Key Rotation

Platforms: All platforms

Parent topic: Encrypting Sensitive Data

Key rotation is recommended for security. This procedure shows how to encrypt data with an initial version of a key, then create a new key, encrypt all of the data with the new key, and drop the old key. This cycle of steps is a simple solution for rotating keys periodically.

For example, create a `staff` table that contains employee ID and social security number (SSN) columns. Assume that the SSN values need to be encrypted and protected from decryption by some users. In this case, both the ID values and the SSN values are unique. If your encrypted table does not have a unique ID, you can generate one by using a sequence.

1. Create the `staff` table:

```
premdb=# create table staff(id bigint, ssn varchar(100));
CREATE TABLE
```

2. Insert some rows:

```
premdb=# insert into staff values(91234,'111-01-0987');
INSERT 0 1
premdb=# insert into staff values(85673, '222-99-4321');
INSERT 0 1
premdb=# select * from staff;
  id   |     ssn
-------+-------------
 91234 | 111-01-0987
 85673 | 222-99-4321
(2 rows)
```

3. Create a key:

```
premdb=# create key ssnkey0 secret '0f5689f32';
CREATE KEY
```

4. Create `staff_ks`, an encrypted version of the `staff` table:

```
premdb=# create table staff_ks(id bigint, ssn varchar(100));
CREATE TABLE
```

Note: Make sure the column that will hold encrypted values is wide enough. When a social security number, for example, is encrypted, it will require a much wider column.

5. Insert rows into the encrypted table,

```
premdb=# insert into staff_ks(ssn)
select concat('v0', '-', encrypt_ks(ssn, ssnkey0)) from staff;
INSERT 0 2
premdb=# select * from staff_ks;
-------+--------------------
 91234 | v0-w70ijTHGijjZrh0
```

```
 85673 | v0-vBmijvHIizDX|p0
(2 rows)
```

Use a prefix of the form `xy-` for the encrypted strings. In this example, `v0-` is the prefix. The hyphen character is never used in encrypted strings, so a prefix that ends with `-` is a good choice.

6. Decrypt the encrypted values to verify that encryption and decryption are working.

```
premdb=# select decrypt_ks(substr(ssn,4),ssnkey0) from staff_ks;
 decrypt_ks
-------------
 111-01-0987
 222-99-4321
(2 rows)
```

7. Add some new rows to the `staff` table.

```
premdb=# truncate table staff;
TRUNCATE TABLE
premdb=# insert into staff values(13579, '451-90-3421');
INSERT 0 1
premdb=# insert into staff values(97684, '455-66-2356');
INSERT 0 1
premdb=# select * from staff;
  id   |     ssn
-------+-------------
 13579 | 451-90-3421
 97684 | 455-66-2356
(2 rows)
```

8. Create a new key:

```
premdb=# create key ssnkey1 secret '41f55e912f';
CREATE KEY
premdb=# select * from sys.key;
 key_id |   name   | schema_id | owner_id |         creation_time
--------+----------+-----------+----------+-------------------------------
  16450 | ssnkey0  |      2200 |    16444 | 2019-11-12 11:52:51.871322-08
  16516 | ssnkey1  |      2200 |    16444 | 2019-11-13 13:28:03.204117-08
(4 rows)
```

9. Update the encrypted table, using the new key, `ssnkey1`, for the new rows.

```
premdb=# insert into staff_ks(id, ssn)
select id, concat('v1', '-', encrypt_ks(ssn, ssnkey1)) from staff;
INSERT 0 2
premdb=# select * from staff_ks;
  id   |        ssn
-------+--------------------
 13579 | v1-hHUpDsQqa1yDwA3
 97684 | v1-hHUoDAwra5CCzM3
 91234 | v0-w70ijTHGijjZrh0
 85673 | v0-vBmijvHIizDX|p0
(4 rows)
```

10. Decrypt the rows with a `CASE` expression that applies the appropriate key to the `v0` and `v1` rows.

```
premdb=# select id, case left(ssn,3)
when 'v0-' then decrypt_ks(substr(ssn,4), ssnkey0)
when 'v1-' then decrypt_ks(substr(ssn,4), ssnkey1)
else 'missing key'
end
from staff_ks;
  id   |    case
-------+-------------
 13579 | 451-90-3421
 97684 | 455-66-2356
 91234 | 111-01-0987
 85673 | 222-99-4321
(4 rows)
```

11. Update the `staff_ks` table so that the original old key, `ssnkey0`, is no longer needed. The first update encrypts the `v0-` values with the new key and removes the `v0` prefix. The second statement removes the `v1-` prefix from the rows that are already encrypted with the new key.

```
premdb=# update staff_ks
set ssn=encrypt_ks(decrypt_ks(substr(ssn,4),ssnkey0),ssnkey1)
where left(ssn,3)='v0-';
UPDATE 2
premdb=# select * from staff_ks;
  id   |        ssn
-------+--------------------
 91234 | k1UpDIAqaDiEmI3
 85673 | jDkpDsAsaTCCwA3
 13579 | v1-hHUpDsQqa1yDwA3
 97684 | v1-hHUoDAwra5CCzM3
(4 rows)

premdb=# update staff_ks
set ssn=substr(ssn,4)
where left(ssn,3)='v1-';
UPDATE 2
premdb=# select * from staff_ks;
  id   |       ssn
-------+-----------------
 13579 | hHUpDsQqa1yDwA3
 97684 | hHUoDAwra5CCzM3
 91234 | k1UpDIAqaDiEmI3
 85673 | jDkpDsAsaTCCwA3
(4 rows)
```

12. Drop the old key.

```
premdb=# drop key ssnkey0 cascade;
DROP KEY
```

SQL Encryption Examples

This section presents some examples that show how to use the `ENCRYPT_KS` and `DECRYPT_KS` functions, how to assign encryption privileges, and how to rotate keys.

Simple example of encryption and decryption of a single column within one query

The following subquery encrypts the `city` column, then the outer query decrypts it. The query verifies that the decrypted value matches the original value. Because multiple `city` values are the same, the `encrypted` column in the result proves that the same string is encrypted the same way. Note that both functions use all four arguments in this example.

```
premdb=# select city, encrypted,
decrypt_ks(encrypted,yb100key,1,'abcdef0') as decrypted
 from(select city, encrypt_ks(city,yb100key,1,'abcdef0') as encrypted from team) t1
order by 1;
    city      |     encrypted      |   decrypted
--------------+--------------------+--------------
 Barnsley     | mp7{1N7Jx6D        | Barnsley
 Birmingham   | mJ7{2|cJv2Clz2     | Birmingham
 Birmingham   | mJ7{2|cJv2Clz2     | Birmingham
 Birmingham   | mJ7{2|cJv2Clz2     | Birmingham
 Blackburn    | m7twCtcGhgzi       | Blackburn
 Blackpool    | m7twCt6KnUSi       | Blackpool
 Bolton       | mBdvRdcJ           | Bolton
 Bournemouth  | mBt|TZMHpUCgacB    | Bournemouth
 Bradford     | m|swB3sJioC        | Bradford
 Burnley      | mZ6{1hMHd2         | Burnley
 Cardiff      | np7{B|cHu2         | Cardiff
 Coventry     | nB7|AZ6Li6D        | Coventry
 Derby        | sZ7{D|7            | Derby
 Hull         | wZcv33             | Hull
 Ipswich      | xtM{O|sGs2         | Ipswich
 Leeds        | {ZtxBN7            | Leeds
 Leicester    | {ZtuCFsKgsyh       | Leicester
 Liverpool    | {J7|AJ7KnUSi       | Liverpool
 Liverpool    | {J7|AJ7KnUSi       | Liverpool
 London       | {B7vBdcJ           | London
 London       | {B7vBdcJ           | London
 London       | {B7vBdcJ           | London
 ...
```

CREATE VIEW example with an encrypted expression

As a user with `ENCRYPT` and `DECRYPT` privilege, create a view that contains an encrypted expression based on the concatenation of three columns in a table.

```
premdb=# create view encrypted_team as select
encrypt_ks(concat_ws(':',teamid,htid,atid),yb100key) as ids,
name, nickname, city, stadium, capacity, avg_att
from team;
```

As the same user, check the decrypted values:

```
premdb=# select decrypt_ks(ids,yb100key) from encrypted_team limit 5;
-----------------
 1:2:51
 2:3:52
 3:4:53
 4:5:54
 5:6:55
(5 rows)
```

Create a new user with `SELECT` privilege on the table, but not do not grant `DECRYPT` privilege to this user. Query the view as that user:

```
premdb=# create user yb007 password '********';
CREATE ROLE
premdb=# grant select on encrypted_team to yb007;
GRANT
premdb=# \c premdb yb007
Password for user yb007:
You are now connected to database "premdb" as user "yb007".
premdb=> select * from encrypted_team order by 1 limit 5;
    ids     |         name         | nickname  |    city     |     stadium      | capacity | avg_att
------------+----------------------+-----------+-------------+------------------+----------+---------
 cdofaC71   | Bournemouth          | Cherries  | Bournemouth | Vitality Stadium |    11464 |  11.189
 ddohkmN2c3 | Bradford City        | Bantams   | Bradford    | Valley Parade    |    25136 |   0.000
 edIgaCd2   | Blackpool            | Seasiders | Blackpool   | Bloomfield Road  |    17338 |   0.000
 fdYfaCt2   | Bolton Wanderers     | Trotters  | Bolton      | Macron Stadium   |    28723 |   0.000
 g32fg8d1cp3 | West Bromwich Albion | Baggies   | Birmingham  | The Hawthorns    |    27000 |  24.631
(5 rows)
```

Verify that this user cannot decrypt the `ids` column:

```
premdb=# \c premdb yb007
Password for user yb007:
You are now connected to database "premdb" as user "yb007".
premdb=> select decrypt_ks(ids,yb100key) from encrypted_team limit 5;
ERROR:  decrypt_ks: insufficient privileges on key yb100key
```

Unloading encrypted data

Using `ybunload`, you can unload data with a query that contains the `ENCRYPT_KS` function. (This example uses the same concatenated expression in the function as the previous example.)

```
$ ybunload -d premdb --username yb100 -W --format csv -o /home/yb100/premdb_unloads --truncate-existing --select
"select encrypt_ks(concat_ws(':',teamid,htid,atid),yb100key) as ids, name, nickname, city, stadium, capacity, avg_att from team;"
Password for user yb100:
13:01:14.577 [ INFO] ABOUT CLIENT:
   app.cli_args         = "-d" "premdb" "--username" "yb100" "-W" "--format" "csv" "-o" "/home/yb100/premdb_unloads" "--truncate-e
   app.name_and_version = "ybunload version 3.2.0-20064"
   java.home            = "/usr/lib/jvm/java-8-oracle/jre"
   java.version         = "1.8.0_101"
   jvm.memory           = "512.00 MB (max=6.00 GB)"
   jvm.name_and_version = "Java HotSpot(TM) 64-Bit Server VM (build 25.101-b13, mixed mode)"
   jvm.options          = "-Xms512m, -Xmx6g, -XX:+UseG1GC, -Dapp.name=ybunload, -Dapp.pid=12850, -Dapp.repo=/opt/ybtools/lib, -Dap
   jvm.vendor           = "Oracle Corporation"
   os.name_and_version  = "Linux 4.4.0-31-generic (amd64)"

13:01:14.584 [ INFO] Truncating existing files that start with: unload
13:01:14.586 [ INFO] Verifying unload statement...
```

```
13:01:14.681 [ INFO] Unload statement verified
13:01:14.681 [ INFO] Beginning unload to /home/yb100/premdb_unloads
13:01:16.470 [ INFO] Network I/O Complete. Waiting on file I/O
13:01:16.473 [ INFO] Finalizing...
13:01:16.474 [ INFO] Transfer complete
13:01:16.475 [ INFO] Transferred: 3.29 KB Avg Network BW: 22.59 KB/s Avg Disk write rate: 21.35 KB/s
```

Now check the contents of the unloaded file. The first column is encrypted:

```
$ cd premdb_unloads
yb100@yb100:~/premdb_unloads$ more unload_1_1_.csv
ld2haCN3,Arsenal,Gunners,London,Emirates Stadium,60260,59.944
idIhaCd3,Aston Villa,Villains,Birmingham,Villa Park,42785,33.690
jdYgaCt3,Barnsley,Tykes,Barnsley,Oakwell Stadium,23009,0.000
gdogaC72,Birmingham City,Blues,Birmingham,St. Andrew's,30016,0.000
hd2gaCN2,Blackburn Rovers,Rovers,Blackburn,Ewood Park,31367,0.000
edIgaCd2,Blackpool,Seasiders,Blackpool,Bloomfield Road,17338,0.000
fdYfaCt2,Bolton Wanderers,Trotters,Bolton,Macron Stadium,28723,0.000
cdofaC71,Bournemouth,Cherries,Bournemouth,Vitality Stadium,11464,11.189
...
```

You can reload the data as is with `ybload`, then decrypt it with a query or a `CTAS` statement that uses the `DECRYPT_KS` function. (You cannot load and decrypt data in one step with `ybload`.)

Setting up DECRYPT privileges for members of a role

This example shows how to create a role with `DECRYPT` privileges, then create users that belong to the role. These users can decrypt data using a specific key, but they cannot encrypt data.

First create a role, then grant select privileges on tables to the role, then grant `DECRYPT` on a specific key to the role:

```
premdb=# create role decrypters;
CREATE ROLE
premdb=# grant select on all tables in schema public to decrypters;
GRANT
premdb=# grant decrypt on key yb100key to decrypters;
GRANT
```

Now create users who belong to the role:

```
premdb=# create user scofield login in role decrypters password 'I am a decrypter';
CREATE ROLE
premdb=# create user olivier login in role decrypters password 'I am also a decrypter';
CREATE ROLE
```

Now log in as one of those users and run a query that decrypts an encrypted expression:

```
premdb=# \c premdb scofield
Password for user scofield:
You are now connected to database "premdb" as user "scofield".
premdb=> select decrypt_ks(ids,yb100key) from encrypted_team limit 5;
 decrypt_ks
------------
 1:2:51
 2:3:52
 3:4:53
 4:5:54
```

```
 5:6:55
(5 rows)
```

Finally, attempt to encrypt data:

```
premdb=> select encrypt_ks(name,yb100key) from encrypted_team limit 5;
ERROR:  encrypt_ks: insufficient privileges on key yb100key
```

LDAP Integration

Yellowbrick Documentation > How-to Guides > Administer Yellowbrick > Databases > LDAP Integration

Platforms: EE: All appliance platforms, EE: All cloud platforms

Parent topic: Databases

In this section:

- LDAP Authentication
- Synchronizing Users and Groups
- Importing LDAP Certificates (Appliance)
- LDAP Sample Schema

Lightweight Directory Access Protocol (LDAP) is an application protocol for directory access (like a telephone book) that provides authentication and membership information for users and groups within the directory. Yellowbrick can be configured to use directory service providers such as Microsoft Active Directory (AD) and OpenLDAP for authentication and group information.

LDAP (unencrypted), LDAPS (LDAP over SSL), and LDAP+TLS communication protocols are all supported. If you want to use LDAPS or LDAP+TLS, the first step is to import your security certificates.

Local Compared to LDAP Users and Groups

Yellowbrick users and groups may be local to the Yellowbrick cluster, LDAP-based, or a combination of both. *Enabling LDAP on your cluster does not prohibit the use of locally authenticated users and groups.*

- *Local users* are those who are authenticated by the database using a password stored in the database.
- *LDAP users* are those authenticated by the LDAP database.
- *Local groups* are those groups that have been created locally and are not being replicated from the LDAP server. An LDAP group is equivalent to a Yellowbrick database *role*.
- *LDAP groups* are groups that you have designated within Yellowbrick to be replicated from the LDAP server.

Before any user can be authenticated, that user must exist within the Yellowbrick cluster. You can create users in two ways:

- Manually within the database (via a `CREATE USER` command or the SMC)
- Automatically as part of an LDAP synchronization operation (via the SMC, with user-defined filter criteria)

Yellowbrick distinguishes between local and LDAP users based on whether or not they have a local password. Local Yellowbrick users will have a password. If a local user is created without a password or a password that is subsequently set to `null`, the user can be authenticated only through LDAP. If LDAP is not enabled, no user with a `null` password will be granted database access. In either case, the authentication is transparent to users; a user will always see the same database-generated authentication regardless of where the authentication happens.

A superuser can convert an existing local user to an LDAP authenticated user by setting the user's password to `null`. A superuser can also change an existing LDAP authenticated user to a locally authenticated user by setting a non-`null` password for that user.

Getting Started with Yellowbrick and LDAP

The following sections explain the process of importing certificates (for SSL and TLS configurations), setting up LDAP authentication for superusers and regular users, and optionally synchronizing LDAP users and groups with database users and roles.

LDAP Authentication

Platforms: EE: All appliance platforms, EE: All cloud platforms

Parent topic: LDAP Integration

In this section:

When LDAP authentication is enabled, LDAP is used to transparently authenticate logins for users who do not have local passwords. Any user account that has a database local password will be authenticated by the database, bypassing LDAP authentication. In either case, the user must first exist in Yellowbrick. If you want LDAP users to be automatically created in Yellowbrick, you can enable LDAP synchronization.

Before enabling LDAP authentication, Yellowbrick recommends that you create one or more superusers with local passwords. These user accounts will bypass LDAP authentication; regardless of the functional state of the LDAP server, these designated superusers will be able to log in to and administer the database. Therefore, the `yellowbrick` superuser is typically not set up as an LDAP user.

If you have existing database user accounts that are currently authenticated by the database but you want them to switch to LDAP authentication, set their passwords to `NULL`. For example, if `brick` is a database user with the local password `yellow`, and you want future logins with this account to use LDAP authentication, run the following ALTER ROLE command:

```
premdb# alter role brick with password null;
ALTER ROLE
```

Alternatively, you can use the **Users** tab in the SMC to make this change. (Note that the authentication change may not be immediate and could take a minute to apply in the system.)

For information about default users that are created on a Yellowbrick appliance, see User Accounts.

Common Errors with LDAP Authentication

If a login attempt is made for a user account that exists in the LDAP directory but does not exist in Yellowbrick, the system returns an error. For example:

```
FATAL: role "dummy.user" does not exist
```

If a login attempt occurs with a username that does not exist in the LDAP directory, or an incorrect password is applied, the system returns an error. For example:

```
LDAP authentication failed for user "some_user"
```

LDAP Authentication Modes

Platforms: EE: All appliance platforms, EE: All cloud platforms

Parent topic: LDAP Authentication

Yellowbrick supports two modes for LDAP authentication:

- Bind
- Search, then Bind

See also LDAP Authentication Settings.

Bind Mode

In Bind mode, the Yellowbrick cluster builds a user's full credential from a prefix, the username, and a suffix. Typically, these components build a distinguished name that uniquely identifies a user in the LDAP Directory Server. A distinguished name is a unique set of attributes that define a path to the user in the LDAP structure.

For example, a username `analyst1` in the `OrgUsers` organizational unit (OU) at the root of the directory structure might have the following distinguished name:

```
uid=analyst1,ou=OrgUsers,dc=test,dc=yellowbrick,dc=io
```

The DN prefix is `uid=`, and the DN suffix is `,ou=OrgUsers,dc=test,dc=yellowbrick,dc=io`. When `analyst1` logs into the cluster, the username is submitted to the LDAP server as follows:

```
uid=analyst1,ou=OrgUsers,dc=test,dc=yellowbrick,dc=io
```

Alternatively, the DN prefix may specify `cn=`. This CN (common name) value can be either the username or the given name and surname of the user.

If you are using a Microsoft Active Directory (AD) environment, instead of using a fully qualified distinguished name, you can use the user principal name (UPN), which is a unique identifier in AD. The UPN consists of the username (`sAMAccountName`) and domain. For example, `analyst1` in the `test.yellowbrick.io` domain has the UPN `analyst1@test.yellowbrick.io`. In this case, `@test.yellowbrick.io` would be your DN suffix for login and you would leave the DN prefix blank. For more information about UPNs, see the MSDN documentation.

Note: Bind mode is most useful if all of your users are in a single organizational unit (OU), or if they are in AD. If users are in multiple organizational units and you are not using AD, Yellowbrick Data recommends you use the Search, then Bind mode, as described in the next section.

Search, then Bind Mode

This mode allows the Yellowbrick cluster to connect to the LDAP server, search the tree for an attribute containing the username, then retrieve that user's distinguished name for authentication. The base DN is the location in the tree to start searching from, and it must specify the fully qualified distinguished name.

For example, if your tree looks like this:

```
test.yellowbrick.io
    OrgUsers
       <individual users>
    SecurityGroups
```

```
        Database
           <separate groups>
```

you can set your Base DN to `ou=OrgUsers,dc=test,dc=yellowbrick,dc=io` and limit your searches to the `OrgUsers` branch only. Typically, it is best to pick the highest common branch: `OrgUsers` in this case, instead of `test.yellowbrick.io`.

The bind DN is the user who will search the LDAP directory, and must be either a fully distinguished name (or a UPN in the case of an AD environment).

For example:

```
uid=admin,ou=OrgUsers,dc=test,dc=yellowbrick,dc=io
```

or the UPN:

```
admin@test.yellowbrick.io
```

The search attribute is the LDAP attribute that defines the username. For example, in many UNIX-based directory servers, the attribute will be `uid`. For AD servers, it will be `sAMAccountName`.

LDAP Authentication Settings

Platforms: EE: All appliance platforms, EE: All cloud platforms

Parent topic: LDAP Authentication

Use this section for reference when setting up LDAP authentication.

Bind Settings

LDAP Server

Enter the hostname or IP address of your LDAP server. For example: `test.yellowbrick.io`

Server Port

Enter the LDAP server port number, or leave it blank for the default (389 for unsecured mode or TLS, 636 for LDAPS (SSL).

Secure Mode

Choose from **Unsecured**, **LDAPS**, or **TLS**.

DN Prefix

The DN prefix is the first component of a fully qualified distinguished name. For example: `cn=` or `uid=`. Depending on your LDAP environment, the CN (common name) value may be either a username or the given name and surname of the user. A UID (user ID) is an LDAP account attribute that stores a username. Both CN and UID formats work for OpenLDAP configurations.

Note: No prefix is required for Active Directory configurations.

DN Suffix

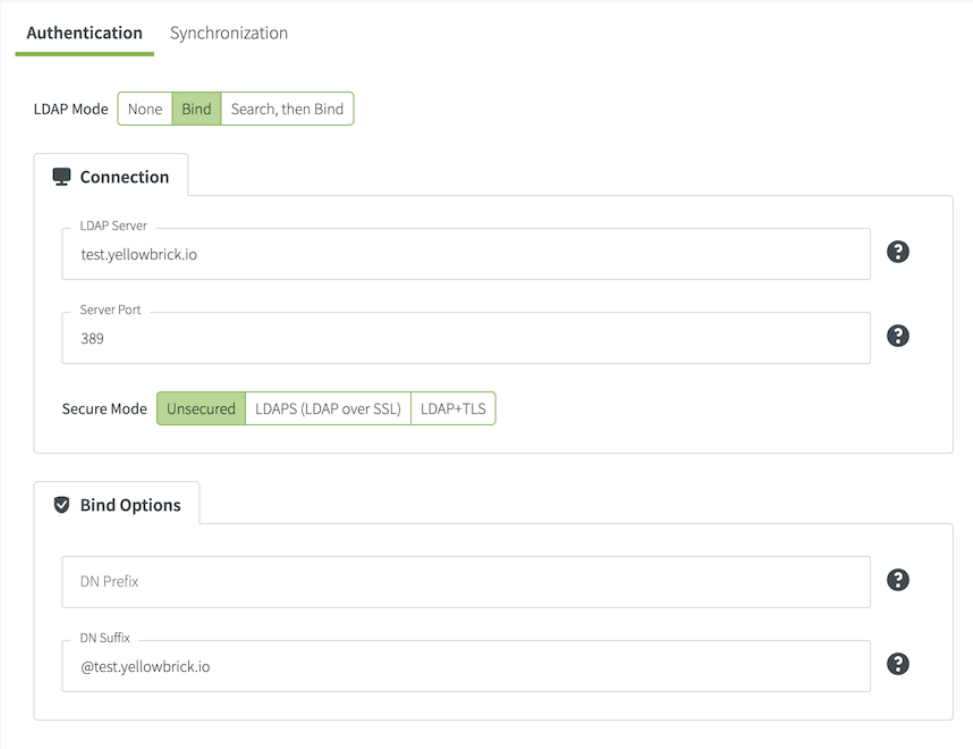
The DN suffix is the remaining piece of a fully qualified distinguished name. A DN suffix may consist of an OU (organizational unit) and domain components (DCs). For example: `,ou=OrgUsers,dc=test,dc=yellowbrick,dc=io`. Alternatively, you can use the UPN (user principal name) format: `@domain`. For example: `@test.yellowbrick.io`. For Active Directory configurations, use the UPN suffix.

Bind Examples

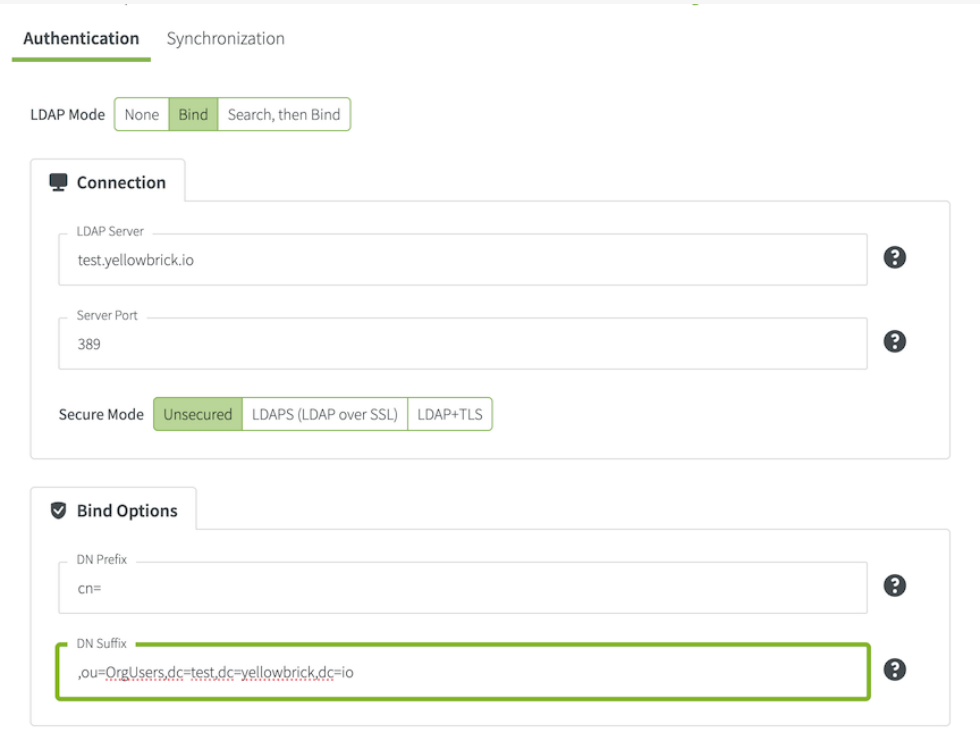
The following settings apply to the Bind mode. See also LDAP Authentication Modes.

Visit the LDAP page of the instance management dashboard in Yellowbrick Manager.

The first example is for a user of a Microsoft Active Directory Server:



The second example is for a user of an OpenLDAP Server:



Search, then Bind Settings

The following settings apply only to the Search, then Bind mode. See also LDAP Authentication Modes.

Base DN
Enter the base search tree DN for locating LDAP entries: the distinguished name where the directory search should begin.

Bind DN
Enter the DN for initial binding to the LDAP server, or leave blank for anonymous binding. (Anonymous binding allows a client to connect and search the directory without logging in.)

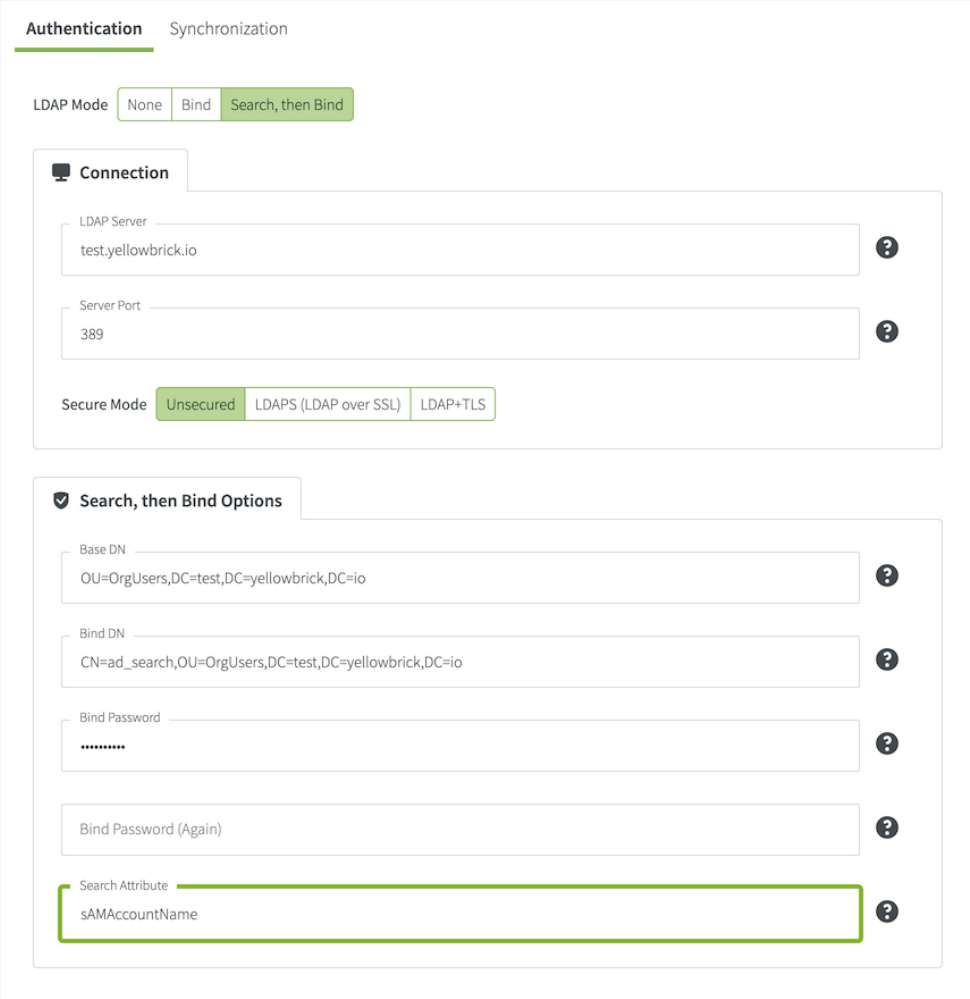
Bind Password (two fields)
Enter the password for the initial binding (twice).

Search Attribute
Enter a search attribute, such as `cn`, `uid`, or `sAMAccountName`, which is an Active Directory user account field.

Search, then Bind Examples

Visit the LDAP page of the instance management dashboard in Yellowbrick Manager.

Here is an example for a user of a Microsoft Active Directory Server:


Authentication
Synchronization
LDAP Mode
None
Bind
Search, then Bind
Connection
LDAP Server
test.yellowbrick.io
Server Port
389
Secure Mode
Unsecured
LDAPS (LDAP over SSL)
LDAP+TLS
Search, then Bind Options
Base DN
OU=OrgUsers,DC=test,DC=yellowbrick,DC=io
Bind DN
CN=ad_search,OU=OrgUsers,DC=test,DC=yellowbrick,DC=io
Bind Password
Bind Password (Again)
Search Attribute
sAMAccountName

LDAP Bind Examples

Platforms: EE: All appliance platforms, EE: All cloud platforms

Parent topic: LDAP Authentication

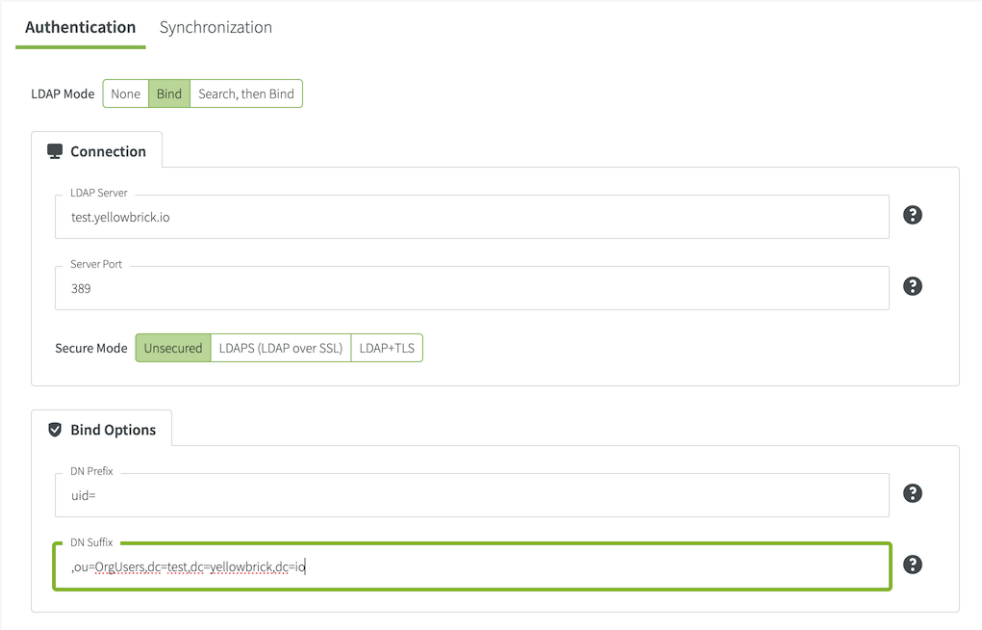
This section contains LDAP authentication examples for Bind mode. See LDAP Sample Schema for information about the LDAP groups and users associated with these examples.

Open LDAP Example with a Distinguished Name

This example shows the Open LDAP settings for a login that uses a distinguished name (DN) to uniquely identify the user account. To configure LDAP settings for user `analyst1` in the `OrgUsers` organizational unit (OU) at the directory root, set the following prefix and suffix:

- **DN Prefix**: `uid=`
- **DN Suffix**: `,ou=OrgUsers,dc=test,dc=yellowbrick,dc=io`

Note: You must include the comma before `ou=` in the suffix string. The `dc` entries are the components of the company domain.



When `analyst1` logs into the cluster, the generated LDAP username is submitted to the LDAP server as follows:

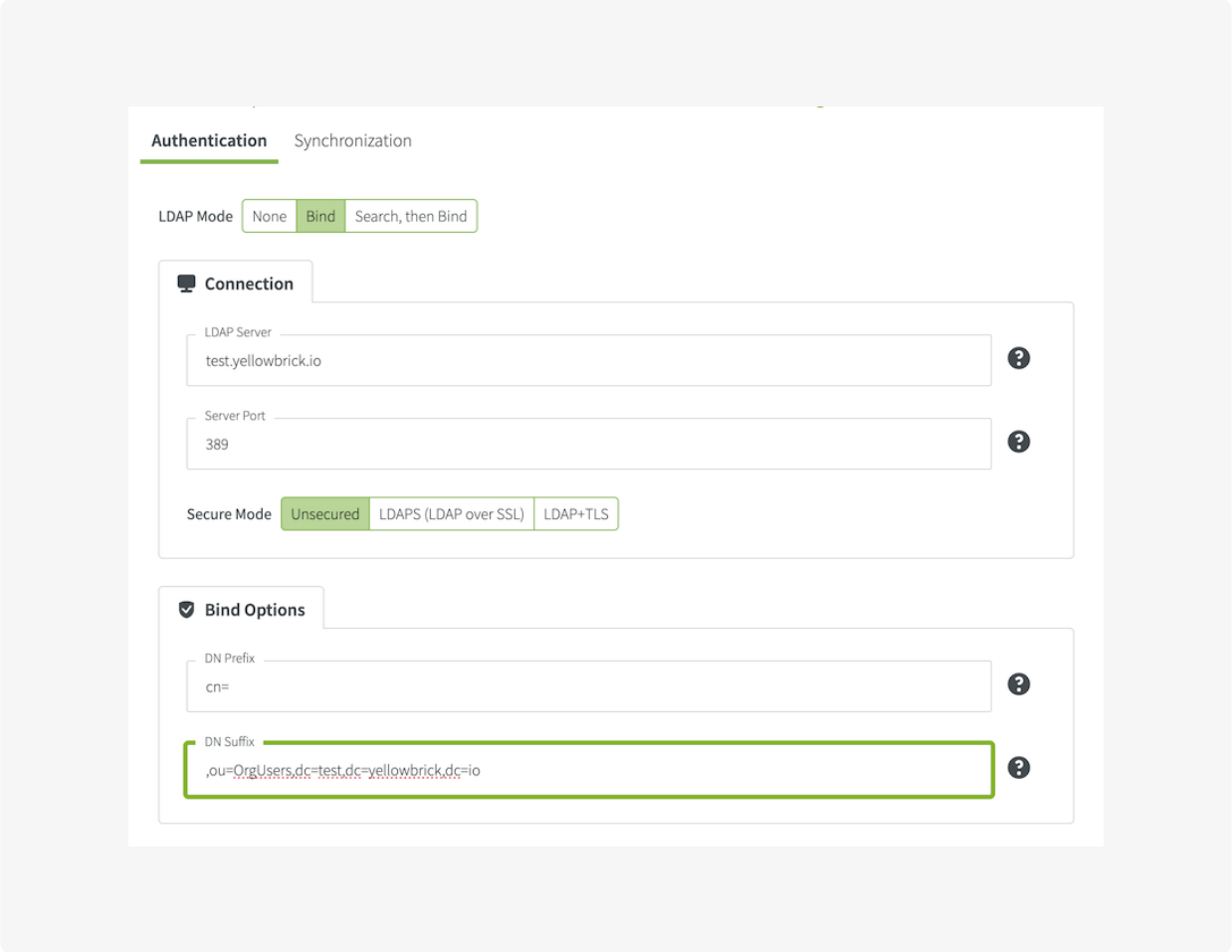
```
uid=analyst1,ou=OrgUsers,dc=test,dc=yellowbrick,dc=io
```

Active Directory (AD) Example with a Distinguished Name

This example shows the Microsoft Active Directory (AD) settings for a login that uses a distinguished name (DN) to uniquely identify the user account. To configure LDAP settings for user `analyst_1` in the `OrgUsers` organizational unit (OU) at the directory root, set the following prefix and suffix:

- **DN Prefix**: `cn=`
- **DN Suffix**: `,ou=OrgUsers,dc=test,dc=yellowbrick,dc=io`

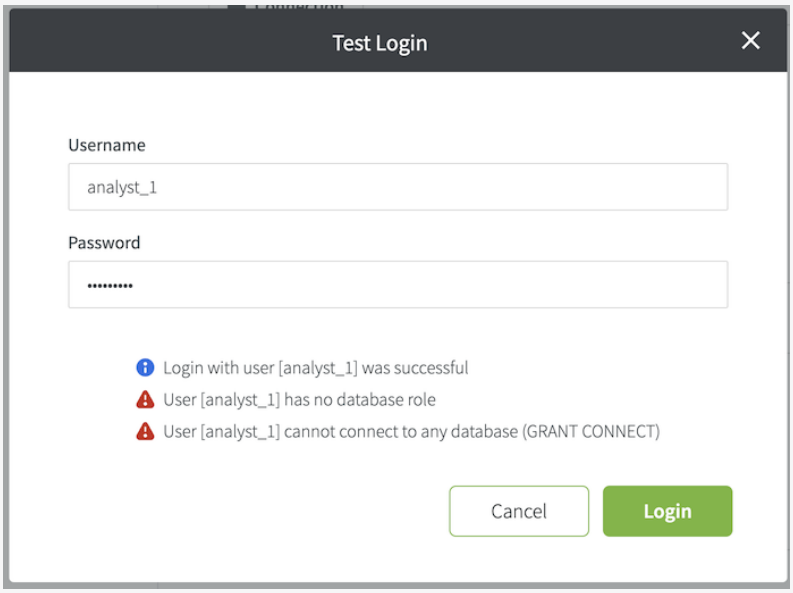
Note: You must include the comma before `ou=` in the suffix string. The `dc` entries are the components of the company domain.



When `analyst_1` logs into the cluster, the generated LDAP username is submitted to the LDAP server as follows:

```
cn=analyst_1,ou=OrgUsers,dc=test,dc=yellowbrick,dc=io
```

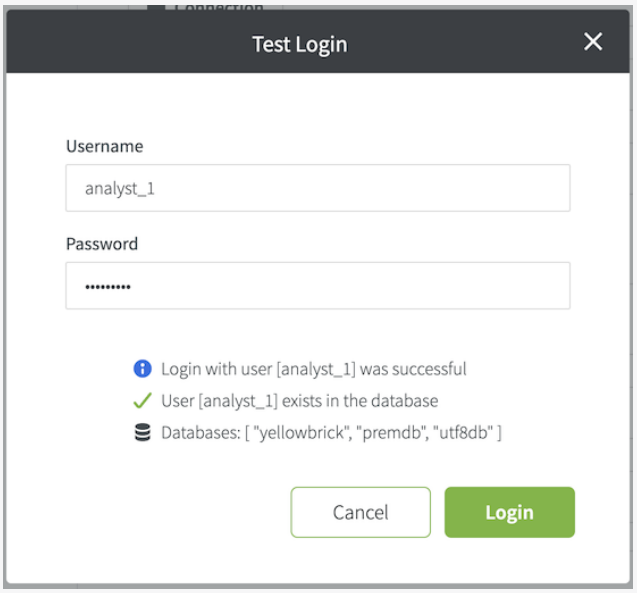
Note that `analyst_1` exists only in LDAP. There is no equivalent database user. Therefore, the login is successful, but two warnings are displayed:



To allow this user to authenticate via LDAP *and* connect to the Yellowbrick database, you have to create the user in the database (without a local password) and grant the user access to tables and other objects as needed:

```
premdb=# create user analyst_1;
CREATE ROLE
premdb=# grant select on match to analyst_1;
GRANT
...
```

Now you can retest the LDAP login:

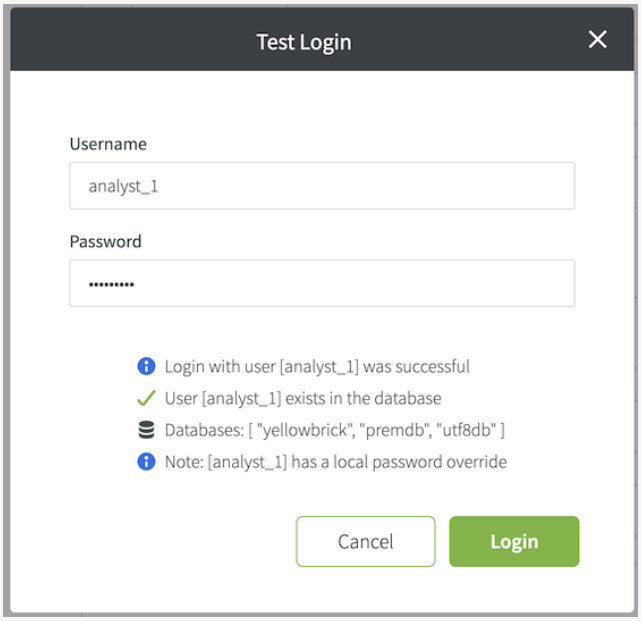


The user can now log in via LDAP, exists as a database user, and can connect to databases.

Finally, note what would happen if an administrator gave a local password to this user:

```
premdb=# alter user analyst_1 password 'local db pswd';
ALTER ROLE
```

The LDAP login test would still succeed but the user would now be using the local password, not the LDAP password, to authenticate:

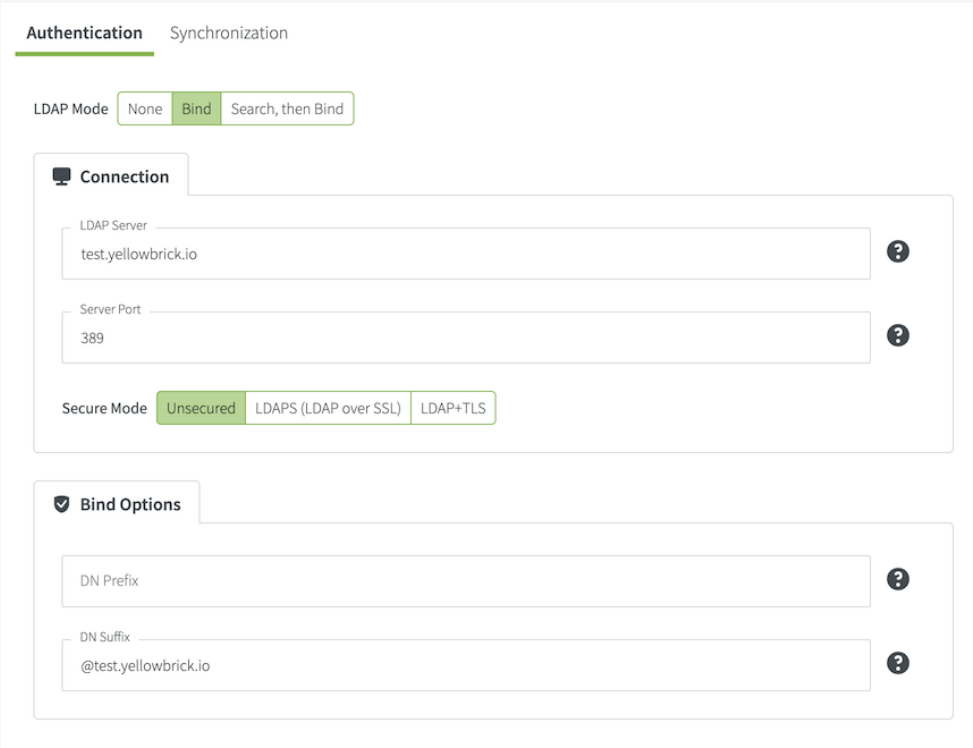


Active Directory (AD) Example with a User Principal Name (UPN)

An alternative approach for authentication is to use the user principal name (UPN), which maps exactly to the concatenation of the LDAP `sAMAccountName` for the user and the `@domain` string (or email domain).

To configure LDAP settings for user `analyst1`, set the following prefix and suffix:

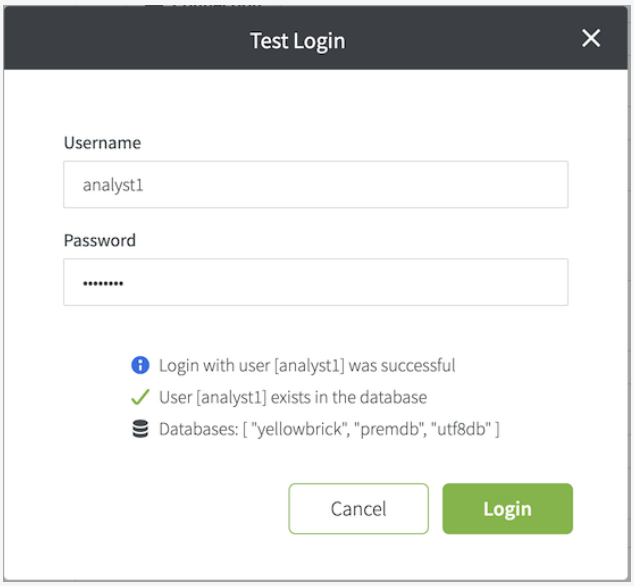
- **DN Prefix**: *no prefix*
- **DN Suffix**: `@test.yellowbrick.io`



When `analyst1` logs into the cluster, the generated LDAP username is submitted to the LDAP server as follows:

```
analyst1@test.yellowbrick.io
```

The LDAP test login will succeed, and if `analyst1` exists in the database, the login will return no warnings:


Test Login
Username
analyst1
Password
Login with user [analyst1] was successful
User [analyst1] exists in the database
Databases: ["yellowbrick", "premdb", "utf8db"]
Cancel
Login

Search, then Bind LDAP Example

Platforms: EE: All appliance platforms, EE: All cloud platforms

Parent topic: LDAP Authentication

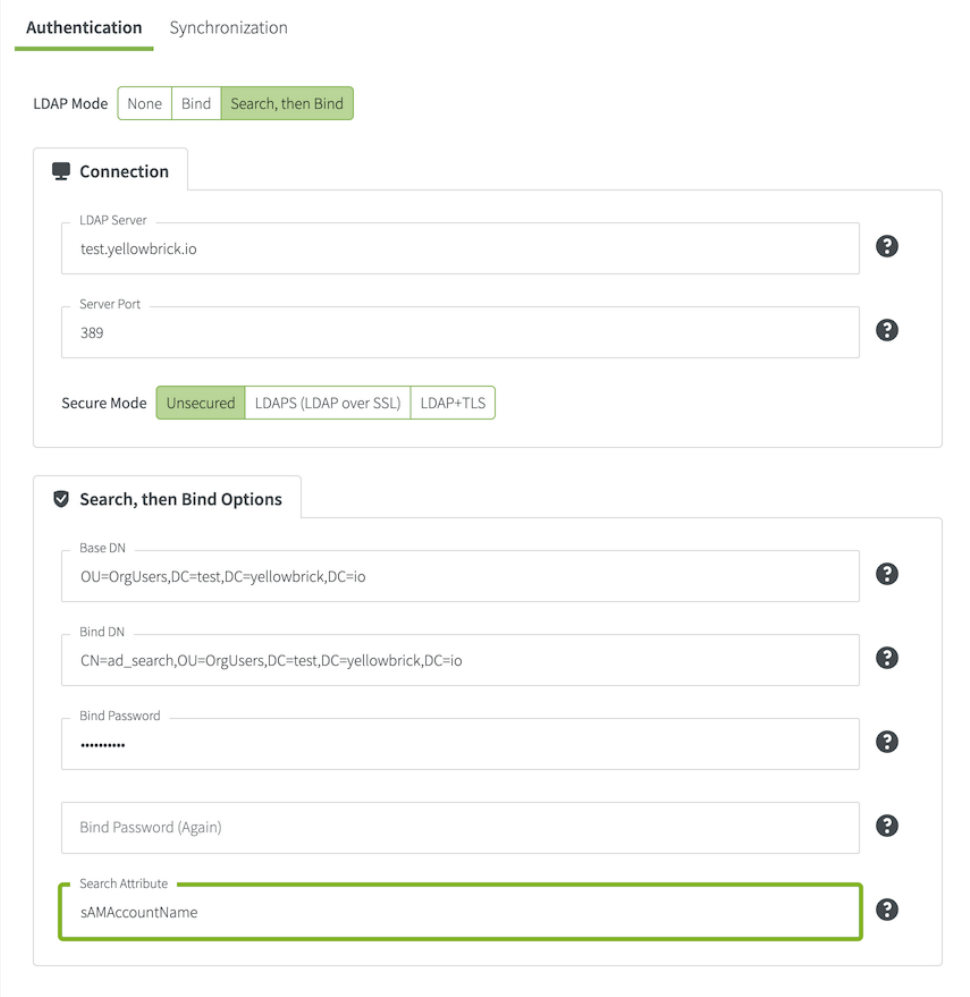
This section contains an LDAP authentication example for Search, then Bind mode with an Active Directory (AD) LDAP server. See LDAP Sample Schema for information about the LDAP groups and users associated with this example.

In this mode, you specify up front a login user whose username and password will be used to search the LDAP directory when other LDAP users attempt to log in with their LDAP usernames and passwords. The **Bind DN** and **Bind Password** fields define the initial user. The search starts from a specified base directory (**Base DN**).

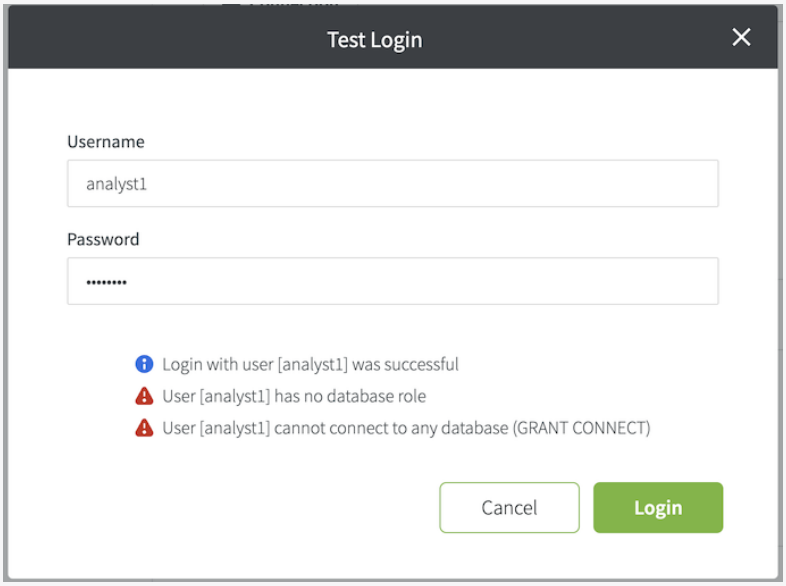
When a login is attempted, the server binds to the LDAP directory with this fixed username and password, then performs a search for the user who is trying to log in to the database. The search tries to find an exact match on a specified "search attribute." In this example, the username supplied when the user logs in must match that user's `sAMAccountName` in the LDAP directory.

The following settings are used in this example:

- **Base DN**: `ou=OrgUsers,dc=test,dc=yellowbrick,dc=io`
- **Bind DN**: `cn=ad_search,ou=OrgUsers,dc=test,dc=yellowbrick,dc=io` (where `ad_search` is the common name (`CN`) of the user who will be authorized for the search)
- **Bind Password**: the password for the Bind DN user (specified twice)
- **Search Attribute**: `sAMAccountName` , which is the LDAP attribute that will be used to match login usernames



In this example, analyst1 logs in successfully. In the background, the search for a match on `sAMAccountName=analyst1` was conducted after authorizing the initial login for user `ad_search`.



Note that the login was successful, but to allow this user to both authenticate via LDAP *and* connect to the Yellowbrick database, you have to create the user in the database (without a local password) and grant the user access to tables and other objects as needed:

```
premdb=# create user analyst1;
CREATE ROLE
premdb=# grant select on match to analyst1;
GRANT
...
```

Then you can retest the LDAP login.

Setting Up LDAP Authentication

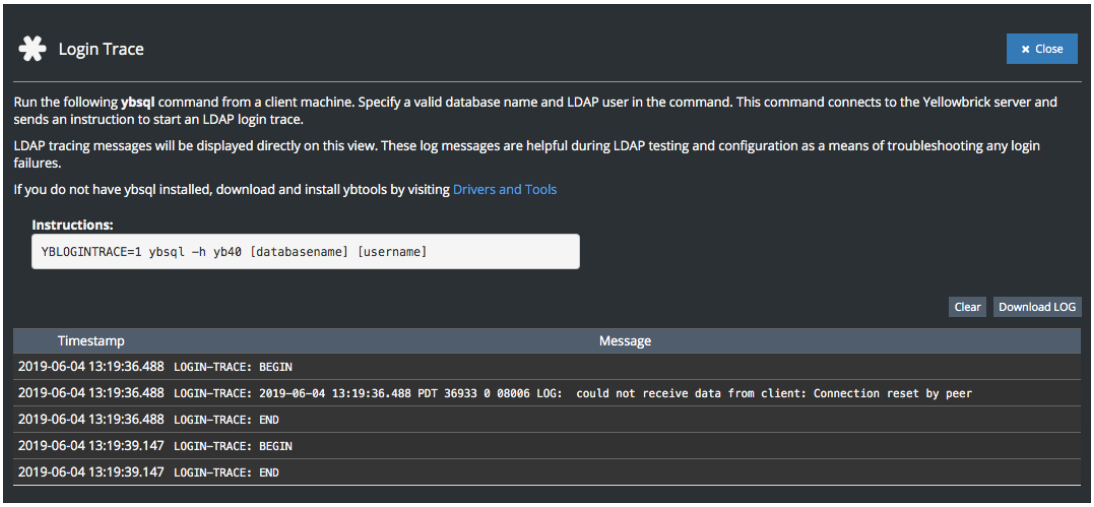
Platforms: EE: All appliance platforms

Parent topic: LDAP Authentication

To set up LDAP authentication, follow these steps:

1. Log in to a database on your cluster as the `yellowbrick` user and manually create the *superusers* that your system needs to authenticate (via LDAP credentials). Use the CREATE ROLE command or the SMC for this purpose; do not provide passwords for these users. The usernames you create must match the usernames in your LDAP directory.
2. In the SMC, go to **Configure** > **LDAP** > **Authentication**.
3. Optionally, set up a login trace that runs on the Yellowbrick server and displays messages for LDAP login attempts directly via the SMC. Click **Login Trace** in the upper right corner of the screen, then run the `ybsql` command from a client machine as instructed.

In this example, the `ybsql` command has been run, and the login trace is returning messages to the SMC:



4. Choose the LDAP mode: **Bind**, or **Search, then Bind**. Then fill out the associated fields. The requirements vary based on both the LDAP mode and the type of LDAP Server (Active Directory, OpenLDAP, and so on). See LDAP Authentication Modes.
5. Test the login for the superusers you created. You should see a "login successful" message, and confirmation that the user exists and belongs to at least one database. (The user may be able to log in but not connect to a database; you may need to create or sync the user in the database and grant privileges as needed. See the following steps.)

Note: You may be able to import a trusted certificate when you test LDAP logins. If you have imported a root or intermediate certificate for the authority that issued the LDAP server certificate, but the server certificate itself has not been imported, click **Set Trusted Certificates**. (See also Importing Certificates.)

6. If the test was successful, click **Save Settings**. If not, go back and check all of your settings for accuracy.
7. Create all of your regular database users either manually *or* by synchronizing with the LDAP server:

- Log in to the database as a new LDAP-authenticated superuser and use that account to create all of the regular users on the cluster. Use CREATE ROLE or the SMC to create database users.

  Note: Create users *without* passwords; LDAP passwords will be used.

- Follow the steps under Synchronizing Users and Groups. **Note:** Any users that are local to the Yellowbrick database and have local passwords will be able to log in to the database without LDAP authentication.

Synchronizing Users and Groups

Platforms: EE: All appliance platforms, EE: All cloud platforms

Parent topic: LDAP Integration

In this section:

First complete the steps under Setting Up Authentication.

This procedure is an alternative to creating non-superusers manually in the database. The Yellowbrick cluster synchronizes with the users and groups stored in an LDAP directory and creates users and roles with the same credentials. You can synchronize specific users and groups by defining search criteria and filters. Ideally, your organization will have an LDAP group defined that specifically contains the users that need to be synchronized to the Yellowbrick database.

You can also use the sys.ldap_sync SQL function to synchronize users and groups based on your current LDAP configuration. The following procedure explains how to use the Yellowbrick Manager (cloud platform) or SMC (appliance platform), which provides several synchronization options.

1. In Yellowbrick Manager, open the instance management dashboard, and navigate to the **LDAP** section. Choose the tab labeled **Synchronization**.
2. Select **Synchronize Users/Groups**.
3. Fill out the required fields. Some entries are carried over from the authentication procedure.

Note the following synchronization options:

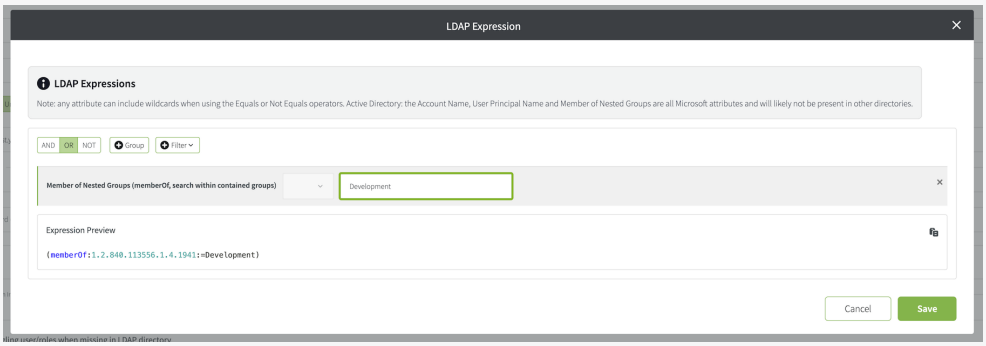
- Drop dangling user/roles when missing in LDAP directory
- Force user/roles to lowercase
- Filter users using discovered groups (AD only/using memberOf)

See LDAP Synchronization Settings for details.

4. Set up group and user filters to define which LDAP members to synchronize.

Use the Expression Builder to simplify the process of searching through multiple LDAP groups; click the **+ Expression** button to the right of the two filter fields. This interface provides building blocks for the notation that is commonly used in LDAP searches.

For example:



Tip: Use an LDAP browser to help identify groups and users based on complex distinguished name (DN) strings and other LDAP attributes.

5. Test the synchronization settings or click **Synchronize Now**.

LDAP Synchronization Settings

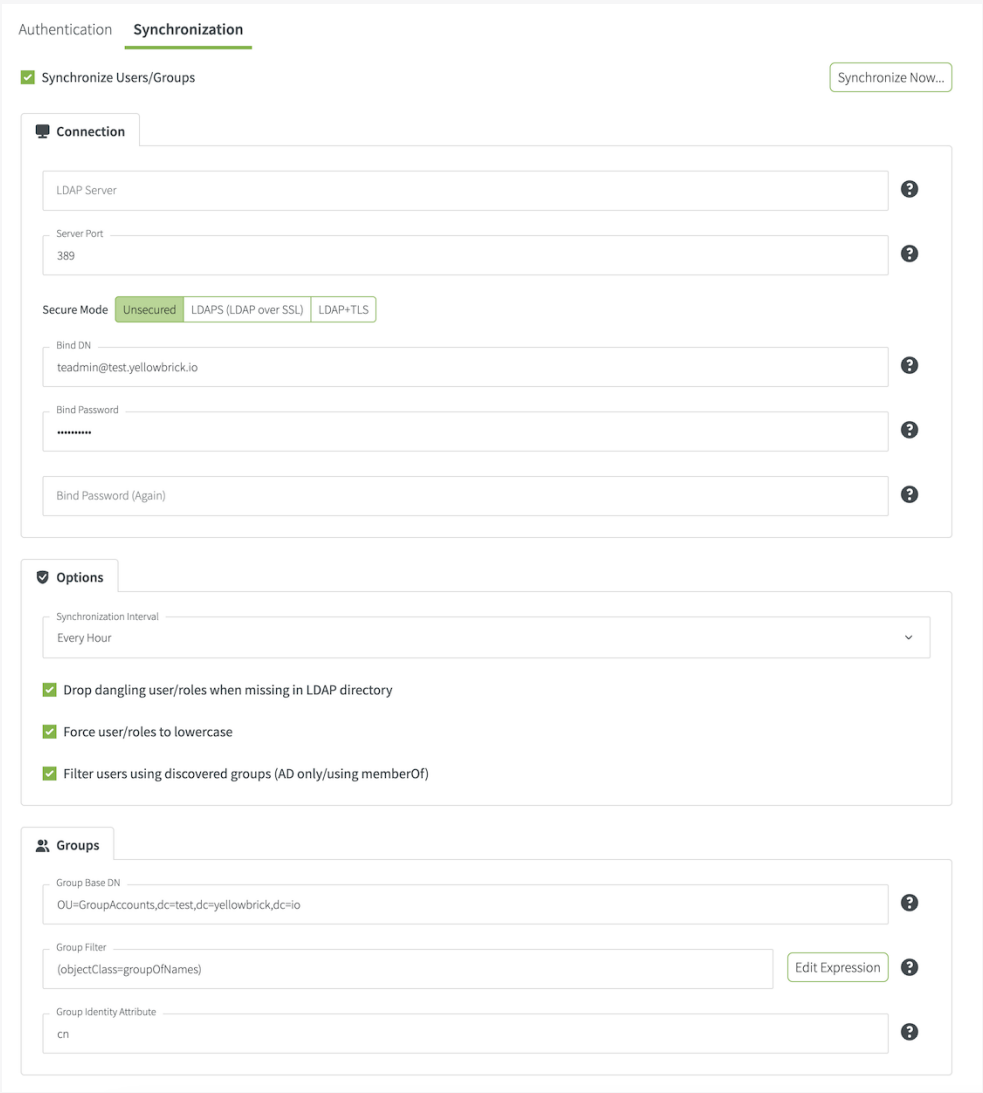
Platforms: EE: All appliance platforms, EE: All cloud platforms

Parent topic: Synchronizing Users and Groups

You must specify the Bind DN and Bind Password settings, unless you are using anonymous bind. To locate users and groups in the target directory, specify the base DN and filter for each type. You can also specify an optional identity attribute that identifies each principal type.

The following settings are specific to synchronization. See LDAP Authentication Settings for descriptions of some common fields. See also LDAP Synchronization: Basic Example and LDAP Synchronization: Examples with Filters.

Be sure to select the **Synchronize Users/Groups** checkbox:



Synchronization Interval

Select an interval from the dial to specify the frequency of synchronizing LDAP users and groups with database users and roles.

Drop dangling users/roles when missing in LDAP directory

This option drops users and roles from the Yellowbrick database when they are not found in the LDAP directory. If this option is not selected, LDAP-synchronized users and roles that were added to the database at some point but do not exist in LDAP continue to exist in the database. This option does not drop local users that were created manually with CREATE USER or CREATE ROLE and were not synchronized via LDAP.

In effect, when this option is *not* selected, synchronization only creates new LDAP roles and users.

Note: When synchronization is enabled *and* this option is chosen, note the following behavior. If you delete a user from your LDAP directory but the equivalent database user owns objects, you cannot drop the database user. For example, if `testuser1` is the owner of `test_database` and you delete `testuser1` in LDAP, Yellowbrick

returns an error of the form:

```
Could not drop DB group: testuser1: SQL []; role "testuser1" cannot be dropped because
        some objects depend on it
```

Force user/roles to lowercase

This option is generally recommended and forces uppercase or mixed-case LDAP user and role names to all lowercase in the Yellowbrick database. Although LDAP searches are not case-sensitive, Yellowbrick user and role names are. They work the same as other database identifiers in this respect. Names that are not double-quoted are forced to lower case. For example, when **Force user/roles to lowercase** is used, a user named `ADUser` in LDAP is created as `aduser` in Yellowbrick, avoiding the need for quoted identifiers to correctly handle the mixed-case LDAP username.

Filter users using discovered groups

This option optimizes group filtering for very large LDAP directories. All groups that qualify for the group filter will have all their members searched iteratively, including members of nested groups. This feature works with any LDAP server that supports `memberOf` filtering, such Active Directory (AD). For example, the initial LDAP search might return 10,000 groups. To retrieve matched users from those groups, 10,000 subsequent searches would be run, one on each group.

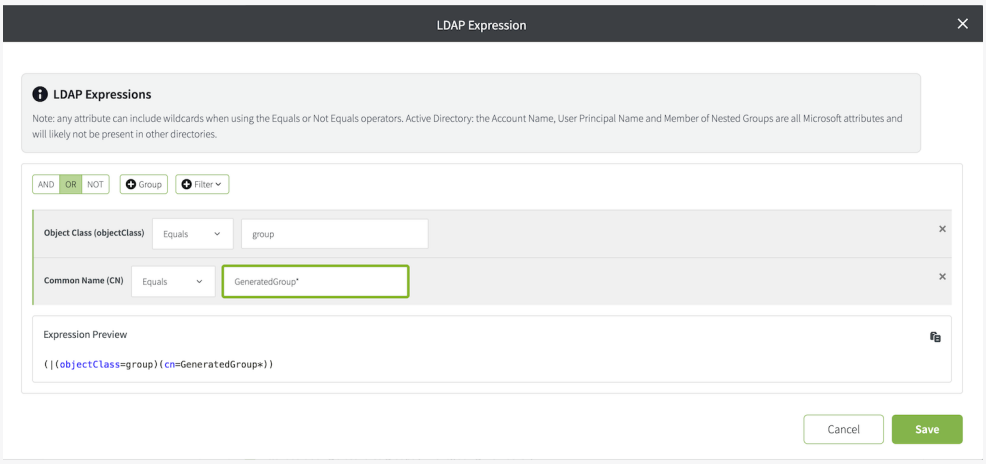
Group Base DN

Enter the base search tree DN for locating LDAP group entries.

Group Filter

Enter the LDAP filter for locating LDAP group entries. If you leave this field blank, the default filter is `(objectClass=group)`.

Use the Expression Builder to simplify the process of searching through multiple LDAP groups; click the **+ Expression** button to the right of the two filter fields. This interface provides building blocks for the notation that is commonly used in LDAP searches.



See also LDAP Synchronization: Examples with Filters.

Group Identity Attribute

Enter the LDAP attribute for identifying LDAP group entries. If you leave this field blank, the default attribute is `cn`.

User Filter

Enter the LDAP filter for locating LDAP user entries. If you leave this field blank, the default filter is `(objectClass=user)`.

Use the Expression Builder to simplify the process of searching for LDAP users; click the **+ Expression** button to the right of the two filter fields. This interface provides building blocks for the notation that is commonly used in LDAP searches.

User Base DN

Enter the base search tree DN for locating LDAP user entries.

User Identity Attribute

Enter the LDAP attribute for identifying LDAP group entries. If you leave this field blank, the default attribute is `cn`. For Active Directory setups, consider using `sAMAccountName`.

LDAP Synchronization: Basic Example

Platforms: EE: All appliance platforms, EE: All cloud platforms

Parent topic: Synchronizing Users and Groups

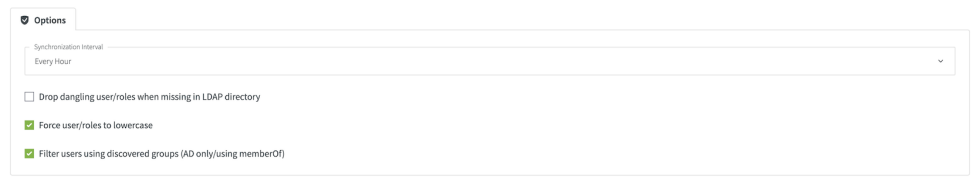
This section presents a simple example of LDAP synchronization. This example uses an AD LDAP server and synchs all groups and all users that belong to the specified OU. See LDAP Sample Schema for information about the LDAP groups and users associated with this example.

1. Set up LDAP authentication as directed in one of the previous AD examples.
2. Visit the LDAP configuration user interface for your platform, and configure synchronization.

In Yellowbrick Manager, navigate to instance management for your instance, choose **LDAP**, choose the **Synchronization** tab, then select **Synchronize Users/Groups**. Fill out the details as shown and click **Save Settings**.

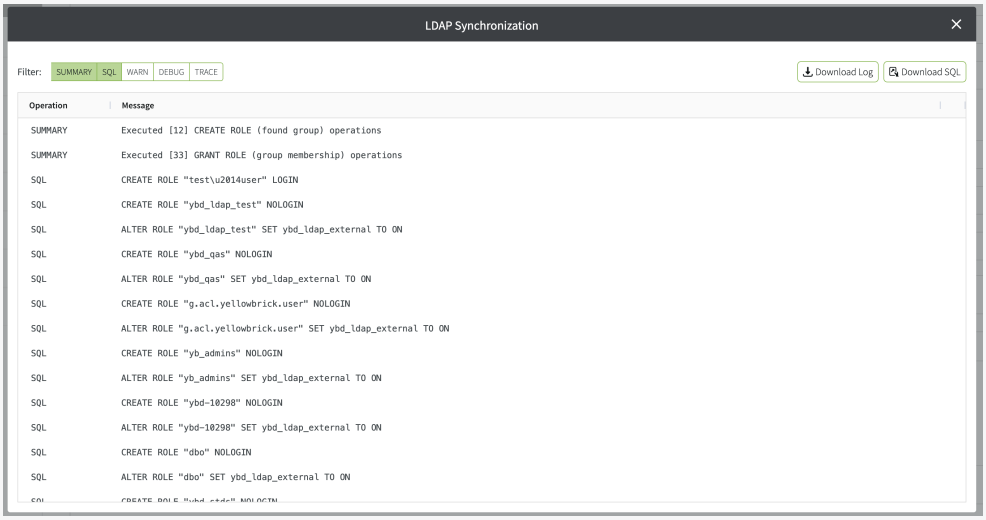
Configure Connection

Choose Options



Specify Group and User Filter Criteria

3. Click **Test Synchronization Settings**. A test results page is displayed.



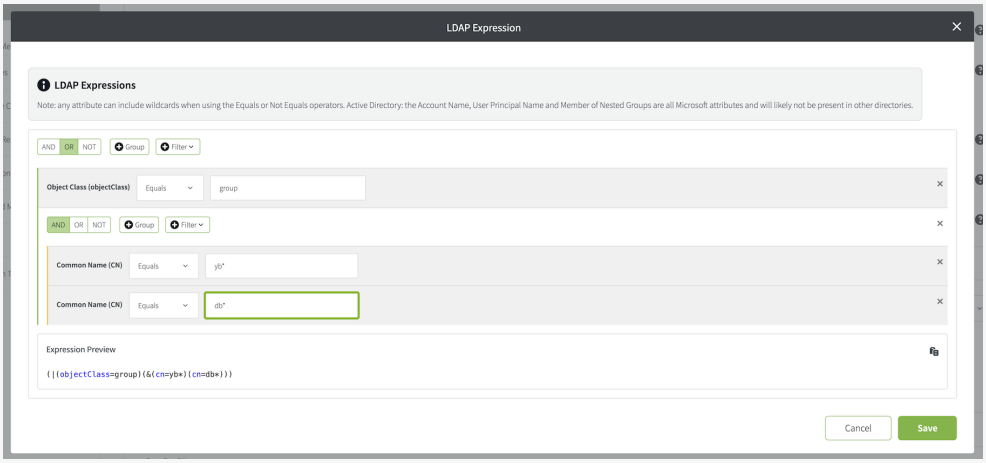
LDAP Synchronization: Examples with Filters

Platforms: EE: All appliance platforms, EE: All cloud platforms

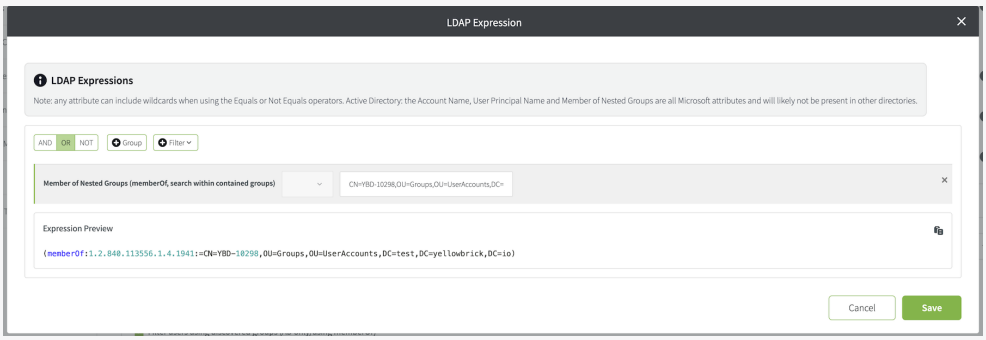
Parent topic: Synchronizing Users and Groups

Following on from the previous example, this section shows how to filter groups and users for LDAP synchronization. You can use the Expression Builder to simplify this task. See LDAP Sample Schema for information about the LDAP groups and users associated with these examples.

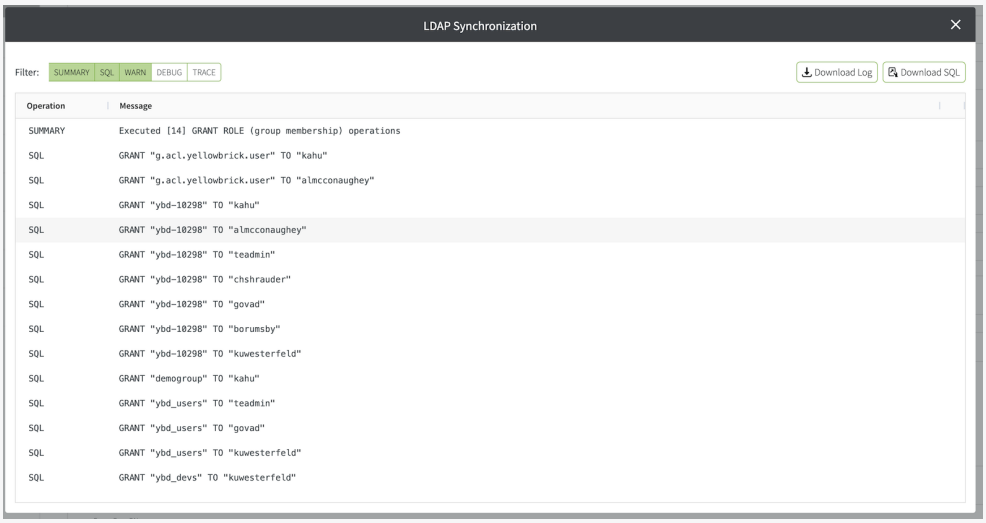
In this case, assume that your organization's LDAP directory contains additional non-Yellowbrick groups under `Database/SecurityGroups`, such as `mstr_users` and `tableau_users`, that you want to filter out during synchronization. To synchronize only those groups whose names begin with `yb` or `db`, for example, you can use the LDAP Expression Builder as shown here. (Alternatively, you can construct the expression manually).



Here is a similar example for the user filter, which looks for all users who are a member of the group `yb_all_users` or one of its descendant groups (**Member of Nested**).



After defining the group and user filters, click **Test Synchronization Settings**. A test results page is displayed.



Importing LDAP Certificates

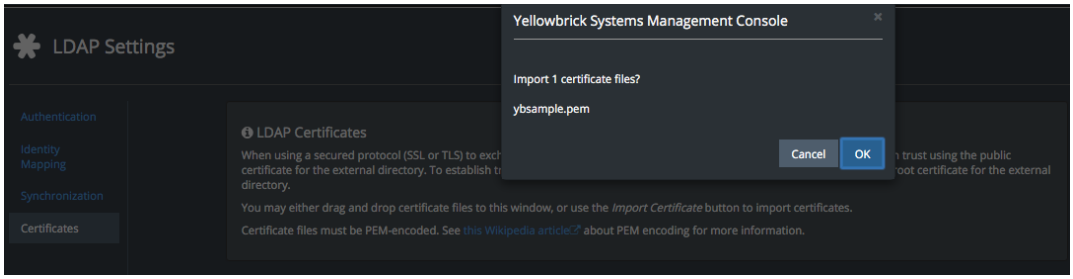
If you are using LDAP over Secure Sockets Layer (SSL) or LDAP with Transport Layer Security (TLS), you must follow these steps before setting up authentication.

When a secured protocol (SSL or TLS) is used to exchange information with an external LDAP directory, Yellowbrick must establish trust using the public certificate for the external directory. To establish trust, import one or more certificates for the external directory or the signing root and intermediate certificates for the external directory. A full certificate chain must be installed; intermediate certificates by themselves are not sufficient to establish trust.

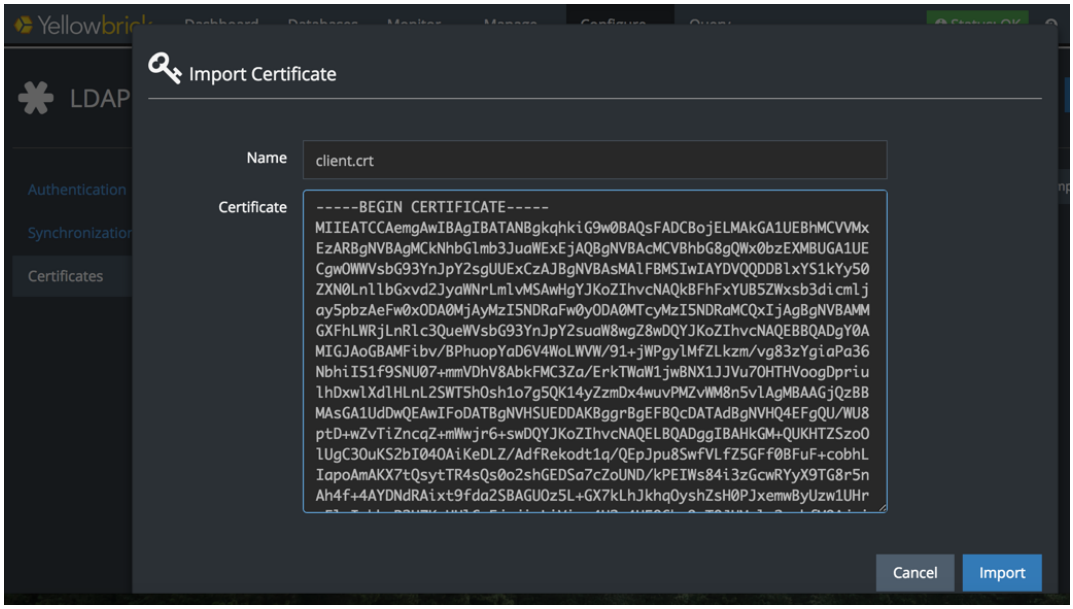
Certificate files must be PEM-encoded.

1. Export the SSL certificate from the LDAP Server.
2. Log in to the SMC and go to **Configure** > **LDAP** > **Certificates**.
3. Import the certificate into the SMC in one of the following ways:

- Drag and drop the file directly to the LDAP certificates screen, then click **OK**. For example:

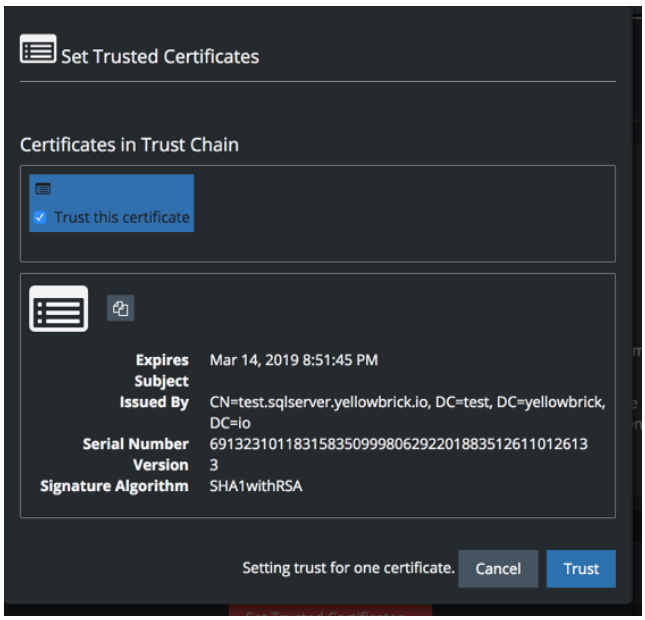


- Click **Import Certificate**, then copy and paste the contents of the certificate. The certificate content must begin with `-----BEGIN CERTIFICATE-----` and end with `-----END CERTIFICATE-----`. For example:



You can also change and delete certificates.

Note: You may be able to bypass this import certificate step and import a trusted certificate when you test LDAP logins. If you have imported a root or intermediate certificate for the authority that issued the LDAP server certificate, but the server certificate itself has not been imported, click **Set Trusted Certificates** on the **Test LDAP Login** screen. For example:



See also Setting Up Authentication.

LDAP Sample Schema

Platforms: EE: All appliance platforms, EE: All cloud platforms

Parent topic: LDAP Integration

The following table represents a portion of a Microsoft Active Directory (AD) instance. This instance contains groups and users that appear throughout the examples in this section.

- This sample schema is for Microsoft Active Directory. In an OpenLDAP server, `UID` would be used instead of `sAMAccountName`
- The user `ad_search` is used for LDAP searches but is not a database user.
- For illustrative purposes, the `CN` and `sAMAccountName` are different. It is common for them to actually be the same.

LDAP Groups

```
Groups OU (OU=Database,OU=SecurityGroups,DC=test,DC=yellowbrick,DC=io)
```

ObjectClass	Member Of	CN	sAMAccountName	Group level
group		yb_all_users	yb_all_users	Top-level group
group	yb_all_users	yb_db_admins	yb_db_admins	Second-level group
group	yb_all_users	yb_db_developers	yb_db_developers	Second-level group
group	yb_all_users	yb_prod_users	yb_prod_users	Second-level group
group	yb_all_users	yb_svc_logins	yb_svc_logins	Second-level group
group	yb_prod_users	db_analyst_role	db_analyst_role	Third-level group
group	yb_prod_users	db_marketing_role	db_marketing_role	Third-level group
group	yb_prod_users	db_sales_role	db_sales_role	Third-level group

LDAP Users

```
Users OU (OU=OrgUsers,DC=test,DC=yellowbrick,DC=io)
```

ObjectClass	Member Of	CN	sAMAccountName	User principal name
InetOrgPerson	db_analyst_role	analyst_1	analyst1	analyst1@test.yellowbrick.io
InetOrgPerson	yb_db_developers	developer_1	developer1	developer1@test.yellowbrick.io
InetOrgPerson	db_marketing_role	marketing_1	marketing1	marketing1@test.yellowbrick.io
InetOrgPerson	db_sales_role	sales_1	sales1	sales1@test.yellowbrick.io
InetOrgPerson	yb_db_admins	user_1_dba	user1-dba	user1-dba@test.yellowbrick.io

ObjectClass	Member Of	CN	sAMAccountName	User principal name
InetOrgPerson	yb_prod_users	elt_user	eltuser	eltuser@test.yellowbrick.io
group	yb_svc_logins	web_user	webuser	webuser@test.yellowbrick.io
user	none	ad_search	ad_search	ad_search@test.yellowbrick.io

Metering

Yellowbrick Documentation > How-to Guides > Administer Yellowbrick > Databases > Metering

Platforms: EE: All cloud platforms

Parent topic: Databases

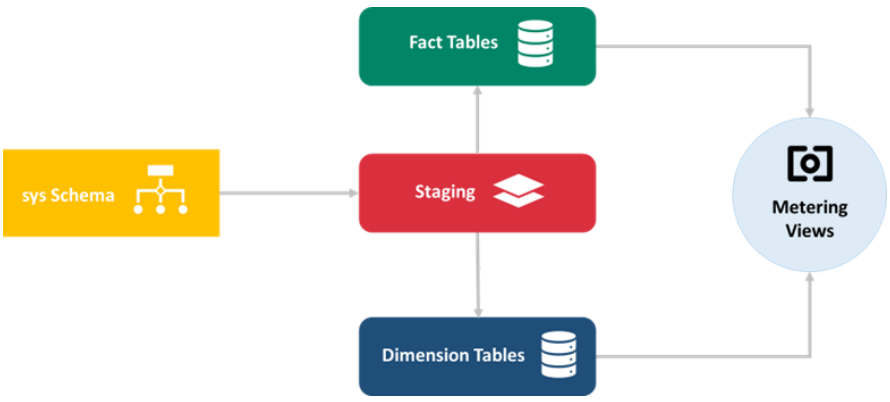
In this section:

Yellowbrick cloud platforms are licensed and billed by quantity of compute virtual CPU (vCPU) consumed by elastic compute clusters. vCPU is consumed when a cluster is resumed, live and able to execute queries. The metering utilities in Yellowbrick allow you to write queries to evaluate who resumed and paused compute clusters, how long clusters have been running, the capacity and vCPU consumption of clusters, and even estimations of vCPU usage by query to provide transparency and help understand consumption.

Metering Overview

Metering is the process that takes transactional data (for example, node count, start and end time of cluster, core count, log queries) from `sys` schema and loads the data into respective views by performing certain operations (for example `vcpu_seconds` calculations) according to business rules. Hourly reports are generated, incorporating the utilization of `vcpu_seconds` based on the views. Metering can be thought of as a traditional ETL process which populates a local data mart with a dimensinoal model for reporting.

Metering data is stored for a continuous year, encompassing the entirety of the present year and the preceeding 12 months (from 1st of January of the prior year). This allows users to predict usage patterns and compare the current year with previous year.



- The data to be analysed from the source (`sys` schema) is extracted into temporary data storage called `staging` .

- The data modelling (denormalising tables, cleaning up bad data, standardising data layouts and so on) is performed on staging to populate the dimension and fact tables.
- In `metering` schema, multiple views are created based on several tables from `fact` and `dim`. These views are created by aggregating several values from the various `fact` and `dim` tables which provide quick information about the consumption and usage of cluster vCPU and so on. This is used for billing the customers.

Terminology

Instance

- An instance is a container of databases, users/roles, all other related data and metadata. It has an IP address, hostname and is the “service” you connect to when you connect to Yellowbrick. Each instance has a special database called `yb_metering` which contains all relevant billing data.

Cluster

- Instances contain compute clusters. The users with the right privileges can create, drop, resume and suspend clusters. Compute clusters are the “work horses” that execute queries on behalf of users.
- All queries executed against a Yellowbrick instance are executed on a compute cluster.

Node

- A cluster contains at least one node, which runs on a hardware instance.

Hardware Instance Type

- The type of machine that a cluster runs on is called a hardware instance. The cluster has a name that's recognised by the cloud (`cloud_provider`) and it also has a name that's recognised by Yellowbrick (`provider_name`). The hardware instance type is recognised by combination of these two fields.

The `hardware_instance_type` is a `sys` schema view that returns a list of available hardware instance types.

Column Name	Data Type	Description	Example
`hardware_instance_type_id`	UUID	UUID for a hardware instance type.	3166615f-fe48-4d82-882b-8300dc963a4e
`cloud_provider`	VARCHAR	Which cloud provider?	azure
`cloud_region`	VARCHAR	Which cloud region is the instance type in?	east-us
`hardware_instance_name`	VARCHAR	Name of the hardware instance.	small-v1
`provider_name`	VARCHAR	Provider name for the instance type.	Standard_L16s_v3
`ram_bytes`	BIGINT	Amount of RAM allocated to this instance type, in bytes.	107374182400
`vcpu_cores`	INTEGER	Number of virtual CPU cores per node.	16
`cpu_cores`	INTEGER	Number of CPU cores per node.	8
`network_access`	VARCHAR		accel_udp
`storage_access`	VARCHAR		accel_nvme
`storage_size_bytes`	BIGINT	Size of data storage, in bytes.	3958241859994
`is_system`	BOOLEAN	Whether this instance type was created by the system.	TRUE

vcpu_cores

- A hardware instance type contains `vcpu_cores`. Depending on cloud provider, the number of `vcpu_cores` may vary. For example,
  - In Azure,
    - A small v1 cluster with one node has 16 `vcpu_cores`.
    - A large v1 cluster with one node has 80 `vcpu_cores`.
  - In AWS,
    - A small v1 cluster with one node has 16 `vcpu_cores`.
    - A large v1 cluster with one node has 96 `vcpu_cores`.

vcpu_seconds

- A vCPU second is one CPU core online, for one second. Fractional seconds consumed are rounded up (for example, consuming 1.5 seconds of 1 CPU core online counts as 2 `vcpu_seconds` consumed). The vCPU usage is unaffected by the type of vCPU you deploy in the cluster.
- `vcpu_seconds` calculation is based on the number of cores in the hardware instance. For example, A query running for 5 seconds on the instance with a small cluster (1 node with 16 cores) should calculate `vcpu_seconds` as 5x1x16 = 80 sec.

vcpu_estimate

- Running queries does not directly consume vCPU. The customers are only billed for the cluster being online, not by the queries that run on that cluster. However, it can be useful to understand what queries are consuming the most CPU so you can properly scale the running clusters and optimise `vcpu_seconds`.

Metering Views

- The `metering` schema views allow customers to query how they will be billed. It also measures which users and clusters consume the most vCPU and when that vCPU was consumed. The views are located in the `yb_metering` sys database.
- metering.vcpu_consumption - contains the `vcpu_seconds` used for billing.
- metering.vcpu_estimate_per_query - contains the `vcpu_estimate` for the queries using the most vCPU on a running cluster.

The `metering` scehma views can be queried directly with SQL or data can be extracted to reporting tools.

Automatic Uploads and Downloads

Platforms: EE: All cloud platforms

Parent topic: Metering

Yellowbrick contains functionality to automatically upload metering data to a central Yellowbrick Metering Service. If you're making use of on-demand billing with Yellowbrick, you'll need to send us your usage metering data periodically so that we can calculate your bill. Alternatively, if you're working in a highly secure network where Yellowbrick is air-gapped from the internet, you can manually download metering data to e-mail to us.

Automatic Uploads

The easiest way to send metering data to Yellowbrick is to allow your Yellowbrick platform to do so automatically. In order to configure this, open Yellowbrick Manager and navigate to *Configuration → Usage Metering*.

Choose *Enable Automatic Usage Metering Upload* to enable automatic uploads. Your Yellowbrick platform will try to upload the metering data hourly. The data volume is typically tiny, measured in kilobytes not megabytes.

You'll need to make sure you allow outbound traffic to `metering.lm.yellowbrick.com` on port 443, otherwise the uploads will fail.

Manual Downloads

If your Yellowbrick installation is in an air-gapped network where it can't access the internet, you can also download metering data and e-mail it to us. The downloads will be larger than the automatic uploads, since we include a longer timeframe, but will still be small enough to e-mail. You can download a day at a time or all historic data.

To download metering data, open Yellowbrick Manager and navigate to *Configuration → Usage Metering*. Choose which timeframe to download, then e-mail `metering@yellowbrick-metering.com` attaching the downloaded archive.

dim.application

Platforms: EE: All cloud platforms

Parent topic: Metering

The `dim.application` consists of applications accessing the system (as in the `application_name` string in `sys.log_query`). Application is likely to be matched to various business units in large systems.

Column Name	Data Type	Description	Example
`application_sk`	BIGINT	A sequencer generated unique ID for a client.	108544
`name`	VARCHAR	Name of the application.	glinda
`valid_from`	TIMESTAMPTZ	The validity start date of the record according to the timestamp at the source system.	2023-12-19T11:05:31.164920Z
`metering_run`	TIMESTAMPTZ	The metering run (one per hour) that created the record.	2023-12-19T11:05:31.164920Z

dim.client

Platforms: EE: All cloud platforms

Parent topic: Metering

The `dim.client` contains one row per client IP address and information about the client that accessed the system. This information is sourced from `sys.log_session`.

Column Name	Data Type	Description	Example
`client_sk`	BIGINT	A sequencer generated unique ID for a client.	1024
`ip`	BIGINT	IP address as in `client_ip_address` in `sys.log_session` that can be accessed via `session_id` from `sys.log_query`.	49.248.77.118
`hostname`	VARCHAR	Host name as in `hostname` in `sys.log_session`.	
`valid_from`	TIMESTAMPTZ	The validity start date of the record according to the timestamp at the source system.	2023-12-19T11:05:31.164920Z
`metering_run`	TIMESTAMPTZ	The metering run (one per hour) that created the record.	2023-12-19T11:05:31.164920Z

dim.cluster

Platforms: EE: All cloud platforms

Parent topic: Metering

The `dim.cluster` gives information about the cluster which is being used for running the queries. It comprises a row for each cluster which is being utilised (using Type 1 dimension schema).

Note: Type 1 Slowly Changing Dimension method overwrites the existing value with the new value and does not retain history.

Column Name	Data Type	Description	Example
`cluster_sk`	BIGINT	A sequencer generated unique ID for each cluster.	25600
`name`	VARCHAR	The cluster name per `cluster_name` in `sys.log_query` and `sys.cluster`.	default_cluster
`cluster_id`	VARCHAR	ID of the cluster, we use varchar instead of UUID as some extractor tools operate better on this.	fb96868a-cbea-422a-99b4-af485e9b43a9
`valid_from`	TIMESTAMPTZ	The validity start date of the record according to the timestamp at the source system.	2023-12-19T11:05:31.164920Z
`metering_run`	TIMESTAMPTZ	The metering run (one per hour) that created the record.	2023-12-19T11:05:31.164920Z

dim.date

Platforms: EE: All cloud platforms

Parent topic: Metering

The `dim.date` is a standard date dimension present in most data warehouses. It allows you to analyze the trends and patterns in your data over a specific timeframe. While computing values with different functions, it's typically better to maintain a precalculated calender.

Column Name	Data Type	Description	Example
`date_sk`	DATE	The actual date. One row for every day from the year 2020 till year 2069.	09-12-2069
`date_start`	DATE	The start of the day to join tables.	09-12-2069
`date_end`	DATE	The end of the day to join tables	10-12-2069
`time_start`	TIME	The time at which the day begins.	00:00:00
`time_end`	TIME	The time at which the day ends.	23:59:59
`year`	INT	Year of the date.	2069
`month`	INT	Month of the date.	12
`day`	INT	Day of the date.	9
`dow`	INT	Day of week, with Sunday being day 1.	2
`week`	INT	The US week number.	50
`iso_dow`	INT	Day of week, with Monday being day 1.	1
`iso_week`	INT	The ISO weekday (Starting monday).	50
`dow_name`	VARCHAR(50)	The name of the day of week.	Monday
`yyyymm`	VARCHAR(7)	Date in the format “YYYY-MM”.	2069-12
`yyyymmdd`	VARCHAR(10)	Date in the format “YYYY-MM-DD”.	09-12-2069
`is_weekend`	BOOLEAN	Is this day a weekend?	False

ISO-8601 (International Organization for Standardization) calendars have a consistent number of weeks in each quarter and a consistent number of days each week. The purpose of the ISO-8601 calendar is to provide a consistent and clear method to represent and calculate dates. ISO-8601 calendars divide dates into years, quarters, weeks, and weekdays. The standard provides a well-defined, unambiguous method of representing calendar dates and times in worldwide communications, especially to avoid misinterpreting numeric dates and times when such data is transferred between countries with different conventions for writing numeric dates and times.

dim.hardware_instance_type

Platforms: EE: All cloud platforms

Parent topic: Metering

The `dim.hardware_instance_type` contains row per hardware instance that has been used in any cluster.

Column Name	Data Type	Description	Example
`hardware_instance_type_sk`	BIGINT	A sequencer generated unique ID for hardware instance type.	7168
`hardware_instance_type_id`	UUID	The UUID of the hardware instance type.	3166615f-fe48-4d82-882b-8300dc963a4e
`name`	VARCHAR	The YB name for the instance.	small-v1
`provider_name`	VARCHAR	The name as used by the cloud providers.	Standard_L16s_v3
`provider`	VARCHAR	Which cloud provider?	azure
`region`	VARCHAR	Which cloud region is the instance type in?	east-us
`valid_from`	TIMESTAMPTZ	The validity start date of the record according to the timestamp at the source system.	2023-12-19T11:05:31.164920Z
`metering_run`	TIMESTAMPTZ	The metering run (one per hour) that created the record.	2023-12-19T11:05:31.164920Z

dim.query

Platforms: EE: All cloud platforms

Parent topic: Metering

The `dim.query` consists of a row for each query associated with the `plan_id` obtained from `sys.log_query`. Before executing a query, the system formulates a plan outlining how to execute it, resulting in the generation of a unique `plan_id`.

Column Name	Data Type	Description	Example
`query_sk`	BIGINT	A sequencer generated ID for a client.	26624
`plan_id`	VARCHAR(64)	Unique ID of the plan as in `plan_id` in `sys.log_query`. This is a hash value in hex format.	lJTTXH5v2BEncBKBSbE2CZOcKwh2pzk9HdXoyAyF9j4=
`query_text`	VARCHAR(10000)	The SQL query text, if necessary has been trimmed to 10,000 chars. If characters exceed the limit of 10,000 characters then entire query text can be found in `sys.log_query`.	ANALYZE HLL test_schema3.table2_1000_col_100000_rows
`valid_from`	TIMESTAMPTZ	The validity start date of the record according to the timestamp at the source system.	2023-12-19T11:05:31.164920Z
`metering_run`	TIMESTAMPTZ	The metering run (one per hour) that created the record.	2023-12-19T11:05:31.164920Z

dim.user

Platforms: EE: All cloud platforms

Parent topic: Metering

The `dim.user` contains one row per user identity using a Type 1 dimension schema.

- Upon user creation, a record is inserted into this table.
- Upon user deletion, a corresponding entry is removed from the table.

A user can be deleted, and a record with the same username may reappear later. Presently, it's challenging to discern that it's the same user.

Column Name	Data Type	Description	Example
`sk_user`	BIGINT	A sequencer generated unique ID for a cluster.	16384
`username`	VARCHAR	The name of user who has been provided access to the instance.	testuser@yellowbrick.com
`valid_from`	TIMESTAMPTZ	The validity start date of the record according to the timestamp at the source system.	2023-12-19T11:05:31.164920Z
`metering_run`	TIMESTAMPTZ	The metering run (one per hour) that created the record.	2023-12-19T11:05:31.164920Z

fact.instance_metering

Platforms: EE: All cloud platforms

Parent topic: Metering

The `fact.instance_metering` contains all the information for instance level metering. The data is stored as intervals where events are valid (examples given below). The intervals get updated on every metering run to the latest load date.

A few examples of valid events for any cluster:

1. Created > Resumed > Suspended
2. Created > Resumed > Dropped
3. Altered > Resumed > Suspended
4. Altered > Resumed > Dropped

Column Name	Data Type	Description	Example
`metering_run`	TIMESTAMPTZ	Assigned on each load of the table from the `curr_load_ts`.	2023-12-21T05:50:12.650643Z
`datetime_start`	TIMESTAMPTZ	The date time when the cluster started. One row is created for each cluster even if the event crosses multiple days.	2023-12-21T05:31:47Z
`datetime_end`	TIMESTAMPTZ	The end time when the cluster is dropped/ suspended/ altered.	2023-12-21T05:50:12Z
`user_sk`	BIGINT NOT NULL	A sequencer generated unique ID for each user. Foreign key to `dim.user`	21504
`cluster_sk`	BIGINT NOT NULL	A sequencer generated unique ID for each cluster. Foreign key to `dim.cluster` pointing to the actual config of the cluster.	25600
`hardware_instance_sk`	BIGINT NOT NULL	A sequencer generated unique ID for each hardware instance. Foreign key to `dim.hardware_instance` tracking which hardware instance, the cluster is running on.	7168
`vcpu_cores`	BIGINT NOT NULL	The number of vCPU provided for each node of the cluster based on hardware instance type.	16
`node_count`	BIGINT	Number of nodes for the cluster as requested by the user. The user can change the number of nodes by altering the cluster.	1
`vcpu_seconds`	BIGINT	Total number of `vcpu_seconds` consumed by the cluster.	17680

fact.top_query

Platforms: EE: All cloud platforms

Parent topic: Metering

The `fact.top_query` allows a user to query for top 100 queries that ran across multiple billing runs and see the queries in each of them.

The query date/time columns are taken from `submit_time` in `sys.log_query` if

- the query has completed running when the metering runs.
- the submit time is larger than the last metering date.

However, if the query ran for more than an hour (over multiple extraction times), then the date/time of the `metering_run` will be considered.

Column Name	Data Type	Description	Example
`metering_run`	TIMESTAMPTZ	The date time at which metering begins.	2023-12-21T05:50:12.650643Z
`datetime_start`	TIMESTAMPTZ	The `submit_time` of the query. Mapped via `sys.log_query`.	22528
`query_sk`	BIGINT	A sequencer generated unique ID for each query. Foreign key to `dim.query`.	153600
`user_sk`	BIGINT	A sequencer generated unique ID for each user. Foreign key to `dim.user`. Mapped via `username` column in `sys.log_query`.	22528
`cluster_sk`	BIGINT	A sequencer generated unique ID for each cluster. Foreign key to `sys.dim_cluster` pointing to the actual config of the cluster. Mapped to the surrogate key via the newly introduced `cluster_id` column in `sys.log_query`.	25600
`client_sk`	BIGINT	A sequencer generated unique ID for each client. Foreign key to `sys.dim_client`. Mapped via `client_ip` which is found via multiple joins to `sys.session` and `session_id` in `sys.log_query`	1024
`application_sk`	BIGINT	A sequencer generated unique ID for each application. Foreign key to `sys.dim_application`. Mapped via `application_name` in `sys.log_query`.	108544
`vcpu_estimate_seconds`	BIGINT	The total `vcpu_seconds` consumed while executing the query.	306112
`query_id`	BIGINT	The query id according to the entry logged in `sys.log_query` or `sys.query`.	4707296

Metering Use Cases

Platforms: EE: All cloud platforms

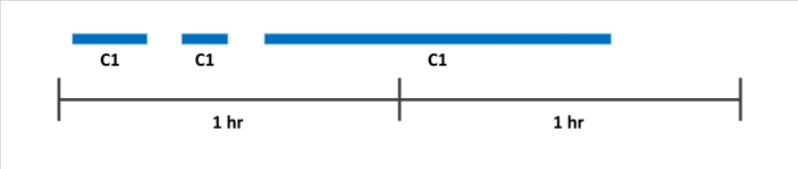
Parent topic: Metering

The metering scenarios within this file serve as a reference.

Scenario 1: Single Cluster Running

Description:

The cluster **C1** undergoes multiple resumptions and suspensions within a one-hour timeframe, persisting in the running state even beyond the one-hour period.



Preconditions:

- Valid user credentials.
- Operational system.

Sample Query:

```sql
SELECT metering_hour, SUM(vcpu_seconds) AS vcpu
FROM metering.vcpu_consumption
GROUP BY 1
```

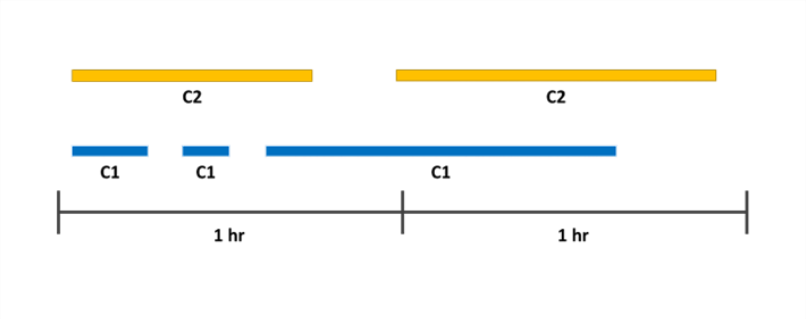
Expected Outcome:

The bill gets split into two segments, one for the initial hour and another for subsequent hour. All metering is aggregated to the hour level.

Scenario 2: Multiple Clusters Running Simultaneously

Description:

Multiple clusters **C1** and **C2** are running simultaneously.



Preconditions:

- Valid user credentials.
- Operational system.

Sample Query:

```sql
SELECT metering_hour, cluster_name, SUM(vcpu_seconds) AS vcpu
FROM metering.vcpu_consumption
GROUP BY 1,2
```

Expected Outcome:

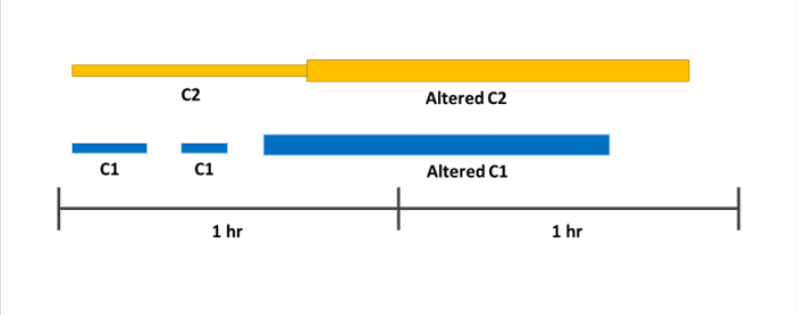
The bill for both **C1** and **C2** clusters is calculated by aggregating the runtime of both clusters (**C1**+**C2**) for the inital hour and for subsequent hour (**C2**+part of **C1** overlapping in second hour).

You can break down the bill into each cluster by aggregating on the `cluster_name` column.

Scenario 3: Altered Clusters

Description:

The cluster can be altered while it is suspended or also while it is running. Cluster **C1** gets altered when suspended however **C2** gets altered while running. The alteration changes the instance type



Preconditions:

- User permissions to alter the cluster.
- Valid user credentials.
- Operational system.

Sample Query:

```sql
SELECT metering_hour, cluster_name, yb_instance_name, SUM(vcpu_seconds) AS vcpu
FROM metering.vcpu_consumption
GROUP BY 1, 2, 3
```

Expected Outcome:

The bill generated is different for the cluster before and after alteration, but they get aggregated in every hour cycle. The bill is also broken down by the instance used

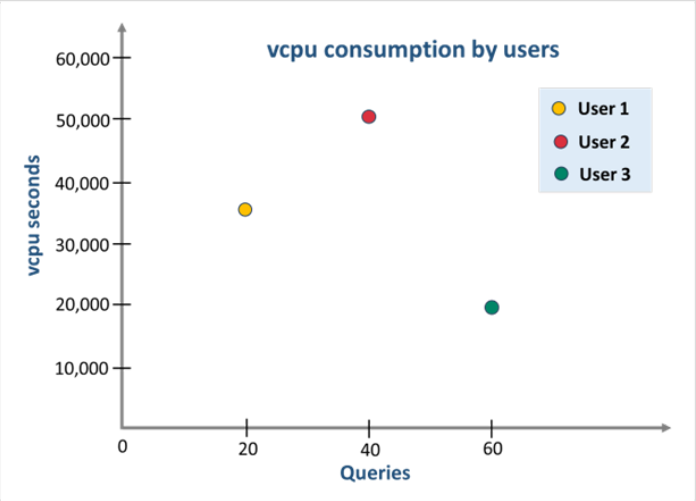
---

Scenario 4: Identify Users Utilising Queries With The Highest "vcpu Consumption"

Description:

For billing, the information about the top users that are consuming the most `vcpu_estimate_seconds` can be obtained using a query. Also additional information such as number of queries each user has executed, cluster name and cluster id for each of the users can be seen.

The `vcpu_estimate_seconds` consumed is not directly proportional to the number of queries executed by the user. It may vary based on the complexity of the query. For example, in the graph below, **User 2** ran fewer queries than **User 3**, but the vCPU consumption for **User 2** is higher than that for **User 3**.



Preconditions:

- Valid user credentials.
- Operational system.

Sample Query:

```sql
SELECT username
  , count(query_id) AS total_queries
  , sum(vcpu_estimate_seconds) AS total_vcpu_seconds
FROM metering.vcpu_estimate_per_query
GROUP BY 1
ORDER BY 3 DESC;
```

Expected Outcome:

The names of users gets displayed in the descending order based on the consumption of `vcpu_estimate_seconds` from `metering.vcpu_estimate_per_query` view.

Note: The top 100 queries based on maximum execution time (`run_ms`) are available.

metering.vcpu_consumption

Platforms: EE: All cloud platforms

Parent topic: Metering

The `metering.vcpu_consumption` view contains information about how much vCPU has been consumed by the running clusters.

The amount of vCPU consumed every second by an instance is determined by:

- The number of clusters in that instance.
- The number of nodes in each cluster.
- The number of actual cpu cores that the hardware instance type exposes to each node.

When the user executes any query, vCPU is consumed only when the cluster is resumed. Active resuming turns cluster into a running cluster. There are two events that will cause the consumption to stop counting:

1. Resume and Suspend: In this case the cluster stays around.
2. Resume and dropped: In this case the cluster is deleted.

This view contains information about how much vCPU has been consumed by the running clusters.

Column Name	Data Type	Description	Example
`metering_hour`	TIMESTAMPTZ	The hour of the metering event. All metering is aggregated to the hour level.	2023-12-19T08:00:00Z
`interval_start`	TIMESTAMPTZ	The start of the hour in `metering_hour`.	2023-12-19T08:00:00Z
`interval_end`	TIMESTAMPTZ	The end of the hour in `metering_hour`.	2023-12-19T09:00:00Z
`username`	VARCHAR	The user who caused the metering event. Either by starting or altering a cluster.	testuser@yellowbrick.com
`cluster_name`	VARCHAR	Name of the cluster that is running at this `metering_hour`.	default_cluster
`cluster_id`	UUID	The unique ID number of the cluster that was running at this `metering_hour`.	fb96868a-cbea-422a-99b4-af485e9b43a9
`yb_instance_name`	VARCHAR	The Yellowbrick name of the cloud instance that the cluster was created with.	small-v1
`region`	VARCHAR	Cloud region where the cluster is located.	east-us
`cloud_provider`	VARCHAR	The cloud provider that hosts the cluster.	azure
`cloud_provider_instance_name`	VARCHAR	The cloud providers name of the instance used to create the cluster.	Standard_L16s_v3
`vcpu_seconds`	INTEGER	Number of `vcpu_seconds` consumed in the `metering_hour`.	55200

metering.vcpu_estimate_per_query

Platforms: EE: All cloud platforms

Parent topic: Metering

The `metering.vcpu_estimate_per_query` view contains information about the queries that use the most vCPU on the system. On every metering execution, the top 100 queries based on maximum execution time (`run_ms`) will get loaded and stored in this view.

> The `vcpu_estimate_seconds` is calculated by using the formula:
>
> `vcpu_estimate_seconds` = (`run_ms` – `wait_run_cpu_ms`) x No. of workers x cores per compute node
>
> - `run_ms` : Time required for query to get executed. It tells us how long the query was running on the longest-running compute node. This value represents the total time the query spent in the run state.
> - `wait_run_cpu_ms` : Duration needed for processing the query.
>
> **Note:** The value (`run_ms` – `wait_run_cpu_ms`) is rounded up.
>
> The `run_ms` value is **NULL** before the query enters the `run` state, then it is initialized to **0**.

This view can be used to understand if the cluster capacity is properly utilised.

Column Name	Data Type	Description	Example
`datetime_start`	TIMESTAMPTZ	The time the query submitted. Maps to `submit_time` in sys.log_query.	2023-12-19T06:56:56.842390Z
`query_id`	BIGINT	The id of the query that ran. Maps to `query_id` of sys.log_query.	247202
`query_text`	VARCHAR	The first 10000 characters of the query text that ran. This is truncated to keep the table small.	ANALYZE HLL test_schema3.table2_1000_col_100000_rows
`username`	VARCHAR	The user running the query.	testuser@yellowbrick.com
`cluster_name`	VARCHAR	Name of the cluster the query ran on.	default_cluster
`cluster_id`	UUID	Id of the cluster the query ran on.	fb96868a-cbea-422a-99b4-af485e9b43a9
`ip`	VARCHAR	IP address of the user running the query.	49.248.77.119
`hostname`	VARCHAR	Hostname (if available) of the user running the query.	
`application_name`	VARCHAR	Application name, as set by `SET application_name` for the application that ran the query.	ym
`vcpu_estimate_seconds`	INTEGER	Estimated number of `vcpu_seconds` the query needed on the cluster to run. This value takes into account concurrency and only measures the time the query actually used the scheduler.	17968

System Views

Platforms: All platforms

Parent topic: Databases

In this section:

sys.query_analyze
sys.query_explain
sys.query_rule_event
sys.remote_server
sys.replica
sys.replica_status
sys.restore
sys.retention
sys.role
sys.schema
sys.sequence
sys.session
sys.storage
sys.table
sys.table_flush_summary
sys.table_info
sys.table_partition
sys.table_storage
sys.unload
sys.user
sys.view
sys.wlm_active_pool
sys.wlm_active_profile
sys.wlm_active_rule
sys.wlm_pending_pool
sys.wlm_pending_rule
sys.worker

This section is a reference for a set of system views that capture information about resource utilization, inflight queries, active loads, and other operational statistics. You can query these views via the Query Editor in Yellowbrick Manager, `ybsql`, or other SQL query tools.

Note: The `information_schema` tables and views are provided only for compatibility with Postgres-compliant tools. For database administration, Yellowbrick recommends using the views in the `sys` schema, as documented in this section. These views provide significant Yellowbrick-specific information that is not available under `information_schema`.

The system views belong to the `sys` schema. For example, in `ybsql` run this command:

```
premdb=# set schema 'sys';
SET
```

Then list the views that belong to the schema:

```
\dvS
                List of relations
 Schema |            Name             | Type | Owner
--------+-----------------------------+------+--------
 sys    | alert                       | view | ybuser
 sys    | backup                      | view | ybuser
 sys    | backup_chain                | view | ybuser
 sys    | backup_snapshot             | view | ybuser
 sys    | blade                       | view | ybuser
 sys    | cluster                     | view | ybuser
 sys    | cluster_node_status         | view | ybuser
 sys    | cluster_usage               | view | ybuser
 sys    | column                      | view | ybuser
...
```

Alternatively, you can specify the schema in the query itself. For example:

```
premdb=# select * from sys.log_query;
...
```

No one can create and modify tables in the `sys` schema.

```
premdb=# create table table2(c1 int, c2 int) distribute on(c1) sort on(c2);
ERROR:  permission denied to create "sys.table2"
DETAIL:  System catalog modifications are currently disallowed.
```

You cannot combine user-defined tables and system tables or views in the same query. One exception to this rule is the sys.const table, which can be queried in conjunction with regular tables.

Retention policy

A log retention policy determines the amount of history that is retained in the following `sys.log_*` views:

- sys.log_analyze
- sys.log_backup
- sys.log_authentication
- sys.log_cluster_event
- sys.log_load
- sys.log_query
- sys.log_query_analyze
- sys.log_replica_status
- sys.log_restore
- sys.log_retention
- sys.log_session
- sys.log_unload

For more details, see Retention policies for system relations.

Visibility of Data in System Views

Note that all users can see the list of system views:

```
premdb=# set schema 'sys';
SET
premdb=# \dS
              List of relations
 Schema |         Name         | Type |  Owner
--------+----------------------+------+----------
 sys    | alert                | view | ybdadmin
 sys    | backup               | view | ybdadmin
...
```

However, regular users may not have access to the data they return. Superusers can see all rows for all databases. The visibility of system views for non-superusers depends on the privileges those users have been granted.

To grant access to the data in system views, use the GRANT command. For example, you can grant `VIEW QUERY TEXT` or `TRACE QUERY` privileges so that non-superusers can see rows for queries that they did not run themselves:

```
premdb=# create user tracer1 password '******';
CREATE ROLE
premdb=# grant trace query on system to tracer1;
GRANT
```

User `tracer1` can now see all queries that are running or have been run on the *current database* but has visibility into all databases on the system. The user would have to connect to different databases to see the queries for each database.

The following table defines the visibility of different system views for regular users (non-superusers):

System View(s)	Regular Users Can See:
`sys.query`, `sys.log_query`, `sys.query_explain`, `sys.query_analyze`	- All queries on the current database if they have `TRACE QUERY` privilege either `ON SYSTEM` or on the current database (for example, `ON DATABASE devdb`). - Their own queries on the current database if they do not have one of these privileges.
`sys.session`, `sys.log_session`	- All sessions on the current database if they have `TRACE QUERY ON SYSTEM`, `TRACE QUERY` on the current database, or `CONTROL ANY SESSION ON SYSTEM` privileges. - Their own sessions on the current database if they do not have one of these privileges.
`sys.database`	Databases on which they have one or more of the following privileges: - Any privilege on the database (note that `CONNECT` is a default privilege on all databases when users are created) - `CREATE DATABASE ON SYSTEM` - `ALTER ANY DATABASE ON SYSTEM` - `DROP ANY DATABASE ON SYSTEM`
`sys.schema`	Schemas in the current database if they have one or more of the following privileges: - `USAGE ON SCHEMA` - `ALTER ANY SCHEMA ON DATABASE` - `DROP ANY SCHEMA ON DATABASE`
`sys.table`	Their own tables and any tables they have been granted access to in the current database. Users who have `SELECT` privilege on any column of a table will be able to see that table in any view that represents the table's metadata (such as `sys.column` or `sys.table_info`).
`sys.role`, `sys.user`	- All roles in the system if they have one or more of the following privileges: - `ALTER ANY ROLE ON SYSTEM` - `DROP ANY ROLE ON SYSTEM` - `CREATE ROLE ON SYSTEM` - Specific roles that they have been given any privilege to see via `GRANT...ON ROLE`. - Their own roles only if they do not have one of these privileges.

sys.lock

Platforms: All platforms

Parent topic: System Views

In this section:

Types of Locks

This system view shows the current locks on all tables in the database. You can use this view to find lock conflicts on tables.

Column Name	Data Type	Description
table_id	bigint	Unique ID of the table.
database_id	bigint	Unique ID of the database.
object_type	text(9)	Always `TABLE`.
session_id	bigint	Unique database session ID.
is_granted	boolean	Whether the requested lock was granted.
lock_type	text	Type of lock requested.
blocked_by_session_id	text	Session ID that is blocking the requested lock (`NULL` if not blocked).

Note: For more information on the lock types and conflicting lock modes, Click Here.

Examples

```
yellowbrick_test=# select * from sys.lock
order by table_id, database_id;
 table_id  | database_id | object_type | session_id | is_granted |        lock_type         | blocked_by_session_id
-----------+-------------+-------------+------------+------------+--------------------------+----------------------
      1247 |       17106 | TABLE       |      39525 | t          | AccessShareLock          |
      1259 |       17106 | TABLE       |      39525 | t          | AccessShareLock          |
      1260 |           0 | TABLE       |      16175 | t          | AccessShareLock          |
      1262 |           0 | TABLE       |      16175 | t          | AccessShareLock          |
      2615 |       17106 | TABLE       |      39525 | t          | AccessShareLock          |
      2662 |       17106 | TABLE       |      39525 | t          | AccessShareLock          |
      2663 |       17106 | TABLE       |      39525 | t          | AccessShareLock          |
      2671 |           0 | TABLE       |      16175 | t          | AccessShareLock          |
      2672 |           0 | TABLE       |      16175 | t          | AccessShareLock          |
      2676 |           0 | TABLE       |      16175 | t          | AccessShareLock          |
      2677 |           0 | TABLE       |      16175 | t          | AccessShareLock          |
      2684 |       17106 | TABLE       |      39525 | t          | AccessShareLock          |
      2685 |       17106 | TABLE       |      39525 | t          | AccessShareLock          |
      2703 |       17106 | TABLE       |      39525 | t          | AccessShareLock          |
      2704 |       17106 | TABLE       |      39525 | t          | AccessShareLock          |
      2840 |       17106 | TABLE       |       4867 | t          | ShareUpdateExclusiveLock |
      2841 |       17106 | TABLE       |       4867 | t          | RowExclusiveLock         |
      2964 |           0 | TABLE       |      16175 | t          | AccessShareLock          |
      2965 |           0 | TABLE       |      16175 | t          | AccessShareLock          |
      3455 |       17106 | TABLE       |      39525 | t          | AccessShareLock          |
      5003 |           0 | TABLE       |      35946 | t          | RowExclusiveLock         |
```

```
     5003 |           0 | TABLE       |      35946 | t          | AccessShareLock          |
...
```

```
yellowbrick_test=# select * from sys.lock
where blocked_by_session_id is not null;
 table_id | database_id | object_type | session_id | is_granted |    lock_type    | blocked_by_session_id
----------+-------------+-------------+------------+------------+-----------------+----------------------
  1812902 |       17106 | TABLE       |      28200 | f          | AccessShareLock |                28264
   17137 |       17106 | TABLE       |      28770 | f          | ExclusiveLock   |                35873
   17137 |       17106 | TABLE       |      28770 | f          | ExclusiveLock   |                35873
(3 rows)
```

```
yellowbrick_test=# select sct.table_id, sct.name, ssl.lock_type
from sys.table sct join sys.lock ssl on sct.table_id=ssl.table_id and sct.database_id=ssl.database_id;
 table_id |                 name                 |    lock_type
----------+--------------------------------------+-----------------
   56889 | ThirdPartyDataGeneralPopulationOverlap | AccessShareLock
   17395 | TargetingDataUniqueThirdPartyUsers     | AccessShareLock
   17395 | TargetingDataUniqueThirdPartyUsers     | ExclusiveLock
   17386 | TargetingData                          | AccessShareLock
   56865 | LookAlikeModelResult                   | AccessShareLock
   56865 | LookAlikeModelResult                   | ExclusiveLock
...
```

Types of Locks

Platforms: All platforms

Parent topic: sys.lock

Yellowbrick provides various lock modes to control concurrent access to data in tables. The list of **Table Level Locks** is given below:

Table Level Locks

Access Share (`AccessShareLock`)

The `SELECT` command acquires a lock of this mode on referenced tables. In general, any query that only reads a table and does not modify it will acquire this lock mode.

Share Update Exclusive (`ShareUpdateExclusiveLock`)

Acquired by `VACUUM` (without `FULL`), `ANALYZE` , `UPDATE` and `ALTER TABLE` variants. This mode protects a table against concurrent schema changes and `VACUUM` runs.

Access Exclusive (`AccessExclusiveLock`)

Acquired by the `DROP TABLE` , `TRUNCATE` . `ALTER TABLE` also acquire a lock at this level. This mode guarantees that the holder is the only transaction accessing the table in any way.

Row Exclusive (`RowExclusiveLock`)

The commands `UPDATE` , `DELETE` and `INSERT` acquire this lock mode on the target table. In general, this lock mode will be acquired by any command that modifies data in a table.

Conflicting Table Level Lock Modes

	Existing Lock Mode			
Requested Lock Mode	Access Share	Share Update Exclusive	Access Exclusive	Row Exclusive
Access Share			x	
Share Update Exclusive		x	x	
Access Exclusive	x	x	x	
Row Exclusive			x	

Retention Policies for System Relations

Platforms: All platforms

Parent topic: System Views

Some system relations can grow in space with time, because of the number of queries being run, sessions being opened, etc. We provide a retention mechanism for some of these relations:

- Removing all records older than a given age.
- Removing the oldest records when the relation takes more than a given amount of space in the storage.

Interface

The current retention configuration can be observed in sys.retention.

The retention history is stored in sys.log_retention.

To modify the retention configuration for a given relation, an `ALTER TABLE` statement needs to be run, for example:

```sql
ALTER TABLE <relation> SET RETENTION_SIZE <size>
```

See ALTER TABLE for the complete SQL syntax.

Notes

- The retention process will trigger every hour.
- When removing records because of space limit, no record younger than a day will be removed.
- The retention policy configuration does not survive upgrades and will need to be re-applied.

Default configuration

By default, the following retention policies are set up:

Relation	Age Retention	Size Retention	Minimum Age Retention	Minimum Size Retention
sys.log_analyze	3 months	1GB	1 day	128MB
sys.log_authentication	Disabled	Disabled	1 day	128MB
sys.log_backup	3 months	1GB	1 day	128MB
sys.log_cluster_event	3 months	1GB	1 day	128MB
sys.log_load	3 months	1GB	1 day	128MB
sys.log_query	1 month	32GB	1 day	1GB
sys.log_query_analyze	1 month	32GB	1 day	128MB

Relation	Age Retention	Size Retention	Minimum Age Retention	Minimum Size Retention
sys.log_query_usage	1 month	32GB	1 day	128MB
sys.log_restore	3 months	1GB	1 day	128MB
sys.log_replica_status	3 months	1GB	1 day	128MB
sys.log_retention	3 months	1GB	1 day	128MB
sys.log_session	Disabled	Disabled	1 day	128MB
sys.log_unload	3 months	1GB	1 day	128MB

sys.alert

Platforms: EE: All appliance platforms

Parent topic: System Views

This system view returns information about active alerts on the appliance. Alerts with a `CLOSED` status do not appear in this view. See also sys.log_alert, which maintains an alert history.

Column Name	Data Type	Description
alert_id	text	A unique UUID string that identifies the alert.
alert_name	text	Name of the alert, such as `Test Alert` or `Database State`.
open_time	timestamptz	When the alert was opened (first issued).
close_time	timestamptz	When the alert was closed, which depends on the type of alert. Alerts are closed when their state changes. For example, Database State alerts close when the database comes back `ONLINE`. Some alerts, such as WLM Rule alerts, are issued then closed immediately (they do not change state).
last_update_time	timestamptz	When the alert was last updated.
duration_ms	numeric(18,3)	Length of time, in milliseconds, from when the alert was opened to when it was closed (if closed).
resource	text	The location or source of the alert, such as a drive, a power supply, or a WLM rule.
severity	text	Severity of the alert: `INFORMATIONAL`, `MINOR`, `MAJOR`, `CRITICAL`
status	text	Status of the alert: `OPEN`, `CLOSED`
state	text	State of the resource for the alert, such as `offline`, `error`, `degraded`, `stopped`.
message	text	Warning or error message associated with the alert.
acknowledged	boolean	Whether the alert has been acknowledged in the SMC (via **Manage Cluster Alerts** or **Manage Query Alerts**).
acknowledged_time	timestamptz	When the alert was acknowledged in the SMC.
acknowledged_by	text	The user who acknowledged the alert in the SMC.
acknowledged_text	text	Text associated with the acknowledgment, if any.

Example

For example:

```
yellowbrick_test=# select * from sys.alert;
-[ RECORD 1 ]-----+-------------------------------------
alert_id          | 4ed3eada-9665-4180-996c-9c7cf684f51c
open_time         | 2018-07-04 01:02:59-07
close_time        | [NULL]
last_update_time  | 2018-07-10 19:17:47-07
duration_ms       | 584695178.636
resource          | CLUSTER_MANAGER|Cluster Manager|status
```

```
severity          | MAJOR
status            | OPEN
state             | stopped
message           | status is in stopped
acknowledged      | t
acknowledged_time | 2018-07-10 19:17:47.896-07
acknowledged_by   | yellowbrick
acknowledged_text |
```

sys.backup

Platforms: All platforms

Parent topic: System Views

This system view captures information about active backup (`ybbackup`) operations. The columns in this view are the same as the columns in sys.log_backup.

Example

In this example, the `ybsql \watch` command tracks the progress of a running backup operation and returns results every 5 seconds.

```
premdb=# select * from sys.backup;
(0 rows)

premdb=# \watch 5
(0 rows)

(0 rows)

Watch every 5s  Tue Feb 18 15:07:51 2020

-[ RECORD 1 ]---------+------------------------------------------------------------------
session_key           | DzTlIEcb4E1EYW0BfHbo1mzMj9vuPskwWriWw3JowgAVscEKXVHO3y0gx5KG83a5
session_id            | 607424
client_hostname       | localhost
client_username       | yb100
user_name             | yb100
user_id               | 16418
database_name         | premdb
database_id           | 16395
state                 | USER_DATA_RUNNING
error_string          | [NULL]
start_time            | 2020-02-18 15:07:47-08
end_time              | [NULL]
elapsed_ms            | 4739
remaining_ms          | 28000
sent_bytes            | 275221394
sent_bytes_per_sec    | 58075836
storage_bytes         | 275312522
storage_bytes_per_sec | 58095068
backup_point_id       | Feb18Full
chain_id              | February2020
type                  | F

Watch every 5s  Tue Feb 18 15:07:56 2020

-[ RECORD 1 ]---------+------------------------------------------------------------------
session_key           | DzTlIEcb4E1EYW0BfHbo1mzMj9vuPskwWriWw3JowgAVscEKXVHO3y0gx5KG83a5
session_id            | 607424
client_hostname       | localhost
client_username       | yb100
user_name             | yb100
user_id               | 16418
database_name         | premdb
database_id           | 16395
state                 | USER_DATA_RUNNING
error_string          | [NULL]
```

```
start_time            | 2020-02-18 15:07:47-08
end_time              | [NULL]
elapsed_ms            | 9754
remaining_ms          | 10000
sent_bytes            | 837437948
sent_bytes_per_sec    | 85855848
storage_bytes         | 837715148
storage_bytes_per_sec | 85884264
backup_point_id       | Feb18Full
chain_id              | February2020
type                  | F

Watch every 5s  Tue Feb 18 15:08:02 2020

-[ RECORD 1 ]---------+------------------------------------------------------------------
session_key           | DzTlIEcb4E1EYW0BfHbo1mzMj9vuPskwWriWw3JowgAVscEKXVHO3y0gx5KG83a5
session_id            | 607424
client_hostname       | localhost
client_username       | yb100
user_name             | yb100
user_id               | 16418
database_name         | premdb
database_id           | 16395
state                 | USER_DATA_RUNNING
error_string          | [NULL]
start_time            | 2020-02-18 15:07:47-08
end_time              | [NULL]
elapsed_ms            | 14767
remaining_ms          | 12000
sent_bytes            | 1398798687
sent_bytes_per_sec    | 94724632
storage_bytes         | 1399260807
storage_bytes_per_sec | 94755928
backup_point_id       | Feb18Full
chain_id              | February2020
type                  | F

Watch every 5s  Tue Feb 18 15:08:07 2020

-[ RECORD 1 ]---------+------------------------------------------------------------------
session_key           | DzTlIEcb4E1EYW0BfHbo1mzMj9vuPskwWriWw3JowgAVscEKXVHO3y0gx5KG83a5
session_id            | 607424
client_hostname       | localhost
client_username       | yb100
user_name             | yb100
user_id               | 16418
database_name         | premdb
database_id           | 16395
state                 | USER_DATA_RUNNING
error_string          | [NULL]
start_time            | 2020-02-18 15:07:47-08
end_time              | [NULL]
elapsed_ms            | 19829
remaining_ms          | 25000
sent_bytes            | 1746924788
sent_bytes_per_sec    | 88099488
storage_bytes         | 1747502012
storage_bytes_per_sec | 88128600
backup_point_id       | Feb18Full
chain_id              | February2020
type                  | F

Watch every 5s  Tue Feb 18 15:08:12 2020

-[ RECORD 1 ]---------+------------------------------------------------------------------
session_key           | DzTlIEcb4E1EYW0BfHbo1mzMj9vuPskwWriWw3JowgAVscEKXVHO3y0gx5KG83a5
session_id            | 607424
client_hostname       | localhost
```

```
client_username       | yb100
user_name             | yb100
user_id               | 16418
database_name         | premdb
database_id           | 16395
state                 | DONE
error_string          | [NULL]
start_time            | 2020-02-18 15:07:47-08
end_time              | 2020-02-18 15:08:08-08
elapsed_ms            | 21014
remaining_ms          | 0
sent_bytes            | 1806698912
sent_bytes_per_sec    | 85975968
storage_bytes         | 1807292528
storage_bytes_per_sec | 86004216
backup_point_id       | Feb18Full
chain_id              | February2020
type                  | F
...
```

sys.backup_chain

Platforms: All platforms

Parent topic: System Views

This system view returns a list of all backup chains in the current database.

Column Name	Data Type	Description
database_id	bigint	The unique ID for the database.
chain_name	text	Name of the backup chain. This view contains records for backup chains associated with database replicas and/or defined with `ybbackup` operations. In the replication case, the default chain name is the same as the replica name. For backups, the chain name is as specified in the `ybbackup` command.
policy	text	Empty JSON string `{}`, except for backup chains that exclude schemas (`ybbackup --exclude` option). When schemas are excluded, this column contains a string of the following form: `{"excludedSchemas": ["schema1","schema2"]}`
oldest_backup_point_id	text	Unique ID for the oldest backup point for this database. (Without this backup snapshot, the backup chain cannot function.)
oldest_rollback_point_id	text	Unique ID for the oldest backup point that this database can be rolled back to with the ROLLBACK DATABASE command.
inprogress_backup_point_id	text	Unique ID for a backup that is in progress, if any.
creation_time	timestamptz	When the backup chain was created.

For example, here is an example from a system running regular backups:

```
premdb=# select * from sys.backup_chain;
 database_id |  chain_name  | policy | oldest_backup_point_id | oldest_rollback_point_id |         creation_time
-------------+--------------+--------+------------------------+--------------------------+-------------------------------
       16391 | December2019 | {}     | Dec17Full              | [NULL]                   | 2019-12-17 18:03:01.511742-08
(1 row)
```

Here is an example from a system running database replication:

```
premdb=# select * from sys.backup_chain;
 database_id |      chain_name       | policy |         oldest_backup_point_id          |        oldest_rollback_point_id
-------------+-----------------------+--------+-----------------------------------------+-----------------------------------------
       16432 | premdb_local_replica1 | {}     | premdb_local_replica1_20_01_14_00_50_56 | premdb_local_replica1_20_01_14_00_50_56 |
(1 row)
```

The following example shows the chain for a full backup that excludes two schemas:

```
premdb=# select * from sys.backup_chain;
 database_id | chain_name |                  policy                  |    oldest_backup_point_id     | oldest_rollback_point_id |
-------------+------------+------------------------------------------+-------------------------------+--------------------------+-
       16396 | May2022    | {"excludedSchemas":["sys","matchstats"]} | May23FullExcludeSysMatchstats | [NULL]                   | [
(1 row)
```

sys.backup_snapshot

This system view returns a list of backup snapshots in the current database. Backup snapshots in this view derive from both database replication cycles and regular backup operations.

Column Name	Data Type	Description
database_id	bigint	The unique ID for the database.
backup_txid	bigint	The transaction ID for the backup snapshot.
backup_txsnap	text	*This value is for internal use only.*
snapshot_name	text	The name of the backup snapshot. - Snapshots that start with `ybd_` and end with `_epoch` are used by the system internally during a `full` backup. - Backup snapshots that are the results of full, cumulative, or incremental backups taken with the `ybbackup` tool have the names they were given when those backups were run (`--name` option). - Backup snapshots for replication cycles are prefixed with the replica name.
creation_time	timestamptz	When the snapshot was created. Replication snapshots are created at regular intervals, based on the `frequency` set for the replica.

Examples

For example, `Dec17Full` is the result of a full backup:

```
premdb=# select * from sys.backup_snapshot;
 database_id | backup_txid | backup_txsnap |                snapshot_name                |          creation_time
-------------+-------------+---------------+---------------------------------------------+-------------------------------
       16391 |        1260 | 3:3:          | ybd_af97315a-622b-ab81-ad4d-59ac80ec6fc5_epoch | 2019-12-17 18:03:01.502767-08
       16391 |        1262 | 1262:1262:    | Dec17Full                                   | 2019-12-17 18:03:01.511742-08
(2 rows)
```

(The initial snapshot that begins with `ybd_` is an internally generated snapshot.)

The entries in this example that are prefixed with `premdb_local_replica1` are all the result of replication cycles:

```
premdb=# select * from sys.backup_snapshot;
 database_id | backup_txid |  backup_txsnap   |                snapshot_name                |          creation_time
-------------+-------------+------------------+---------------------------------------------+-------------------------------
       16432 |        4119 | 0:3:3:           | ybd_0035c89e-ccec-9fe5-f1f4-59fe687a960f_epoch | 2020-01-13 16:07:56.195664-08
       16432 |        4123 | 0:4123:4123:     | premdb_local_replica1_20_01_14_00_07_56        | 2020-01-13 16:07:56.204702-08
       16432 |        4542 | 0:4542:4542:     | premdb_local_replica1_20_01_14_00_08_56        | 2020-01-13 16:08:56.058381-08
       16432 |        4581 | 0:4581:4581:     | premdb_local_replica1_20_01_14_00_09_56        | 2020-01-13 16:09:56.058157-08
       16432 |        4624 | 0:4624:4624:     | premdb_local_replica1_20_01_14_00_10_56        | 2020-01-13 16:10:56.065316-08
       16432 |        4709 | 0:4709:4709:     | premdb_local_replica1_20_01_14_00_11_56        | 2020-01-13 16:11:56.071239-08
       16432 |        4749 | 0:4749:4749:     | premdb_local_replica1_20_01_14_00_12_56        | 2020-01-13 16:12:56.05596-08
       16432 |        4788 | 0:4788:4788:     | premdb_local_replica1_20_01_14_00_13_56        | 2020-01-13 16:13:56.054972-08
```

```
      16432 |        4827 | 0:4827:4827:     | premdb_local_replica1_20_01_14_00_14_56        | 2020-01-13 16:14:56.064773-08
      16432 |        4866 | 0:4866:4866:     | premdb_local_replica1_20_01_14_00_15_56        | 2020-01-13 16:15:56.063238-08
      16432 |        4926 | 0:4926:4926:     | premdb_local_replica1_20_01_14_00_16_56        | 2020-01-13 16:16:56.038004-08
      16432 |        5043 | 0:5043:5043:     | premdb_local_replica1_20_01_14_00_17_56        | 2020-01-13 16:17:56.056254-08
      16432 |        5222 | 0:5222:5222:     | premdb_local_replica1_20_01_14_00_18_56        | 2020-01-13 16:18:56.040732-08
...
```

The following example joins this view with `sys.database` to return the database name. The query also constrains on snapshots that have the prefix `premrep`, which is a replica with a frequency of 90 seconds:

```
premdb=# select name dbname, snapshot_name, creation_time
from sys.backup_snapshot sb, sys.database sd
where sb.database_id=sd.database_id and snapshot_name like 'premrep%'
order by creation_time;
 dbname |       snapshot_name       |         creation_time
--------+---------------------------+-------------------------------
 premdb | premrep_20_04_01_13_52_13 | 2020-04-01 13:52:13.345599-07
 premdb | premrep_20_04_01_13_53_42 | 2020-04-01 13:53:42.200495-07
 premdb | premrep_20_04_01_13_55_12 | 2020-04-01 13:55:12.204403-07
 premdb | premrep_20_04_01_13_56_42 | 2020-04-01 13:56:42.217729-07
 premdb | premrep_20_04_01_13_58_12 | 2020-04-01 13:58:12.203774-07
 ...
```

Logging of Backup Snapshots

The backup snapshots logged by this view are filtered. Snapshots that are no longer usable as the basis for a subsequent backup are removed. For example, consider a database that is both the target of a sequence of backups and the source of database replication to another system. At this point in the log recorded by the view, one full backup and two incremental backups have been taken in quick succession (snapshots beginning with `April1`). Meanwhile replication is occurring on a regular schedule for this database:

```
premdb=# select * from sys.backup_snapshot order by creation_time;
 database_id | backup_txid |  backup_txsnap   |                 snapshot_name                  |         creation_time
-------------+-------------+------------------+------------------------------------------------+-------------------------------
      16395 |      137080 | 0:3:3:           | ybd_f89f1179-08cc-43f9-a1c2-1ed724d886f1_epoch | 2020-03-31 18:43:12.156275-07
      16395 |      186750 | 0:186750:186750: | April1Full                                     | 2020-04-01 13:19:34.955359-07
      16395 |      186855 | 0:186855:186855: | premrep_20_04_01_13_20_42                      | 2020-04-01 13:20:42.200568-07
      16395 |      189364 | 0:189364:189364: | premrep_20_04_01_13_22_12                      | 2020-04-01 13:22:12.194519-07
      16395 |      189495 | 0:189495:189495: | April1Inc                                      | 2020-04-01 13:23:22.827597-07
      16395 |      189537 | 0:189537:189537: | premrep_20_04_01_13_23_42                      | 2020-04-01 13:23:42.207404-07
      16395 |      189659 | 0:189659:189659: | premrep_20_04_01_13_25_12                      | 2020-04-01 13:25:12.21102-07
      16395 |      190025 | 0:190025:190025: | premrep_20_04_01_13_26_42                      | 2020-04-01 13:26:42.197756-07
      16395 |      190211 | 0:190211:190211: | April1Inc2                                     | 2020-04-01 13:26:54.542625-07
      16395 |      190300 | 0:190300:190300: | premrep_20_04_01_13_28_12                      | 2020-04-01 13:28:12.197437-07
      16395 |      190351 | 0:190351:190351: | premrep_20_04_01_13_29_42                      | 2020-04-01 13:29:42.18738-07
      16395 |      190400 | 0:190400:190400: | premrep_20_04_01_13_31_12                      | 2020-04-01 13:31:12.20491-07
      16395 |      190528 | 0:190528:190528: | premrep_20_04_01_13_32_42                      | 2020-04-01 13:32:42.195353-07
...
```

Next, some changes are made to the database and a new cumulative backup is taken. Now the results of the system view query look like this:

```
premdb=# select * from sys.backup_snapshot order by creation_time;
 database_id | backup_txid |  backup_txsnap   |                 snapshot_name                  |         creation_time
-------------+-------------+------------------+------------------------------------------------+-------------------------------
      16395 |      137080 | 0:3:3:           | ybd_f89f1179-08cc-43f9-a1c2-1ed724d886f1_epoch | 2020-03-31 18:43:12.156275-07
      16395 |      191574 | 0:191574:191574: | April1Cumu                                     | 2020-04-01 13:52:05.523348-07
      16395 |      191621 | 0:191621:191621: | premrep_20_04_01_13_52_13                      | 2020-04-01 13:52:13.345599-07
(3 rows)
```

The new backup, `April1Cumu` resets the "horizon" for backup and restore operations. Snapshots before `April1Cumu` cannot be used as the starting point of a new backup, so the previous three backups are all removed from the history. The same rule applies to the replication cycles that occurred before the cumulative backup. All of those snapshots are obsolete as well.

A few minutes later the query is run again:

```
premdb=# select * from sys.backup_snapshot order by creation_time;
 database_id | backup_txid |  backup_txsnap   |                snapshot_name                 |         creation_time
-------------+-------------+------------------+----------------------------------------------+-------------------------------
       16395 |      137080 | 0:3:3:           | ybd_f89f1179-08cc-43f9-a1c2-1ed724d886f1_epoch | 2020-03-31 18:43:12.156275-07
       16395 |      191574 | 0:191574:191574: | April1Cumu                                   | 2020-04-01 13:52:05.523348-07
       16395 |      191621 | 0:191621:191621: | premrep_20_04_01_13_52_13                    | 2020-04-01 13:52:13.345599-07
       16395 |      191761 | 0:191761:191761: | premrep_20_04_01_13_53_42                    | 2020-04-01 13:53:42.200495-07
       16395 |      191810 | 0:191810:191810: | premrep_20_04_01_13_55_12                    | 2020-04-01 13:55:12.204403-07
(5 rows)
```

Two new snapshots have been added, as created by replication cycles that run every 90 seconds. These snapshots, and subsequently created snapshots, are all valid and remain in the view until a new full or cumulative backup marks them obsolete.

Although backup snapshots may be removed from the view history, this behavior has *no effect* on backups that are already persisted and stored in backup bundles. These physical backups may be required for the purpose of restoring a database. For example, if you run the `ybbackupctl --list` command for this scenario, you will see the following results:

```
Restore points:

       type                              snapshot_name            backup_timestamp   size (GB)
       FULL                                 April1Full 2020-04-01T20:19:34.955359Z        1.00
 INCREMENTAL                                  April1Inc 2020-04-01T20:23:22.827597Z        0.51
 INCREMENTAL                                 April1Inc2 2020-04-01T20:26:54.542625Z        2.02
  CUMULATIVE                                 April1Cumu 2020-04-01T20:52:05.523348Z        3.61
```

A full database restore to snapshot `April1Cumu` will start restoring from snapshot `April1Full`, regardless of the fact that this snapshot is no longer present in the results of the `sys.backup_snapshot` view.

sys.blade

Platforms: EE: All appliance platforms

Parent topic: System Views

This system view returns status information about the blades in an appliance.

Column Name	Data Type	Description
blade_id	text	UUID for the blade
chassis_id	integer	`0` for a single-chassis appliance; `0` or `1` for a dual-chassis appliance.
name	text	Name: `dm1`, `dm2`, and so on.
status	text	Current status, such as `ok` or `not_installed`.
blade_state	text	Current state, such as `WORKER_REGISTERED` or `NOT_INSTALLED`.
blade_state_timestamp	timestamptz	When the blade went into the current state
powered_on	boolean	True or false
bios_version	text	BIOS version
ip_address	text	IP address
mac_address	text	MAC address
bay_slot	bigint	Bay number
led_yellow	boolean	True or false
led_blue	boolean	True or false
led_red	boolean	True or false
operating_system	text	`ybos` or `off`

Example

For example:

```
yellowbrick_test=# select * from sys.blade where chassis_id=1;
-[ RECORD 1 ]---------+-------------------------------------
blade_id              | 00000000-0000-0000-0000-38b8ebd00cad
chassis_id            | 1
name                  | dm1
status                | ok
blade_state           | WORKER_REGISTERED
blade_state_timestamp | 2018-07-07 01:27:44.031-07
powered_on            | t
bios_version          | 05.04.21.0038.00.004
ip_address            | 192.168.10.10
mac_address           | 38:b8:eb:d0:0c:ad
```

```
bay_slot              | 1
led_yellow            | t
led_blue              | f
led_red               | f
operating_system      | ybos
-[ RECORD 2 ]---------+-------------------------------------
blade_id              | 00000000-0000-0000-0000-38b8ebd00c76
chassis_id            | 1
name                  | dm10
status                | ok
blade_state           | WORKER_REGISTERED
blade_state_timestamp | 2018-07-07 01:27:50.399-07
powered_on            | t
bios_version          | 05.04.21.0038.00.004
ip_address            | 192.168.10.19
mac_address           | 38:b8:eb:d0:0c:76
bay_slot              | 10
led_yellow            | t
led_blue              | f
led_red               | f
operating_system      | ybos
...
```

Blade States

NOT_INSTALLED
: The blade is not installed and the bay is empty.

POWERED_ON
: The blade is booting up, but is not ready yet.

POWERED_ON_OFFLINE
: The blade is powered on but is unavailable.

POST
: Performing self-test after powering on.

YBOS_BOOTING
: The operating system is booting up.

YBOS_BOOTED
: The operating system is booted and running.

KERNEL_BOOTED
: System is booted, freeing unused memory.

FLASHING_BIOS
: Maintenance state only.

FLASHING_INFINIBAND
: Maintenance state only.

WORKER_STARTING
: The compute node is starting up.

NETWORK_READY
: The network link is ready.

WORKER_FAILED
: Compute node failed and is restarting.

WORKER_FAILED_REGISTRATION
: The compute node was unable to register with the cluster manager.

WORKER_ERROR

Compute node error, reported from the Lime compiler.

WORKER_REGISTERED

The compute node is registered with the cluster manager and is ready.

POWERED_OFF

The blade is stopped.

ERROR

The Chassis Manager Processor is reporting that the blade is in an error state.

UNKNOWN_EVENT

Unable to determine state.

sys.cluster

Platforms: All platforms

Parent topic: System Views

This system view returns state information about each compute cluster in the current instance. See compute clusters for more information on the role of compute clusters, the lifecycle of cluster nodes and the meaning of the values below.

If a compute cluster's `state` is `SUSPENDED`, the various node counts will be null.

Column Name	Data Type	Description
cluster_id	uuid	UUID for the cluster.
cluster_name	name	Unique name of the cluster.
nodes	smallint	Number of nodes requested for the cluster.
prepared_nodes	bigint	The number of cluster nodes we have received from the cloud provider.
configured_nodes	bigint	The number of cluster nodes that have the database software pod and image.
ready_nodes	bigint	Number of cluster nodes with the database software starting up.
ready_workers	bigint	Number of cluster nodes that have started the software and are ready to execute queries.
default_wlm_profile_name	name	Name of the default WLM profile that was defined for the cluster.
active_wlm_profile_name	varchar(64000)	Name of the active WLM profile currently assigned to the cluster.
hardware_instance_type_id	uuid	UUID for the hardware instance type selected for the cluster.
hardware_instance_name	varchar(256)	Name of the hardware instance type selected for the cluster: `small-v1` or `large-v1`.
is_default_cluster	boolean	Whether this cluster is the default cluster. See Default and System Clusters.
is_system_cluster	boolean	Whether this cluster is the system cluster. See Default and System Clusters.
auto_suspend	integer	Auto-suspend timeout, in seconds. `0` means that auto-suspend is disabled (as for the default cluster).
auto_resume	boolean	Whether this cluster resumes automatically when a query or some other operation attempts to use it.
max_spill_pct	integer	Maximum percentage of total disk that can be used for spill space.
max_cache_pct	integer	Maximum percentage of total disk that can be used for the object store cache.
type	text	Always `COMPUTE` (virtual compute cluster).
state	text	Cluster state: `CREATING`, `RUNNING`, `SUSPENDED`.

Examples

Show the number of nodes we've requested but not yet received from the cloud provider. This can show lack of free nodes in a certain availability zone, or account quota issues:

```
SELECT cluster_name, nodes - prepared_nodes from sys.cluster;
```

sys.cluster_status

Platforms: All platforms

Parent topic: System Views

Column Name	Data Type	Description
cluster_id	uuid	UUID for the cluster.
is_online	boolean	Whether the cluster is online from a hardware resource perspective.
is_ready	boolean	Whether the cluster is operationally ready from a compute node perspective.
status		Cluster hardware status.
offline_reason		Reason why the cluster is offline.
state	text	Cluster state: `RUNNING`, `SUSPENDED`, and so on.

Examples

Which clusters are currently running?

```
premdb=# select * from sys.cluster_status where state='RUNNING';
             cluster_id              | is_online | is_ready |    status    | offline_reason |  state
-------------------------------------+-----------+----------+--------------+----------------+---------
 da88c52f-c876-43f9-9777-6121e1b0da0e | t         | t        | NOT_DEGRADED | NOT_OFFLINE    | RUNNING
 7b942fdd-7293-47d6-86f4-de994daa300d | t         | t        | NOT_DEGRADED | NOT_OFFLINE    | RUNNING
(2 rows)
```

Join `sys.cluster`, `sys.cluster_status`, and `sys.worker` to add cluster names and compute node roles to the information available in the status view:

```
premdb=# select c.cluster_id, c.status, c.offline_reason, c.state, w.role, cl.cluster_name
from sys.cluster_status c join sys.worker w on c.cluster_id=w.cluster_id
join sys.cluster cl on c.cluster_id=cl.cluster_id;
             cluster_id              |    status    |      offline_reason      |   state   |  role  |      cluster_name
-------------------------------------+--------------+--------------------------+-----------+--------+------------------------
 da88c52f-c876-43f9-9777-6121e1b0da0e | NOT_DEGRADED | NOT_OFFLINE              | RUNNING   | MEMBER | cn-uat-beta1-cluster-01
 7b942fdd-7293-47d6-86f4-de994daa300d | NOT_DEGRADED | TOO_MANY_MISSING_WORKERS | SUSPENDED | SPARE  | bobr-may3-small-cluster
 7b942fdd-7293-47d6-86f4-de994daa300d | NOT_DEGRADED | TOO_MANY_MISSING_WORKERS | SUSPENDED | SPARE  | bobr-may3-small-cluster
(3 rows)
```

sys.column

This view returns information about all of the columns in the user tables in the current database.

Column Name	Data Type	Description
table_id	bigint	The unique ID for the table. Tables in different schemas within the same database may have the same name, so you can use this ID to uniquely identify a table.
column_id	integer	The ID for the column in the table: a positional number starting with `1` for the first named column.
name	text	Name of the column.
type	text	Data type of the column.
nullable	boolean	A value of `t` means the column accepts `null` values. A value of `f` means that the column was defined with a `NOT NULL` constraint.
distribution_key	integer	`1` if the column is the distribution key for the table; otherwise `null`.
sort_key	integer	`1` if the column is the sort key for the table; otherwise `null`.
partition_key	smallint	`1` or greater if the column is a partitioning key for the table; otherwise `null`.
cluster_key	integer	`1-4` if a column is a cluster key for the table; otherwise `null`.
encrypted	boolean	A value of `t` means the column is encrypted; `f` means it is not encrypted.

Examples

The following query returns an ordered list of the columns in six user tables in the current database:

```
premdb=# select name, table_id from sys.column order by 2,1;
       name        | table_id
-------------------+----------
 numteams          |    46958
 season_name       |    46958
 seasonid          |    46958
 winners           |    46958
 atid              |    46962
 avg_att           |    46962
 capacity          |    46962
 city              |    46962
 htid              |    46962
 name              |    46962
 nickname          |    46962
 stadium           |    46962
 teamid            |    46962
 htid              |    46966
 name              |    46966
 atid              |    46970
 name              |    46970
```

```
 atid               |    46974
 ftscore            |    46974
 htid               |    46974
 htscore            |    46974
 matchday           |    46974
 seasonid           |    46974
 avg_mins_per_match |    47071
 cob                |    47071
 dob                |    47071
 firstname          |    47071
 lastname           |    47071
 matches_played     |    47071
 playerid           |    47071
 position           |    47071
 seasonid           |    47071
 teamid             |    47071
 weekly_wages       |    47071
(34 rows)

premdb=# \d
           List of relations
 Schema |   Name   | Type  |    Owner
--------+----------+-------+-------------
 public | awayteam | table | yellowbrick
 public | hometeam | table | yellowbrick
 public | match    | table | yellowbrick
 public | player   | table | yellowbrick
 public | season   | table | yellowbrick
 public | team     | table | yellowbrick
(6 rows)
```

The following example joins `sys.column` to `sys.database`, `sys.schema`, and `sys.table` to get the database name, schema name, and table name (and to make sure that the results are for a single table, schema, and database combination):

```
premdb=# select d.database_id, d.name database_name, d.owner_id,
s.name schema_name, t.name table_name, t.table_id,
c.name column_name, c.type, c.distribution_key
from sys.database d
inner join
sys.table t on (d.database_id = t.database_id)
inner join
sys.column c on (t.table_id = c.table_id)
inner join
sys.schema s on (s.schema_id = t.schema_id) and (s.database_id = d.database_id)
where t.name = 'match' and s.name = 'public'
order by c.column_id;
 database_id | database_name | owner_id | schema_name | table_name | table_id | column_name |            type             | distri
-------------+---------------+----------+-------------+------------+----------+-------------+-----------------------------+-------
       16459 | premdb        |    16007 | public      | match      |    16512 | seasonid    | smallint                    |
       16459 | premdb        |    16007 | public      | match      |    16512 | matchday    | timestamp without time zone |
       16459 | premdb        |    16007 | public      | match      |    16512 | htid        | smallint                    |
       16459 | premdb        |    16007 | public      | match      |    16512 | atid        | smallint                    |
       16459 | premdb        |    16007 | public      | match      |    16512 | ftscore     | character(3)                |
       16459 | premdb        |    16007 | public      | match      |    16512 | htscore     | character(3)                |
(6 rows)
```

sys.const

This persistent table stores one row with one column and a constant integer value of `1`. Because this table has only one row, you can use it to return exactly one row for the result of a function or constant expression. You can use any other table in your database to compute the same result, but the result will be returned once for every row in that table. By using the `sys.const` table, you guarantee that only one row will be returned.

The Yellowbrick planner is aware of the `sys.const` table so queries may gain a performance benefit from using `sys.const` rather than some other table.

```
premdb=# \d sys.const
     Table "sys.const"
 Column |  Type   | Modifiers
--------+---------+-----------
 one    | integer | not null
```

```
premdb=# select * from sys.const;
 one
-----
   1
(1 row)
```

```
premdb=# select current_date from sys.const;
   date
------------
 2016-07-13
(1 row)
```

Tip: This table is similar in functionality to the `dual` table in Oracle databases.

sys.database

A system view that returns information about the databases in the system.

Column Name	Data Type	Description
database_id	bigint	The unique ID for the database.
name	text	Name of the database. (Note that the `yellowbrick` database row contains information about some internal tables.)
owner_id	bigint	Unique ID of the user who owns the database. Join to `sys.user` to get the name of the user.
encoding	text	Database encoding; either `LATIN9` or `UTF8`.
collation	text	Database collation type; currently always `C`.
is_readonly	boolean	Whether the database is in `READONLY` mode.
readonly_reason	text	Why the database is in `READONLY` mode. The database may have been set to `READONLY` explicitly by a database administrator.
is_hot_standby	boolean	Whether the database is in `HOT_STANDBY` mode.
access_privileges	text	User access privileges (ACLs) defined for the database, using the following format: `role=privileges/dbowner` You can use the HAS_DATABASE_PRIVILEGE function to get information about database ACLs. For a complete list of ACLs and their abbreviations, see Abbreviations for ACLs.
table_count	bigint	Number of user tables and temporary tables in the database.
rows_columnstore	bigint	Total number of rows in the column store for this database (backend storage).
rows_rowstore	bigint	Total number of rows in the row store for this database (rows not yet flushed).
compressed_bytes	bigint	Total compressed bytes of actual, live user data on the workers. See also sys.table_storage
snapshot_backup_bytes	bigint	Backup and restore system data stored in internal system tables. Managed by the backups and only decreases when full backups occur. *Not supported in this release.*
delete_info_bytes	bigint	Deleted and updated system data on the workers, which is kept in specialized internal data blocks.
reclaimable_bytes	bigint	Deleted or updated user data on the workers that will be freed up in the background.
uncompressed_bytes	bigint	Total uncompressed bytes of actual, live user data on the workers. See also sys.table_storage
max_size_bytes	bigint	Maximum size of the database set by the quota. `NULL` if no disk usage limit is set.
default_location	text	Default storage location for the database (currently, always `LOCAL`).

Note: Aggregations in `sys.database` view are calculated based on the user's access privileges to the underlying tables. Tables where the user lacks `SELECT` privileges on any column are excluded from the aggregation.

Examples

Return information for a database named `premdb`:

```
premdb=# select * from sys.database where name = 'premdb';
-[ RECORD 1 ]---------+---------
database_id           | 16634
name                  | premdb
owner_id              | 16007
encoding              | LATIN9
collation             | C
is_readonly           | f
readonly_reason       | [NULL]
is_hot_standby        | f
access_privileges     | [NULL]
table_count           | 5
rows_columnstore      | 100
rows_rowstore         | 8681
compressed_bytes      | 4194304
snapshot_backup_bytes | 0
delete_info_bytes     | 0
reclaimable_bytes     | 0
uncompressed_bytes    | 3314
max_size_bytes        | [NULL]
default_location      | LOCAL
```

Return information about any read-only databases:

```
premdb=# select database_id, name, is_readonly, readonly_reason, rows_columnstore from sys.database where is_readonly='t';
 database_id |  name  | is_readonly |                                    readonly_reason                                    |
-------------+--------+-------------+-------------------------------------------------------------------------------------+
       31384 | repldb | t           | Replication topology validation failure. Target database premdb_yb95 is not in HOT_STANDBY |
(1 row)
```

Return access privileges for the `premdb` database:

```
yellowbrick=# select name, access_privileges from sys.database where name='premdb';
  name  |    access_privileges
--------+--------------------------
 premdb | =Tc/ybrick1             +
        | ybrick1=CTcpqQuO/ybrick1+
        | bobr=CTcpqQuO/ybrick1   +
        | bar=uO/ybrick1          +
        | yb007=C/ybrick1         +
        | yb008=q/ybrick1
(1 row)
```

In this example, `ybrick1` is the database owner. The users and roles with privileges on this database are `bobr`, `bar`, `yb007`, and `yb008`.

To return the total space usage on the workers for a given database, add the values of `compressed_bytes`, `delete_info_bytes`, and `snapshot_backup_bytes`:

```
premdb=# select compressed_bytes, delete_info_bytes, snapshot_backup_bytes,
compressed_bytes+delete_info_bytes+snapshot_backup_bytes as total_space
from sys.database where name = 'premdb';
-[ RECORD 1 ]---------+---------
```

```
compressed_bytes      | 33554432
snapshot_backup_bytes | 4194304
delete_info_bytes     | 0
total_space           | 37748736
```

In this example, the total space usage is `37748736` bytes.

The following example demonstrates database visibility for a new user. By default, new users have `CONNECT` and `TEMP` privilege on all databases through membership in `PUBLIC`:

```
premdb=# create user noprivs;
CREATE ROLE
premdb=# alter user noprivs password 'noprivs';
ALTER ROLE
premdb=# \c premdb noprivs
Password for user noprivs:
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)
You are now connected to database "premdb" as user "noprivs".
premdb=> select database_id, name from sys.database
 database_id |         name
-------------+----------------------
        4400 | yellowbrick
       16392 | yellowbrick_test
       16393 | yellowbrick_test_utf8
       19123 | premdb
(4 rows)
```

To limit visibility for the user `noprivs`, use a `REVOKE` command:

```
premdb=> \c premdb yellowbrick
Password for user yellowbrick:
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)
You are now connected to database "premdb" as user "yellowbrick".
premdb=# revoke connect, temp on database yellowbrick, yellowbrick_test, yellowbrick_test_utf8 from public;
REVOKE
premdb=# \c premdb noprivs
Password for user noprivs:
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)
You are now connected to database "premdb" as user "noprivs".
premdb=> select database_id, name from sys.database
 database_id |  name
-------------+--------
       19123 | premdb
(1 row)
```

Note that this `REVOKE` command applies to all users who belong to `PUBLIC`, not just the new user `noprivs`.

sys.drive_summary

Platforms: EE: All appliance platforms

Parent topic: System Views

This system view returns a summary of storage information for the drives on each compute node.

Column Name	Data Type	Description
worker_id	uuid	Compute node identifier.
chassis_id	integer	`0` for a single-chassis appliance; `0` or `1` for a dual-chassis appliance.
drive	bigint	Drive number or name on the compute node.
total_bytes	bigint	Total number of bytes of storage available.
bytes_used	bigint	Total number of bytes of storage in use.
drive_wear	bigint	A percentage that describes the health of the drive. `0` means there is no wear on the drive.

Example

```
yellowbrick_test=# select * from sys.drive_summary where chassis_id=0 order by 3;
             worker_id               | chassis_id | drive |  total_bytes  |  bytes_used  | drive_wear
-------------------------------------+------------+-------+---------------+--------------+------------
 00000000-0000-0000-0000-38b8ebd00c1c |          0 |     0 | 2048347078656 | 356606017536 |          0
 00000000-0000-0000-0000-38b8ebd00c6c |          0 |     0 | 2048347078656 | 345254133760 |          0
 00000000-0000-0000-0000-38b8ebd00bf9 |          0 |     0 | 2048347078656 | 339386302464 |          0
 00000000-0000-0000-0000-38b8ebd007fd |          0 |     0 | 2048347078656 | 357673467904 |          0
 00000000-0000-0000-0000-38b8ebd00c35 |          0 |     0 | 2048347078656 | 429966491648 |          0
 00000000-0000-0000-0000-38b8ebd00c0d |          0 |     0 | 2048347078656 | 365672005632 |          0
 00000000-0000-0000-0000-38b8ebd00c58 |          0 |     0 | 2048347078656 | 315751399424 |          0
 00000000-0000-0000-0000-38b8ebd00c53 |          0 |     0 | 2048347078656 | 326126010368 |          0
 00000000-0000-0000-0000-38b8ebd00ca8 |          0 |     0 | 2048347078656 | 328330117120 |          0
 00000000-0000-0000-0000-38b8ebd00c30 |          0 |     0 | 2048347078656 | 327034077184 |          0
 00000000-0000-0000-0000-38b8ebd00c2b |          0 |     0 | 2048347078656 | 359581876224 |          0
 00000000-0000-0000-0000-38b8ebd00c2b |          0 |     1 | 2048347078656 | 356343873536 |          0
 00000000-0000-0000-0000-38b8ebd00c6c |          0 |     1 | 2048347078656 | 344658542592 |          0
 00000000-0000-0000-0000-38b8ebd00c1c |          0 |     1 | 2048347078656 | 363593728000 |          0
 00000000-0000-0000-0000-38b8ebd00c58 |          0 |     1 | 2048347078656 | 317699653632 |          0
 00000000-0000-0000-0000-38b8ebd00c53 |          0 |     1 | 2048347078656 | 327791149056 |          0
 00000000-0000-0000-0000-38b8ebd007fd |          0 |     1 | 2048347078656 | 360108261376 |          0
 00000000-0000-0000-0000-38b8ebd00bf9 |          0 |     1 | 2048347078656 | 342580264960 |          0
 00000000-0000-0000-0000-38b8ebd00c35 |          0 |     1 | 2048347078656 | 427101782016 |          0
 00000000-0000-0000-0000-38b8ebd00c30 |          0 |     1 | 2048347078656 | 331408736256 |          0
 00000000-0000-0000-0000-38b8ebd00c0d |          0 |     1 | 2048347078656 | 362717118464 |          0
 00000000-0000-0000-0000-38b8ebd00ca8 |          0 |     1 | 2048347078656 | 330909614080 |          0
...
```

sys.external_authentication

A system view that returns information about the external authentication objects in the instance. External authentication objects are used to configure access to external identity providers (IDPs) when configuring single sign-on (SSO). See also Single Sign-On and CREATE EXTERNAL AUTHENTICATION.

Column Name	Data Type	Description
id	oid	Identity of the external authentication record
name	name	Name of the external authentication record
owner_id	oid	Owner of this external authentication record
is_enabled	boolean	This external authentication record is enabled or not
issuer	text	Issuer to match for token signature validation and for public key discovery using JWKS if `public_key` is not specified
is_user_auto_create	boolean	Auto create the mapped db user if it doesn't exist
user_mapping_claim	name	Claim key whose value in the jwt maps to the db user, or is used to create user if `is_user_auto_create` is set
audiences_claim	text[1]	Array of possible audience claim values in the jwt. One must match for authentication to proceed. Scaling consideration: Limited to 5000 strings
roles_granted	text[1]	Array of role names to grant user if `is_user_auto_create` is set, and user is created during login. Scaling consideration: Limited to 5000 names
role_grant_claim	name	Claim key whose value in the jwt contains a json array of grant roles
roles_allowed_login	text[1]	Array of role name to allow login if found in jwt. See `role_grant_claim`. Scaling consideration: Limited to 5000 role names
roles_disallowed_login	text[1]	Array of role names to disallow login if found in jwt. See `role_grant_claim`. Scaling consideration: Limited to 5000 role names
azp_claim	name	Authorized party value that is expected to be found in a valid jwt
public_key	text	Permanent public key. Set this to the public key of the issuer signing authority for tokens if the JWKS endpoint is not available as specified by `issuer`.
is_disable_trust	boolean	Disable TLS trust verification between PG and JWKS endpoint disabled or not.

sys.external_format

A system view that returns information about the external format objects in the instance.

Column Name	Data Type	Description
external_format_id	bigint	Unique ID of the external format object.
name	text	Name of the external format object.
owner_id	bigint	Unique ID of the user who owns the external format object. Join key for `sys.user` and `sys.role`.
type	text	Type of format: `CSV`, `TEXT`, or `BCP`, `PARQUET`.
format_options	text	JSON string that lists all of the load options that were defined for this external format object.
creation_time	timestamptz	When the external format object was created.
modify_time	timestamptz	This value is currently the same as `creation_time` because external format objects cannot be modified.

Example

```
premdb=# select * from sys.external_format where name='premdbs3format';
-[ RECORD 1 ]------+--------------------------------------------------------------------------------
external_format_id | 137328
name               | premdbs3format
owner_id           | 137192
type               | csv
format_options     | {"version":1,"type":"csv","opts":{"skip_blank_lines":"true","delimiter":","}}
creation_time      | 2022-06-14 11:58:22.382366-07
modify_time        | 2022-06-14 11:58:22.382366-07
```

sys.external_location

Platforms: All platforms

Parent topic: System Views

A system view that returns information about the external location objects in the database. This view does not return information about `PRIMARY` storage locations.

Column Name	Data Type	Description
external_location_id	bigint	Unique ID of the external location object.
name	text	Name of the external location object.
owner_id	bigint	Unique ID of the user who owns the external location object. Join key for `sys.user` and `sys.role`.
external_storage_id	bigint	Unique ID of the external storage object. Join key for `sys.external_storage`.
external_format_id	bigint	Unique ID of the external format object. Join key for `sys.external_format`.
path	text	Path to the location of the S3 files.
creation_time	timestamptz	When the external location object was created.
modify_time	timestamptz	This value is currently the same as `creation_time` because external location objects cannot be modified.
usage_type	varchar(256)	Always shows `EXTERNAL`. Although every instance has a `PRIMARY` storage location, these locations are not shown in this view.

Example

```
premdb=# select * from sys.external_location where name='premdbs3data';
-[ RECORD 1 ]--------+----------------------------
external_location_id | 137329
name                 | premdbs3data
owner_id             | 137192
external_storage_id  | 137327
external_format_id   | 137328
path                 | yb-sampledata-d4f1c23ea7
creation_time        | 2022-06-14 11:58:22.390847-07
modify_time          | 2022-06-14 11:58:22.390847-07
usage_type           | EXTERNAL
```

sys.external_storage

A system view that returns information about the external storage objects in the instance.

Column Name	Data Type	Description
external_storage_id	bigint	Unique ID of the external storage object.
name	text	Name of the external storage object.
owner_id	bigint	Unique ID of the user who owns the external storage object. Join key for `sys.user` and `sys.role`.
type	text	Type of external storage. Currently, only `S3` is supported.
endpoint	text	Endpoint URI for object storage.
region	text	Name of an AWS S3 region.
identity	text	Access key ID for S3 endpoints, or a storage account name for Azure endpoints.
credential	text(12)	Credential for accessing object storage, if specified, such as a secret key or a SAS token. Always displayed as ********.
creation_time	timestamptz	When the external storage object was created.
modify_time	timestamptz	This value is currently the same as `creation_time` because external storage objects cannot be modified.

Example

```
premdb=# select external_storage_id, name, type, region, identity, creation_time
 from sys.external_storage where name='premdbs3' or name='sf10000';
 external_storage_id |   name   | type |  region   |       identity       |         creation_time
---------------------+----------+------+-----------+----------------------+-------------------------------
               16402 | sf10000  |  s3  | us-east-1 | AKIA4JMEBK6CTP7MB4FZ | 2022-05-31 01:23:57.480927-07
              137327 | premdbs3 |  s3  | us-west-2 |                      | 2022-06-14 11:58:22.073316-07
(2 rows)
```

sys.hardware_instance_assignment

This system view maps clusters to hardware instance types (by ID).

Column Name	Data Type	Description
cluster_id	uuid	UUID for the cluster.
hardware_instance_type_id	uuid	UUID for the hardware instance type defined for the cluster.

Examples

How many clusters are set up to use each instance type?

```
yellowbrick=# select hardware_instance_type_id, count(cluster_id)
from sys.hardware_instance_assignment group by 1;
     hardware_instance_type_id       | count
-------------------------------------+-------
 f43d64aa-828e-42cc-b402-60e008c544a3 |     8
 b7da0afd-2c98-4f57-888d-98a51975a412 |     2
(2 rows)
```

Join to the `sys.cluster` view to get the names of each hardware instance type:

```
yellowbrick=# select c.hardware_instance_type_id, hardware_instance_name, count(c.cluster_id)
from sys.hardware_instance_assignment h join sys.cluster c on h.cluster_id=c.cluster_id group by 1, 2;
     hardware_instance_type_id       | hardware_instance_name | count
-------------------------------------+------------------------+-------
 f43d64aa-828e-42cc-b402-60e008c544a3 | small-v1               |     8
 b7da0afd-2c98-4f57-888d-98a51975a412 | large-v1               |     2
(2 rows)
```

sys.hardware_instance_limits

Platforms: All platforms

Parent topic: System Views

Column Name	Data Type	Description
hardware_instance_type_id	varchar(256)	ID for a hardware instance type.
max_nodes	integer	Maximum number of nodes for this hardware instance type, if a limit is set. `-1` indicates no limit.
reserved_nodes	integer	Number of reserved nodes for the instance. `-1` indicates that no nodes were reserved.

Examples

```
yellowbrick_test=> select * from sys.hardware_instance_limits;
     hardware_instance_type_id       | max_nodes | reserved_nodes
-------------------------------------+-----------+----------------
 9ae12f61-a18f-43b4-9e4d-180e8ed0611a |        -1 |             -1
 e1e22106-487a-4603-850e-07708222948a |        -1 |             -1
(2 rows)
```

```
yellowbrick=# select l.hardware_instance_type_id, hardware_instance_name, max_nodes
from sys.hardware_instance_limits l
join sys.hardware_instance_assignment a on l.hardware_instance_type_id=a.hardware_instance_type_id
join sys.cluster c on c.cluster_id=a.cluster_id order by 1,2;
     hardware_instance_type_id       | hardware_instance_name | max_nodes
-------------------------------------+------------------------+-----------
 b7da0afd-2c98-4f57-888d-98a51975a412 | large-v1               |        -1
 b7da0afd-2c98-4f57-888d-98a51975a412 | large-v1               |        -1
 f43d64aa-828e-42cc-b402-60e008c544a3 | small-v1               |        -1
 f43d64aa-828e-42cc-b402-60e008c544a3 | small-v1               |        -1
 f43d64aa-828e-42cc-b402-60e008c544a3 | small-v1               |        -1
 f43d64aa-828e-42cc-b402-60e008c544a3 | small-v1               |        -1
 f43d64aa-828e-42cc-b402-60e008c544a3 | small-v1               |        -1
 f43d64aa-828e-42cc-b402-60e008c544a3 | small-v1               |        -1
 f43d64aa-828e-42cc-b402-60e008c544a3 | small-v1               |        -1
 f43d64aa-828e-42cc-b402-60e008c544a3 | small-v1               |        -1
(10 rows)
```

sys.hardware_instance_type

Platforms: All platforms

Parent topic: System Views

This system view returns a list of available hardware instance types.

Column Name	Data Type	Description
hardware_instance_type_id	uuid	UUID for a hardware instance type.
cloud_provider	varchar(256)	`aws` in the current release.
cloud_region	varchar(256)	AWS region, such as `us-west.`
hardware_instance_name	varchar(256)	`large-v1` or `small-v1` in the current release.
provider_name	varchar(256)	Provider name for the instance type: `i3en.metal` or `m5dn.4xlarge` .
ram_bytes	bigint	Amount of RAM allocated to this instance type, in bytes.
vcpu_cores	integer	Number of virtual CPU cores per node.
cpu_cores	integer	Number of CPU cores per node.
network_access	varchar(256)	`accel_udp`
storage_access	varchar(256)	`accel_nvme`
storage_size_bytes	bigint	Size of data storage, in bytes.
is_system	boolean	Whether this instance type was created by the system; always `true` in the current release.

Example

```
yellowbrick=# select * from sys.hardware_instance_type;
-[ RECORD 1 ]-------------+-------------------------------------
hardware_instance_type_id | b7da0afd-2c98-4f57-888d-98a51975a412
cloud_provider            | aws
cloud_region              | us-west
hardware_instance_name    | large-v1
provider_name             | i3en.metal
ram_bytes                 | 793495207936
vcpu_cores                | 95
cpu_cores                 | 48
network_access            | accel_udp
storage_access            | accel_nvme
storage_size_bytes        | 68169720922112
is_system                 | t
-[ RECORD 2 ]-------------+-------------------------------------
hardware_instance_type_id | f43d64aa-828e-42cc-b402-60e008c544a3
cloud_provider            | aws
cloud_region              | us-west
hardware_instance_name    | small-v1
provider_name             | m5dn.4xlarge
```

```
ram_bytes                | 57982058496
vcpu_cores               | 15
cpu_cores                | 8
network_access           | accel_udp
storage_access           | accel_nvme
storage_size_bytes       | 644245094400
is_system                | t
```

sys.hardware_instance_usage

Platforms: All platforms

Parent topic: System Views

Column Name	Data Type	Description
hardware_instance_type_id	uuid	UUID for a hardware instance type.
hardware_instance_name	varchar(256)	`large-v1` or `small-v1` in the current release.
max_nodes	integer	Maximum number of nodes for this hardware instance type, if a limit is set. `-1` indicates no limit. (This value is also defined in `sys.hardware_instance_limits`.)
current_nodes	integer	Number of nodes assigned to this instance (per instance type). All of the nodes for a given instance must be the same type. The node number is "current" because the instance may be scaled up or down.
online_nodes	integer	Number of nodes currently online for this instance.
available_nodes	bigint	The number of available nodes is the remainder of `max_nodes - current_nodes`.
requested_reserved_nodes	integer	Number of *reserved nodes* that were requested when the instance was created. See Reserved Nodes.
over_subscribed	boolean	Whether the current usage represents an over-subscription of available nodes.

Example

```
yellowbrick_test=> select * from sys.hardware_instance_usage;
     hardware_instance_type_id       | hardware_instance_name | max_nodes | current_nodes | online_nodes | available_nodes | reque
-------------------------------------+------------------------+-----------+---------------+--------------+-----------------+-----
 9ae12f61-a18f-43b4-9e4d-180e8ed0611a | small-v1               |        -1 |             2 |           10 |              -3 |
 e1e22106-487a-4603-850e-07708222948a | large-v1               |        -1 |             0 |              |              -1 |
(2 rows)
```

sys.instance_id

Return the UUID, name, and creation time of the current data warehouse instance. For example:

```
yellowbrick=> select * from sys.instance_id;
                 id                  |    name    |             when
-------------------------------------+------------+-------------------------------
 ab2161d0-62ae-4cd6-a621-125e20006a59 | bobr070623 | 2023-07-06 22:35:23.864468+00
(1 row)
```

sys.key

This system view returns information about the keys for which the current user has encrypt or decrypt rights or is the owner.

Column Name	Data Type	Description
key_id	bigint	The unique ID of the key
name	text	The name of the key
schema_id	bigint	The unique ID of the schema
owner_id	bigint	The unique ID of the user who owns the key
creation_time	timestamp with time zone	When the key was created

For example:

```
premdb=> select * from sys.key;
 key_id |   name   | schema_id | owner_id |         creation_time
--------+----------+-----------+----------+-------------------------------
  16480 | yb100key |      2200 |    16007 | 2019-11-07 15:26:30.577896-08
(1 row)
```

sys.load

This view captures information about active bulk load (`ybload`) operations. The `sys.load` and `sys.log_load` views contain the same columns. See sys.log_load for details. Note that the `end_time` column for an active load displays the same timestamp as the `start_time` column.

Examples

How many rows per second are being loaded into table `16496` ?

```
premdb=# select table_id, inserted_rows, inserted_rows_per_second from sys.load;
 table_id | inserted_rows | inserted_rows_per_second
----------+---------------+--------------------------
   16435 |      25990128 |                4464890.5
(1 row)
```

The following example uses the `ybsql \watch` command to poll the view every 5 seconds as a load is progressing. The results are displayed in `ybsql` expanded mode for readability.

```
premdb=# select * from sys.load;
...
-[ RECORD 3 ]-------------+------------------------------------------------------------------
session_key              | C557WQQ6MK72T-sGpavzoStA_iUOYG7WmYugI5jmn70oKSvEiU-SYTifUo9_VCxU
state                    | RUNNING
error_string             | [NULL]
table_id                 | 16435
schema_id                | 2200
database_id              | 16423
user_id                  | 16434
session_id               | 16747
table_name               | newmatchstats
schema_name              | public
database_name            | premdb
username                 | bobr
client_hostname          | brumsby
client_username          | brumsby
start_time               | 2018-05-02 16:14:50-07
last_activity_time       | 2018-05-02 16:14:50-07
elapsed_ms               | 2604.000
remaining_ms             | 0.000
transaction_first        | 45637
transaction_last         | 45637
num_transactions         | 1
parsed_rows              | 0
parsed_bytes             | 0
parsed_rows_per_second   | 0
parsed_bytes_per_second  | 0
inserted_rows            | 27421
inserted_bytes           | 1048376
inserted_rows_per_second | 10530.337891
inserted_bytes_per_second | 402602.15625
error_rows               | 0
```

```
sent_bytes                | 0
sent_bytes_per_second     | [NULL]

Watch every 5s  Wed May  2 16:14:58 2018
...
-[ RECORD 3 ]-------------+-----------------------------------------------------------------
session_key               | C557WQQ6MK72T-sGpavzoStA_iUOYG7WmYugI5jmn70oKSvEiU-SYTifUo9_VCxU
state                     | DONE
error_string              | [NULL]
table_id                  | 16435
schema_id                 | 2200
database_id               | 16423
user_id                   | 16434
session_id                | 16747
table_name                | newmatchstats
schema_name               | public
database_name             | premdb
username                  | bobr
client_hostname           | brumsby
client_username           | brumsby
start_time                | 2018-05-02 16:14:50-07
last_activity_time        | 2018-05-02 16:14:56-07
elapsed_ms                | 5873.012
remaining_ms              | 0.000
transaction_first         | 45637
transaction_last          | 45637
num_transactions          | 1
parsed_rows               | 25990128
parsed_bytes              | 659797724
parsed_rows_per_second    | 4425349
parsed_bytes_per_second   | 112344008
inserted_rows             | 25990128
inserted_bytes            | 727723568
inserted_rows_per_second  | 4286678
inserted_bytes_per_second | 120026976
error_rows                | 0
sent_bytes                | 727723568
sent_bytes_per_second     | [NULL]

Watch every 5s  Wed May  2 16:15:03 2018

...
```

sys.log_alert

Platforms: EE: All appliance platforms

Parent topic: System Views

This system view returns information about alerts that have been recorded on the appliance. See also sys.alert.

Example

For example:

```
yellowbrick_test=# select * from sys.log_alert where acknowledged='t';
-[ RECORD 1 ]-----+-------------------------------------
alert_id          | 67f3713b-476a-40e1-87a4-625b570eb16d
alert_name        | Power Supply Alert
open_time         | 2018-08-14 01:21:08-07
close_time        | [NULL]
last_update_time  | 2018-08-15 15:55:40-07
duration_ms       | 138882336.927
resource          | chassis0:power4
severity          | CRITICAL
status            | OPEN
state             | error
message           | The power supply failed: alert_status
acknowledged      | t
acknowledged_time | 2018-08-15 15:55:40.993-07
acknowledged_by   | yellowbrick
acknowledged_text |
-[ RECORD 2 ]-----+-------------------------------------
alert_id          | d9ba42e0-51ca-4953-b08d-023f27a0cd6a
alert_name        | Power Supply Alert
open_time         | 2018-08-14 01:21:08-07
close_time        | [NULL]
last_update_time  | 2018-08-15 15:55:40-07
duration_ms       | 138882336.927
resource          | chassis0:power3
severity          | CRITICAL
status            | OPEN
state             | error
message           | The power supply failed: alert_status
acknowledged      | t
acknowledged_time | 2018-08-15 15:55:40.993-07
acknowledged_by   | yellowbrick
acknowledged_text |
-[ RECORD 3 ]-----+-------------------------------------
...
```

sys.log_analyze

Yellowbrick Documentation > How-to Guides > Administer Yellowbrick > Databases > System Views > sys.log_analyze

Platforms: All platforms

Parent topic: System Views

This view records a history of analyze operations on tables (auto-analyze and manually run commands).

Column Name	Data Type	Description
table_id	bigint	The unique ID for the table. Tables in different schemas within the same database may have the same name, so you can use this ID to uniquely identify a table.
database_id	bigint	Unique database ID.
user_id	bigint	Unique ID of the user who ran the analyze command.
start_time	timestamptz	When the analyze command started running.
end_time	timestamptz	When the analyze command completed its execution.
rows	bigint	The number of rows in the table when the command was run.
rows_analyzed	bigint	The number of rows that were scanned in order to update statistics for the table.

Example

```
premdb=# select * from sys.log_analyze;
 table_id | database_id | user_id |          start_time           |           end_time            | rows | rows_analyzed
----------+-------------+---------+-------------------------------+-------------------------------+------+---------------
    16884 |       16874 |   16006 | 2017-06-19 14:17:09.448803-07 | 2017-06-19 14:17:11.427635-07 |   50 |            50
    16881 |       16874 |   16006 | 2017-06-19 14:17:11.431026-07 | 2017-06-19 14:17:13.376706-07 |   50 |            50
    16893 |       16874 |   16006 | 2017-06-19 15:37:15.650933-07 | 2017-06-19 15:37:16.972667-07 |    1 |             1
    16887 |       16874 |      10 | 2017-06-19 17:54:36.917482-07 | 2017-06-19 17:54:42.338015-07 |    0 |             0
    16887 |       16874 |      10 | 2017-06-20 12:49:15.774682-07 | 2017-06-20 12:49:20.240464-07 | 8606 |           172
(5 rows)
```

Note: Internal system users have five-digit user IDs that begin with `16`.

sys.log_audit

Yellowbrick Documentation > How-to Guides > Administer Yellowbrick > Databases > System Views > sys.log_audit

Platforms: EE: All appliance platforms

Parent topic: System Views

This view records audit events that occur on the appliance and their associated messages.

Column Name	Data Type	Description
event_id	uuid	Unique ID of the audit event.
event_time	timestamptz	Timestamp for the event.
query_id	bigint	Unique query ID, if applicable.
severity	integer	Severity gradient, where `0` =OK, `25` =Informational, `50` =Minor, `75` =Major, `100` =Critical.
acknowledged	boolean	Whether the user has acknowledged the event on the SMC System Events screen, where this data is also visible.
audit_code	text	In many cases, `system` . In other cases, an internal name for the source of the audit message, such as `CPU
audit_msg	text	The text of the message itself.

Example

```
premdb=# select * from sys.log_audit;
               event_id               |         event_time         | query_id | severity | acknowledged | audit_code |
--------------------------------------+----------------------------+----------+----------+--------------+------------+------------
 001d42ea-b97a-43cb-8f3a-e831d2790ad5 | 2017-06-19 14:12:07.406-07 |          |        0 | f            | system     | Initialized
 f1b61530-dc7d-4eaa-be2f-94904b2d5764 | 2017-06-19 14:12:14.134-07 |          |        0 | f            | system     | Added worke
 fd4e0ace-b020-4615-941e-74a58c40fa85 | 2017-06-19 14:12:14.142-07 |          |        0 | f            | system     | Worker 0000
 966a4ca0-06ff-4605-afd2-65a4fee15877 | 2017-06-19 14:12:17.191-07 |          |        0 | f            | system     | Cluster was
 2ada1d60-8162-4def-9095-d2309c9fd79c | 2017-06-19 14:12:17.202-07 |          |        0 | f            | system     | Worker 0000
 798a36a8-78e0-4e70-97dd-e2bc3bcc671d | 2017-06-19 14:12:17.204-07 |          |        0 | f            | system     | Computing n
 55429d42-85e9-4962-90ec-7a43c9f53d11 | 2017-06-19 14:12:17.221-07 |          |        0 | f            | system     | Found 1 new
 b733219c-791e-4206-9458-f81771919ab0 | 2017-06-19 14:12:17.229-07 |          |        0 | f            | system     | Starting ne
 c75ba733-7d32-4ade-b0e5-546e3552cdda | 2017-06-19 14:12:17.277-07 |          |        0 | f            | system     | Computed 2
 8fcd9daf-c902-48ec-82d0-26057181a27e | 2017-06-19 14:12:17.315-07 |          |        0 | f            | system     | Worker 0000
 b1b70644-7434-48ff-9710-2cb7aa107008 | 2017-06-19 14:12:17.465-07 |          |        0 | f            | system     | Cluster was
(11 rows)
```

sys.log_authentication

Yellowbrick Documentation > How-to Guides > Administer Yellowbrick > Databases > System Views > sys.log_authentication

Platforms: All platforms

Parent topic: System Views

This view records the history of users' attempts to authenticate with the database and logs success or failure details in each case.

Column Name	Data Type	Description
start_time	timestamptz	When the authentication attempt started.
user_name	name	The name of the user for the connection.
database_name	name	The database name.
event	text	`AUTHENTICATION`
client_ip_address	text	IP address for remote connections (or `[local]` for local connections).
client_port	text	Port number for remote connections.
status	integer	`0` for success; `-1` for failure.
status_detail	text	Message associated with failures (or `SUCCESS` for successful authentications).

Examples

Return authentication records for non-system users:

```
premdb=# select * from sys.log_authentication where user_name not like 'sys%';
        start_time         |  user_name  | database_name |     event      | client_ip_address | client_port | status |
----------------------------+-------------+---------------+----------------+-------------------+-------------+--------+----------
 2018-07-20 11:26:52.798-07 | brumsby     | yellowbrick   | Authentication | [local]           |             |      0 | SUCCESS
 2018-07-20 11:26:59.22-07  | brumsby     | premdb        | Authentication | [local]           |             |      0 | SUCCESS
 2018-07-20 18:43:31.995-07 | brumsby     | premdb        | Authentication | [local]           |             |      0 | SUCCESS
 2018-07-20 18:44:37.141-07 | bobr        | premdb        | Authentication | [local]           |             |      0 | SUCCESS
 2018-07-20 18:46:12.87-07  | brumsby     | premdb        | Authentication | [local]           |             |      0 | SUCCESS
 2018-07-23 18:10:19.87-07  | brumsby     | premdb        | Authentication | [local]           |             |      0 | SUCCESS
 2018-07-23 18:24:56.439-07 | yellowbrick | yellowbrick   | Authentication | 127.0.0.1         | 53884       |      0 | SUCCESS
 2018-07-24 11:33:45.312-07 | brumsby     | premdb        | Authentication | [local]           |             |      0 | SUCCESS
 2018-07-24 18:16:17.993-07 | bobr        | premdb        | Authentication | [local]           |             |     -1 | password a
(9 rows)
```

Return failed authentication records:

```
premdb=# select * from sys.log_authentication where status !=0;
        start_time         | user_name | database_name |     event      | client_ip_address | client_port | status |
----------------------------+-----------+---------------+----------------+-------------------+-------------+--------+----------
 2018-07-24 18:16:17.993-07 | bobr      | premdb        | Authentication | [local]           |             |     -1 | password aut
(1 row)
```

sys.log_backup

This view records the history of backup (`ybbackup`) operations.

Column Name	Data Type	Description
session_key	text	Unique session key generated for the backup operation.
session_id	bigint	Unique ID of the `ybbackup` session, as logged in `sys.session` and `sys.log_session` .
client_hostname	text	Client host name, if available.
client_username	text	Operating system user on the client system who is running the backup tool.
user_name	text	Name of the database user, as passed in on the `ybbackup` command line (or set with `YBUSER`).
user_id	bigint	Unique ID of the database user.
database_name	text	Name of the database that was backed up.
database_id	bigint	Unique ID of the database that was backed up.
state	text	Current state of the backup operation, such as `USER_DATA_RUNNING` or `DONE` .
error_string	text	Error message reported for a failed backup. For example, if a backup is canceled before it completes, the view may report an `External data socket error` .
start_time	timestamptz	When the `ybbackup` command started running.
end_time	timestamptz	When the `ybbackup` command completed.
elapsed_ms	bigint	Time spent running the backup (in milliseconds).
remaining_ms	bigint	Time remaining to complete the backup (in milliseconds).
sent_bytes	bigint	Number of bytes of backup data transferred over the network, including some overhead for the backup and restore communication protocol. (`sent_bytes` and `storage_bytes` are not expected to match, either during a backup or when it completes.)
sent_bytes_per_sec	double precision	Number of bytes per second transferred over the network.
storage_bytes	bigint	Number of bytes of backup data written to the storage system.
storage_bytes_per_sec	double precision	Number of bytes per second being written to the storage system.
backup_point_id	text	Backup name as specified with the `ybbackup --name` option.
chain_id	text	Name of the backup chain.
type	text	Backup type: `F` (full), `I` (incremental), `C` (cumulative).

Example

Return information about the backups in a specific backup chain:

```
premdb=# select * from sys.log_backup where chain_id='February2020';
-[ RECORD 1 ]---------+------------------------------------------------------------------
session_key           | DOl8lbUDTtILPRROKMN_jxrgAWO1DUJzJeKyijalbb-CHZgO2crMI8yh4Y_HqQM5
session_id            | 18646
client_hostname       | localhost
client_username       | yb100
user_name             | yb100
user_id               | 16418
database_name         | premdb
database_id           | 16395
state                 | DONE
error_string          | [NULL]
start_time            | 2020-02-14 16:37:56.752-08
end_time              | 2020-02-14 16:38:05.154-08
elapsed_ms            | 8402
remaining_ms          | 0
sent_bytes            | 35281796
sent_bytes_per_sec    | 4199213.99666746
storage_bytes         | 35289956
storage_bytes_per_sec | 4200185.19400143
backup_point_id       | Feb14Full
chain_id              | February2020
type                  | F
-[ RECORD 2 ]---------+------------------------------------------------------------------
session_key           | BNJyx7CXbkRr9STxe8FCgER2EdZUyjf8L5aZXoHYraWVtuUoZJQF_lrOHWLuwSA6
session_id            | 335742
client_hostname       | localhost
client_username       | yb100
user_name             | yb100
user_id               | 16418
database_name         | premdb
database_id           | 16395
state                 | DONE
error_string          | [NULL]
start_time            | 2020-02-16 21:09:25.728-08
end_time              | 2020-02-16 21:10:47.788-08
elapsed_ms            | 82060
remaining_ms          | 0
sent_bytes            | 1771417116
sent_bytes_per_sec    | 21586852.4981721
storage_bytes         | 1772002572
storage_bytes_per_sec | 21593986.9851328
backup_point_id       | Feb16Inc
chain_id              | February2020
type                  | I
```

sys.log_cluster_event

This system view returns the history of operations on clusters, such as suspend and resume events.

Column Name	Data Type	Description
cluster_id	uuid	UUID for the cluster.
cluster_name	name	Unique name of the cluster.
event_time	timestamptz	When the operation on the cluster occurred.
user_id	bigint	ID of the user account running the operation (`0` for `system` account).
old_nodes	smallint	Number of nodes before the event.
new_nodes	smallint	Number of nodes after the event.
old_hardware_instance	uuid	UUID of the instance before the event.
new_hardware_instance	uuid	UUID of the instance after the event.
type	text	Always `COMPUTE`.
event_action	text	What state the cluster was put in: `suspended`, `resumed`, `created`.
username	text	User account running the operation (can be `system` if system-driven).

Example

```
yellowbrick=# select * from sys.log_cluster_event where cluster_name like 'small%';
-[ RECORD 1 ]---------+-------------------------------------
cluster_id            | eb9aa671-583c-451a-8267-15ecf2e4a344
cluster_name          | small-2node-cluster
event_time            | 2022-04-20 20:49:31.818+00
user_id               | 16432
old_nodes             | 0
new_nodes             | 2
old_hardware_instance | 00000000-0000-0000-0000-000000000000
new_hardware_instance | f43d64aa-828e-42cc-b402-60e008c544a3
type                  | COMPUTE
event_action          | created
username              | hkane@tottenhamfc.com
-[ RECORD 2 ]---------+-------------------------------------
cluster_id            | eb9aa671-583c-451a-8267-15ecf2e4a344
cluster_name          | small-2node-cluster
event_time            | 2022-04-20 20:57:39.326+00
user_id               | 0
old_nodes             | 0
new_nodes             | 0
old_hardware_instance | f43d64aa-828e-42cc-b402-60e008c544a3
new_hardware_instance | f43d64aa-828e-42cc-b402-60e008c544a3
```

```
type                 | COMPUTE
event_action         | suspended
username             | system
-[ RECORD 3 ]--------+-----------------------------------
cluster_id           | eb9aa671-583c-451a-8267-15ecf2e4a344
cluster_name         | small-2node-cluster
event_time           | 2022-04-23 00:57:55.449+00
user_id              | 16432
old_nodes            | 2
new_nodes            | 2
old_hardware_instance | f43d64aa-828e-42cc-b402-60e008c544a3
new_hardware_instance | f43d64aa-828e-42cc-b402-60e008c544a3
type                 | COMPUTE
event_action         | resumed
username             | hkane@tottenhamfc.com
...
```

sys.log_error

Platforms: EE: All appliance platforms

Parent topic: System Views

This view captures system-level errors that are logged on the appliance. This view does not capture SQL errors.

Column Name	Data Type	Description
event_id	uuid	Unique event ID for the error.
event_time	timestamptz	Timestamp when the event (error) occurred.
query_id	bigint	Unique query ID, if any was associated with this error.
acknowledged	boolean	Whether the user has acknowledged the event on the SMC System Events screen, where this data is also visible.
error_code	text	Internal error code.
error_msg	text	Text of the error message.

Example

```
yellowbrick=# select * from sys.log_error where error_code !='system';
              event_id               |         event_time         | query_id | acknowledged |                      error_code
-------------------------------------+----------------------------+----------+--------------+-------------------------------------
 0194213f-f847-46ec-b97a-cce8b392ea98 | 2017-06-13 17:08:33.58-07  |          | f            | MEMORY|yb77-mgr0.yellowbrick.io|mem
 3571ce6e-dbe0-4aac-988e-a5e4e63d6939 | 2017-06-14 19:58:58.972-07 |          | f            | connection factory
 fd64a358-4e6a-4efd-9946-d3ed26247717 | 2017-06-14 20:13:38.975-07 |          | f            | connection factory
(3 rows)
```

sys.log_load

This system view logs information about completed bulk load (`ybload`) operations. The `sys.load` view contains the same information for `ybload` operations that are in progress.

Column Name	Data Type	Description
session_key	text	Unique authentication key for the `ybload` operation. This is different from the session ID, which is the ID tracked for the user session in the `sys.session` and `sys.log_session` views. (A unique query ID is not assigned until the bulk load transaction is executed.)
state	text	State of the load operation, such as `STARTING` , `DONE` , or `ERROR` .
error_string	text	Error message for a failed load.
table_id	bigint	The unique ID for the table. Tables in different schemas within the same database may have the same name, so you can use this ID to uniquely identify a table.
schema_id	bigint	Unique ID of the schema that contains the table being loaded.
database_id	bigint	Unique ID of the database that contains the table being loaded.
user_id	bigint	Unique ID of the user who ran the `ybload` command.
session_id	bigint	Session ID; foreign key to the `sys.session` and `sys.log_session` views.
table_name	text	Name of the table being loaded.
schema_name	text	Name of the schema that the table belongs to.
database_name	text	Name of the database that the table belongs to.
username	text	Name of the user who ran the `ybload` command.
client_hostname	text	Host name of the client system where the load was started.
client_username	text	Client OS user running the `ybload` command.
start_time	timestamptz	When the `ybload` command started running.
end_time	timestamptz	When the `ybload` command completed its execution.
elapsed_ms	decimal(18,3)	Duration of the load, in milliseconds.
remaining_ms	decimal(18,3)	Estimated number of milliseconds remaining to complete the load. (This value is always 0.0 for completed loads in this view.)
transaction_first	bigint	First transaction ID for the load. Bulk load operations may be split into multiple transactions.
transaction_last	bigint	Last transaction ID for the load. Bulk load operations may be split into multiple transactions.
num_transactions	bigint	Total number of transactions for the load operation.

Column Name	Data Type	Description
parsed_rows	bigint	Number of rows that have been parsed.
parsed_bytes	bigint	Amount of table data read from the source file (in uncompressed bytes).
parsed_rows_per_second	double precision	Number of rows parsed per second.
parsed_bytes_per_second	double precision	Amount of table data read from the source file per second (in uncompressed bytes).
inserted_rows	bigint	The number of rows that were loaded successfully into the table.
inserted_bytes	bigint	Amount of table data written to the table (in compressed bytes). `inserted_bytes` may be greater than `parsed_bytes`, given the overhead of the size of a shard (unit of data storage).
inserted_rows_per_second	double precision	Number of rows that were loaded successfully per second.
inserted_bytes_per_second	double precision	Amount of table data written to the table per second, in bytes.
error_rows	bigint	The number of rows that could not be loaded into the table (see the `.bad` rows file for details).
sent_bytes	bigint	Number of bytes transferred over the network during the load.
sent_bytes_per_second	double precision	Number of bytes transferred per second during the load.
tags	text	Query tags in the system, as defined with a `SET YBD_QUERY_TAGS` command. Tags may be logged based on WLM rule processing. `NULL` if the query received no tags.

Examples

The following example shows information for a load of 26 million rows into the `newmatchstats` table. (Expanded `ybsql` display is used in this example for readability.)

```
premdb=# select * from sys.log_load where table_name = 'newmatchstats';
-[ RECORD 1 ]--------------+----------------------------------------------------------------
session_key                | A0iOTdPuAXDBlGY_IJcXrFZXoxncCDlG1Spo5477TVa9P-Y9IE8k30FarQ9iCx1e
state                      | DONE
error_string               | [NULL]
table_id                   | 16496
schema_id                  | 2200
database_id                | 16404
user_id                    | 16415
session_id                 | 18913
table_name                 | newmatchstats
schema_name                | public
database_name              | premdb
username                   | bobr
client_hostname            | ybload100
client_username            | ybuser100
start_time                 | 2018-03-26 16:42:09.233-07
end_time                   | 2018-03-26 16:42:16.012-07
elapsed_ms                 | 6779.000
remaining_ms               | 0.000
transaction_first          | 179110
transaction_last           | 179110
num_transactions           | 1
parsed_rows                | 25990128
parsed_bytes               | 659797724
```

```
parsed_rows_per_second    | 3820717.46658389
parsed_bytes_per_second   | 96994546.8717621
inserted_rows             | 25990128
inserted_bytes            | 727723568
inserted_rows_per_second  | 3833917.66216401
inserted_bytes_per_second | 107349692.180362
error_rows                | 0
sent_bytes                | 727723568
sent_bytes_per_second     | 106980086.712245

premdb=# select count(*) from newmatchstats;
  count
----------
 25990128
(1 row)
```

The following example returns the duration, in seconds, of the same load into the same table:

```
premdb=# select table_id, start_time, end_time, trunc(elapsed_ms/1000,2) as duration, inserted_rows, error_rows, inserted_bytes
from sys.log_load where table_name='newmatchstats' and session_id=18913;
 table_id |         start_time         |          end_time          | duration | inserted_rows | error_rows | inserted_bytes
----------+----------------------------+----------------------------+----------+---------------+------------+----------------
   16496 | 2018-03-26 16:42:09.233-07 | 2018-03-26 16:42:16.012-07 |     6.77 |      25990128 |          0 |      727723568
(1 row)
```

sys.log_query

This view records the history of SQL operations and bulk loads performed over the past seven days (by default).

Note: The statistical values recorded in this view are also available via the **Query Details** tabs in Yellowbrick Manager.

User accounts only see the queries they submit. Superuser accounts see all queries. (Commands executed by internal superusers, such as `yellowbrick`, are not logged.)

Column Name	Data Type	Description
query_id	bigint	Unique query ID. When queries are restarted, multiple entries are logged to `sys.log_query` under the same query ID. In `sys.query`, a query ID always identifies one unique row. Query IDs are logged for `ybload` operations as well as SQL queries.
session_id	bigint	Unique ID of the session. Foreign key reference to sys.session and sys.log_session.
transaction_id	bigint	Unique transaction ID. `NULL` if transaction context has not yet been acquired.
plan_id	varchar(64)	Unique query plan ID.
state	varchar(50)	The current state of the query. In `sys.log_query`, this state is always a completion state: `done`, `cancel`, `error`. In `sys.query`, this state may be any of the states in the query lifecycle. When the `BLOCK` row store action is set for a table, inserts tracked by `sys.query` are always in the `plan` state. See ALTER TABLE.
blocked	varchar(255)	The reason the query is blocked; `NULL` if not blocked. For example, this column contains the lock requested when a query is waiting for an object-level lock. **Note:** This column is only found in `sys.query`; it is not needed in `sys.log_query`.
username	varchar(128)	User who ran the query.
application_name	varchar(128)	Name of the application that ran the query, such as `ybsql`, `ym`, or `ybload`. The application name for replication operations is `replication service`.
database_name	varchar(128)	Name of the database where the query was run. (The database ID is not tracked because the database may be dropped when the query is complete and recorded in `sys.log_query`.)
cluster_name	varchar(128)	Name of the virtual compute cluster that was used for the query.
type	varchar(128)	Type of query or operation, such as `select`, `insert`, `update`, `delete`, `load`, `ctas`, `drop table`, `backup`, `restore`. The `maintenance` type refers to background system operations.
tags	varchar(128)	Query tags in the system, as defined with a `SET YBD_QUERY_TAGS` command. Tags may be logged based on WLM rule processing. `NULL` if the query received no tags.
error_code	varchar(5)	Yellowbrick error code.

Column Name	Data Type	Description
error_message	varchar(255)	Text of the error message, if any.
query_text	varchar(60000)	SQL text of the query. For readability the text is broken into multiple lines, with each line separated by a plus sign (+).
pool_id	varchar(128)	Resource pool that the query was assigned to.
priority	varchar(30)	WLM query priority: `normal`, `low`, `high`, or `critical`.
slot	bigint	Execution slot number for the query, within its resource pool. Each pool has a fixed maximum and minimum number of slots.
num_workers	integer	Total number of compute nodes that participated in, or are participating in, running the query. `NULL` until the query leaves the `assemble` or `acquire_resources` state. Set to `1` for single-compute node queries after the `assemble` phase. For multi-compute node queries, the actual compute node count is not set until after the `acquire_resources` phase. This column is set to `0` for front-end queries, including internal system queries.
longest_worker_id	varchar(38)	The compute node that is currently taking the longest time to finish executing this query (or the compute node that was the last to finish when the query was done). `NULL` until the query starts executing. In a distributed system, compute nodes may run for different durations. This column helps identify which of the compute nodes matter from a tuning perspective. The "longest-running compute node" is the compute node that finished last when the query was completed and moved to `sys.log_query`. While the query is running, the longest-running compute node is either the compute node with the highest `io_spill_space_bytes` (if spilling is occurring) or the compute node with the highest `run_ms` value (if spilling is not occurring).
compile_percent	integer	The amount of resources (as a percentage of the total used) that the query used from when it started compiling until the last fragment was done. `NULL` until the query enters the `compile` state.
cpu_percent	integer	How effectively the CPU was used on the compute node that dominated runtime (the longest compute node). `(run_ms - wait_cpu_ms - wait_io_ms) / (run_ms - wait_cpu_ms)`
cpu_percent_max	integer	Calculated in the same way as `cpu_percent`, but using the largest CPU % of the cluster. For a perfectly non-skewed execution, this value will be the same as `cpu_percent`. For single-compute node queries, this value should be equal to `cpu_percent` on that compute node.
num_restart	integer	Number of restarts. When the query is restarted or moved by a user process, this number is incremented by 1. Initialized to 0. This value is incremented when: - A WLM rule triggers a `w.moveToResourcePool()` action, the target pool is full or has insufficient resources, and the query must restart. - A WLM rule triggers a `w.restartInResourcePool()` action. - A WLM profile change is activated (with the option to cancel running queries), the query is currently running, and is restarted. - An administrator runs a SQL `MOVE` or `RESTART` command. See also Moving Queries and Restarting Queries.
num_error	integer	Number of error-driven restarts. When the query errors out and is restarted by an error-recovery process, this number is incremented by 1. Initialized to 0. See also the description for the `num_restart` column.
parse_ms	decimal(18,3)	Time spent parsing the query text and acquiring locks. Initialized to 0 when the query is submitted.
wait_parse_ms	decimal(18,3)	Time spent waiting for the parser to become available. Initialized to 0 when the query is submitted.

Column Name	Data Type	Description
wait_lock_ms	decimal(18,3)	Time spent blocked while waiting for locks during the parse/lock phase. Initialized to 0 when the query is submitted.
plan_ms	decimal(18,3)	Time spent generating the query plan. This measurement does not include time spent generating the XML serialization for the query compiler. Initialized to 0 when the query is submitted.
wait_plan_ms	decimal(18,3)	Time spent waiting for the query planner to become available. Initialized to 0 when the query is submitted.
assemble_ms	decimal(18,3)	Time spent assembling the query. The time starts just before the XML is serialized in the front-end database and ends when the generated C++ code is available. This time may be close to 0 when the query plan is found in the cache. `NULL` before entering the `assemble` state, then initialized to 0.
wait_assemble_ms	decimal(18,3)	Time spent waiting for assembly/preparation steps (XML serialisation, AST analysis, and C++ code generation) to be available to the query. This value also measures time where the assemble steps did not run for various reasons (such as waiting for internal locks). `NULL` before entering the queue, then initialized to 0.
compile_ms	decimal(18,3)	Duration of time when at least one query fragment is being compiled. The timer starts when the first fragment begins to compile and ends when the last fragment has finished compiling. `NULL` before compilation, then initialized to 0. Also 0 if the query is found in the compile cache.
wait_compile_ms	decimal(18,3)	Duration of time when at least one fragment could have been compiled, but it did not start compiling because the compile resource was not available. `NULL` before entering the `compile` state, then initialized to 0.
acquire_resources_ms	decimal(18,3)	How long it took to acquire resources to run the query, such as memory, spill space, and compute node slots. `NULL` before entering the `acquire_resources` state, then initialized to 0.
run_ms	decimal(18,3)	How long the query was running on the *longest-running* compute node. This value represents the total time the query spent in the `run` state. Note that wait-time statistics are included in the `run_ms` total: `run_ms = wait_run_cpu_ms + wait_run_io_ms + wait_run_spool_ms` To get the true CPU consumption for the query (how much time the CPU spent doing actual work), calculate: `run_ms - wait_run_cpu_ms - wait_run_io_ms` The `run_ms` value is `NULL` before the query enters the `run` state, then it is initialized to `0`.
wait_run_cpu_ms	decimal(18,3)	Time spent waiting for the CPU to become available. The query is not actively running though in the `run` state. This measurement is taken on the longest-running compute node. `NULL` before entering the `run` state, then initialized to 0.
wait_run_io_ms	decimal(18,3)	Time spent waiting for I/O or other operations while the query is scheduled to run and is in the `run` state. This measurement is taken on the longest-running compute node. `NULL` before entering the `run` state, then initialized to 0.

Column Name	Data Type	Description
wait_run_spool_ms	decimal(18,3)	Time spent spooling the data when the compute node has finished running the query. The time spent waiting for spool overlaps with the runtime of the query; spooling may occur while other runtime work is being done. `NULL` until spooling starts.
client_ms	decimal(18,3)	Duration of time from when the query finishes spooling data to when the client receives the last row in the result set.
wait_client_ms	decimal(18,3)	Total time spent waiting for the client. This time only starts counting when the last row has been spooled. This time goes up when:- Rows are not sent to the client because the client has not requested them. - Spooling is blocked (out of spool space) until the client has received more data. `NULL` before entering the `client_wait` state, then initialized to 0.
total_ms	decimal(18,3)	The total duration of the query. The following timings overlap and do not contribute to the `total_ms` measurement: `cancel_ms`, `restart_ms`, `wlm_runtime_ms`, and `spool_ms`. `NULL` until the query is in the `done` state.
cancel_ms	decimal(18,3)	Time spent waiting for the query to move out of the `cancel` state; how long it takes to completely cancel and tear down the query.
restart_ms	decimal(18,3)	Time spent waiting for a query to restart, once requested.
wlm_runtime_ms	decimal(18,3)	How long it takes to evaluate WLM runtime rules.
spool_ms	decimal(18,3)	How long it takes for the entire result set to be sent to the manager node for consumption by the client. This column returns `NULL` for queries that do not spool and returns `NULL` before the query starts spooling.
cache_efficiency	decimal(18,3)	Efficiency of the data cache, expressed as a percentage, where 100% means that all of the data blocks for the query were cached (held in local NVMe SSDs).
submit_time	timestamptz	When the query was submitted by the user. This timestamp remains constant during restarts.
done_time	timestamptz	When the query finished (and moved to the `done` state).
state_time	timestamptz	When the query entered the current state. This value is set to the `submit_time` when the query first enters the system. If the query restarts, this is the timestamp when the restart was initiated.
restart_time	timestamptz	When the query was restarted. `NULL` for queries that were not restarted.
io_read_bytes	bigint	Number of bytes read by the query (aggregated across all compute nodes). This value does not include temporary storage reads.
io_write_bytes	bigint	Number of bytes written by the query (aggregated across all compute nodes). This value does not include temporary storage writes.
io_spill_read_bytes	bigint	Number of bytes the query has read back from temporary storage on disk (aggregate of all spilling reads). This number may be lower than `io_spill_write_bytes` (for example, if the query has a `LIMIT` clause). This column returns `0` for queries that run on the compute nodes without spilling. For queries that did not reach the compute nodes, this column returns `NULL`.
io_spill_write_bytes	bigint	Number of bytes the query has spilled to disk, using temporary storage (aggregate of all spilling writes). This column returns `0` for queries that run on the compute nodes without spilling. For queries that did not reach the compute nodes, this column returns `NULL`.

Column Name	Data Type	Description
io_network_bytes	bigint	Number of bytes moved internally over the network on the cluster (manager node to compute nodes and vice versa). This number does not include data being returned to the client.
io_client_read_bytes	bigint	Total number of bytes sent by the client to the server (for example, via a `ybsql \copy` or `ybload` command).
io_client_write_bytes	bigint	Total number of bytes sent to the client.
io_spool_write_bytes	bigint	Total size of spool space written by the query.
rows_inserted	bigint	Number of rows inserted by the query (for `UPDATE`, `INSERT`, `CTAS`, `ybload`, `\ycopy` operations).
rows_deleted	bigint	Number of rows deleted by the query. For `UPDATE` statements, both `rows_deleted` and `rows_inserted` are both `>0`.
rows_returned	bigint	Number of rows the query returned to the client.
memory_bytes	bigint	Amount of memory consumed by the longest-running compute node that ran this query.
memory_bytes_max	bigint	Maximum amount of memory used by any compute node that ran this query.
io_spill_space_bytes	bigint	Amount of temporary space consumed by the longest-running compute node that ran this query. This value is very useful when you are setting up spill space allocation for WLM resource pools.
io_spill_space_bytes_max	bigint	The maximum amount of temporary space consumed by any compute node that ran this query.
io_spill_space_granted_bytes	bigint	The amount of spill space allowed by the query. `NULL` for front-end queries and queries that have not yet been granted space.
memory_estimated_bytes	bigint	The estimated memory *per compute node* from the query plan. `NULL` until the query exits the `compile` state.
memory_required_bytes	bigint	The minimum amount of memory required *per compute node* to start the query. `NULL` until the query exits the `compile` state.
memory_granted_bytes	bigint	Amount of memory *per compute node* actually given to the query (during the `acquire_resources` phase) in its assigned resource pool and execution slot. `NULL` until the query enters the `acquire_resources` state.
memory_estimate_confidence	varchar(16)	`None` (no confidence), `Low`, `High`, or `Unknown`. These values describe the degree to which the estimated memory is thought to be reliable.

Examples

You can monitor query activity at the cluster level with the `sys.query` and `sys.log_query` views, using the `cluster_name` column. For example:

```
premdb=> select username, application_name, database_name, cluster_name, rows_returned, submit_time
from sys.log_query
where
database_name='premdb'
and rows_returned >0
and submit_time like '2022-04-27%'
and cluster_name like 'bobrum%';
 username  |          application_name          | database_name |      cluster_name       | rows_returned |          submit_time
-----------+------------------------------------+---------------+-------------------------+---------------+-------------------------
 ybtools10 | DBeaver 22.0.3 - Main <premdb>     | premdb        | bobrum-beta1-cluster-01 |             1 | 2022-04-27 22:30:24.96
 ybtools10 | DBeaver 22.0.3 - Metadata <premdb> | premdb        | bobrum-beta1-cluster-01 |             1 | 2022-04-27 22:30:26.21
 ybtools10 | DBeaver 22.0.3 - Metadata <premdb> | premdb        | bobrum-beta1-cluster-01 |             1 | 2022-04-27 22:30:26.73
 ybtools10 | ybsql                              | premdb        | bobrum-beta1-cluster-01 |             1 | 2022-04-27 21:29:24.14
 ybtools10 | ybsql                              | premdb        | bobrum-beta1-cluster-01 |             1 | 2022-04-27 21:22:50.66
 ybtools10 | ybsql                              | premdb        | bobrum-beta1-cluster-01 |            14 | 2022-04-27 21:18:07.95
 ybtools10 | ybsql                              | premdb        | bobrum-beta1-cluster-01 |            13 | 2022-04-27 21:17:32.58
```

```
 ybtools10 | ybsql                          | premdb       | bobrum-beta1-cluster-01 |           22 | 2022-04-27 21:14:34.36
(8 rows)
```

Return the total running time for queries run by a specific user:

```
premdb=> select query_id, state, username, total_ms, substr(query_text,1,50)
from sys.log_query where username='bobr' order by query_id desc;
 query_id | state | username | total_ms |                      substr
----------+-------+----------+----------+-----------------------------------------------------
   248790 | done  | bobr     |  293.061 | select t1.season_name, t1.winners, homegoals+awayg
   247096 | done  | bobr     |  115.999 | select season_name, numteams, sum(substr(ftscore,1
(3 rows)
```

Return runtime statistics and the SQL text for `UPDATE` queries run by a single user:

```
premdb=> select query_id, username, application_name, type, state, substr(query_text,1,25) qrytxt, parse_ms, plan_ms, assemble_ms,
from sys.log_query
where type='update' and username='bobr'
order by 1 desc;
 query_id | username | application_name |  type  | state |          qrytxt           | parse_ms | plan_ms | assemble_ms | compile_
----------+----------+------------------+--------+-------+---------------------------+----------+---------+-------------+---------
  1143640 | bobr     | ybsql            | update | done  | update newmatchstats set  |    0.852 |   2.345 |      49.589 |   1474.5
  1141949 | bobr     | ybsql            | update | done  | update newmatchstats set  |    0.528 |   2.475 |      27.465 |   1319.9
  1130126 | bobr     | ybsql            | update | done  | update newmatchstats set  |    0.492 |   2.377 |      21.911 |      1.6
  1095945 | bobr     | ybsql            | update | done  | update newmatchstats set  |    0.380 |   1.504 |      26.710 |      0.6
(4 rows)
```

Join the `sys.log_query`, `sys.log_query_analyze`, and `sys.log_query_explain` views to return statistics and plan information for a single query.

```
premdb=> select a.*, e.query_plan
from sys.log_query_analyze a join sys.log_query q on q.query_id = a.query_id join sys.log_query_explain e on e.plan_id = q.plan_id
where q.query_id = 258345
order by node_id;
-[ RECORD 1 ]--------+-----------------------------------------------------------------------------------------------------------------
query_id             | 258345
node_id              | 0
rows_planned         | 774540
rows_actual          | 0
memory_planned_bytes | 0
memory_actual_bytes  | 524288
io_read_bytes        | 0
io_write_bytes       | 0
io_network_bytes     | 0
io_network_count     | 0
runtime_ms           | 0.000
skew                 | 0
detail               | [NULL]
query_plan           | SEQUENCE
                     | distribute none
-[ RECORD 2 ]--------+-----------------------------------------------------------------------------------------------------------------
query_id             | 258345
node_id              | 1
rows_planned         | 774540
rows_actual          | 0
memory_planned_bytes | 8388696
memory_actual_bytes  | 524464
io_read_bytes        | 0
io_write_bytes       | 0
io_network_bytes     | 0
io_network_count     | 8
runtime_ms           | 0.000
skew                 | 0
```

```
detail               | [NULL]
query_plan           | WRITE ROW STORE
                     | (newmatchstats.seasonid, newmatchstats.matchday, newmatchstats.htid, newmatchstats.atid, newmatchstats.momen
                     | distribute (newmatchstats.seasonid)
-[ RECORD 3 ]--------+-----------------------------------------------------------------------------------------------------------
query_id             | 258345
node_id              | 2
rows_planned         | 774540
rows_actual          | 0
memory_planned_bytes | 3221225472
memory_actual_bytes  | 495452160
io_read_bytes        | 0
io_write_bytes       | 182235136
io_network_bytes     | 0
io_network_count     | 0
runtime_ms           | 0.000
skew                 | 0
detail               | [NULL]
query_plan           | INSERT INTO newmatchstats
                     | (newmatchstats.seasonid, newmatchstats.matchday, newmatchstats.htid, newmatchstats.atid, newmatchstats.momen
                     | distribute (newmatchstats.seasonid)
-[ RECORD 4 ]--------+-----------------------------------------------------------------------------------------------------------
query_id             | 258345
node_id              | 3
rows_planned         | 774540
rows_actual          | 49570560
memory_planned_bytes | 2097152
memory_actual_bytes  | 1048576
io_read_bytes        | 0
io_write_bytes       | 0
io_network_bytes     | 0
io_network_count     | 0
runtime_ms           | 4.854
skew                 | 0.18614919823379
detail               | [NULL]
query_plan           | APPEND SCAN
                     | (newmatchstats.seasonid, newmatchstats.matchday, newmatchstats.htid, newmatchstats.atid, newmatchstats.momen
                     | distribute (newmatchstats.seasonid)
...
```

Return the 10 longest-running operations in terms of total running time:

```
premdb=> select query_id, type, submit_time, done_time, total_ms, rank() over(order by total_ms desc) from sys.log_query limit 10;
 query_id |  type  |        submit_time         |         done_time          | total_ms | rank
----------+--------+----------------------------+----------------------------+----------+------
   258345 | insert | 2020-09-09 17:59:01.871-07 | 2020-09-09 17:59:07.801-07 | 5930.091 |    1
   116626 | select | 2020-09-09 16:28:15.35-07  | 2020-09-09 16:28:18.41-07  | 3059.581 |    2
   258043 | insert | 2020-09-09 17:58:57.191-07 | 2020-09-09 17:59:00.22-07  | 3029.446 |    3
   140372 | select | 2020-09-09 16:47:55.137-07 | 2020-09-09 16:47:58.087-07 | 2950.050 |    4
   130453 | select | 2020-09-09 16:39:08.884-07 | 2020-09-09 16:39:11.778-07 | 2894.107 |    5
   234651 | select | 2020-09-09 17:48:01.518-07 | 2020-09-09 17:48:04.087-07 | 2568.231 |    6
   282685 | select | 2020-09-09 18:11:07.228-07 | 2020-09-09 18:11:09.47-07  | 2241.723 |    7
   276565 | select | 2020-09-09 18:08:03.255-07 | 2020-09-09 18:08:05.291-07 | 2036.606 |    8
   265009 | select | 2020-09-09 18:02:18.264-07 | 2020-09-09 18:02:20.3-07   | 2035.756 |    9
   150281 | select | 2020-09-09 16:52:50.278-07 | 2020-09-09 16:52:52.312-07 | 2033.339 |   10
(10 rows)
```

Return information for a query that was restarted in a different pool:

```
yellowbrick=# yellowbrick=# select query_id, transaction_id, state, pool_id, num_restart, submit_time, done_time, restart_ms, tota
 query_id | transaction_id |  state  |      pool_id      | num_restart |        submit_time         |         done_time          |
----------+----------------+---------+-------------------+-------------+----------------------------+----------------------------+
  2573711 |     4294974926 | done    | bobr_pool         |           1 | 2020-09-18 16:56:28.829-07 | 2020-09-18 17:01:22.792-07 |
```

```
   2573711 |     4294974926 | restart | bobr_profile: mix |          0 | 2020-09-18 16:56:28.829-07 | 2020-09-18 16:56:51.945-07 |
(2 rows)
```

sys.log_query_alert

Log alerts that are triggered specifically by WLM rules that have a `Log ERROR` or `Log WARN` action.

Column Name	Data Type	Description
alert_id	uuid	Unique ID for the alert.
query_id	bigint	Unique query ID associated with this alert.
alert_time	timestamptz	Timestamp when the alert occurred.
acknowledged	boolean	N/A (always `false`).
source	varchar(255)	The name of the WLM rule that triggered the alert.
severity	integer	Severity gradient, where `0`=OK, `25`=Informational, `50`=Minor, `75`=Major, `100`=Critical.
message	varchar(63488)	Text of the alert.

For example:

```
premdb=# select * from sys.log_query_alert;
              alert_id               | query_id |         alert_time         | acknowledged |      source       | severity |    me
-------------------------------------+----------+----------------------------+--------------+-------------------+----------+-----
 e44d60f0-5cab-48da-b0d0-77e7262cf809 |    49040 | 2018-09-11 14:55:04.421-07 | f            | Bob's premdb rule |       50 | prem
 17e4a5d0-c985-4627-8397-0b0691e3a6f4 |    48998 | 2018-09-11 14:55:04.356-07 | f            | Bob's premdb rule |       50 | prem
 6830c650-a5a8-4c33-af7e-705fbdd6ee0e |    52054 | 2018-09-11 14:57:20.431-07 | f            | Bob's premdb rule |       50 | prem
 02fb578d-bb94-4324-9af1-5e58c11c61c2 |    49029 | 2018-09-11 14:55:04.394-07 | f            | Bob's premdb rule |       50 | prem
(4 rows)
```

sys.log_query_analyze

Platforms: All platforms

Parent topic: System Views

This view records runtime statistics for each execution step (or node) in the query plan. A unique row in this table is identified by a combination of the query ID and node ID. *All of the reported statistics are per plan node.*

For descriptions of the columns in this view, see sys.query_analyze. Note that the `sys.log_query_analyze` view does not have a `worker_id` column.

Example

```
premdb=# select query_id, node_id, rows_actual, runtime_ms from sys.log_query_analyze where query_id=1078487 order by node_id;
 query_id | node_id | rows_actual | runtime_ms
----------+---------+-------------+------------
  1078487 |       0 |          19 |      0.000
  1078487 |       2 |          19 |      0.187
  1078487 |       3 |          19 |      0.083
  1078487 |       4 |          19 |      0.100
  1078487 |       5 |          19 |      0.100
  1078487 |       6 |          19 |      0.105
  1078487 |       7 |          19 |      0.288
  1078487 |       8 |          19 |      0.080
  1078487 |       9 |           0 |      0.094
  1078487 |      10 |          22 |      0.155
  1078487 |      11 |          25 |      0.279
  1078487 |      12 |          22 |      0.050
  1078487 |      13 |           0 |      0.106
  1078487 |      14 |           0 |      1.417
  1078487 |      15 |           0 |      1.414
  1078487 |      16 |        7220 |      0.447
  1078487 |      17 |        7220 |      0.019
  1078487 |      18 |        7220 |      0.811
  1078487 |      19 |        7220 |      0.588
  1078487 |      20 |        7220 |      0.019
  1078487 |      22 |           0 |      1.623
(21 rows)
```

sys.log_query_explain

This view records the plan for each query and contains a row for each node in the plan. A unique row in this table is identified by a combination of plan ID and node ID. The rows in the table map to the node-level text output that is returned when you run an EXPLAIN ANALYZE command.

Column Name	Data Type	Description
plan_id	varchar(64)	Query plan ID. You can join this column to the `plan_id` column in `sys.log_query`.
node_id	integer	Node ID within the query plan, representing an execution step.
parent_id	bigint	Parent node ID, to show the parent-child relationships among nodes within a single query plan.
index	bigint	Zero-based index for the nodes in a plan.
type	varchar(255)	Type of execution step, such as `hash distribute`, `filter`, or `limit`.
workers	varchar(6)	Workers processing the plan node: `single` or `all` for each node of the query plan
query_plan	varchar(64000)	Query plan text for each node

Examples

```
premdb=# select plan_id, node_id, substr(query_plan, 1, 50)
from sys.log_query_explain order by plan_id, node_id;
                plan_id                   | node_id |                       substr
------------------------------------------+---------+---------------------------------------------------
 0XeN2C-FjU6lxnkPh9ry4jOuU9Ad+s5O0dNO+T7uOdA= |       0 | SELECT                                           +
                                          |         | (match.matchday, DATEADD(month, 1, match.ma
 0XeN2C-FjU6lxnkPh9ry4jOuU9Ad+s5O0dNO+T7uOdA= |       1 | APPEND SCAN                                      +
                                          |         | (match.matchday, DATEADD(month, 1, mat
 0XeN2C-FjU6lxnkPh9ry4jOuU9Ad+s5O0dNO+T7uOdA= |       2 | | SCAN match                                     +
                                          |         |   (match.matchday, match.atid, match.
 0XeN2C-FjU6lxnkPh9ry4jOuU9Ad+s5O0dNO+T7uOdA= |       3 | |-TRANSPOSE                                      +
                                          |         |   (match.matchday, DATEADD(month, 1, m
 0XeN2C-FjU6lxnkPh9ry4jOuU9Ad+s5O0dNO+T7uOdA= |       4 | |-DISTRIBUTE RANDOM                              +
                                          |         |   (match.matchday, DATEADD(mon
 0XeN2C-FjU6lxnkPh9ry4jOuU9Ad+s5O0dNO+T7uOdA= |       5 |   SCAN ROW STORE                                 +
                                          |         |   (match.matchday, match.htid, ma
 0XeN2C-FjU6lxnkPh9ry4jOuU9Ad+s5O0dNO+T7uOdA= |       6 |   FILTER WHERE ((match.htid::INT4 = $2) AND (match
 1LOUm6k0L5L+xZU-duWpJjzPtHWXsNLIpwNKR4FXL4k= |       0 | WRITE ROW STORE                                  +
                                          |         | (team.atid, team.name)                           +
                                          |         | distribute
 1LOUm6k0L5L+xZU-duWpJjzPtHWXsNLIpwNKR4FXL4k= |       1 | INSERT INTO awayteam                              +
                                          |         | (team.atid, team.name)                           +
                                          |         | distri
 1LOUm6k0L5L+xZU-duWpJjzPtHWXsNLIpwNKR4FXL4k= |       2 | APPEND SCAN                                      +
                                          |         | (team.atid, team.name)                           +
                                          |         | distribute repl
 1LOUm6k0L5L+xZU-duWpJjzPtHWXsNLIpwNKR4FXL4k= |       3 | | SCAN team                                      +
                                          |         |   (team.atid, team.name)                         +
```

```
                                          |         |   distribute
...
```

The following example joins to the `sys.log_query` view to return plan information for a query based on its specific SQL text:

```
premdb=# select slq.plan_id, slqe.node_id, slqe.type from sys.log_query_explain slqe, sys.log_query slq
where slq.plan_id=slqe.plan_id and slq.query_text like 'select season_name, numteams, sum%';
                 plan_id                  | node_id |         type
------------------------------------------+---------+----------------------
 WnbWYG5kKxbVuJOPqBB0p7fd5ViubYLAHFxV0akPIdA= |       0 | SELECT
 WnbWYG5kKxbVuJOPqBB0p7fd5ViubYLAHFxV0akPIdA= |       0 | SELECT
 WnbWYG5kKxbVuJOPqBB0p7fd5ViubYLAHFxV0akPIdA= |       2 | DISTRIBUTE SORT
 WnbWYG5kKxbVuJOPqBB0p7fd5ViubYLAHFxV0akPIdA= |       2 | DISTRIBUTE SORT
 WnbWYG5kKxbVuJOPqBB0p7fd5ViubYLAHFxV0akPIdA= |       3 | SORT
 WnbWYG5kKxbVuJOPqBB0p7fd5ViubYLAHFxV0akPIdA= |       3 | SORT
 WnbWYG5kKxbVuJOPqBB0p7fd5ViubYLAHFxV0akPIdA= |       4 | EXPRESSION
 WnbWYG5kKxbVuJOPqBB0p7fd5ViubYLAHFxV0akPIdA= |       4 | EXPRESSION
 WnbWYG5kKxbVuJOPqBB0p7fd5ViubYLAHFxV0akPIdA= |       5 | GROUP BY
 WnbWYG5kKxbVuJOPqBB0p7fd5ViubYLAHFxV0akPIdA= |       5 | GROUP BY
 WnbWYG5kKxbVuJOPqBB0p7fd5ViubYLAHFxV0akPIdA= |       6 | DISTRIBUTE HASH
 WnbWYG5kKxbVuJOPqBB0p7fd5ViubYLAHFxV0akPIdA= |       6 | DISTRIBUTE HASH
 WnbWYG5kKxbVuJOPqBB0p7fd5ViubYLAHFxV0akPIdA= |       7 | GROUP BY PARTIAL
 WnbWYG5kKxbVuJOPqBB0p7fd5ViubYLAHFxV0akPIdA= |       7 | GROUP BY PARTIAL
 WnbWYG5kKxbVuJOPqBB0p7fd5ViubYLAHFxV0akPIdA= |      14 | INNER JOIN
 WnbWYG5kKxbVuJOPqBB0p7fd5ViubYLAHFxV0akPIdA= |      14 | INNER JOIN
 WnbWYG5kKxbVuJOPqBB0p7fd5ViubYLAHFxV0akPIdA= |      16 | DISTRIBUTE HASH
 WnbWYG5kKxbVuJOPqBB0p7fd5ViubYLAHFxV0akPIdA= |      16 | DISTRIBUTE HASH
 WnbWYG5kKxbVuJOPqBB0p7fd5ViubYLAHFxV0akPIdA= |      15 | APPEND SCAN
 WnbWYG5kKxbVuJOPqBB0p7fd5ViubYLAHFxV0akPIdA= |      15 | APPEND SCAN
...
```

sys.log_replica_status

Platforms: All platforms

Parent topic: System Views

This system view records the history of replication activity on the source system.

See the sys.replica_status description for details about the columns in the `sys.log_replica_status` view.

Examples

Return records for replication cycles that have sent bytes from the source system to the target system:

```
premdb=# select * from sys.log_replica_status where sent_bytes >0 order by sent_bytes desc;
-[ RECORD 1 ]-------+------------------------------------------------------------------
database_id         | 16391
replica_id          | 16415
snapshot_id         | premdb_replica1_19_12_12_17_34_37
elapsed_ms          | 223745
remaining_ms        | 0
start_time          | 2019-12-12 17:34:37.123-08
end_time            | 2019-12-12 17:38:20.868-08
state               | DONE
error_string        | [NULL]
backup_session_key  | DmabhirRiu7r_o0DEH4YguDjg0A0oBnESEKjvH40CXVP8W4Ug1ODvtFD-2YVbjnd
restore_session_key | AxtTYw1nAb0qqRFi93rSEA0PWU9K4O-d2RRGm2fNkZg34qyHwHZ5s23wevDI51iI
sent_bytes          | 5815083399
write_bytes         | 5815083399
retries             | 0
-[ RECORD 2 ]-------+------------------------------------------------------------------
database_id         | 16391
replica_id          | 16415
snapshot_id         | premdb_replica1_19_12_12_17_29_59
elapsed_ms          | 47561
remaining_ms        | 0
start_time          | 2019-12-12 17:29:59.848-08
end_time            | 2019-12-12 17:30:47.409-08
state               | DONE
error_string        | [NULL]
backup_session_key  | Dn5KkhJ0_5Wt_7qahaFAFYVCCFDJ-3Y3dtBRsvqNsD1hzTNRdWjJo61VxDTzLbLD
restore_session_key | BAZOiGXIj8NtxayHGrUPtOZSBxsuKg0ykj7jtsTdCuQKUb-Yd7_tPH5gfm2a0m18
sent_bytes          | 968731450
write_bytes         | 968731450
retries             | 0
-[ RECORD 3 ]-------+------------------------------------------------------------------
database_id         | 16391
replica_id          | 16415
snapshot_id         | premdb_replica1_19_12_12_17_29_13
elapsed_ms          | 46109
remaining_ms        | 0
start_time          | 2019-12-12 17:29:13.729-08
end_time            | 2019-12-12 17:29:59.838-08
state               | DONE
error_string        | [NULL]
backup_session_key  | AeL5WMy1W36EBiv0ea0BKhYxW0sV2Xlmrvzg0ucLUIKcg2JxlsU4zFnlBfIBWcbM
restore_session_key | CWW8BbTi2bi9KWq-GdupA9fYok_B1anAGy5fuA8OcsDlPD_GHN01FRftqet0_zFF
sent_bytes          | 413449071
```

```
write_bytes         | 413449071
retries             | 0
...
```

Return the five longest-running replication cycles for a replicated database named `premdb_hot`. This query joins three system views in order to return the source database name and the alias (target) database name.

```
premdb=# select d.database_id, d.name as srcdbname, r.replica_id, r.alias, elapsed_ms, sent_bytes
from
sys.log_replica_status lrs,
sys.database d,
sys.replica r
where lrs.database_id=d.database_id
and lrs.replica_id=r.replica_id
and r.alias='premdb_hot'
order by elapsed_ms desc
limit 5;
 database_id | srcdbname | replica_id |   alias    | elapsed_ms | sent_bytes
-------------+-----------+------------+------------+------------+------------
       16432 | premdb    |      16460 | premdb_hot |      48158 | 8433800688
       16432 | premdb    |      16460 | premdb_hot |      36261 | 6272871179
       16432 | premdb    |      16460 | premdb_hot |      13666 |    8210360
       16432 | premdb    |      16460 | premdb_hot |      12133 | 1956897451
       16432 | premdb    |      16460 | premdb_hot |       2138 |  120741116
(5 rows)
```

sys.log_restore

This view records the history of restore (`ybrestore`) operations.

Column Name	Data Type	Description
session_key	text	Unique session key generated for the restore operation.
session_id	bigint	Unique ID of the `ybrestore` session.
client_hostname	text	Client host name, if available.
client_username	text	Operating system user on the client system who is running the restore tool.
user_name	text	Name of the database user, as passed in on the `ybrestore` command line (or set with `YBUSER`).
user_id	bigint	Unique ID of the database user.
database_name	text	Name of the restored database.
database_id	bigint	Unique ID of the restored database.
state	text	Current state of the restore operation, such as `USER_DATA_RUNNING` or `DONE` .
error_string	text	Error message reported for a failed restore. For example, if a restore is canceled before it completes, the view may report an `External data socket error` .
start_time	timestamptz	When the `ybrestore` command started running.
end_time	timestamptz	When the `ybrestore` command completed.
elapsed_ms	bigint	Time spent running the restore (in milliseconds).
remaining_ms	bigint	Time remaining to complete the restore (in milliseconds).
received_bytes	bigint	Number of bytes transferred over the network.
received_bytes_per_sec	double precision	Number of bytes per second transferred over the network.
storage_bytes	bigint	Number of bytes stored in the target location.
storage_bytes_per_sec	double precision	Number of bytes per second being stored in the target location.
backup_point_id	text	Backup name as specified with the `ybbackup --name` option.
chain_id	text	Name of the backup chain.
source_database_name	text	Name of the source database for the restore (not the restored database name in the case of a database being restored to an alias).

Column Name	Data Type	Description
source_appliance_name	text	Name of the appliance or data warehouse instance where the database was backed up.
table_name	text	Name of the table. For database-level restores, this column is `NULL`.

Examples

The following query shows that the database restore operation occurred with `premdb_restored` as the target database.

```sql
yellowbrick=# select * from sys.log_restore where database_name='premdb_restored';
-[ RECORD 1 ]----------+-----------------------------------------------------------------
session_key            | Dcli8wrwpy1WbsvrMNUzH9jPEjRdtLwpg8rBxGRRatRgN8BuOw7e2rMn3UYf7Xvs
session_id             | 54337
client_hostname        | localhost
client_username        | yellowbrick
username               | yellowbrick
user_id                | 16007
database_name          | premdb_restored
database_id            | 40619
state                  | DONE
error_string           | [NULL]
start_time             | 2021-06-22 14:37:56.185-07
end_time               | 2021-06-22 14:38:08.746-07
elapsed_ms             | 12561
remaining_ms           | 0
received_bytes         | 1040150443
received_bytes_per_sec | 82807932.728286
storage_bytes          | 1039832202
storage_bytes_per_sec  | 82782597.0862193
backup_point_id        | June22Full
chain_id               | premdbJune2021
source_database_name   | premdb
source_appliance_name  | localhost
table_name             | [NULL]
-[ RECORD 2 ]----------+-----------------------------------------------------------------
session_key            | CEyf1a9giaaA128eqmYHrdR35eo1mKybN4VliZkEcNhHWsvXf_nHJ2FrS_dUviPe
session_id             | 54449
client_hostname        | localhost
client_username        | yellowbrick
username               | yellowbrick
user_id                | 16007
database_name          | premdb_restored
database_id            | 40646
state                  | DONE
error_string           | [NULL]
start_time             | 2021-06-22 14:42:45.212-07
end_time               | 2021-06-22 14:42:47.389-07
elapsed_ms             | 2177
remaining_ms           | 0
received_bytes         | 62664469
received_bytes_per_sec | 28784781.3504823
storage_bytes          | 62647477
storage_bytes_per_sec  | 28776976.1139182
backup_point_id        | June22Full
chain_id               | premdbJune2021
source_database_name   | premdb
source_appliance_name  | localhost
table_name             | public.match_restored
```

sys.log_retention

This view records the history of the retention policy process. See also Retention Policies for System Relations.

Column Name	Data Type	Description
database_id	bigint	The unique ID for the database.
class_id	bigint	The unique ID for the relation. Relations in different schemas within the same database may have the same name, so you can use this ID to uniquely identify a relation.
start_time	timestamp with time zone	When the retention process started running against this table.
delete_count	bigint	The number of records removed by this retention cycle.
retention_ms	numeric(18,3)	The duration of this retention cycle, in milliseconds.
delete_ms	numeric(18,3)	The duration of the `DELETE` statement for this retention cycle, in milliseconds.
vacuum_ms	numeric(18,3)	The duration of the `VACUUM` statement for this retention cycle, in milliseconds.

Examples

We can join `sys.log_retention` with `sys.schema` and `sys.view` to have string names instead of IDs:

```sql
SELECT v.name AS relation, r.start_time, r.delete_count, r.retention_ms
    FROM sys.log_retention r
    JOIN sys.view v ON r.class_id = v.view_id AND r.database_id = v.database_id
    ORDER BY r.start_time DESC
    LIMIT 10;
```

```console
premdb=#
      relation      |          start_time           | delete_count | retention_ms
--------------------+-------------------------------+--------------+--------------
 log_restore        | 2023-11-20 10:46:33.451834+00 |            0 |        1.528
 log_query_analyze  | 2023-11-20 10:46:33.451834+00 |            0 |        1.551
 log_cluster_event  | 2023-11-20 10:46:33.451834+00 |            0 |        1.329
 log_backup         | 2023-11-20 10:46:33.451834+00 |            0 |        1.418
 log_analyze        | 2023-11-20 10:46:33.451834+00 |            0 |        1.505
 log_unload         | 2023-11-20 10:46:33.451834+00 |            0 |        1.337
 log_query          | 2023-11-20 10:46:33.451834+00 |            0 |      481.642
 log_load           | 2023-11-20 10:46:33.451834+00 |            0 |        1.760
 log_retention      | 2023-11-20 10:46:33.451834+00 |            0 |        0.882
 log_replica_status | 2023-11-20 10:46:33.451834+00 |            0 |        2.648
(10 rows)
```

sys.log_session

This view records the history of individual database sessions.

Column Name	Data Type	Description
session_id	bigint	Unique ID of the session.
application_name	varchar(256)	Name of the application that is running the session, such as `ybsql`, `ym`, `ybload`, or `ybunload`. You will also see internal application names such as `yb-lime`. (Note that an unload session reports `ybunload` as the application name while it is running. When the unload is complete, the view reports `yb-lime` as the application name.)
client_hostname	varchar(256)	Client host name, if available.
client_ip_address	varchar(256)	Client IP address.
database_id	bigint	Unique ID for the database.
user_id	bigint	Unique ID for the database user.
start_time	timestamptz	Timestamp for when the session started.
end_time	timestamptz	Timestamp for when the session ended, if it is not still running.
process_id	bigint	Unique process ID for the session. Used internally for debugging purposes.
secure_connection	boolean	Whether the database connection is secure (SSL).
last_statement	timestamptz	When the last SQL statement was started. This column is `NULL` if the session did not execute any statements.
state	varchar(256)	If the session was shut down because of idle_in_transaction_timeout, the state is `25P03`. If the session was shut down because of idle_session_timeout, the state is `25P04`. Other states are `idle` and `active`. See also Managing Idle Sessions.
secure_details	varchar(256)	SSL version, cipher, and key size.

Example

Join `sys.log_session` to `sys.database` to return information about `ybsql` sessions that lasted longer than 1 second:

```
premdb=# select sd.name dbname, sd.database_id, user_id, session_id, application_name, end_time-start_time duration
from sys.database sd, sys.log_session sls where sd.database_id=sls.database_id
and application_name ='ybsql' and duration>'1 sec'
order by sd.database_id, session_id;
     dbname      | database_id | user_id | session_id | application_name |    duration
-----------------+-------------+---------+------------+------------------+-----------------
 yellowbrick_test |       16588 |   16590 |      17232 | ybsql            | 00:00:11.774066
 yellowbrick_test |       16588 |   16590 |      18279 | ybsql            | 00:00:19.486412
 yellowbrick_test |       16588 |   16590 |      19941 | ybsql            | 00:00:30.966864
 yellowbrick_test |       16588 |   16590 |      23253 | ybsql            | 00:18:08.167864
 premdb           |       26259 |   26261 |      23285 | ybsql            | 00:00:35.007097
```

```
 premdb          |       26259 |   26261 |      23445 | ybsql           | 00:00:54.950788
(6 rows)
```

sys.log_unload

This system view captures information about completed unload (`ybunload`) operations.

Column Name	Data Type	Description
session_key	text	Unique authentication key for the `ybunload` operation. This is different from the session ID, which is the ID tracked for the user session in the `sys.session` and `sys.log_session` views.
state	text	State of the unload operation, such as `STARTING` , `DONE` , or `ERROR` .
error_string	text	Error message for a failed unload.
database_id	bigint	Unique ID of the database that contains the data being unloaded.
user_id	bigint	Unique ID of the user who ran the `ybunload` command.
session_id	bigint	Session ID; foreign key to the `sys.session` and `sys.log_session` views.
database_name	text	Name of the database that the data belongs to.
username	text	Name of the user who ran the `ybunload` command.
client_hostname	text	Host name of the client system where the unload was started.
client_username	text	Client OS user running the `ybunload` command.
start_time	timestamptz	When the `ybunload` command started running.
end_time	timestamptz	When the `ybunload` command completed its execution. (The `sys.unload` view has a `last_activity_time` column instead.)
elapsed_ms	bigint	Duration of the unload, in milliseconds.
sql_query	text	The SELECT statement that was run to unload data.
sent_rows	bigint	Number of rows transferred over the network during the unload.
sent_bytes	bigint	Number of bytes transferred over the network during the unload.
sent_rows_per_second	double precision	Number of rows transferred per second during the unload.
sent_bytes_per_second	double precision	Number of bytes transferred per second during the unload.

Example

The following example tracks the progress of an unload every 2 seconds:

```
premdb=# select * from sys.unload where start_time>'2018-05-02 18:00:00-07';
(0 rows)

premdb=# \watch
(0 rows)

Watch every 2s  Wed May  2 18:00:13 2018

-[ RECORD 1 ]---------+----------------------------------------------------------------
session_key           | CDil7ZNCMkZDvlgSPiHZn7qR4Wv7KLkXM3DLpIIMYXd6arJT0L0OFBDKPBNSaojf
state                 | RUNNING
error_string          | [NULL]
database_id           | 16423
user_id               | 16434
session_id            | 16792
database_name         | premdb
username              | bobr
client_hostname       | localhost
client_username       | yb100
start_time            | 2018-05-02 18:00:11-07
last_activity_time    | 2018-05-02 18:00:11-07
elapsed_ms            | 1860
sql_query             | select * from "newmatchstats"
sent_rows             | 0
sent_bytes            | 0
sent_rows_per_second  | 0
sent_bytes_per_second | 0

Watch every 2s  Wed May  2 18:00:15 2018
...

-[ RECORD 1 ]---------+----------------------------------------------------------------
session_key           | CDil7ZNCMkZDvlgSPiHZn7qR4Wv7KLkXM3DLpIIMYXd6arJT0L0OFBDKPBNSaojf
state                 | COMPLETE
error_string          | [NULL]
database_id           | 16423
user_id               | 16434
session_id            | 16792
database_name         | premdb
username              | bobr
client_hostname       | localhost
client_username       | yb100
start_time            | 2018-05-02 18:00:11-07
last_activity_time    | 2018-05-02 18:00:21-07
elapsed_ms            | 9851
sql_query             | select * from "newmatchstats"
sent_rows             | 25990128
sent_bytes            | 659797740
sent_rows_per_second  | 2599012.8
sent_bytes_per_second | 65979774

...
Watch every 2s  Wed May  2 18:00:23 2018
...

-[ RECORD 1 ]---------+----------------------------------------------------------------
session_key           | CDil7ZNCMkZDvlgSPiHZn7qR4Wv7KLkXM3DLpIIMYXd6arJT0L0OFBDKPBNSaojf
state                 | DONE
error_string          | [NULL]
database_id           | 16423
user_id               | 16434
session_id            | 16792
database_name         | premdb
username              | bobr
client_hostname       | localhost
client_username       | yb100
start_time            | 2018-05-02 18:00:11-07
```

```
last_activity_time    | 2018-05-02 18:00:21-07
elapsed_ms            | 10353
sql_query             | select * from "newmatchstats"
sent_rows             | 25990128
sent_bytes            | 659797740
sent_rows_per_second  | 2599012.8
sent_bytes_per_second | 65979774

...
```

sys.log_warning

Platforms: EE: All appliance platforms

Parent topic: System Views

This view logs system-level warning messages logged on the appliance. This view does not log SQL warnings.

Column Name	Data Type	Description
event_id	uuid	Unique ID for the event (warning).
event_time	timestamptz	Timestamp when the event (warning) occurred.
acknowledged	boolean	Whether the user has acknowledged the warning on the SMC System Events screen, where this data is also visible.
warning_code	text	Warning code, such as `system`.
warning_msg	text	Text of the warning message.

Example

For example:

```
yellowbrick=# select * from sys.log_warning where warning_code='system';
-[ RECORD 1 ]+-------------------------------------------------------------------------------------------
event_id     | e138fda6-e691-4825-834c-3e25be2829bb
event_time   | 2020-10-09 09:23:47.713-07
query_id     | [NULL]
acknowledged | f
warning_code | system
warning_msg  | Worker 00000000-0000-0000-0000-38b8ebd002a3 logical id: 17 has gone offline at 7858526678238244
-[ RECORD 2 ]+-------------------------------------------------------------------------------------------
event_id     | 75a6fe72-c0e3-49fd-9a43-53baf1a95d9f
event_time   | 2020-10-11 14:34:00.662-07
query_id     | [NULL]
acknowledged | f
warning_code | system
warning_msg  | Worker 00000000-0000-0000-0000-38b8ebd002a3 logical id: 17 has gone offline at 8049939633529310
-[ RECORD 3 ]+-------------------------------------------------------------------------------------------
event_id     | 0890c719-2613-4a48-8f8d-a84cc29ac840
event_time   | 2020-10-21 23:47:12.386-07
query_id     | [NULL]
acknowledged | f
warning_code | system
warning_msg  | Worker 00000000-0000-0000-0000-38b8ebd002a3 logical id: 17 has gone offline at 8947131356151761
-[ RECORD 4 ]+-------------------------------------------------------------------------------------------
event_id     | d75affa6-d29b-4a23-a2e1-966d45a6116e
event_time   | 2020-10-22 18:16:43.571-07
query_id     | [NULL]
acknowledged | f
warning_code | system
warning_msg  | Worker 00000000-0000-0000-0000-38b8ebd002a3 logical id: 17 has gone offline at 9013702550085108
-[ RECORD 5 ]+-------------------------------------------------------------------------------------------
event_id     | de63aeea-cd4e-4aa1-85be-bd93ff9ccdf5
event_time   | 2020-10-22 18:53:35.607-07
query_id     | [NULL]
```

```
 acknowledged | f
 warning_code | system
 warning_msg  | Worker 00000000-0000-0000-0000-38b8ebd002a3 logical id: 17 has gone offline at 9015914587018051
```

sys.mount

Platforms: EE: All appliance platforms

Parent topic: System Views

This system view returns a list of NFS mount points that have been created for working with external tables.

Column Name	Data Type	Description
ref_name	text	Reference name for the mount point.
resource	text	Actual path to the mount point.
options	text	Options used to create the mount point.

For example:

```
premdb=# select * from sys.mount;
       ref_name        |           resource            |        options
-----------------------+-------------------------------+-----------------------
   /qumulo/yb100       | nfs://qumulo:/data/yb100/     | {"gid" : 0, "uid" : 0}
(2 rows)
```

sys.password_policy

This system view has one row that represents the system-level policy, with additional rows present for each user who has had their policy overwritten. In this view, the users can see their own password policies however they cannot see the system-wide policies.

Password policies are a new feature that must be enabled by setting the `enable_password_policy` configuration parameter, after which a full database restart is required.

Column Name	Data Type	Description
object id	oid	The ID (if any) of the object the policy applies to. For the SYSTEM policy, this is `NULL`.
user_name	name	Name of the user or role. For the SYSTEM policy, this is `NULL`.
object_type	text	Either `SYSTEM` for the system level policy or `USER` if the policy is set for a user.
min_length	smallint	Minimum length of the password in characters.
min_letters	smallint	Minimum number of letters in the password.
min_lower_case	smallint	Minimum number of lowercase letters in the password. If larger than `min_letter`, this takes precedence.
min_upper_case	smallint	Minimum number of uppercase letters in the password.
min_symbols	smallint	Minimum number of special characters that are not letters or digits.
min_digits	smallint	Minimum number of digits in the password.
max_reuse	integer	Maximum number of times a password can be reused by the user. If set to zero, any number of reuses are allowed.
max_attempts	integer	Maximum number of failed login attemps before the account is locked out.
unlock_after_s	integer	How many seconds until the account automatically unlocks if locked out? If set to zero, unlocking must be done manually. **Note:** This interval starts from the time of a successful login.
expire_user_after_s	integer	How long, in seconds, a user will be automatically expired after a successful login.
expire_password_after_s	integer	How long, in seconds, a password will be automatically expired after it was last set.

Note: Existing users will be able to use their current passwords until they expire. After a password expires, the new password must comply with the policy. Yellowbrick does not verify passwords that are already in use when the new password policy is implemented.

The largest of `min_length`, `min_letters`, `min_lower_case`, `min_upper_case`, `min_symbols` and `min_digits` take precendence when determining the smallest allowed password.

Access Control

The following rules apply to the rows in this view:

- `superuser` and `sysadmin` can read all rows.
- Users with `CREATE USER`, `ALTER ANY USER` or `CONTROL ANY USER` privilege can read all rows with `object_type` = **USER**

Data Retention

The system-level data in this table does not expire. If a user is deleted, the entry (if any) for that user in this view is deleted too.

Replication and Backup/Restore

The rows with `object_type` = **USER** are replicated or backed up if that user is backed up. The row with `object_type` = **SYSTEM** is not replicated. This means that the primary and secondary system can have different password policies.

See also: Password Policies for a general overview of the feature. `enable_password_policy` for enabling the feature.

sys.procedure

A system view that returns information about the user-defined stored procedures in the database.

Column Name	Data Type	Description
procedure_id	bigint	Unique ID of the stored procedure.
database_id	bigint	Unique ID of the database where the procedure was created.
schema_id	bigint	Unique ID of the schema where the view was created. Join to `sys.schema` to get the schema name.
owner_id	bigint	Unique ID of the user who owns the view. Join to `sys.user` to get the name of the user.
name	text	Name of the stored procedure.
definition	text	Signature of the stored procedure, including the data type of each parameter.
creation_time	timestamptz	When the procedure was created.
database_name	text	Name of the database where the procedure was created.

For example:

```
premdb=# select * from sys.procedure;
--------------+-------------+-----------+-----------------+-------------------------------------------+--------------------------
       19120 |       16395 |      2200 | addteam         | addteam(character varying)                | 2020-02-17 20:42:52.74990
       19121 |       16395 |      2200 | insert_hometeam | insert_hometeam(integer, character varying) | 2020-02-17 20:43:03.99308
(2 rows)
```

sys.query

This view returns similar information about queries to the `sys.log_query` view, but for actively running queries or submitted but not yet executed queries. See the sys.log_query page for column descriptions.

Note: The `sys.query` view has a `blocked` column. When a query is blocked, this column shows the reason. The `sys.log_query` view does not have this column. All other columns in these two views are the same.

User accounts only see the queries they submit. Superuser accounts see all queries.

Examples

The following `UPDATE` query is repeated every second, returning different results as queries progress on the system:

```
premdb=> select query_id, username, application_name, type, state, substr(query_text,1,25) qrytxt, parse_ms, plan_ms, assemble_ms,
from sys.query where type='update' and username='bobr'
order by 1 desc;
...
                                                    Watch every 1s    Thu Sep 10 18:52:16 2020

 query_id | username | application_name |  type  |  state  |          qrytxt           | parse_ms | plan_ms | assemble_ms | compil
----------+----------+------------------+--------+---------+---------------------------+----------+---------+-------------+-------
  1141949 | bobr     | ybsql            | update | compile | update newmatchstats set  |    0.528 |   2.475 |      27.465 |    513
(1 row)

                                                    Watch every 1s    Thu Sep 10 18:52:17 2020

 query_id | username | application_name |  type  | state |          qrytxt           | parse_ms | plan_ms | assemble_ms | compile_
----------+----------+------------------+--------+-------+---------------------------+----------+---------+-------------+---------
  1141949 | bobr     | ybsql            | update | run   | update newmatchstats set  |    0.528 |   2.475 |      27.465 |   1319.9
(1 row)

                                                    Watch every 1s    Thu Sep 10 18:52:18 2020

 query_id | username | application_name |  type  | state |          qrytxt           | parse_ms | plan_ms | assemble_ms | compile_
----------+----------+------------------+--------+-------+---------------------------+----------+---------+-------------+---------
  1141949 | bobr     | ybsql            | update | run   | update newmatchstats set  |    0.528 |   2.475 |      27.465 |   1319.9
(1 row)
...
```

sys.query_analyze

This view records runtime statistics for each execution step (or node) in the query plan. A unique row in this table is identified by a combination of the query ID, compute node ID, and node ID. *All of the reported statistics are per plan node per compute node.*

Column Name	Data Type	Description
query_id	bigint	Query execution ID
worker_id	text	Compute Node ID (this column does not exist in sys.log_query_analyze)
node_id	integer	Node ID within the query plan
rows_planned	bigint	Estimated number of rows that will be output by this query node
rows_actual	bigint	Actual number of rows output from this query node
memory_planned_bytes	bigint	Estimated amount of memory required for this query node
memory_actual_bytes	bigint	Amount of memory consumed by this query node
io_read_bytes	bigint	Number of bytes read
io_write_bytes	bigint	Number of bytes written
io_network_bytes	bigint	Number of bytes sent over the network
io_network_count	bigint	Number of network calls made
runtime_ms	decimal(18,3)	Wall clock time in milliseconds: how long the plan node was running on the compute nodes
skew	double precision	A value between 0 and 1 that indicates how evenly distributed the input is to the query node: `0` indicates perfect distribution and `1` indicates all data is on a single node.**Important:** Values above `0.3` should be investigated.
detail	varchar(1024)	Textual data that describes additional statistics for this query node.

Examples

```
premdb=# select query_id, worker_id, node_id, memory_actual_bytes, io_read_bytes, io_write_bytes, io_network_bytes, io_network_cou
from sys.query_analyze order by worker_id, query_id, node_id;
 query_id |              worker_id               | node_id | memory_bytes | io_read_bytes | io_write_bytes | io_network_bytes | io
----------+--------------------------------------+---------+--------------+---------------+----------------+------------------+---
    25832 | 00000000-0000-0000-0000-000000000002 |       0 |      3145936 |             0 |              0 |       8091360528 |
    25832 | 00000000-0000-0000-0000-000000000002 |       1 |     17301376 |             0 |              0 |                0 |
    25832 | 00000000-0000-0000-0000-000000000002 |       2 |       262144 |             0 |              0 |                0 |
    25832 | 00000000-0000-0000-0000-000000000002 |       3 |       262144 |             0 |              0 |                0 |
    25832 | 00000000-0000-0000-0000-000000000002 |       4 |        61440 |        131072 |              0 |                0 |
    25832 | 00000000-0000-0000-0000-000000000002 |       5 |      1048576 |             0 |              0 |                0 |
    25832 | 00000000-0000-0000-0000-000000000002 |       7 |      1310864 |             0 |              0 |                0 |
    25832 | 00000000-0000-0000-0000-000000000002 |       8 |      1310720 |             0 |              0 |                0 |
    25832 | 00000000-0000-0000-0000-000000000002 |       9 |      2097152 |             0 |              0 |                0 |
```

```
   25832 | 00000000-0000-0000-0000-000000000002 |      10 |       524288 |            0 |            0 |            0 |
   25832 | 00000000-0000-0000-0000-000000000002 |      11 |       262144 |            0 |            0 |            0 |
   25832 | 00000000-0000-0000-0000-000000000002 |      12 |      1078272 |            0 |            0 |            0 |
   25832 | 00000000-0000-0000-0000-000000000002 |      13 |      1048576 |            0 |            0 |            0 |
   25832 | 00000000-0000-0000-0000-000000000002 |      15 |      1310828 |            0 |            0 |            0 |
   25832 | 00000000-0000-0000-0000-000000000002 |      16 |      1310720 |            0 |            0 |            0 |
...
```

The following example joins `sys.query_analyze` to `sys.query` and `sys.query_explain`. The output shown here is only the first few rows for the first compute node:

```
yellowbrick=# select a.*, e.query_plan
from sys.query_analyze a
left outer join sys.query q
on a.query_id = q.query_id
left outer join sys.query_explain e
on e.plan_id = q.plan_id and e.node_id = a.node_id
order by a.query_id, a.worker_id, a.node_id;

-[ RECORD 1 ]--------+-----------------------------------------------------------------------------------------------------------------------
query_id             | 19798710
worker_id            | 00000000-0000-0000-0000-38b8ebd000cd
node_id              | 0
rows_planned         | 774540
rows_actual          | 0
memory_planned_bytes | 0
memory_actual_bytes  | 0
io_read_bytes        | [NULL]
io_write_bytes       | [NULL]
io_network_bytes     | 0
io_network_count     | 0
runtime_ms           | 0.000
skew                 | 0
detail               | [NULL]
query_plan           | SEQUENCE
                     | distribute none
-[ RECORD 2 ]--------+-----------------------------------------------------------------------------------------------------------------------
query_id             | 19798710
worker_id            | 00000000-0000-0000-0000-38b8ebd000cd
node_id              | 1
rows_planned         | 774540
rows_actual          | 0
memory_planned_bytes | 33554784
memory_actual_bytes  | 352
io_read_bytes        | [NULL]
io_write_bytes       | [NULL]
io_network_bytes     | 0
io_network_count     | 0
runtime_ms           | 0.000
skew                 | 0
detail               | [NULL]
query_plan           | WRITE ROW STORE
                     | (newmatchstats.seasonid, newmatchstats.matchday, newmatchstats.htid, newmatchstats.atid, newmatchstats.momen
                     | distribute (newmatchstats.seasonid)
-[ RECORD 3 ]--------+-----------------------------------------------------------------------------------------------------------------------
query_id             | 19798710
worker_id            | 00000000-0000-0000-0000-38b8ebd000cd
node_id              | 2
rows_planned         | 774540
rows_actual          | 0
memory_planned_bytes | 3221225472
memory_actual_bytes  | 1048576
io_read_bytes        | [NULL]
io_write_bytes       | [NULL]
io_network_bytes     | 0
io_network_count     | 0
runtime_ms           | 0.000
```

```
skew                | 0
detail              | [NULL]
query_plan          | INSERT INTO newmatchstats
                    | (newmatchstats.seasonid, newmatchstats.matchday, newmatchstats.htid, newmatchstats.atid, newmatchstats.momen
                    | distribute (newmatchstats.seasonid)
-[ RECORD 4 ]-------+-------------------------------------------------------------------------------------------------------------
query_id            | 19798710
worker_id           | 00000000-0000-0000-0000-38b8ebd000cd
node_id             | 3
rows_planned        | 774540
rows_actual         | 0
memory_planned_bytes | 8388608
memory_actual_bytes | 786432
io_read_bytes       | [NULL]
io_write_bytes      | [NULL]
io_network_bytes    | 0
io_network_count    | 0
runtime_ms          | 0.000
skew                | 0.154261304864766
detail              | [NULL]
query_plan          | APPEND SCAN
                    | (newmatchstats.seasonid, newmatchstats.matchday, newmatchstats.htid, newmatchstats.atid, newmatchstats.momen
                    | distribute (newmatchstats.seasonid)
-[ RECORD 5 ]-------+-------------------------------------------------------------------------------------------------------------
query_id            | 19798710
worker_id           | 00000000-0000-0000-0000-38b8ebd000cd
node_id             | 4
rows_planned        | 774540
rows_actual         | 0
memory_planned_bytes | 285212672
memory_actual_bytes | 8126464
io_read_bytes       | [NULL]
io_write_bytes      | [NULL]
io_network_bytes    | 0
io_network_count    | 29
runtime_ms          | 0.000
skew                | 0.154261304864766
detail              | [NULL]
query_plan          | DISTRIBUTE ON HASH(newmatchstats.seasonid) (degraded only)
                    | (newmatchstats.seasonid, newmatchstats.matchday, newmatchstats.htid, newmatchstats.atid, newmatchstats.momen
                    | distribute (newmatchstats.seasonid)
...
```

sys.query_explain

This view records the query plan for each currently active query and contains a row for each node in the plan. A unique row in this table is identified by a combination of plan ID and node ID. The rows in the table map to the node-level text output that is returned when you run an EXPLAIN ANALYZE command.

See sys.log_query_explain for a description of the columns in the view.

For example:

```
premdb=# select * from sys.query_explain;
 plan_id | node_id | index | type | workers | query_plan
---------+---------+-------+------+---------+------------
(0 rows)

premdb=# \watch
       Watch every 2s   Thu Feb  1 15:34:20 2018

 plan_id | node_id | index | type | workers | query_plan
---------+---------+-------+------+---------+------------
(0 rows)
...
                                                                                Watch every 2s    Thu Feb  1 15:34:28 2018

                 plan_id                    | node_id | index |      type       | workers |
--------------------------------------------+---------+-------+-----------------+---------+-----------------------------------
 v2i-xVAwlrav2rBiQmxF50CLB0fWdHyuoBvhOj7WMnU= |       0 |     0 | WRITE ROW STORE | all     | WRITE ROW STORE
                                            |         |       |                 |         | (newmatchstats.seasonid, newmatchstat
                                            |         |       |                 |         | distribute none
 v2i-xVAwlrav2rBiQmxF50CLB0fWdHyuoBvhOj7WMnU= |       1 |     1 | INSERT INTO     | all     | INSERT INTO newmatchstats
                                            |         |       |                 |         | (newmatchstats.seasonid, newmatchstat
 v2i-xVAwlrav2rBiQmxF50CLB0fWdHyuoBvhOj7WMnU= |       3 |     2 | DISTRIBUTE HASH | all     | DISTRIBUTE HASH (degraded only)
                                            |         |       |                 |         | (newmatchstats.seasonid, newmatchstat
                                            |         |       |                 |         | distribute (newmatchstats.seasonid)
 v2i-xVAwlrav2rBiQmxF50CLB0fWdHyuoBvhOj7WMnU= |       2 |     3 | APPEND SCAN     | all     | APPEND SCAN
                                            |         |       |                 |         | (newmatchstats.seasonid, newmatchstat
 v2i-xVAwlrav2rBiQmxF50CLB0fWdHyuoBvhOj7WMnU= |       5 |     4 | TRANSPOSE       | all     | |-TRANSPOSE
                                            |         |       |                 |         |   (newmatchstats.seasonid, newmatchst
 v2i-xVAwlrav2rBiQmxF50CLB0fWdHyuoBvhOj7WMnU= |       4 |     5 | SCAN            | all     | | SCAN newmatchstats
                                            |         |       |                 |         |   (newmatchstats.seasonid, newmatchst
                                            |         |       |                 |         |   distribute (newmatchstats.seasonid)
 v2i-xVAwlrav2rBiQmxF50CLB0fWdHyuoBvhOj7WMnU= |       7 |     6 | SCAN ROW STORE  | single  | |-SCAN ROW STORE
                                            |         |       |                 |         |   (newmatchstats.seasonid, newmatchst
                                            |         |       |                 |         |   distribute none
(7 rows)
...
```

sys.query_rule_event

This view captures information about WLM rule processing (as also displayed in Yellowbrick Manager). A single row identifies a unique rule-processing event for a specific query.

Use this view to see which rules and related actions are in effect when specific queries are run.

Column Name	Data Type	Description
query_id	bigint	Query ID, which can be joined with the `query_id` column in `sys.log_query` or `sys.query`.
rule_name	text	Name of the WLM rule being applied. The default value of `rule` appears in this column for `begin` and `end` event types.
event_time	timestamptz	When each event occurred.
event_type	text	Possible values for each event are `begin`, `disabled`, `end`, `error`, `ignore`, `info`, `move`, `set`, `timeout`, `warn`
group_timestamp	timestamptz	Timestamp for a group of rules that are run for a query during a given state in its life cycle.
group_index	integer	Index number that defines the order of rules that are run for a given group (0 to *n*).
event	text	Rule processing events include `Rule processing starting`, `Rule processing complete`, `Setting...`, `Move to resource pool...`, `Rule is disabled`, and so on.

Examples

Return all rule events for a specific query:

```
premdb=> select * from sys.query_rule_event where query_id=23177566;
 query_id |             rule_name              |          event_time           | event_type |        group_timestamp        | grou
----------+------------------------------------+-------------------------------+------------+-------------------------------+-----
 23177566 | rule                               | 2022-09-15 15:28:56.595608-07 | begin      | 2022-09-15 15:28:56.595699-07 |
 23177566 | default_logLongRunningQuery        | 2022-09-15 15:28:56.595611-07 | disabled   | 2022-09-15 15:28:56.595699-07 |
 23177566 | global_setupYMWorkDone             | 2022-09-15 15:28:56.595691-07 | info       | 2022-09-15 15:28:56.595699-07 |
 23177566 | rule                               | 2022-09-15 15:28:56.595698-07 | end        | 2022-09-15 15:28:56.595699-07 |
 23177566 | rule                               | 2022-09-15 15:28:48.616755-07 | begin      | 2022-09-15 15:28:48.617378-07 |
 23177566 | default_cancelAfterOneMinute       | 2022-09-15 15:28:48.616756-07 | disabled   | 2022-09-15 15:28:48.617378-07 |
 23177566 | default_superUserToAdmin           | 2022-09-15 15:28:48.616759-07 | ignore     | 2022-09-15 15:28:48.617378-07 |
 23177566 | global_throttleExternalTables      | 2022-09-15 15:28:48.616843-07 | info       | 2022-09-15 15:28:48.617378-07 |
 23177566 | global_throttleReplication         | 2022-09-15 15:28:48.616865-07 | ignore     | 2022-09-15 15:28:48.617378-07 |
 23177566 | default_barToSystem                | 2022-09-15 15:28:48.616866-07 | ignore     | 2022-09-15 15:28:48.617378-07 |
 23177566 | default_mapToSmall                 | 2022-09-15 15:28:48.616907-07 | info       | 2022-09-15 15:28:48.617378-07 |
 23177566 | global_mapAAToLowPriority          | 2022-09-15 15:28:48.61691-07  | ignore     | 2022-09-15 15:28:48.617378-07 |
 23177566 | global_setupDCSWork                | 2022-09-15 15:28:48.61691-07  | ignore     | 2022-09-15 15:28:48.617378-07 |
 23177566 | global_setupSMCWork                | 2022-09-15 15:28:48.616911-07 | ignore     | 2022-09-15 15:28:48.617378-07 |
 23177566 | global_setupSystemWork             | 2022-09-15 15:28:48.616912-07 | ignore     | 2022-09-15 15:28:48.617378-07 |
 23177566 | global_setupYMWork                 | 2022-09-15 15:28:48.616944-07 | info       | 2022-09-15 15:28:48.617378-07 |
 23177566 | global_setupStatsWork              | 2022-09-15 15:28:48.616946-07 | ignore     | 2022-09-15 15:28:48.617378-07 |
 23177566 | default_dropTableToAdmin           | 2022-09-15 15:28:48.617167-07 | info       | 2022-09-15 15:28:48.617378-07 |
 23177566 | global_mapAnalyzeToSystem          | 2022-09-15 15:28:48.61717-07  | ignore     | 2022-09-15 15:28:48.617378-07 |
 23177566 | global_mapFlusherToSystem          | 2022-09-15 15:28:48.617171-07 | ignore     | 2022-09-15 15:28:48.617378-07 |
 23177566 | global_mapGCToSystem               | 2022-09-15 15:28:48.617172-07 | ignore     | 2022-09-15 15:28:48.617378-07 |
 23177566 | global_mapParityToSystem           | 2022-09-15 15:28:48.617173-07 | ignore     | 2022-09-15 15:28:48.617378-07 |
```

```
 23177566 | global_restartErrorPolicy          | 2022-09-15 15:28:48.61723-07  | set        | 2022-09-15 15:28:48.617378-07 |
 23177566 | global_restartErrorPolicySuperuser | 2022-09-15 15:28:48.617235-07 | ignore     | 2022-09-15 15:28:48.617378-07 |
 23177566 | global_setupReplicatorWork         | 2022-09-15 15:28:48.617236-07 | ignore     | 2022-09-15 15:28:48.617378-07 |
 23177566 | global_defaultRowLimit             | 2022-09-15 15:28:48.61737-07  | set        | 2022-09-15 15:28:48.617378-07 |
 23177566 | rule                               | 2022-09-15 15:28:48.617376-07 | end        | 2022-09-15 15:28:48.617378-07 |
 23177566 | rule                               | 2022-09-15 15:28:48.613373-07 | begin      | 2022-09-15 15:28:48.613981-07 |
 23177566 | global_throttleConcurrentQueries   | 2022-09-15 15:28:48.613521-07 | throttle   | 2022-09-15 15:28:48.613981-07 |
 23177566 | urgent_reports_rule                | 2022-09-15 15:28:48.61397-07  | info       | 2022-09-15 15:28:48.613981-07 |
 23177566 | rule                               | 2022-09-15 15:28:48.613977-07 | end        | 2022-09-15 15:28:48.613981-07 |
(31 rows)
```

Join `sys.query_rule_event` to `sys.log_query` and return results based on specific query text:

```
premdb=> select s1.query_id, s1.query_text, s2.rule_name, s2.event_type, s2.event
from sys.log_query s1, sys.query_rule_event s2
where s1.query_id=s2.query_id and query_text ='select * from match;';
 query_id |      query_text      |             rule_name              | event_type |                            event
----------+----------------------+------------------------------------+------------+-----------------------------------------------
 23259120 | select * from match; | rule                               | begin      | Rule [completion] processing for 23259120 sta
 23259120 | select * from match; | default_logLongRunningQuery        | disabled   | Rule is disabled
 23259120 | select * from match; | global_setupYMWorkDone             | info       | Rule changed no query settings
 23259120 | select * from match; | rule                               | end        | Rule [completion] processing for 23259120 com
 23259120 | select * from match; | rule                               | begin      | Rule [compile] processing for 23259120 starti
 23259120 | select * from match; | default_cancelAfterOneMinute       | disabled   | Rule is disabled
 23259120 | select * from match; | default_superUserToAdmin           | ignore     | Rule configured for superuser queries (query
 23259120 | select * from match; | global_throttleExternalTables      | info       | Rule changed no query settings
 23259120 | select * from match; | global_throttleReplication         | ignore     | Rule configured for superuser queries (query
 23259120 | select * from match; | default_barToSystem                | ignore     | Rule configured for superuser queries (query
 23259120 | select * from match; | default_mapToSmall                 | set        | Setting ResourcePool to small
 23259120 | select * from match; | global_mapAAToLowPriority          | ignore     | Rule configured for superuser queries (query
 23259120 | select * from match; | global_setupDCSWork                | ignore     | Rule configured for superuser queries (query
 23259120 | select * from match; | global_setupSMCWork                | ignore     | Rule configured for superuser queries (query
 23259120 | select * from match; | global_setupSystemWork             | ignore     | Rule configured for superuser queries (query
 23259120 | select * from match; | global_setupYMWork                 | info       | Rule changed no query settings
 23259120 | select * from match; | global_setupStatsWork              | ignore     | Rule configured for superuser queries (query
 23259120 | select * from match; | default_dropTableToAdmin           | info       | Rule changed no query settings
 23259120 | select * from match; | global_mapAnalyzeToSystem          | ignore     | Rule configured for superuser queries (query
 23259120 | select * from match; | global_mapFlusherToSystem          | ignore     | Rule configured for superuser queries (query
 23259120 | select * from match; | global_mapGCToSystem               | ignore     | Rule configured for superuser queries (query
 23259120 | select * from match; | global_mapParityToSystem           | ignore     | Rule configured for superuser queries (query
 23259120 | select * from match; | global_restartErrorPolicy          | set        | Setting ErrorRecoverable to true
 23259120 | select * from match; | global_restartErrorPolicySuperuser | ignore     | Rule configured for superuser queries (query
 23259120 | select * from match; | global_setupReplicatorWork         | ignore     | Rule configured for superuser queries (query
 23259120 | select * from match; | global_defaultRowLimit             | set        | Setting MaximumRowLimit to 5000000
 23259120 | select * from match; | rule                               | end        | Rule [compile] processing for 23259120 comple
 23259120 | select * from match; | rule                               | begin      | Rule [submit] processing for 23259120 startin
 23259120 | select * from match; | global_throttleConcurrentQueries   | throttle   | Throttle 500 accesses
 23259120 | select * from match; | urgent_reports_rule                | info       | Rule changed no query settings
 23259120 | select * from match; | rule                               | end        | Rule [submit] processing for 23259120 complet
(31 rows)
```

The following example filters events in which a query was moved to another pool:

```
yellowbrick=# select query_id, rule_name, event_time, event_type, event
from sys.query_rule_event where event like 'Move%';
 query_id |    rule_name    |         event_time         | event_type |           event
----------+-----------------+----------------------------+------------+---------------------------
 13450004 | system          | 2022-08-24 14:39:59.967-07 | move       | Move to resource pool long
 13245785 | flex_moveToLong | 2022-08-24 12:35:18.543-07 | move       | Move to resource pool long
(2 rows)
```

The following example finds events for a specific completion rule when it was not disabled or ignored:

```
yellowbrick=# select query_id, rule_name, event_time, event_type, event
from sys.query_rule_event
where rule_name='completion_error' and event_type not in('ignore','disabled');
 query_id |    rule_name     |         event_time         | event_type |               event
----------+------------------+----------------------------+------------+-----------------------------------
 13584534 | completion_error | 2020-09-24 15:59:05.644-07 | error      | Do not use the yellowbrick database!
 13583194 | completion_error | 2020-09-24 15:58:50.191-07 | error      | Do not use the yellowbrick database!
 13580925 | completion_error | 2020-09-24 15:57:36.721-07 | error      | Do not use the yellowbrick database!
 13579296 | completion_error | 2020-09-24 15:57:13.002-07 | info       | Rule changed no query settings
 13578087 | completion_error | 2020-09-24 15:57:08.935-07 | error      | Do not use the yellowbrick database!
 13577467 | completion_error | 2020-09-24 15:56:27.999-07 | info       | Rule changed no query settings
...
```

sys.remote_server

This view captures information about remote servers that have been created for database replication. See also CREATE REMOTE SERVER.

Column Name	Data Type	Description
remote_server_id	bigint	Unique ID for the remote server.
name	varchar(256)	User-defined name for the remote server object.
host	varchar(256)	Host name of the remote server (the target system).
port	integer	Database port on the remote server. Default is `5432`.
control_port	integer	`NULL`. (Refers to JDBC control port on the remote server.)
no_hostname_check	boolean	Whether strict checking of host names is disabled (`true`) or enabled (`false`). The default is `false`.

For information about Yellowbrick ports, see Opening Network Ports for Clients.

For example:

```
premdb=# select * from sys.remote_server;
 remote_server_id |       name        |          host          | port | control_port | no_hostname_check
------------------+-------------------+------------------------+------+--------------+-------------------
            16417 | yb100             | yb100.yellowbrick.io | 5432 |       [NULL] | t
            16418 | br_yb_io          | 10.30.22.203         | 5432 |       [NULL] | t
            16419 | yblocal_checkhost | localhost            | 5432 |       [NULL] | f
(3 rows)
```

sys.replica

This view captures information about replicas that have been created for database replication. See also ALTER DATABASE ADD REPLICA.

Column Name	Data Type	Description
replica_id	bigint	Unique ID of the replica.
database_id	bigint	Unique ID of the source database.
remote_server_id	bigint	Unique ID of the remote server.
backup_chain_id	varchar(256)	Unique ID of the backup chain for the replica.
name	varchar(256)	Name of the replica.
last_replication_time	timestamptz	When the snapshot was created for the last successful replication cycle that was run for this replica. This timestamp corresponds to the `creation_time` value in the `sys.backup_snapshot` view.
status	varchar(256)	Status of replication for this replica: `RUNNING`, `PAUSED`, `NOT_STARTED`, `ERROR`, `FORCED`, or `UNKNOWN`.
frequency	integer	Frequency interval for replication, such as 30 seconds or 5 minutes.
alias	varchar(256)	Name of the database being replicated to on the remote system.
bandwidth_limit_bytes	bigint	Limit set for maximum speed of transmission traffic.
exclude_filter	varchar(256)	Schemas excluded from replication for this database, if any.
security_mode	varchar(256)	`ALL` (the default) or `NONE`, to indicate whether security-related objects and attributes are replicated (users, roles, ACLs, GRANTs).
user_resolution_mode, user_duplicate_resolution_mode	varchar(256)	*Not applicable in this release.* Default values are `db_owner` and `rename`.
is_sync_users, is_sync_roles, is_sync_grants, is_sync_acls	boolean	*Not applicable in this release.* Default values are `true` when `security_mode` is `all` and `false` when `security_mode` is `none`.
is_sync_superusers	boolean	*Not applicable in this release.* Default value is `false`.
is_table_level_restart	boolean	*Not applicable in this release.*Default value is `false`.
reverse_replica	varchar(256)	Name of a reverse replica associated with this replica, if any; otherwise `NULL`.

Examples

```
premdb=# select * from sys.replica where name='premdb_replica';
-[ RECORD 1 ]----------------+-----------------
replica_id                   | 41547
database_id                  | 41524
```

```
remote_server_id               | 16410
backup_chain_id                | [NULL]
name                           | premdb_replica
last_replication_time          | [NULL]
status                         | NOT_STARTED
frequency                      | 10
alias                          | premdb_replica_db
bandwidth_limit_bytes          | [NULL]
exclude_filter                 | [NULL]
security_mode                  | all
user_resolution_mode           | db_owner
user_duplicate_resolution_mode | rename
is_sync_users                  | t
is_sync_roles                  | t
is_sync_grants                 | t
is_sync_acls                   | t
is_sync_superusers             | f
is_table_level_restart         | f
reverse_replica                | [NULL]
```

```
yellowbrick=# select sr.replica_id, sr.name replica_name, sr.database_id, sd.name db_name, sr.status
from sys.replica sr, sys.database sd where sr.database_id=sd.database_id;
 replica_id | replica_name | database_id |     db_name      |   status
------------+--------------+-------------+------------------+-------------
      16555 | replication  |       16553 | repl_source      | RUNNING
      16449 | replication2 |       16388 | yellowbrick_test | NOT_STARTED
(2 rows)
```

sys.replica_status

This system view records current replication activity, per cycle, on the source system. See also sys.log_replica_status.

Column Name	Data Type	Description
database_id	bigint	Unique ID of the source database.
replica_id	bigint	Unique ID of the replica.
snapshot_id	varchar(256)	Unique ID of the backup snapshot.
elapsed_ms	bigint	Elapsed time for the replication cycle (in milliseconds).
remaining_ms	bigint	Estimated remaining time for the replication cycle (in milliseconds), based on an estimation of the size of the backup that is being restored. (This column always returns `0` in `sys.log_replica_status`.)
start_time	timestamptz	When the replication cycle started.
end_time	timestamptz	When the replication cycle ended.
state	varchar(256)	Status of replication for this replica, such as `RUNNING`, `PAUSED`, `NOT_STARTED`.
error_string	varchar(256)	Error message, if there was a failure.
backup_session_key	varchar(256)	Unique session key for the backup used for replication.
restore_session_key	varchar(256)	Unique session key for the restore used for replication.
sent_bytes	bigint	Number of bytes sent over the network.
write_bytes	bigint	Number of bytes written to the replica database.
retries	integer	*Not applicable in this release.* Default value is `0`.

Examples

After starting replication on an empty database, when nothing has been replicated except the system catalog, you will see an entry like this:

```
premdb=# select * from sys.replica_status;
-[ RECORD 1 ]-------+------------------------------------------------------------------
database_id         | 16391
replica_id          | 16415
snapshot_id         | premdb_replica1_19_12_12_16_58_14
elapsed_ms          | 37868
remaining_ms        | 0
start_time          | 2019-12-12 16:58:14-08
end_time            | [NULL]
state               | CATALOG_COMPLETED
error_string        | [NULL]
backup_session_key  | A1pETqXv8ndvwu8z-I2_b_HjCqXKbO-tAr5Dav7M0QpUx5k2H25yEd2jMmWDof5N
restore_session_key | CmeN0iX2HD-t--YfAm9huHsjkraIYlpw9dZeZPT9-xFFSqwzswEsd8ZPlNXMAb-6
```

```
sent_bytes          | 0
write_bytes         | 0
retries             | 0
```

The snapshot ID identifies a backup that is used for replication on this database. The following records show replication in flight based on data captured in new snapshots:

```
premdb=# select * from sys.replica_status;
-[ RECORD 1 ]-------+----------------------------------------------------------------
database_id         | 16391
replica_id          | 16415
snapshot_id         | premdb_replica1_19_12_12_17_14_58
elapsed_ms          | 8784
remaining_ms        | 0
start_time          | 2019-12-12 17:14:58-08
end_time            | [NULL]
state               | USER_DATA_RUNNING
error_string        | [NULL]
backup_session_key  | DDK-d7UVx8UMmxKPl9iPO65La7lvj-X207JY1QaWi1KIZoLbbYpnfFno3Gc-HVt0
restore_session_key | DC4O8zCY6sFB1SSQILTUvZOfGMZFiOgj_XxbA6__gi9SM4ixM7xGViFp26XoxOD1
sent_bytes          | 0
write_bytes         | 0
retries             | 0
```

```
premdb=# select * from sys.replica_status;
-[ RECORD 1 ]-------+----------------------------------------------------------------
database_id         | 16391
replica_id          | 16415
snapshot_id         | premdb_replica1_19_12_12_17_28_18
elapsed_ms          | 11162
remaining_ms        | 0
start_time          | 2019-12-12 17:28:18-08
end_time            | [NULL]
state               | USER_DATA_RUNNING
error_string        | [NULL]
backup_session_key  | C1XW4jcEnywf8LQKycQLIW-_6RyYEiVUENQtqQmvnwEILJU7AJxcpeuuUaI-HSHy
restore_session_key | DH6janEMJMwUgCn3qtlS6030IGO_DdTefLrHin9BdbZHVdQSYL5aTF9Ki5U9DFOv
sent_bytes          | 24731019
write_bytes         | 24731019
retries             | 0
```

```
premdb=# select * from sys.replica_status;
-[ RECORD 1 ]-------+----------------------------------------------------------------
database_id         | 16391
replica_id          | 16415
snapshot_id         | premdb_replica1_19_12_12_17_29_13
elapsed_ms          | 34215
remaining_ms        | 0
start_time          | 2019-12-12 17:29:13-08
end_time            | [NULL]
state               | USER_DATA_RUNNING
error_string        | [NULL]
backup_session_key  | AeL5WMy1W36EBiv0ea0BKhYxW0sV2Xlmrvzg0ucLUIKcg2JxlsU4zFnlBfIBWcbM
restore_session_key | CWW8BbTi2bi9KWq-GdupA9fYok_B1anAGy5fuA8OcsDlPD_GHN01FRftqet0_zFF
sent_bytes          | 138412896
write_bytes         | 138412896
retries             | 0
```

```
premdb=# select * from sys.replica_status;
-[ RECORD 1 ]-------+----------------------------------------------------------------
database_id         | 16391
replica_id          | 16415
```

```
snapshot_id          | premdb_replica1_19_12_12_17_34_37
elapsed_ms           | 207203
remaining_ms         | 0
start_time           | 2019-12-12 17:34:37-08
end_time             | [NULL]
state                | USER_DATA_RUNNING
error_string         | [NULL]
backup_session_key   | DmabhirRiu7r_o0DEH4YguDjg0A0oBnESEKjvH40CXVP8W4Ug1ODvtFD-2YVbjnd
restore_session_key | AxtTYw1nAb0qqRFi93rSEA0PWU9K4O-d2RRGm2fNkZg34qyHwHZ5s23wevDI51iI
sent_bytes           | 5580363831
write_bytes          | 5580363831
retries              | 0
```

Note that the replica database is in `HOT_STANDBY` mode and must remain in that mode while replication is running:

```
premdb=# \l
                                    List of databases
        Name          |    Owner    | Encoding |      Access privileges      | Read-Only | Hot Standby
----------------------+-------------+----------+-----------------------------+-----------+-------------
 premdb               | os_user     | LATIN9   |                             | f         | f
 premdb_replica1_db   | yellowbrick | LATIN9   |                             | f         | t
...
```

sys.restore

Platforms: All platforms

Parent topic: System Views

This system view captures information about active restore (`ybrestore`) operations. The columns in this view are the same as the columns in sys.log_restore.

Example

In this example, the `ybsql \watch` command tracks the progress of a running database restore operation and returns results every 5 seconds.

```
premdb=# select * from sys.restore;
(0 rows)

premdb=# \watch 5
(0 rows)

(0 rows)

Watch every 5s  Tue Jun 22 15:15:42 2021

-[ RECORD 1 ]----------+------------------------------------------------------------------
session_key            | TED9yW3bVpOtR-syMZcm_3K5smzR2mTqqbE16MOCP8xRSi_9EVFNPjottbzjojY=
session_id             | 608313
client_hostname        | localhost
client_username        | yb100
user_name              | yb100
user_id                | 16418
database_name          | premdb_full_restore
database_id            | 19579
state                  | USER_DATA_RUNNING
error_string           | [NULL]
start_time             | 2021-06-22 15:15:14-08
end_time               | [NULL]
elapsed_ms             | 3375
remaining_ms           | -1
received_bytes         | 12941
received_bytes_per_sec | 3834.370361
storage_bytes          | 3741
storage_bytes_per_sec  | 1108.444458
backup_point_id        | June22Full
chain_id               | June2021
source_database_name   | premdb
source_appliance_name  | localhost
table_name             | [NULL]

Watch every 5s  Tue Jun 22 15:15:47 2021

-[ RECORD 1 ]----------+------------------------------------------------------------------
session_key            | TED9yW3bVpOtR-syMZcm_3K5smzR2mTqqbE16MOCP8xRSi_9EVFNPjottbzjojY=
session_id             | 608313
client_hostname        | localhost
client_username        | yb100
user_name              | yb100
user_id                | 16418
database_name          | premdb_full_restore
database_id            | 19579
```

```
state                 | USER_DATA_RUNNING
error_string          | [NULL]
start_time            | 2021-06-22 15:15:14-08
end_time              | [NULL]
elapsed_ms            | 8392
remaining_ms          | 8912000
received_bytes        | 182719258
received_bytes_per_sec | 21773028
storage_bytes         | 552786
storage_bytes_per_sec | 65870.59375
backup_point_id       | June22Full
chain_id              | June2021
source_database_name  | premdb
source_appliance_name | localhost
table_name            | [NULL]

Watch every 5s  Tue Jun 22 15:15:52 2021

-[ RECORD 1 ]----------+-----------------------------------------------------------------
session_key           | TED9yW3bVpOtR-syMZcm_3K5smzR2mTqqbE16MOCP8xRSi_9EVFNPjottbzjojY=
session_id            | 608313
client_hostname       | localhost
client_username       | yb100
user_name             | yb100
user_id               | 16418
database_name         | premdb_full_restore
database_id           | 19579
state                 | USER_DATA_RUNNING
error_string          | [NULL]
start_time            | 2021-06-22 15:15:14-08
end_time              | [NULL]
elapsed_ms            | 13405
remaining_ms          | 22000
received_bytes        | 487804302
received_bytes_per_sec | 36389728
storage_bytes         | 552786
storage_bytes_per_sec | 41237.300781
backup_point_id       | June22Full
chain_id              | June2021
source_database_name  | premdb
source_appliance_name | localhost
table_name            | [NULL]

Watch every 5s  Tue Jun 22 15:15:57 2021

-[ RECORD 1 ]----------+-----------------------------------------------------------------
session_key           | TED9yW3bVpOtR-syMZcm_3K5smzR2mTqqbE16MOCP8xRSi_9EVFNPjottbzjojY=
session_id            | 608313
client_hostname       | localhost
client_username       | yb100
user_name             | yb100
user_id               | 16418
database_name         | premdb_full_restore
database_id           | 19579
state                 | USER_DATA_RUNNING
error_string          | [NULL]
start_time            | 2021-06-22 15:15:14-08
end_time              | [NULL]
elapsed_ms            | 43490
remaining_ms          | 2000
received_bytes        | 1807300426
received_bytes_per_sec | 41556688
storage_bytes         | 1806697658
storage_bytes_per_sec | 41542828
backup_point_id       | June22Full
chain_id              | June2021
source_database_name  | premdb
source_appliance_name | localhost
```

```
table_name             | [NULL]

Watch every 5s  Tue Jun 22 15:16:02 2021

-[ RECORD 1 ]----------+-----------------------------------------------------------------
session_key            | TED9yW3bVpOtR-syMZcm_3K5smzR2mTqqbE16MOCP8xRSi_9EVFNPjottbzjojY=
session_id             | 608313
client_hostname        | localhost
client_username        | yb100
user_name              | yb100
user_id                | 16418
database_name          | premdb_full_restore
database_id            | 19579
state                  | DONE
error_string           | [NULL]
start_time             | 2021-06-22 15:15:14-08
end_time               | 2021-06-22 15:16:00-08
elapsed_ms             | 45717
remaining_ms           | 0
received_bytes         | 1807301704
received_bytes_per_sec | 39532376
storage_bytes          | 1806698912
storage_bytes_per_sec  | 39519192
backup_point_id        | June22Full
chain_id               | June2021
source_database_name   | premdb
source_appliance_name  | localhost
table_name             | [NULL]
```

sys.retention

This system view captures information about the retention policies on system relations. See also Retention policies for system relations.

Column Name	Data Type	Description
database_id	bigint	The unique ID for the database.
class_id	bigint	The unique ID for the relation. Relations in different schemas within the same database may have the same name, so you can use this ID to uniquely identify a relation.
retention_ms	bigint	The retention age for that relation, in milliseconds. See ALTER TABLE to set that field.
retention_bytes	bigint	The retention age for that relation, in bytes. See ALTER TABLE to set that field.
enabled	boolean	Whether this retention policy is enabled.
last_retention_run	timestamp with time zone	The last time the retention process run against this relation. See sys.log_retention for more history information.

Examples

We can join `sys.retention` with `sys.view` to have string names instead of IDs:

```sql
SELECT v.name AS relation, r.retention_ms, r.retention_bytes, r.enabled, r.last_retention_run
    FROM sys.retention r
    JOIN sys.view v ON r.class_id = v.view_id AND r.database_id = v.database_id
    ORDER BY relation;
```

```console
premdb=#
      relation      | retention_ms | retention_bytes | enabled |      last_retention_run
--------------------+--------------+-----------------+---------+------------------------------
 log_analyze        |   7776000000 |      1073741824 | t       | 2023-11-19 13:05:24.226491+00
 log_authentication |              |                 | f       |
 log_backup         |   7776000000 |      1073741824 | t       | 2023-11-19 13:06:12.004477+00
 log_cluster_event  |   7776000000 |      1073741824 | t       | 2023-11-19 13:06:12.001678+00
 log_load           |   7776000000 |      1073741824 | t       | 2023-11-19 13:06:11.995431+00
 log_query          |   2592000000 |     34359738368 | t       |
 log_query_analyze  |   2592000000 |     34359738368 | t       | 2023-11-19 13:05:24.214316+00
 log_replica_status |   7776000000 |      1073741824 | t       | 2023-11-19 13:06:12.012928+00
 log_restore        |   7776000000 |      1073741824 | t       | 2023-11-19 13:06:12.00742+00
 log_retention      |   7776000000 |      1073741824 | t       | 2023-11-19 13:06:12.010452+00
 log_session        |              |                 | f       |
 log_unload         |   7776000000 |      1073741824 | t       | 2023-11-19 13:06:11.998813+00
(12 rows)
```

sys.role

Platforms: All platforms

Parent topic: System Views

This system view returns a list of roles.

Superusers can see all roles in this view. Users and roles with `CREATE ROLE`, `ALTER ANY ROLE`, or `DROP ANY ROLE` privilege can also see all roles in this view.

Users and roles that are not superusers and do not have (or inherit) `CREATE ROLE`, `ALTER ANY ROLE`, or `DROP ANY ROLE` privilege can only see their own accounts in this view.

Note: This view returns only those roles and users that are created with the `NOLOGIN` attribute. Roles and users with `LOGIN` access are not included. See also sys.user.

Column Name	Data Type	Description
role_id	bigint	Unique role ID.
name	text	Name of the role.
create_role	boolean	Whether this role has the privilege to create other roles.
create_database	boolean	Whether this role has the privilege to create databases.
superuser	boolean	Whether this role has superuser privileges.
ldap_synchronized	boolean	Whether this role is synchronized via an LDAP server group.
default_cluster	text	Default virtual compute cluster that was set for the role, if any.
memberof	name[]	Which roles this role is a member of, if any.

Examples

For example:

```
premdb=# select * from sys.role;
 role_id |  name  | create_role | create_database | superuser | ldap_synchronized | memberof
---------+--------+-------------+-----------------+-----------+-------------------+----------
   16486 | qa     | t           | t               | f         | f                 | {}
   16485 | backup | f           | f               | f         | f                 | {}
(2 rows)
```

The following role is not listed in `sys.role` because it was created with the `login` attribute:

```
premdb=# create role bar_role login;
CREATE ROLE
premdb=# select * from sys.role where name='bar_role';
 role_id | name | create_role | create_database | superuser | ldap_synchronized | memberof
---------+------+-------------+-----------------+-----------+-------------------+----------
(0 rows)
```

sys.schema

A system view that returns information about the schemas in the database.

Column Name	Data Type	Description
schema_id	bigint	Unique ID of the schema. User-defined schemas have IDs greater than `16000`.
database_id	bigint	Unique ID of the database where the schema belongs. This view returns schemas for all databases.
owner_id	bigint	Unique ID of the user who owns the schema. Join to `sys.user` to get the name of the user.
name	varchar(256)	Name of the schema.
creation_time	timestamptz	When the schema was created. `NULL` for `sys` and `public` schemas.
max_size_bytes	bigint	Maximum size of schema set by the quota. `NULL` if no disk usage limit is set.
default_location	name	`LOCAL`

For example:

```
yellowbrick=# select * from sys.schema order by 4,2,1;
 schema_id | database_id | owner_id |    name     |         creation_time         | max_size_bytes
-----------+-------------+----------+-------------+-------------------------------+----------------
     16552 |       16520 |    16007 | premdb_bobr | 2020-10-26 14:41:36.499661-07 |         [NULL]
     16553 |       16520 |    16007 | premdb_test | 2020-10-26 14:44:52.985893-07 |         [NULL]
      2200 |        4400 |       10 | public      | [NULL]                        |         [NULL]
      2200 |       16392 |       10 | public      | [NULL]                        |         [NULL]
      2200 |       16393 |       10 | public      | [NULL]                        |         [NULL]
      2200 |       16520 |       10 | public      | [NULL]                        |         [NULL]
      3300 |        4400 |       10 | sys         | [NULL]                        |         [NULL]
      3300 |       16392 |       10 | sys         | [NULL]                        |         [NULL]
      3300 |       16393 |       10 | sys         | [NULL]                        |         [NULL]
      3300 |       16520 |       10 | sys         | [NULL]                        |         [NULL]
(10 rows)
```

sys.sequence

This view captures information about the sequences that exist in the system. See Generating Values with Sequences.

Column Name	Data Type	Description
database_id	bigint	The unique ID for the database.
schema_id	bigint	The unique schema ID for the sequence.
sequence_id	bigint	The unique ID for the sequence.
name	name	The user-defined name of the sequence.
fullname	text	The user-defined name of the sequence, pre-pended with the schema name.
start_value	bigint	The current starting value for the sequence.
last_value	bigint	The last value of the sequence, as calculated below.
min_value	bigint	The minimum value that the sequence can generate.
max_value	bigint	The maximum value that the sequence can generate.
is_temp	boolean	Whether the sequence was created as a temporary sequence (`f` =persistent; `t` =temporary).
creation_time	timestamptz	When the sequence was created.

About *last_value*

The value in the `last_value` column is derived using the following formula:

```sql
last_value = (NEXTVAL('my_seq_name') - sys.worker_id()) / 1024
```

Where:

- `NEXTVAL` returns the next number from the sequence.
- `sys.worker_id()` returns the compute node ID.

Example

```
premdb=# create sequence matchid start 50000;
CREATE SEQUENCE
premdb=> select * from sys.sequence;
 database_id | schema_id | sequence_id | owner_id |  name   |    fullname    | start_value | last_value |      min_value       |
-------------+-----------+-------------+----------+---------+----------------+-------------+------------+----------------------+--
       16400 |     24638 |       25507 |    25048 | matchid | premdb.matchid |          49 |         49 | -9223372036854775807 | 92
(1 row)
```

```
premdb=> select nextval('matchid');
 nextval
---------
   51199
(1 row)
```

sys.session

This view returns a list of current database sessions.

Column Name	Data Type	Description
session_id	bigint	Unique ID of the session.
application_name	varchar(256)	Name of the application that is running the session, such as `ybsql`, `ym`, `ybload`, or `ybunload`. You will also see internal application names such as `yb-lime`. (Note that an unload session reports `ybunload` as the application name while it is running. When the unload is complete, the view reports `yb-lime` as the application name.) The application name for replication operations is `replication service`.
client_hostname	varchar(256)	Client host name, if available.
client_ip_address	varchar(256)	Client IP address.
database_id	bigint	Unique ID for the database.
user_id	bigint	Unique ID for the database user.
start_time	timestamptz	When the session started.
process_id	bigint	Unique process ID for the session. Used internally for debugging purposes.
secure_connection	boolean	Whether the database connection is secure (SSL).
last_statement	timestamptz	When the last SQL statement was started. This column is `NULL` if the session did not execute any statements.
state	varchar(256)	If the session was shut down because of `idle_in_transaction_timeout`, the state is `25P03`. If the session was shut down because of `idle_session_timeout`, the state is `25P04`. Other states are `idle` and `active`. See also Managing Idle Sessions.
secure_details	varchar(256)	SSL version, cipher, and key size.
cluster_name	varchar(256)	Name of the compute cluster where the session is running.

Examples

Join the `sys.session`, `sys.user`, and `sys.database` views to return the details about current `ybsql` sessions:

```
premdb=> select ss.session_id, su.user_id, su.name username, sd.database_id, sd.name dbname
from sys.session ss, sys.user su, sys.database sd
where ss.user_id=su.user_id and ss.database_id=sd.database_id
 session_id | user_id |          username           | database_id | dbname
------------+---------+-----------------------------+-------------+--------
   1499660 |   16399 | trebor@yellowbrickcloud.com |       16400 | premdb
(1 row)
```

Return all active `ybsql` sessions:

```
premdb=> select * from sys.session where state='active' and application_name='ybsql';
-[ RECORD 1 ]-----+--------------------------------------------
session_id        | 1499490
application_name  | ybsql
client_hostname   |
client_ip_address | 10.0.123.231/32
database_id       | 16400
user_id           | 16395
start_time        | 2022-10-03 18:07:07.383345-07
process_id        | 42120
secure_connection | t
last_statement    | 2022-10-03 18:08:44.929571-07
state             | active
secure_details    | TLSv1.2/ECDHE-RSA-AES256-GCM-SHA384/256 bits
cluster_name      | large-default-cluster
```

sys.storage

This system view returns current storage statistics per compute node.

Column Name	Data Type	Description
worker_id	uuid	Unique ID for the compute node.
total_bytes	bigint	Total number of bytes of storage on the node.
used_bytes	bigint	Number of bytes in use on the node. This value is not a precise sum of other values in the view. You cannot derive the exact amount of space used per compute node; `used_bytes` is an approximation, roughly equal to: `distributed_bytes + replicated_bytes + random_bytes + scratch_bytes + some overhead`
free_bytes	bigint	Number of bytes available on the node. Free bytes = `total_bytes` - `used_bytes`.
distributed_bytes	bigint	Total number of bytes of hash-distributed data on the compute node.
replicated_bytes	bigint	Total number of bytes of replicated data on the compute node.
random_bytes	bigint	Total bytes used for randomly distributed data on the compute node.
scratch_bytes	bigint	Total bytes reserved for temporary space (spilling).
overhead_bytes	bigint	Storage overhead.

Example

For example, the cluster in this case has 15 nodes, with either 4TB or 2TB per node.

```
yellowbrick_test=# select worker_id, total_bytes, used_bytes, free_bytes, distributed_bytes from sys.storage;
            worker_id               |  total_bytes  |  used_bytes   |  free_bytes   | distributed_bytes
------------------------------------+---------------+---------------+---------------+------------------
 00000000-0000-0000-0000-38b8ebd000b4 | 2048430964736 |  635417362432 | 1413013602304 |      104893251584
 00000000-0000-0000-0000-38b8ebd0024e | 4096828375040 | 1045096824832 | 3051731550208 |      104891154432
 00000000-0000-0000-0000-38b8ebd009bf | 4096828375040 | 1045094727680 | 3051733647360 |      104895348736
 00000000-0000-0000-0000-38b8ebd001a9 | 2048430964736 |  635432042496 | 1412998922240 |      104901640192
 00000000-0000-0000-0000-38b8ebd0086b | 4096828375040 | 1045119893504 | 3051708481536 |      104914223104
 00000000-0000-0000-0000-38b8ebd001f9 | 4096828375040 | 1045147156480 | 3051681218560 |      104920514560
 00000000-0000-0000-0000-38b8ebd009f1 | 4096828375040 | 1045084241920 | 3051744133120 |      104886960128
 00000000-0000-0000-0000-38b8ebd0023f | 2048430964736 |  635417362432 | 1413013602304 |      104895348736
 00000000-0000-0000-0000-38b8ebd00023 | 2048430964736 |  635417362432 | 1413013602304 |      104899543040
 00000000-0000-0000-0000-38b8ebd0082f | 4096828375040 | 1045113602048 | 3051714772992 |      104910028800
 00000000-0000-0000-0000-38b8ebd00163 | 2048430964736 |  635432042496 | 1412998922240 |      104905834496
 00000000-0000-0000-0000-38b8ebd009a6 | 4096828375040 | 1045080047616 | 3051748327424 |      104884862976
 00000000-0000-0000-0000-38b8ebd00087 | 4096828375040 | 1045096824832 | 3051731550208 |      104899543040
 00000000-0000-0000-0000-38b8ebd00866 | 4096828375040 | 1045088436224 | 3051739938816 |      104884862976
```

```
 00000000-0000-0000-0000-38b8ebd00235 | 4096828375040 | 1045098921984 | 3051729453056 |      104891154432
(15 rows)
```

sys.table

Platforms: All platforms

Parent topic: System Views

A system view that returns information about the user-defined tables in the database.

Note: The `yellowbrick` database stores a set of `yb_*` tables that are visible to superusers only. These tables have four-digit table IDs and appear in results when superusers query `sys.table` and other views that have the `sys.table_` prefix.

Column Name	Data Type	Description
table_id	bigint	The unique ID for the table. Tables in different schemas within the same database may have the same name, so you can use this ID to uniquely identify a table.
database_id	bigint	The unique ID for the database.
name	text	Name of the table.
schema_id	bigint	Unique ID of the database schema where the table was created.
owner_id	bigint	Unique ID of the owner of the table, referring to the user who created it.
distribution	varchar(256)	Distribution type: `replicated`, `hash`, `random`. See Distribution Options.
sort_key	text	Column name if sorted; otherwise `null`. See Sorted and Clustered Tables
distribution_key	text	Column name if distributed by `hash`; otherwise `null`. See Distribution Options.
cluster_keys	text	Columns used as cluster keys, if any. See Sorted and Clustered Tables
partition_keys	text	Columns used as partitioning columns, if any. See Partitioning Options.
is_temp	boolean	Whether the table was created as a temporary table (`f` =persistent; `t` =temporary).
definition	text(4)	*Not implemented*.
is_auto_analyze	varchar	`f` =table not enabled for auto-analyze; `t` =table enabled for auto-analyze.
auto_analyze_policy	text	Name of the auto-analyze policy enabled for the table, if any (or `default_policy`).
last_analyzed	timestamptz	When the table was last analyzed (either manually or automatically).
creation_time	timestamptz	When the table was created.
rowstore_bytes	bigint	Number of bytes committed to the row store for the table.
rowstore_row_count	bigint	Number of rows committed to the row store for the table.
rowstore_disk_usage	bigint	Total size of files currently used in the row store for the table. An entire file (32MB) is reported as in use even if there is only a single row in the table. In general, partially filled files are filled up before new ones are created.
rowstore_size_limit	bigint	Total size allowed in the row store for the table. The default is 500GB. See also ALTER TABLE.

Column Name	Data Type	Description
rowstore_full_action	text	Action taken when the row store size limit is reached for the table: `S` (Slow), `B` (Block), or `C` (Cancel). See Managing the Row Store and ALTER TABLE.
compressed_bytes	bigint	Total compressed bytes of actual, live usage on all transactions on the workers.
uncompressed_bytes	bigint	Total uncompressed bytes of actual, live usage on all transactions on the workers. This does not include bytes in the rowstore on the manager nodes.
snapshot_backup_bytes	bigint	Backup and restore system data, which is stored in internal system tables. Managed by the backups and only decreases when full backups occur. This data is not counted in the quota decision for the current transaction but is counted for subsequent transactions.
delete_info_bytes	bigint	Deleted and updated system data on the workers, which is kept in specialized internal data blocks.
reclaimable_bytes	bigint	Deleted or updated user data on the workers that will be freed up in the background.
max_size_bytes	bigint	Maximum size of the table set by the quota. `NULL` if no disk usage limit is set.
location	name	Storage location of the table (currently always `LOCAL`).

Examples

Return distribution information for tables and the timestamp for when they were last analyzed:

```
premdb=# select table_id, name, distribution, distribution_key, last_analyzed
from sys.table order by table_id;
 table_id |     name      | distribution | distribution_key |         last_analyzed
----------+---------------+--------------+------------------+-------------------------------
   16882 | season        | replicated   |                  | 2017-09-01 11:47:50.685147-07
   16885 | team          | replicated   |                  | 2017-09-01 11:47:56.03602-07
   16888 | hometeam      | replicated   |                  | 2017-09-01 11:44:19.424487-07
   16891 | awayteam      | replicated   |                  | 2017-09-01 11:44:19.032821-07
   16894 | match         | hash         | seasonid         | 2017-09-01 11:48:07.712739-07
   16898 | newmatchstats | hash         | seasonid         | 2017-09-01 11:40:18.967244-07
(6 rows)
```

Return row store information for some tables:

```
premdb_match=# select name, rowstore_bytes, rowstore_row_count, rowstore_disk_usage, rowstore_size_limit, rowstore_full_action
from sys.table where table_id>10000;
 name  | rowstore_bytes | rowstore_row_count | rowstore_disk_usage | rowstore_size_limit | rowstore_full_action
-------+----------------+--------------------+---------------------+---------------------+----------------------
 new   |            231 |                  3 |            33554432 |        549755813888 | S
 match |       33288310 |               6502 |            33554432 |            33554432 | S
(2 rows)
```

Return information for a table named `season`:

```
premdb=# select * from sys.table where name = 'season';
-[ RECORD 1 ]---------+---------
table_id              | 16635
database_id           | 16634
name                  | season
schema_id             | 2200
owner_id              | 16007
distribution          | replicated
```

```
sort_key              | [NULL]
distribution_key      | [NULL]
cluster_keys          | [NULL]
partition_keys        | [NULL]
is_temp               | f
definition            | [NULL]
is_auto_analyze       | t
auto_analyze_policy   | default_policy
last_analyzed         | [NULL]
creation_time         | 2020-10-01 15:01:01.747977-07
rowstore_bytes        | 1917
rowstore_row_count    | 25
rowstore_disk_usage   | 33554432
rowstore_size_limit   | 549755813888
rowstore_full_action  | S
compressed_bytes      | 0
uncompressed_bytes    | 0
snapshot_backup_bytes | 0
delete_info_bytes     | 0
reclaimable_bytes     | 0
max_size_bytes        | [NULL]
location              | LOCAL
```

sys.table_flush_summary

A system view that returns information about the flush status for the user-defined tables in the database. See Managing the Row Store.

Column Name	Data Type	Description
table_id	bigint	The unique ID for the table. Tables in different schemas within the same database may have the same name, so you can use this ID to uniquely identify a table.
rows_pending_insert	bigint	Total number of rows that need to be inserted into the column store (backend database).
size_pending_flush	bigint	Total size of rows (in bytes) that need to be inserted into the column store (backend database).
rows_pending_flush	bigint	Number of rows scheduled for flushing.
queued_for_flush	boolean	Whether the table is pending a needed flush operation. If `true`, some rows have not yet been flushed to the file systems on the compute nodes, and the table is in the queue for the next background flush operation.
flush_time	timestamptz	When the flush operation is scheduled (or was scheduled) to occur.
update_time	timestamptz	When the flush status was last updated (by default, the polling interval is every 30 seconds).

Examples

For example, here the view is queried three times in a row, just after `1049932` rows were copied into table `17017`.

```
premdb=# select * from sys.table_flush_summary;
 table_id | rows_pending_insert | size_pending_flush | rows_pending_flush | queued_for_flush |       flush_time       |      updat
----------+---------------------+--------------------+--------------------+------------------+------------------------+-----------
   16998 |                   0 |                  0 |                  0 | f                |                        | 2017-06-20
   17011 |                   0 |                  0 |                  0 | f                |                        | 2017-06-20
   17017 |             1049932 |                  0 |                  0 | f                | 2017-06-20 17:28:54-07 | 2017-06-20
   16995 |                   0 |                  0 |                  0 | f                |                        | 2017-06-20
   17001 |                   0 |                  0 |                  0 | f                |                        | 2017-06-20
   16992 |                   0 |                  0 |                  0 | f                |                        | 2017-06-20
   17004 |                   0 |                  0 |                  0 | f                |                        | 2017-06-20
   17014 |                   0 |                  0 |                  0 | f                |                        | 2017-06-20
(8 rows)

premdb=# select * from sys.table_flush_summary;
 table_id | rows_pending_insert | size_pending_flush | rows_pending_flush | queued_for_flush |       flush_time       |      updat
----------+---------------------+--------------------+--------------------+------------------+------------------------+-----------
   16998 |                   0 |                  0 |                  0 | f                |                        | 2017-06-20
   17011 |                   0 |                  0 |                  0 | f                |                        | 2017-06-20
   17017 |             1049932 |           71480152 |             100000 | t                | 2017-06-20 17:29:54-07 | 2017-06-20
   16995 |                   0 |                  0 |                  0 | f                |                        | 2017-06-20
   17001 |                   0 |                  0 |                  0 | f                |                        | 2017-06-20
   16992 |                   0 |                  0 |                  0 | f                |                        | 2017-06-20
   17004 |                   0 |                  0 |                  0 | f                |                        | 2017-06-20
   17014 |                   0 |                  0 |                  0 | f                |                        | 2017-06-20
(8 rows)
```

```
premdb=# select * from sys.table_flush_summary;
 table_id | rows_pending_insert | size_pending_flush | rows_pending_flush | queued_for_flush |       flush_time       |      updat
----------+---------------------+--------------------+--------------------+------------------+------------------------+-----------
   16998 |                   0 |                  0 |                  0 | f                |                        | 2017-06-20
   17011 |                   0 |                  0 |                  0 | f                |                        | 2017-06-20
   17017 |                   0 |                  0 |                  0 | f                | 2017-06-20 17:29:54-07 | 2017-06-20
   16995 |                   0 |                  0 |                  0 | f                |                        | 2017-06-20
   17001 |                   0 |                  0 |                  0 | f                |                        | 2017-06-20
   16992 |                   0 |                  0 |                  0 | f                |                        | 2017-06-20
   17004 |                   0 |                  0 |                  0 | f                |                        | 2017-06-20
   17014 |                   0 |                  0 |                  0 | f                |                        | 2017-06-20
(8 rows)
```

sys.table_info

Platforms: All platforms

Parent topic: System Views

This view reports the current state of database tables with respect to background operations that have been run or might need to be run. See also sys.table_storage.

Column Name	Data Type	Description
table_id	bigint	Unique table ID; join key to `sys.table`. Some internal yb_query* tables with four-digit IDs are tracked by this view; you can filter them out in queries: `where table_id>10000;`
rows_rowstore	bigint	Number of rows that have yet to be flushed from the row store to the column store. See also the `rows_columnstore` column in the `sys.table_storage` view.
modified_rows_statistics	bigint	Number of rows that were inserted or deleted since the last ANALYZE operation.
last_analyzed	timestamptz	When statistics were last updated (via an ANALYZE operation).

Example

This query joins the `sys.table_info` view to the `sys.table` and `sys.schema` views.

```
premdb=# select sct.name, sti.* from sys.table_info sti, sys.table sct, sys.schema sch
where (sti.table_id=sct.table_id) and (sct.schema_id=sch.schema_id) and (sct.database_id=sch.database_id)
and sct.table_id>10000;
   name   | table_id | rows_rowstore | modified_rows_statistics |         last_analyzed
----------+----------+---------------+--------------------------+-------------------------------
 awayteam |    17560 |        [NULL] |                   [NULL] | 2020-10-27 12:35:08.986892-07
 hometeam |    17556 |        [NULL] |                   [NULL] | 2020-10-27 12:35:09.147433-07
 team     |    17552 |        [NULL] |                   [NULL] | 2020-10-27 13:10:40.714623-07
 season   |    17548 |        [NULL] |                   [NULL] | 2020-10-27 13:10:48.675918-07
 match    |    17564 |        [NULL] |                   [NULL] | 2020-10-27 15:14:28.525819-07
(5 rows)
```

sys.table_partition

Platforms: All platforms

Parent topic: System Views

This view describes how a partitioned table is distributed by the partitions defined in the CREATE TABLE statement. Each row in this view represents a unique permutation of the partition column values in the partitioned table. The `count` column of the view is a count of the rows that are stored in a single partition. The sum of the `count` column matches the current number of rows in the partitioned table.

You can query this view to check for potential skew in the partitioning scheme for the table. If the `count` column reflects an uneven distribution, you can look for the source of the skew and find ways to fix it. For example, you may consider scaling back the number of partitions or choosing hash partitioning instead of range partitioning.

Note: This view does not account for rows that have not yet been flushed to the column store.

Column Name	Data Type	Description
table_id	bigint	Unique table ID. Only partitioned tables are captured in this view.
n1, n2, n3, n4	integer	Values in these columns are repeated, ordered sets of integers that map to actual values in the partition columns for the table. If fewer than 4 partition columns exist in a table, the unused `n` columns are filled with zeros. If `OUTSIDE RANGE` partitions were created for a table, you will see two additional rows in the view that account for these partitions. If `IS NULL` partitions were created, you will see an additional row for that partition. See Partitioning Options.
count	bigint	Row count for a given combination of partition values for a table. The `sum(count)` result against this view matches the total number of rows in the partitioned table.

Examples

For example, table `16984` is partitioned on one column:

```
premdb=# select * from sys.table_partition where table_id=16984;
 table_id | n1 | n2 | n3 | n4 | count
----------+----+----+----+----+--------
   16984 |  0 |  0 |  0 |  0 | 291060
   16984 |  1 |  0 |  0 |  0 | 291060
   16984 |  2 |  0 |  0 |  0 | 291060
   16984 |  3 |  0 |  0 |  0 | 239400
   16984 |  4 |  0 |  0 |  0 | 239400
   16984 |  5 |  0 |  0 |  0 | 239400
   16984 |  6 |  0 |  0 |  0 | 239400
...
```

The following query returns the `sum` of the `count` column, which is equivalent to the `count(*)` result from the partitioned table.

```
premdb=# select sum(count) from sys.table_partition where table_id=16984;
   sum
---------
 5421780
(1 row)
```

The following example joins to the `sys.table` view to get the table name for table `16895`. The query computes several aggregates over the `count` column to analyze skew in the partitioning.

```
premdb=# select name, stp.table_id, max(count), min(count), avg(count), stddev(count)
from sys.table_partition stp, sys.table sct where stp.table_id=sct.table_id and count <>0
group by name, stp.table_id;
          name            | table_id | max  | min  |          avg          |      stddev
--------------------------+----------+------+------+-----------------------+-------------------
 partitioned_newmatchstats |    16895 | 9576 | 8664 | 8798.9596412556053812 | 324.1986257363057
(1 row)
```

sys.table_storage

This view shows the current storage statistics for each table in the database. See also sys.table_info.

Column Name	Data Type	Description
table_id	bigint	Unique table ID; join key to `sys.table`. Some internal yb_query* tables with four-digit IDs are tracked by this view; you can filter them out in queries: `where table_id>10000;`
worker_id	uuid	Unique ID of the compute node where rows are stored. (The `worker_id` may be `null` when a table has no data, or when the data is temporarily stored in the row store.)
rows_columnstore	bigint	Number of rows in the column store (flushed to the compute nodes). See also the `rows_rowstore` column in the `sys.table_info` view.
compressed_bytes	bigint	Amount of table data stored (in compressed bytes). This number may be greater than `uncompressed_bytes`, especially for smaller tables, given the overhead of the size of a shard (unit of data storage).
uncompressed_bytes	bigint	Estimated amount of table data received (in uncompressed bytes).

Calculation of Uncompressed Bytes

The `uncompressed_bytes` is calculated based on the number of columns in the table, their data type, whether they are nullable, and the number of table rows. The computed metric is typically an estimate because an average row size is subtracted for rows that were deleted from the table, not the actual row size.

The following is a breakdown of how to calculate the number of uncompressed bytes in a table:

- Nullable columns: 1 bit per row for each column, across all rows, rounded up to the nearest byte. For example, if a table has 8 columns that allow `NULL` values, each row requires 1 byte for `NULL` values.
- Bytes per column type for each row:
- Fixed field types:
  - 1 byte: `BOOLEAN`
  - 2 bytes: `SMALLINT`
  - 4 bytes: `INTEGER`, `REAL`, `DATE`
  - 8 bytes: `BIGINT`, `DOUBLE PRECISION`, `TIME`, `TIMESTAMP`, `TIMESTAMPTZ`
  - 16 bytes: `UUID`
- Variable field types:
  - `DECIMAL`
    - 4 bytes: Precision values 1 to 9
    - 8 bytes: Precision values 10 to 18
    - 16 bytes: Precision values 19 to 38
  - `VARCHAR`: 2 bytes to store string length, plus the length of the string in bytes
  - `CHAR`: Same as `VARCHAR` (The `CHAR` data type is only conceptually a fixed-width character string; internally it is handled as a variable-width character string.)

Here is a specific example of the calculated number of uncompressed bytes for a very small table named `hometeam`, which has 2 nullable columns and 50 rows:

Calculation	Uncompressed bytes
Calculation for `htid` (`smallint` data type)	2 * 50 = 100
Calculation for `name` (`varchar(30)` data type)	(2 * 50) + 593 (sum of actual length of strings) = 693
Calculation for both nullable columns	2 bits * 50 = 100 bits = approx. 13 bytes
Total	100 + 693 + 13 = approx. 806 bytes

Deleted Rows

When rows are deleted from a table, they are only logically deleted. They persist in the table's storage structures until space is reclaimed by a periodic background operation. As a result, deleted rows consume space and are included in both the `compressed_bytes` and `uncompressed_bytes` metrics until space is reclaimed. If required, the number of logically deleted but not yet reclaimed rows can be used to estimate the uncompressed bytes of current rows.

Inserted Rows

Rows are included in the `compressed_bytes` and `uncompressed_bytes` metrics only when they are stored in the column store of the database. Rows inserted via the `ybload` bulk loader are accounted for immediately because they are loaded directly into the column store. However, because rows inserted via INSERT statements, or via tools using OBDC or JDBC drivers, are initially stored in the row store, they are not accounted for until they are flushed into the column store. This process typically takes only a few minutes; therefore, these rows appear in the storage calculations after a short delay.

Examples

Return storage information for a specific table:

```
yellowbrick_test=# select sum(rows_columnstore) from sys.table_storage where table_id=16606;
   sum
-----------
 600037902
(1 row)
```

```
yellowbrick_test=# select * from sys.table_storage where table_id=16606;
 table_id |              worker_id               | rows_columnstore | compressed_bytes | uncompressed_bytes
----------+--------------------------------------+------------------+------------------+--------------------
   16606 | 00000000-0000-0000-0000-38b8ebd00023 |         40012122 |       1692401664 |         6040322048
   16606 | 00000000-0000-0000-0000-38b8ebd00087 |         40013865 |       1692401664 |         6042157056
   16606 | 00000000-0000-0000-0000-38b8ebd000b4 |         39998238 |       1690304512 |         6038487040
   16606 | 00000000-0000-0000-0000-38b8ebd00163 |         39990064 |       1692401664 |         6038749184
   16606 | 00000000-0000-0000-0000-38b8ebd001a9 |         40007412 |       1692401664 |         6041108480
   16606 | 00000000-0000-0000-0000-38b8ebd001f9 |         39983517 |       1690304512 |         6035603456
   16606 | 00000000-0000-0000-0000-38b8ebd00235 |         40008626 |       1692401664 |         6041108480
   16606 | 00000000-0000-0000-0000-38b8ebd0023f |         40006680 |       1692401664 |         6038749184
   16606 | 00000000-0000-0000-0000-38b8ebd0024e |         39975472 |       1692401664 |         6034292736
   16606 | 00000000-0000-0000-0000-38b8ebd0082f |         40015874 |       1692401664 |         6043729920
   16606 | 00000000-0000-0000-0000-38b8ebd00866 |         39993230 |       1690304512 |         6040846336
   16606 | 00000000-0000-0000-0000-38b8ebd0086b |         40008190 |       1694498816 |         6038487040
   16606 | 00000000-0000-0000-0000-38b8ebd009a6 |         40010891 |       1692401664 |         6039273472
   16606 | 00000000-0000-0000-0000-38b8ebd009bf |         40017426 |       1692401664 |         6040846336
   16606 | 00000000-0000-0000-0000-38b8ebd009f1 |         39996295 |       1692401664 |         6037700608
(15 rows)
```

The following example joins the `sys.table_storage` view to the `sys.table` and `sys.schema` views. The query returns an aggregated count of bytes stored per table across the cluster.

```
yellowbrick=# select t.name table_name, sc.name schema_name, sum(t.compressed_bytes) bytes_stored
yellowbrick-# from sys.table t, sys.schema sc, sys.table_storage s
yellowbrick-# where (t.table_id = s.table_id) and  (t.schema_id = sc.schema_id) and (t.database_id = sc.database_id)
yellowbrick-# group by 1, 2
yellowbrick-# order by 2, 1;
  table_name   | schema_name | bytes_stored
---------------+-------------+--------------
 newt          | premdb_bobr |            0
 awayteam      | public      |   1887436800
 hometeam      | public      |   1887436800
 match         | public      |   1887436800
 newmatchstats | public      | 2656881868800
 season        | public      |   1887436800
 t1            | public      |            0
 team          | public      |   1887436800
(8 rows)
```

sys.unload

Platforms: All platforms

Parent topic: System Views

This system view captures information about active unload (`ybunload`) operations.

This view has the same columns as sys.log_unload, except for the `last_activity_time` column, which replaces the `end_time` column. The `last_activity_time` column returns a timestamp that shows when unload activity was last reported.

For example:

```
premdb=# select * from sys.unload;
...
-[ RECORD 2 ]---------+-----------------------------------------------------------------
session_key           | Atd2C9bb37QtHfKGgvM4M84OoWsnuyzZOcrOYfA_NZ1-6Md6QHlKZnSgaLxlq0To
state                 | DONE
error_string          | [NULL]
database_id           | 16423
user_id               | 16434
session_id            | 16790
database_name         | premdb
username              | bobr
client_hostname       | localhost
client_username       | yb100
start_time            | 2018-05-02 17:35:36-07
last_activity_time    | 2018-05-02 17:35:47-07
elapsed_ms            | 10453
sql_query             | select * from "newmatchstats"
sent_rows             | 25990128
sent_bytes            | 659797740
sent_rows_per_second  | 2362738.90909091
sent_bytes_per_second | 59981612.7272727
```

sys.user

A system view that returns information about the users in the system. See also sys.role.

Superusers can see all users in this view. Users and roles with `CREATE ROLE`, `ALTER ANY ROLE`, or `DROP ANY ROLE` privilege can also see all users in this view.

Users and roles that are not superusers and do not have (or inherit) `CREATE ROLE`, `ALTER ANY ROLE`, or `DROP ANY ROLE` privilege can only see their own accounts in this view.

Column Name	Data Type	Description
user_id	bigint	The unique ID for the user or role.
name	text	Name of the user or role.
search_path	text	Value of the `search_path` parameter for the user (a comma-delimited list of schemas).
create_role	boolean	Whether the user can create other users.
create_database	boolean	Whether the user can create databases.
superuser	boolean	Whether the user has superuser privileges.
has_local_password	boolean	`true` means that the user has a local password set via `ALTER USER` (or `CREATE USER`). `false` means that the user does not have a local password and may depend on an LDAP password for login.
ldap_synchronized	boolean	`true` means that this user exists in the database via LDAP synchronization.
default_cluster	text	Default virtual compute cluster that was set for the role, if any.
memberof	name[]	Roles that the user is a member of.
connection_limit	smallint	Max number of connections the user can have. This is reserved for future use.
is_inherit	boolean	Is this user or role inherting permission? This is reserved for future use.
is_password_encrypted	boolean	
last_login	timestampz	When the user was last logged in?
last_password_change	timestampz	When did the user last changed their password?
creation_time	timestampz	When the user was created.
login_attempts	integer	Current number of failed login attempts.

Example

```
yellowbrick_test=> select * from sys.user where name='bobr';
 user_id | name |   search_path    | create_role | create_database | superuser | has_local_password | ldap_synchronized | default_
---------+------+------------------+-------------+-----------------+-----------+--------------------+-------------------+---------
```

```
   38165 | bobr | "public, premdb" | f           | f             | f         | f                | f               | default
(1 row)
```

sys.view

A system view that returns information about the system and user-defined views in the database.

Column Name	Data Type	Description
view_id	bigint	Unique ID of the view.
database_id	bigint	Unique ID of the database where the view was created.
schema_id	bigint	Unique ID of the schema where the view was created. Join to `sys.schema` to get the schema name.
name	varchar(255)	Name of the view.
owner_id	bigint	Unique ID of the user who owns the view. Join to `sys.user` to get the name of the user.
definition	text	SQL text of the view.
is_temp	boolean	Whether the view was created as a temporary view (`f` =persistent; `t` =temporary).
creation_time	timestamptz	When the view was created.
database_name	varchar(255)	Name of the database where the procedure was created.

This view also returns the system views and views that belong to the `INFORMATION_SCHEMA` (`schema_id 10000`).

For example:

```
premdb=# select sv.schema_id, ss.name, sv.name, sv.database_name
from sys.view sv join sys.schema ss on ss.schema_id=sv.schema_id and ss.database_id=sv.database_id
where database_name='premdb' and view_id>16000 order by 1,2;
 schema_id |  name  |   name    | database_name
-----------+--------+-----------+---------------
      2200 | public | teamview  | premdb
      2200 | public | matchview | premdb
     16439 | prem   | matchview | premdb
     16439 | prem   | teamview  | premdb
(4 rows)
```

```
premdb=# select * from sys.view
where schema_id=16439 and name='teamview';
 view_id | database_id | schema_id |   name   | owner_id |      definition      | is_temp |        creation_time         | databas
---------+-------------+-----------+----------+----------+----------------------+---------+------------------------------+--------
   16440 |       16402 |     16439 | teamview |    16007 |  SELECT team.teamid,+| f       | 2020-11-09 14:00:34.63173-08 | premdb
         |             |           |          |          |     team.htid,      +|         |                              |
         |             |           |          |          |     team.atid,      +|         |                              |
         |             |           |          |          |     team.name,      +|         |                              |
         |             |           |          |          |     team.nickname,  +|         |                              |
         |             |           |          |          |     team.city,      +|         |                              |
         |             |           |          |          |     team.stadium,   +|         |                              |
         |             |           |          |          |     team.capacity,  +|         |                              |
         |             |           |          |          |     team.avg_att    +|         |                              |
```

```
        |           |          |         |         |    FROM team;       |        |                          |
(1 row)
```

sys.wlm_active_pool

This view returns the currently *active* WLM resource pool per compute cluster and identifies its profile and other attributes, including the number of pending queries for each prioritized queue.

Column Name	Data Type	Description
cluster_id	uuid	Unique ID of the compute cluster.
name	text	Name of the resource pool.
profile_name	text	Name of the profile where the resource pool was created.
memory_requested	text	Percentage or amount of available memory requested for the pool. This column returns one of three character strings: - A percentage, such as `22%` - A value in MB, such as `32768MB` - `remainder`, which means the resource pool is requesting whatever memory is left over (equivalent to `Any` in Yellowbrick Manager).
memory_per_query_bytes	bigint	Actual amount of memory available per query, in bytes.
temp_space_requested	text	Percentage or amount of temporary space requested for spilling, as a percentage. This column returns one of three character strings: - A percentage, such as `22%` - A value in MB, such as `32768MB` - `remainder`, which means the resource pool is requesting whatever temporary space is left over (equivalent to `Any` in Yellowbrick Manager).
temp_space_per_query_bytes	bigint	Actual amount of temporary space for spilling available per query, in bytes.
max_concurrency	bigint	Maximum number of concurrent queries that can run in this pool.
min_concurrency	bigint	Minimum number of concurrent queries that can run in this pool.
queue_size	bigint	Number of queries that can wait in the queue for this pool.
max_row_limit	bigint	Maximum number of rows that a query can return when executed in this pool. `NULL` if not set.
max_wait_time_limit	bigint	Maximum number of seconds that a query can wait in this pool. `NULL` if not set.
max_exec_time_limit	bigint	Maximum duration (number of seconds) for a query that executes in this pool. `NULL` if not set.
running	bigint	Number of queries running on compute nodes assigned to the pool.
low_pending, normal_pending, high_pending, critical_pending	bigint	Number of queries waiting in each prioritized queue (low, normal, high, critical).

Example

For example:

```
premdb=> select name, profile_name, max_concurrency, min_concurrency, queue_size, max_row_limit, max_exec_time_limit
from sys.wlm_active_pool
where profile_name='sqb';
     name      | profile_name | max_concurrency | min_concurrency | queue_size | max_row_limit | max_exec_time_limit
---------------+--------------+-----------------+-----------------+------------+---------------+---------------------
 longpool      | sqb          |               2 |               1 |        100 |               |
 shortquerypool | sqb          |              20 |               1 |        100 |        100000 |                   1
(2 rows)
```

sys.wlm_active_profile

This view returns the currently active WLM profile for each compute cluster, identified by its cluster ID (UUID).

For example:

```
premdb=> select * from sys.wlm_active_profile;
  profile_name  |              cluster_id
----------------+--------------------------------------
 default        | 96842a0a-4627-4e39-aaf0-94968d6b22b4
 instance_small | 56ec98dd-b475-4a33-a499-90b29cef87e8
(2 rows)
```

Join this view to `sys.cluster` for more information:

```
premdb=> select sw.*, sc.cluster_name from sys.wlm_active_profile sw join sys.cluster sc on sw.cluster_id=sc.cluster_id;
  profile_name  |              cluster_id              |     cluster_name
----------------+--------------------------------------+----------------------
 default        | 96842a0a-4627-4e39-aaf0-94968d6b22b4 | large-default-cluster
 instance_small | 56ec98dd-b475-4a33-a499-90b29cef87e8 | small-2node-cluster
(2 rows)
```

sys.wlm_active_rule

This view returns the currently active WLM rules and identifies the JavaScript expression for each rule, rule type, and other attributes. See sys.wlm_pending_rule for a description of the columns.

Example

For example:

```
yellowbrick=# select * from sys.wlm_active_rule where rule_type='submit';
  profile_name  |     rule_name     | rule_type | order | enabled | superuser |                              expression
----------------+-------------------+-----------+-------+---------+-----------+-----------------------------------------------------
 shortquerybias | limit_concurrency | submit    |     1 | t       | f         | if (w.user === 'sqb') {
                |                   |           |       |         |           |         wlm.throttle(2, w.application);
                |                   |           |       |         |           | }
 shortquerybias | log_premdb        | submit    |   100 | t       | f         |  if (w.database === 'premdb') {log.info('premdb qu
(2 rows)
```

sys.wlm_pending_pool

This view returns the current list of pending WLM resource pools (inactive pools) and identifies their profile and other attributes.

Column Name	Data Type	Description
cluster_id	uuid	Unique ID of the compute cluster.
name	text	Name of the resource pool.
profile_name	text	Name of the profile where the resource pool was created.
memory_requested	text	Percentage or amount of available memory requested for the pool. This column returns one of three character strings: - A percentage, such as `22%` - A value in MB, such as `32768MB` - `remainder`, which means the resource pool is requesting whatever memory is left over (equivalent to `Any` in Yellowbrick Manager).
memory_per_query_bytes	bigint	Actual amount of memory available per query, in bytes.
temp_space_requested	text	Percentage or amount of temporary space requested for spilling, as a percentage. This column returns one of three character strings: - A percentage, such as `22%` - A value in MB, such as `32768MB` - `remainder`, which means the resource pool is requesting whatever temporary space is left over (equivalent to `Any` in Yellowbrick Manager).
temp_space_per_query_bytes	bigint	Actual amount of temporary space for spilling available per query, in bytes.
max_concurrency	bigint	Maximum number of concurrent queries that can run in this pool.
min_concurrency	bigint	Minimum number of concurrent queries that can run in this pool.
queue_size	bigint	Number of queries that can wait in the queue for this pool.
max_row_limit	bigint	Maximum number of rows that a query can return when executed in this pool. `NULL` if not set.
max_wait_time_limit	bigint	Maximum number of seconds that a query can wait in this pool. `NULL` if not set.
max_exec_time_limit	bigint	Maximum duration (number of seconds) for a query that executes in this pool. `NULL` if not set.

Example

For example:

```
premdb=> select * from sys.wlm_pending_pool where profile_name='sqb';
            cluster_id              |      name      | profile_name | memory_requested | memory_per_query_bytes | temp_space_requ
------------------------------------+----------------+--------------+------------------+------------------------+---------------
 96842a0a-4627-4e39-aaf0-94968d6b22b4 | shortquerypool | sqb          | remainder        |           34735128576 | remainder
```

```
 96842a0a-4627-4e39-aaf0-94968d6b22b4 | system         | sqb          | 13312MB          |            4652531712 | 98304MB
 96842a0a-4627-4e39-aaf0-94968d6b22b4 | longpool       | sqb          | remainder        |           34736177152 | remainder
 56ec98dd-b475-4a33-a499-90b29cef87e8 | shortquerypool | sqb          | remainder        |            2592079872 | remainder
 56ec98dd-b475-4a33-a499-90b29cef87e8 | longpool       | sqb          | remainder        |            2593128448 | remainder
(5 rows)
```

sys.wlm_pending_rule

This view returns the current list of pending WLM rules (inactive rules) and identifies the JavaScript expression for each rule, rule type, and other attributes.

Column Name	Data Type	Description
profile_name	text	Name of the profile that the rule is associated with, or `(global)` if it belongs to all profiles.
rule_name	name	Name of the rule.
rule_type	text	Type of rule: `submit`, `assemble`, `compile`, `restart_for_error`, `restart_for_user`, `runtime`, or `completion`
order	integer	Rule order, expressed as a number. This number represents the order in which rules are applied, with the lowest applied first and the highest applied last.
enabled	boolean	Whether the rule is enabled (in use).
superuser	boolean	Whether the rule applies to superusers or non-superusers.
expression	text	The JavaScript expression that defines the conditions and actions for the rule.

For example:

```
yellowbrick=# select * from sys.wlm_pending_rule where profile_name='shortquerybias';
  profile_name  |     rule_name     |    rule_type     | order | enabled | superuser |                                expression
----------------+-------------------+------------------+-------+---------+-----------+-------------------------------------------------
 shortquerybias | fastlane          | compile          |   100 | t       | f         |  if (w.database === 'premdb') { w.resourceP
 shortquerybias | log_premdb        | submit           |   100 | t       | f         |  if (w.database === 'premdb') {log.info('pr
 shortquerybias | limit_concurrency | submit           |     1 | t       | f         | if (w.user === 'sqb') {
                |                   |                  |       |         |           |         wlm.throttle(2, w.application);
                |                   |                  |       |         |           | }
 shortquerybias | assemble_rule     | assemble         |     1 | t       | f         | if ((String(w.SQLText).indexOf('newmatchsta
                |                   |                  |       |         |           |         w.maximumExecTimeLimit = 3;
                |                   |                  |       |         |           | }
 shortquerybias | restart_admin     | restart_for_user |     1 | t       | f         | if (w.application === 'ybsql') {
                |                   |                  |       |         |           |         w.statsStored = true;
                |                   |                  |       |         |           | }
(5 rows)
```

sys.worker

This view captures the current state and role of each compute node on each compute cluster.

Column Name	Data Type	Description
worker_id	uuid	UUID for the compute node.
cluster_id	uuid	UUID for the compute cluster.
chassis_id	integer	`0` for a single-chassis system; `0` or `1` for a dual-chassis system.
logical_id	integer	Logical ID number for the compute node.
ip_address	text	IP address for the compute node.
state	text	Current state, such as `IN_SERVICE` or `OFFLINE`.
role	text	Current role, such as `MEMBER` or `SPARE`. This value corresponds to the "Cluster role" for the blade.
state_updated	timestamptz	When the state of the compute changed.
drive_status	text	Status of the compute node drives: `OK` (all drives online) or `DEGRADED` (one or more drives offline).

You can monitor the compute nodes at the cluster level with the `sys.worker` view by joining other system views on the `cluster_id` column.

For example, the role of a compute is `SPARE` when a cluster is suspended and `MEMBER` when it is fully resumed:

```
premdb=# select c.cluster_id, c.status, c.offline_reason, c.state, w.role, cl.cluster_name
from sys.cluster_status c join sys.worker w on c.cluster_id=w.cluster_id
join sys.cluster cl on c.cluster_id=cl.cluster_id;
             cluster_id              |    status    |      offline_reason      |   state   |  role  |      cluster_name
-------------------------------------+--------------+--------------------------+-----------+--------+------------------------
 da88c52f-c876-43f9-9777-6121e1b0da0e | NOT_DEGRADED | NOT_OFFLINE              | RUNNING   | MEMBER | test-may3-cluster-01
 7b942fdd-7293-47d6-86f4-de994daa300d | NOT_DEGRADED | TOO_MANY_MISSING_WORKERS | SUSPENDED | SPARE  | bobr-may3-small-cluster
 7b942fdd-7293-47d6-86f4-de994daa300d | NOT_DEGRADED | TOO_MANY_MISSING_WORKERS | SUSPENDED | SPARE  | bobr-may3-small-cluster
(3 rows)
...
             cluster_id              |    status    | offline_reason |  state  |  role  |      cluster_name
-------------------------------------+--------------+----------------+---------+--------+------------------------
 da88c52f-c876-43f9-9777-6121e1b0da0e | NOT_DEGRADED | NOT_OFFLINE    | RUNNING | MEMBER | test-may3-cluster-01
 7b942fdd-7293-47d6-86f4-de994daa300d | NOT_DEGRADED | NOT_OFFLINE    | RUNNING | MEMBER | bobr-may3-small-cluster
 7b942fdd-7293-47d6-86f4-de994daa300d | NOT_DEGRADED | NOT_OFFLINE    | RUNNING | MEMBER | bobr-may3-small-cluster
(3 rows)
```

Workload Management

Platforms: All platforms

Parent topic: Databases

In this section:

This section describes how to make the best use of system resources by managing workloads.

Creating WLM Resource Pools

Platforms: All platforms

Parent topic: Workload Management

In this section:

The main purpose of a resource pool is to allocate *memory* to queries and other database operations. When a query is queued for execution, it lands in a resource pool based on criteria defined in WLM rules, and that resource pool executes the query if it can.

Note: If a resource pool cannot execute the query within a given time limit or with the memory available to the pool, the system (or an administrator) may move the query into another pool for execution.

A resource pool belongs to a specific profile. Every profile must have a user-defined pool marked as the `default` pool. You can add any number of user-defined pools to a profile, provided that you mark one of them as the `default` pool.

A single `system` pool, which is created by default and cannot be removed or modified, executes work on the default cluster, the running cluster, or the system cluster (if defined). That is, this pool only exists in the profile that is currently activated on the current cluster. When you activate a profile, you will see the `system` pool listed and the total resources of the cluster shared among the other pools plus the `system` pool. In order to avoid wasting resources, the `system` pool is only included on one cluster.

Because different instances and clusters have different hardware instance types, your mileage will vary with respect to resource allocation when a profile is activated for a cluster. Dividing a non-fixed amount of memory for a pool among a given number of slots will have different results on clusters backed by different instance types.

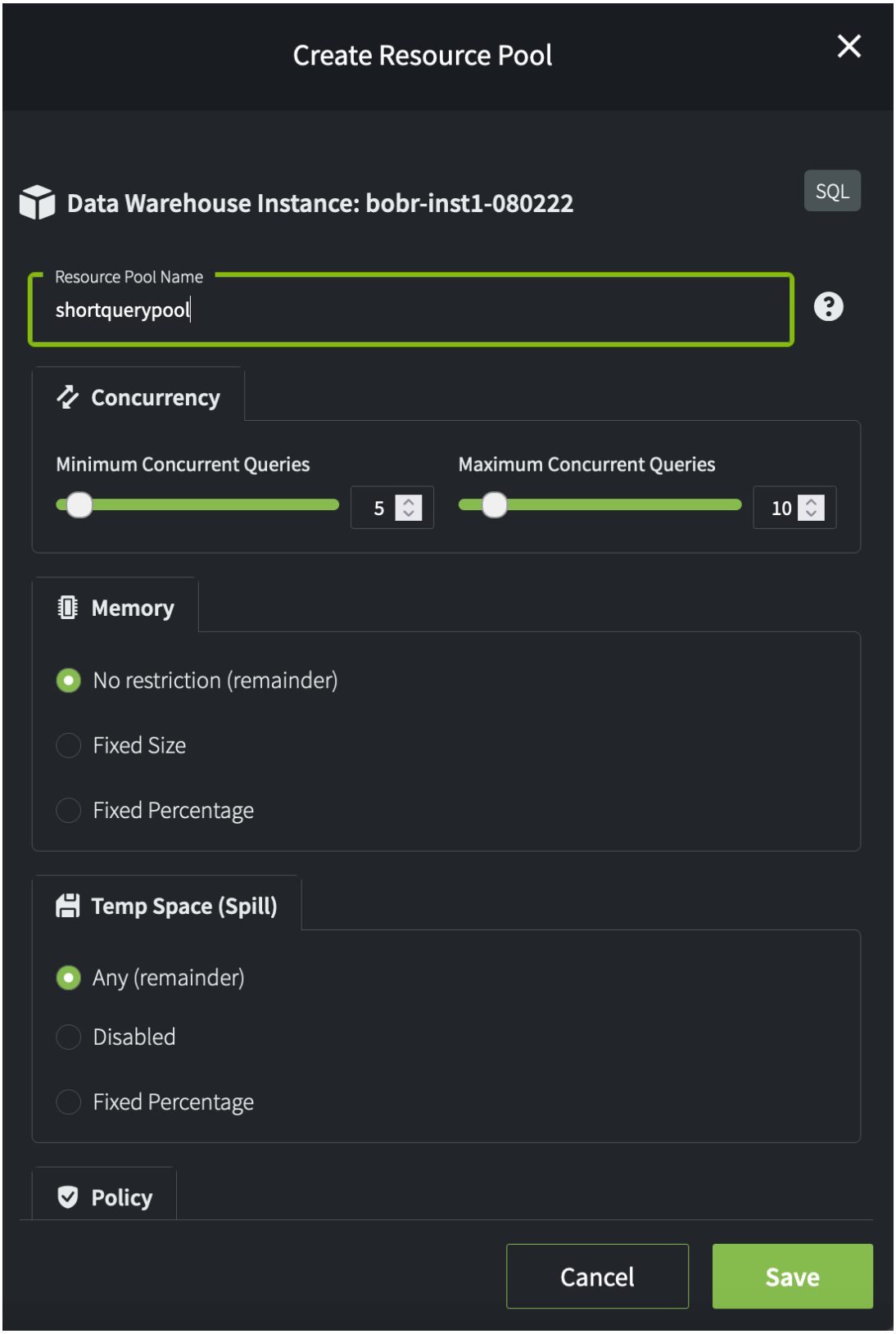
A single query occupies an *execution slot*. You set the minimum and maximum concurrency within a resource pool to some number of slots. You can set up "flex pools" that can expand or contract based on the values that you set. For example, a pool can have values of `12/3`, where `12` is the maximum concurrency and `3` is the minimum. For details, see Examples of Flex Pools.

Resource pools also define several other thresholds:

- Memory use and temporary space; see Allocating Query Memory.
- Queue size: how many queries can be waiting for execution in a given pool before action is taken
- Maximum wait limit: how long a query can wait in a queue before action is taken
- Maximum row limit: how many rows a query can process before action is taken
- Maximum execution limit: how many seconds a query may run before being canceled

To create a resource pool:

- In Yellowbrick Manager, go to **Workload Management**, select a profile, and click **+Resource Pool**. Then fill out the details in the **Create Resource Pool** screen. For example:


Create Resource Pool
Data Warehouse Instance: bobr-inst1-080222
SQL
Resource Pool Name
shortquerypool
Concurrency
Minimum Concurrent Queries
5
Maximum Concurrent Queries
10
Memory
No restriction (remainder)
Fixed Size
Fixed Percentage
Temp Space (Spill)
Any (remainder)
Disabled
Fixed Percentage
Policy
Cancel
Save

- Alternatively, use the SQL command CREATE WLM RESOURCE POOL.

To see a list of resource pools that are currently defined, go to **Workload Management** and select a profile. You can also query the sys.wlm_pending_pool view, or use the `ybsql \dwrp` command, as shown here:

```
premdb=> \dwrp
                                                    List of WLM resource pools
       Name        | Max Concurrency | Min Concurrency | Queue size | Wait limit | Row limit | Exec time limit |    Profile     |
-------------------+-----------------+-----------------+------------+------------+-----------+-----------------+----------------+
 admin             |               1 |                 |       2000 |            |           |                 | default        |
 instance_large_mix |              30 |              15 |       2000 |            |           |                 | instance_large |
 instance_small_mix |               3 |               2 |       2000 |            |           |                 | instance_small |
 large             |               4 |                 |       2000 |            |           |                 | default        |
 long              |               1 |                 |       2000 |            |           |                 | flex           |
 maintenance       |              10 |                 |        100 |            |           |                 | maintenance    |
 mix               |               8 |               4 |       2000 |            |           |                 | flex           |
 small             |               8 |                 |       2000 |            |           |                 | default        |
 system            |               3 |               2 |        100 |            |           |                 |                |
(9 rows)
```

See also Creating WLM Profiles.

Allocating Query Memory

The total memory available on each active compute node of the cluster is divided among active WLM resource pools and a flexible number of query slots within those pools. WLM rules influence how queries are assigned to resource pools and may prohibit or change query execution based on memory consumption.

Each new resource pool that you associate with a profile may cause an adjustment of the memory allocation among all of the pools associated with that profile. The same kind of adjustment may occur when you change an existing resource pool. Within a single resource pool, you can allocate (or restrict) memory in three different ways:

- Request a fixed size (specify an exact amount, in MB)
- Request a percentage
- Set it to Unrestricted, which means the pool will accrue remaining memory when the fixed and percentage-based pools have taken their quota

Memory specifications are subject to the following WLM policies, which are applied in order:

1. An initial minimum amount of memory is allocated to all pools.
2. Memory is allocated to pools requesting fixed amounts.
3. Remaining memory, if any, is allocated proportionally to pools requesting a percentage. (A percentage is calculated from the total memory available, not the memory minus fixed allocations.)
4. Remaining memory, if any, is allocated equally to pools that were defined as Unrestricted.
5. Remaining memory, if any, is allocated proportionally to pools requesting a percentage (repeat of step 3 where applicable).

Whether there is any remaining memory to use in steps 3, 4, and 5 depends on the pools that are currently defined and how they are defined (fixed, percentage-based, or unrestricted).

Memory allocation further depends on the minimum and maximum concurrency defined for the resource pool and the amount of memory on each compute node. For example, assume you have a small 2-node cluster with 58GB per node of available memory on each node. A resource pool with a maximum of 5 slots and a 25% memory allocation, assuming 25% is available, will accrue roughly 14.5GB per node and 2.9GB per slot.

Note: As a best practice, avoid allocating memory such that any query may be allocated less than 1GB. Memory requirements increase when spilling is enabled (see the discussion of spilling later in this section). A minimum of 512MB is enforced for resource pools with spilling disabled, and a minimum of 1GB for pools with spilling enabled.

When a query is planned, three useful pieces of memory information are calculated:

- The *minimum required* amount of memory needed to set up the query (based on the query plan and calculated before any rows are actually processed into or out of memory)
- The *estimated* total amount of memory to run the query (which is the minimum value plus some additional amount of memory, based on estimates of the data that will flow through the plan)
- A *confidence* value, expressed as `None` (no confidence), `Low`, `High`, or `Unknown`. These values describe the degree to which the estimated memory is thought to be reliable.

Given the availability of this information, follow these basic guidelines when you are writing WLM rules that assign queries to resource pools and slots:

- Never map a query to an execution slot whose size is less than a query's required memory
- When using memory estimates for a query to match it to an execution slot, make sure the memory estimate confidence value is high
- If the estimated memory for a query is low and the confidence value is high, choose a pool designated for queries that are expected to run fast and consume low memory.

For example, the following rule directs queries to a `smallquery` pool when the memory usage estimate is low (`<1024`) and the confidence in that estimate is `High` :

```
if (w.memoryEstimate < 1024 && w.memoryEstimateConfidence == 'High')
{ w.resourcePool = 'smallquery'; }
```

This kind of rule is reliable for very simple queries. Memory consumption estimates are less predictable for more complex queries.

Queries That Spill Data to Disk

Some long-running, complex queries may run out of memory and fail to run. To reduce the frequency of out-of-memory conditions, some query steps are capable of "spilling," which means that data is written to disk during processing, then read back into memory to complete query execution.

Spilling is supported for queries that contain only these steps or a combination of these steps:

- Joins
- Aggregates
- Sorts
- UNION ALL

Spilling is not supported for queries that contain:

- Cross-joins
- The following set operations: UNION, EXCEPT, EXCEPT ALL, INTERSECT, INTERSECT ALL
- Window functions
- A WHERE clause condition of the following form:

```
WHERE column [ NOT ] IN (subquery) OR column [ NOT ] IN (subquery)
```

Queries for which spilling is possible effectively have an amplified slot size: the amount of memory available to them appears larger than the memory that is allocated. The memory required by a query may not prevent it from running in a given pool that does not have that much memory available. In this way, spilling prevents or reduces the frequency of out-of-memory conditions.

Every time a query operation spills some data to disk, some metadata has to be saved in memory to keep track of the spilled data and its location. Assume you have a memory allocation of 3GB for a resource pool with 1 slot, and a required memory of 1GB for an incoming query. This means that 2GB of memory is left for use during execution, as rows are processed. In general, Yellowbrick queries can spill up to 0.5TB of data per 1GB of memory. In this case, if needed, the query could safely spill up to 1TB of data without running out of memory.

Spilling may also be prohibited by a simple lack of temporary space. To make sure that complex long-running queries have the *capacity* to spill, set the temporary disk usage appropriately for your resource pools. If queries spill but run out of spill space to use, you may see "spill limit exceeded" errors.

You can allocate temporary space to specific resource pools by choosing one of the following options:

- Any: by default, queries assigned to a pool use any spill space that is available.
- Disabled: Queries cannot use any spill space.
- Fixed percentage: A minimum reservation of 1 to 100% for the pool. For example, if a pool specifies 50%, that percentage of space will always be reserved for that pool, but queries going to that pool could use more. (In effect, this means that the total spill space for all other pools combined may not exceed 50%.)

Resource pools that do not specify a reserved spill percentage contend for *all* of the remaining spill space, not a proportional fraction. Remaining spill space is not divided evenly among the slots for those pools.

Note: The memory configuration is adjusted as needed to ensure a slot size of at least 1GB when spilling is enabled. Slots in a configuration with spilling disabled require a minimum slot size of 512MB.

Optimizing Queries to Reduce Spilling

Queries that spill may need to be optimized. Spilling introduces overhead that causes queries to run slower. Compared to when they run fully in memory, queries that spill may run 1.5x to 4x slower. Also, reducing spilling prolongs the lifetime of the flash storage media on the cluster.

The simplest way to reduce spilling is to allocate more memory to the resource pools where queries that spill are running. To identify queries that are spilling, go to **Query Activity** > **Query Details** in Yellowbrick Manager.

You can see where queries spill (and how much data they spill) because the plan nodes are marked with a warning. You can also use the `EXPLAIN ANALYZE` command or query the sys.log_query view, which has some spilling-related columns:

- `io_spill_write_bytes`: data written to temporary space because of spilling.
- `io_spill_read_bytes`: data read back from the spill area. This value could be less than what was originally written/spilled: for example, if there is a LIMIT clause on the query. (However, if it routinely occurs that `io_spill_read_bytes` is less than `io_spill_write_bytes`, you may need to rewrite the query or redesign a table in some way to avoid this "waste" of spill space.)
- `io_spill_space_bytes`: maximum temporary space used at any given time on any compute node.

Examples of Flex Pools

Platforms: All platforms

Parent topic: Creating WLM Resource Pools

Queries admitted to a flex pool have their memory allocated dynamically, based on minimum and maximum concurrency boundaries and the total memory available to the pool. The `mix` resource pool in the default `flex` profile, with concurrency set to `8/4`, will divide memory among a minimum of 4 concurrent queries when its workload is light, but the same resource pool will start up to 8 queries concurrently when the workload is heavy, adjusting memory allocation accordingly. By default, any query in this pool will be limited to a maximum of ¼ of the available memory, making room for three more queries to run with the same amount of memory. However, if 8 queries join this pool at once and no other queries are running in it, they will each accrue 1/8 of the available memory (and no more).

Memory that is allocated to queries when they start running does not change during execution; therefore, queries that join the pool while other queries are still running must divide up the remaining memory. If insufficient memory is available, new queries cannot start and must be queued (or possibly moved). (Insufficient memory in this context is the amount of memory available divided by some number between the minimum and maximum concurrency values.)

Queries are not necessarily restricted by the minimum concurrency setting. In this 8/4 example, four slots' worth of resources do not need to be kept available all the time. When queries complete and slots *become available* in the pool (busy slots cannot be hijacked when queries are mid-stream), a rule may accrue any specified percentage (or number of MB) of the entire pool's memory allocation. For example, a rule may request 100% of the memory in the `mix` pool, wait until no queries are running in the pool, take all of the memory, and block all other queries from starting in the pool. Alternatively, a query may request 50% of the pool's memory, wait until 4 slots are open, take half the memory, and allow up to 4 more queries to start in the pool.

The following rule specifically requests 50% of the memory when user `bobr` runs `ybload` operations in the `mix` pool:

```
if ((String(w.application).indexOf('ybload') >= 0) &&
   w.user === 'bobr' &&
   w.resourcePool === 'mix') {
    w.requestedMemoryPercent = 50;
}
```

See CREATE WLM RESOURCE POOL for examples of the syntax for flex pools.

Example with concurrency of 8/4, 80GB memory

For example, say the system has `80GB` available for queries in a given pool defined with concurrency of `8/4`:

- Query 1 comes in and takes 20GB of available memory (¼).
- While query 1 is running, queries 2 and 3 come in. Each one also takes 20GB of available memory (¼). 20GB of memory remains.
- While queries 1, 2, and 3 are running, queries 4, 5, and 6 come in. Only two of these can start, and they will take the remaining 10GB each (1/8). (In order to accommodate the maximum number of queries, the pool now uses 8 as the divisor instead of 4.)

Query 6 will wait in the queue until the minimum required memory per query (1/8, or 10GB) becomes available in the pool. Under a high concurrent workload, most queries in this flex pool would be given only 1/8 of the memory.

Example with concurrency of 8/1, 80GB memory

This example is similar to the first example, except that the entire resource pool is available to a single query when there is no queue. This kind of pool is more appropriate for a sequential reporting workload (but not so useful for a mixed workload). User-defined throttling would help allow a query mix to continue to run, except that any long-running query would starve new queries, whether throttling is used or not.

In this case, the system has `80GB` available for queries in a pool defined with concurrency of `8/1`:

- Query 1 comes in and takes all 80GB of available memory.
- While query 1 is running, queries 2 and 3 come in. They wait in the queue.
- While query 1 is running, and queries 2 and 3 are waiting, queries 4, 5, and 6 come in. All five queries wait in the queue.
- Query 1 completes with five queries queued. They all start running with 16GB of memory each.

Example with concurrency of 32/8, 80GB memory

In this high-concurrency example, the system has `80GB` available for queries in a pool defined with concurrency of `32/8`:

- Query 1 comes in and takes 10GB of available memory.
- While query 1 is running, queries 2 and 3 come in. They also take 10GB each. 50GB remains in the pool.
- While queries 1, 2, and 3 are running, queries 4, 5, and 6 come in. They also take 10GB each. 20GB remains in the pool.
- Queries 1, 2, and 3 complete. 50GB remains in the pool.
- Ten more queries come in. They all start with 5GB each.

This pool will always allow up to 8 concurrent queries at 10GB per query, but can subdivide resources to supply up to 32 concurrent queries. At the highest concurrency levels for this pool, 1/32 of the resources is given to each query (approximately 2.5GB).

Creating WLM Rules

Platforms: All platforms

Parent topic: Workload Management

In this section:

Given knowledge of the workloads in your data warehouse, you can write WLM classification rules that define specific scheduling behavior, resource allocation, and other actions that apply to queries at different points in their life cycle (from submission to completion). Rule conditions may look for matches to a particular table referenced in a query or a particular user who is running it. When a query qualifies for the conditions defined in a rule, a given action is taken, such as to place the query in a given resource pool, restrict memory use, or even cancel the query.

To create a rule, use one of the following methods:

- Go to **Workload Management** > **Global Rules** > **+Rule** in Yellowbrick Manager. You can use assisted mode or advanced mode to write the logical conditions for rules.

In assisted mode (during step 2 of 3), JavaScript is generated for the rule based on your choices of conditions and actions. Simple logical operators are provided, such as `equals`, `not equals`, `greater than`, `less than`, and so on. Where appropriate, drop-down lists provide specific choices, such as database names, table names, and names of resource pools. In other cases, you can use regular expressions to write conditions based on pattern matching.

In advanced mode, you can type in the JavaScript logic yourself, making use of the same building blocks that assisted mode provides.

- Use the SQL command CREATE WLM RULE. In this case, you need to know how to write the JavaScript logic for rule conditions and actions (as with advanced mode in Yellowbrick Manager).

Note: Choosing the *rule type* is a critical step. The type affects when the rule is evaluated during the query lifecyle and determines what planning information is available to the rule. In turn, the rule type determines the kinds of actions that can be taken. For details, see Rule Types and Rule Examples.

Yellowbrick Manager shows a list of rules that are currently in the system. You can also query the sys.wlm_pending_rule view or use the `ybsql \dwr` command.

Rule Order

All rules are evaluated for every query. The order in which they run is controlled by the *rule order*. Lower-numbered rules are run first. For example, a rule with a rule order of `10` runs before a rule with a rule order of `100`. Therefore, a rule with a higher rule order number may override settings that were set by a rule with a lower number.

This override behavior may be set on purpose. For example, consider the following rules:

- Rule 1, order 10:

```
w.priority = ‘normal’;
 w.requestedMemoryPercent = 10;
```

- Rule 2, order 100:

```
if (w.application === 'SAS') {
w.priority = 'high';
w.requestedMemoryPercent = 50;
}
```

Both rules set the `priority` and `requestedMemoryPercent` values, but the final values set for the query are governed by the last rule to run.

System Rules and Global Rules

Some WLM rules are defined for each system-defined profile, and these rules are prefixed with the profile name. For example:

- `flex_mapToPenalty`
- `default_mapToSmall`

You cannot remove or modify these rules.

Several global rules belong to all profiles, and these rules are prefixed with `global_`. You can remove or modify these rules. A set of global rules exists for each instance you create. Changing a rule for one instance has no effect on the corresponding rule in another instance.

For example, by default the `global_defaultRowLimit` rule prevents queries run by regular users from returning more than 5,000,000 rows. These queries are aborted and return the following error:

```
premdb=> select * from newmatchstats;
ERROR:   WLM row limit exceeded
```

You can increase this limit globally for a given instance by changing the definition of the rule action. For example:

After changing rules, you have to activate the changes before they will take effect, as discussed in Creating WLM Profiles.

Rule Examples

Platforms: All platforms

Parent topic: Creating WLM Rules

In this section:

This section contains examples of several WLM rules that are created with different *types*. Depending on their type and logic, rules are evaluated and applied at different points in the lifecycle of a query. Remember that rules are also evaluated with respect to each other. It is possible for one rule to cancel out another rule, so make sure you are aware which rules are actually being applied on your system. These examples should help you understand some of the typical behaviors with rule processing.

In some cases, the behavior shown in these examples could be emulated with security settings (ACLs) on different objects. Note that the WLM rule approach provides a conditional and temporary form of security, given that rules can be enabled or disabled, different profiles can be activated for different workloads, and so on.

While debugging rules, you may want to look at the information logged in the sys.query_rule_event view:

```
select * from sys.query_rule_event where rule_name='new_rule_name';
```

Assemble Rule Examples

Platforms: All platforms

Parent topic: Rule Examples

The following examples are evaluated and applied during the `assemble` phase of query execution.

Abort Queries via a SQL Condition

```
Rule Preview:
if ((String(w.SQLText).indexOf('match') >= 0) &&
   w.application === 'ybsql') {
    w.abort('Flex profile: ybsql users cannot query match tables!');
}
```

Any `ybsql` query that contains the string `match` (such as in a table name) will be aborted. For example:

```
premdb=> select count(*) from match;
ERROR:  WLM: Denied by administrator: Flex profile: ybsql users cannot query match tables!
CONTEXT:  select count(*) from match;
premdb=> select count(*) from matchstats;
ERROR:  WLM: Denied by administrator: Flex profile: ybsql users cannot query match tables!
CONTEXT:  select count(*) from matchstats;
premdb=> select count(*) from newmatchstats;
ERROR:  WLM: Denied by administrator: Flex profile: ybsql users cannot query match tables!
CONTEXT:  select count(*) from newmatchstats;
```

This rule is enabled for the `flex` profile only. When a different profile is activated, this rule will have no effect.

Adjust Memory for a User

This example makes sure that a large amount of memory is available to `CTAS` statements run by user `bobr`:

```
Rule Preview:
if (w.memoryEstimate < 20000 &&
   w.user === 'bobr' &&
   w.type === 'ctas') {
    w.requestedMemoryMB = 20000;
}
```

When `bobr` runs a `CTAS` query, requested memory is set accordingly.

The `sys.log_query` view records requested compared to estimated memory use:

```
yellowbrick=# select query_id, state, username, substr(query_text,1,33), memory_estimated_bytes, memory_granted_bytes
from sys.log_query
where query_id=3490373;
 query_id | state | username |             substr              | memory_estimated_bytes | memory_granted_bytes
```

```
----------+-------+----------+-----------------------------------+-----------------------+----------------------
  3490373 | done  | bobr     | create table union_matchstats as  |            3617587200 |          20971520000
(1 row)
```

Note: If you are more concerned about memory requirements during query compilation, you could create an equivalent `Compile` rule that sets the requested memory during that phase of execution.

Compile Rule Examples

The following examples show rules that are evaluated as queries enter the `compile` state.

Throttle Users at Compile Time

This compile rule is called `compile_throttle_users`, and it is designed to make sure that a set of users cannot use up compile-time resources at the expense of other queries and sessions. The condition in this example applies to users with `backup` in their names (such as `backup1` and `backup2`). As queries submitted by these users enter the `compile` state, concurrency is limited to 1, preventing them from running multiple queries concurrently.

```
Rule Preview:
if ((String(w.user).indexOf('backup') >= 0)) {
    wlm.throttle(1, w.user);
}
```

When this rule is applied, the administrator sees a message like this: `Throttle 1 accesses for barrier backup2`. In this case, the "barrier" to concurrency is the user `backup2`.

Move a Query to the "Long" Pool

The `flexMoveToLong` rule is a predefined `Compile` rule for the `flex` profile:

```
setTimeout(function () {
  log.info('Moving long-running query to resource pool "long": {}', w.execId);
  if (w.resourcePool !== 'long' && w.movable) {
   w.moveToResourcePool('long');
  }
}, 15*1000);
```

This rule sets a timeout of 15000 ms for the `run_ms` statistic, which is clocked when the query starts running. If `run_ms` exceeds 15000, the query is moved to the `long` resource pool (assuming that the query is not already running in the `long` pool and is movable; not all types of queries are movable). This rule is evaluated as a `compile` rule but affects runtime processing because of the timeout delay.

When this rule is applied, the timeout is set, then applied when the query runs longer than 15000 ms. The following system view query confirms that a specific query was moved to the `long` pool:

```
premdb=# select pool_id, run_ms from sys.log_query where query_id=3639753;
 pool_id |  run_ms
---------+-----------
 long    | 97083.613
(1 row)
```

Completion Rule Examples

Platforms: All platforms

Parent topic: Rule Examples

The following rules are evaluated and applied when a query goes into any completion state: `done`, `cancel`, or `error`.

Setting a Query Tag

The following completion rule, `completion_tag_short_query`, sets a query tag `Subsecond Query` when queries run by non-superuser roles complete in less than 1 second.

```
if (w.roles !== 'superuser' &&
   w.duration <1000) {
    w.tags = 'Subsecond Query';
}
```

Log a Warning Message for the ybdadmin User

This completion rule, `completion_warn_ybdadmin_user`, tracks all of the queries run by the `ybdadmin` account and logs a warning message.

```
Rule Preview:
if (w.user === 'ybdadmin) {
    log.warn('Running as ybdadmin!');
}
```

For every query run by the `ybdadmin` account, the administrator sees the warning:

```
Running as ybdadmin!
```

Examples of JavaScript for Rules

This section contains a few examples of the JavaScript that is used to represent rule logic when you create them in Yellowbrick Manager or with a SQL command. You can use Advanced mode in Yellowbrick Manager to write your own JavaScript for rules. See JavaScript Properties for WLM Rules.

Note: WLM rules use strict equality comparisons with `===` instead of `==`. For details, see Equality comparisons and sameness.

If the database is `premdb`, set the maximum execution time to `5` seconds:

```
if (w.database === 'premdb') {
    w.maximumExecTimeLimit = 5;
}
```

If the user who runs the query is `bobr`, log an INFO-level message:

```
if (w.user === 'bobr') {
    log.info('bobr ran this query');
}
```

If a `select` query is run from `ybsql` on the `premdb` database, set the maximum row limit to `10000`:

```
if (w.application === 'ybsql' &&
   w.database === 'premdb' &&
   w.command === 'select') {
    w.maximumRowLimit = 100000;
}
```

If the SQL text matches the string `sys.const`, set Short Query Bias (SQB) to `true`:

```
if (String(w.SQLText).match(/sys\.const/)) {
    w.allowSQB = true;
}
```

If the SQL text contains `extract`, cancel the query and log the message: `Contains extract function`.

```
if ((String(w.SQLText).indexOf('extract') >= 0)) {
    w.abort('Contains extract function');
}
```

Note that this rule may "misfire" if a table or a column name contains the string `extract`.

Using log.warn and log.error

If you write a rule that uses `log.warn` or `log.error`, a warning or error is logged in the `sys.log_query_alerts` table.

An example of an execution rule that logs warnings is the system rule named `default_boostForHighMemory`. This rule is defined as follows:

```
false && log.info('Query {} using maxMemory {}', w.execId, w.stats.maxMemory);

if (w.stats.maxMemory > (2*1024*1024*1024) && w.priority !== 'high') {
  w.priority = 'high';
  log.error('Boosted query {} to high priority, because it is using too much memory ({} GB)',
   w.execId, (w.stats.maxMemory / (1024*1024*1024)).toFixed(1));
}
```

This rule examines the current statistics for the query, and if it has used more than 2GB (`2*1024*1024*1024`), and its priority is not already `high`, its priority is set to `high` and an error is logged. (The decision to log an error is arbitrary; a warning might be more appropriate here.)

Restart Rule Examples

Platforms: All platforms

Parent topic: Rule Examples

The following examples describe rules that are evaluated when queries are restarted because of an error.

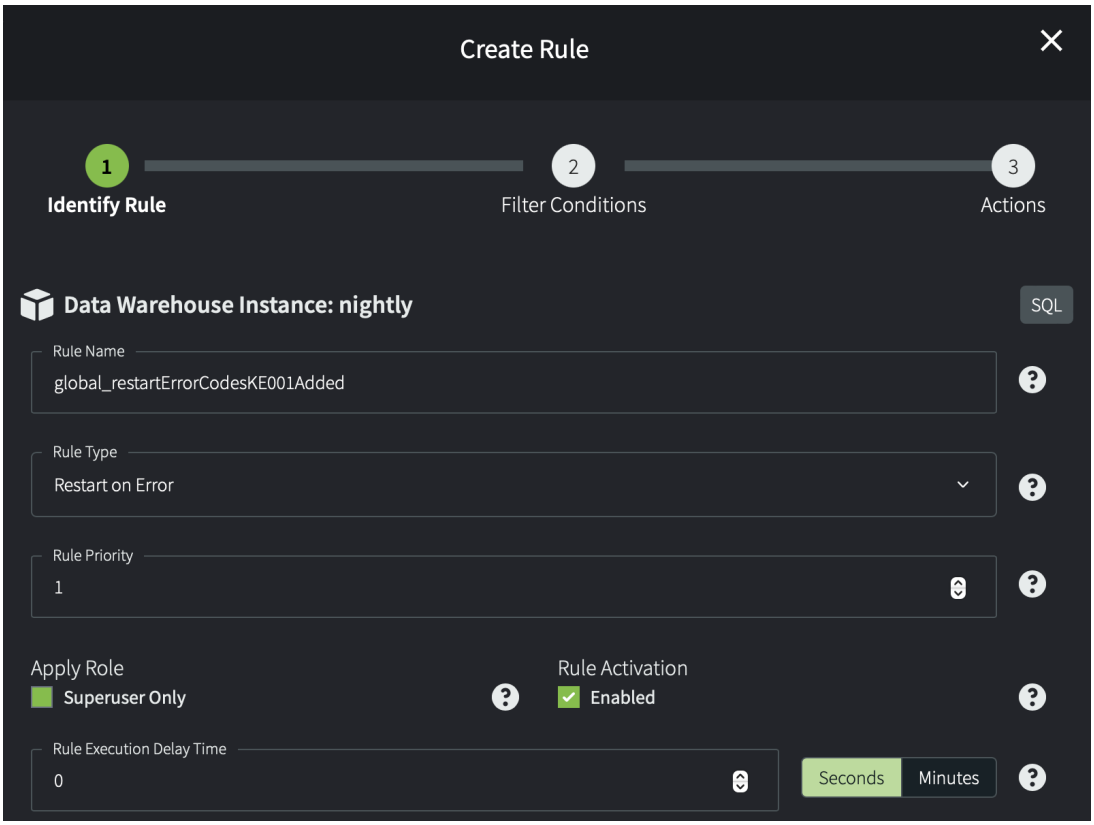
Modified Restart on Error System Rule

The `global_restartErrorCodes` and `global_restartErrorCodesSuperuser` system rules trigger a one-time restart attempt for queries that return a small subset of recoverable error codes. If you want to extend these rules to apply to more error codes, Yellowbrick recommends that you create new rules rather than modify existing rules. (If system rule versions change during an upgrade, your modifications to those existing rules will be lost.)

Start by looking at the default definition of `global_restartErrorCodes`:

```
yellowbrick=# select * from sys.wlm_active_rule where rule_name ='global_restartErrorCodes';
-[ RECORD 1 ]+-----------------------------------------------------------------------------------------------------------
profile_name | (global)
rule_name    | global_restartErrorCodes
rule_type    | restart_for_error
order        | 1
enabled      | t
superuser    | f
expression   | // Recoverable error codes:                                                                              +
             | //  - KE041 # Library file write                                                                       +
             | //  - YB044 # RecoverableGeneric                                                                      +
             | //  - WM001 # ERRCODE_WORKER_OFFLINE                                                                  +
             | //  - KE039 # RPCCHANNELBROKEN                                                                        +
             | //  - KE038 # RPCCHANNELCLOSED                                                                        +
             | //  - P0004 # Assert failure                                                                          +
             | //  - WM002 # System not ready                                                                        +
             | //  - WM003 # Query restarting                                                                        +
             | if (w.errorRecoverable === undefined || w.errorRecoverable == null || w.errorRecoverable) {         +
             |   w.errorRecoverable = String(w.errorCode).match(/KE041|YB044|WM001|KE039|KE038|P0004|WM002|WM003/);+
             | }                                                                                                    +
             |
```

You can use Yellowbrick Manager or a SQL command to create a new rule that copies most of this rule definition. For example, in Yellowbrick Manager, go to **Workload Management** > **Global Rules** > **+Rule**. Use the same main settings as the existing rule but a different rule name:



Use the same rule definition, but add another error code from the recoverable list (`KE001` in this example):

```
if (w.errorRecoverable === undefined || w.errorRecoverable == null || w.errorRecoverable) {
  w.errorRecoverable = String(w.errorCode).match(/KE041|YB044|WM001|KE039|KE038|P0004|WM002|WM003|KE001/);
}
```

Now disable the existing `global_restartErrorCodes` rule and activate changes to make your new rule, `global_restartErrorCodesKE001Added` , take effect instead.

Alternatively you can enable both rules, but in that case your new rule would need a higher rule order, such as `10` , so that the existing rule is applied first. Also you would need to remove the following code from the rule definition:

```
if (w.errorRecoverable === undefined || w.errorRecoverable == null || w.errorRecoverable) {   }
```

Note that there are two instances of the `ErrorCodes` rule: one for superusers and one for non-superusers. You may need to modify both of these to suit your requirements.

Restart and Try to Expand Resources

The `flex` profile has a predefined rule, `flex_expandResourcesErrorHandler` , which attempts to increase the resources available to a query that errors out with one of several specified error codes. The attempt to expand resources happens when the query restarts and only applies to the `flex` profile.

The rule is defined as follows:

```
log.info(w + ' is restarting for error ' + w.errorCode);
if (String(w.errorCode).match(/53200|KE002|YB004|KE032|KE029|YB006|EEOOM/)) {

  // See if we can't expand resources; if we can, lets try the query with more resources.
  if (!wlm.assignMaximumResources(w)) {
    w.errorRecoverable = false;
    log.info(w + ' cannot expand resources; marked as not recoverable');
  } else {
    w.errorRecoverable = true;
    log.info(w + ' expanded resources for restart (memory ' + w.requestedMemoryMB + ', spill ' + w.requestedSpillMB + ')');
  }
}
```

The `wlm.assignMaximumResources(w)` property returns true if expanded resources (memory and spill space) are available from the `flex` pool. Additional resources may or may not be available, depending on concurrent query activity in that pool. This rule also logs appropriate `INFO` messages, either marking the query as not recoverable or listing the resources available on restart.

See also Recoverable Error Codes.

Runtime Rule Examples

Platforms: All platforms

Parent topic: Rule Examples

Runtime rules take actions during actual query execution on the workers. These rules are evaluated only for queries with a duration of at least 5 seconds and are evaluated repetitively at 5-second intervals after queries enter the `run` state.

Set Priority to Critical for Queries Against Specific Tables

The following runtime rule makes sure that CPU resources are available to queries against tables that contain `matchstats` in their names. The rule action sets the priority of these queries to `critical`:

```
Rule Preview:
if (w.referencedTables.contains('matchstats') &&
   w.priority === 'low' ||
   w.priority === 'normal' ||
   w.priority === 'high') {
    w.priority = 'critical';
}
```

Note that if only one query is running on the system, it could be allocated 100% of the available CPU, regardless of its priority. Setting the priority only has an effect when concurrent queries are running.

Log an INFO Message for Large DELETE Queries

The following runtime rule, `runtime_log_info_msg`, logs an `INFO` message when more than 1 million rows are deleted from any table:

```
Rule Preview:
if (w.stats.rowsDeleted > 1000000) {
    log.info('More than 1 million rows deleted');
}
```

The `ybsql` user sees, for example:

```
premdb=> delete from newmatchstats where seasonid=7;
DELETE 849801600
```

The `DELETE` succeeded, but an `INFO` message is logged as expected:

```
More than 1 million rows deleted
```

Note that runtime rules are processed at 5-second intervals.

Do Not Store Stats for a Database

The following runtime rule, `runtime_no_stats_for_premdb`, turns off statistics logging for queries against the `premdb` database. This means that `sys.log_query` will not have a record of these queries.

```
Rule Preview:
if (w.database === 'premdb') {
    w.statsStored = false;
    log.info('No stats for a premdb query!');
}
```

Submit Rule Examples

You can use a submit rule to take actions at the very beginning of the query lifecycle. You can throttle concurrency and cancel queries early on.

Abort Queries Based on Type and User

The following example, `submit_abort_truncate`, is a rule that immediately aborts `TRUNCATE` commands when they are attempted by users in a backup group (users named `backup1`, `backup2`, and so on).

The rule logic is defined as follows:

```
Rule Preview:
if (w.type === 'truncate table' &&
   (String(w.user).indexOf('backup') >= 0)) {
    w.abort('Backup users cannot truncate tables!');
}
```

In `ybsql`, user `backup1` sees the following error when attempting a `TRUNCATE`:

```
yellowbrick=# \c premdb backup1
Password for user backup1:
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)
You are now connected to database "premdb" as user "backup1".
premdb=> truncate matchstats;
ERROR:  canceling statement due to user request
```

Submit Rule That Throttles Concurrency

The following example, `submit_throttle_dml`, is a rule that limits concurrency to 1 for each user in a backup group (users named `backup1`, `backup2`, and so on) when they attempt to run `INSERT`, `UPDATE`, or `DELETE` statements.

The rule logic is defined as follows:

```
Rule Preview:
if ((String(w.user).indexOf('backup') >= 0) &&
   w.type === 'insert' ||
   w.type === 'update' ||
   w.type === 'delete') {
    wlm.throttle(1);
}
```

Note the use of `AND` (`&&`) and `OR` (`||`) conditions in this rule. If user `backup2` tries to run concurrent `INSERT`, `UPDATE`, or `DELETE` statements, the system will throttle those operations and run them serially.

JavaScript Properties for WLM Rules

Platforms: All platforms

Parent topic: Creating WLM Rules

The following JavaScript properties are supported for manually scripting conditions and actions in WLM rules. See also Rule Conditions and Rule Actions; where applicable, each property is linked to a description of the equivalent UI control in Yellowbrick Manager.

The read-only column indicates whether the JavaScript property is settable. `Yes` means that it is not settable; it can only be read from (that is, used in a filter to make decisions).

Name	Description	Data Type	Read-only
`w.abort` or `w.abortReason`	Abort Query action (cancel query and return a reason message)	String	No
w.allowSQB	Set Short Query Bias action (or a condition)	Boolean	No
w.application	Application condition	String	No
`w.authenticatedUsername`	Condition on the user account that originates the session. If `SET ROLE` is used, this value does not change.	String	Yes
w.command	Command condition	String	Yes
w.cost	Cost condition	Number	Yes
w.currentState	Current State condition. The state of a query can be filtered in a rule that is defined for an earlier state (for example, a `submit` rule that sets a timeout and executes when the query is in `compile` or `restart` state).	String	Yes
w.database	Database condition	String	Yes
w.duration or `w.elapsedTime`	Total Duration condition. (`w.duration` and `w.elapsedTime` are synonyms.)	Number	Yes
w.errorCode	Error Code condition	String	Yes
w.errorMessage	Error Message condition	String	Yes
`w.errorRecoverable`	Recoverable Error condition; whether the query can be restarted on error (defined as part of the default error recovery policy).	Boolean	No
`w.execId`	Execution ID condition (where execution ID is the same value as a `query_id` in `sys.query`)	Number	Yes
w.executionDuration	Execution Duration condition	Number	Yes

Name	Description	Data Type	Read-only
`w.explainAnalyze`	`EXPLAIN ANALYZE` condition (whether the query is an `EXPLAIN ANALYZE` command)	Boolean	Yes
`log.debug`, `log.error`, `log.info`, `log.warn`	Action to log messages at different levels	String	Yes
w.lookup	Set Lookup Query action; also a condition	Boolean	No
w.maximumExecTimeLimit	Set Maximum Execution Time Limit action	Number	No
w.maximumRowLimit	Set Maximum Row Limit action	Number	No
w.maximumWaitLimit	Set Maximum Wait Limit action	Number	No
w.memoryEstimate	Memory Estimate condition (value in MB, but equivalent to `memory_estimated_bytes` in `sys.query`)	Number	No
w.memoryEstimateConfidence	Memory Estimate Confidence condition	String	Yes
`w.memoryRequiredMB`	Memory required condition (in MB but maps to `memory_required_bytes` in `sys.query`)	Number	Yes
w.movable	Movable query condition	Boolean	Yes
w.moveToResourcePool(w.resourcePool, w.allowMoveToSamePool)	Move Query to New Resource Pool action Moves a query to a different resource pool, or to the same pool if the second argument is `true` (it is `false` by default).	String	N/A
w.namespace	Schema condition	String	Yes
`w.numRestartError`	Number of restarts on error (`num_error` in `sys.query`)	Number	Yes
`w.numRestartUser`	Number of restarts by user (`num_restart` in `sys.query`)	Number	Yes
w.priority	Priority condition or action	String	No
w.referencedTables	Referenced Tables condition	String (list)	Yes
w.requestedMemoryMB	Set Requested Memory action (in MB)	Number	No
w.requestedMemoryPercent	Set Requested Memory action (as a percentage)	Number	No
w.requestedSpillMB	Set Requested Temp Space action (in MB)	Number	No
w.requestedSpillPercent	Set Requested Temp Space action (as a percentage)	Number	No
w.resourcePool	Resource Pool condition or action. See also Setting and Matching Resource Pools in WLM Rules.	String	No

Name	Description	Data Type	Read-only
w.restartable	Restartable query condition	Boolean	Yes
w.restartInResourcePool(w.resourcePool, w.allowRestartInSamePool)	Restart Query in Resource Pool action Restarts a query in a different resource pool, or in the same pool if the second argument is `true` (it is `false` by default).	String, Boolean	N/A
w.roles	User Roles condition	String	Yes
`w.sessionId`	Session ID condition	Number	
`w.setTimeout(function () ..., timeoutMs)`	Timeout condition (specifies a `timeoutMs` delay on the body of the rule)	String, Number	
w.singleWorker	Single compute node query condition	Boolean	Yes
w.SQLText	SQL condition	String	Yes
`w.stats.AvgCpu`	Average CPU across cluster	Number	Yes
w.stats.bytesNetwork	Bytes Network condition	Number	Yes
w.stats.bytesRead	Bytes Read condition	Number	Yes
`w.stats.BytesReadClient`	Bytes Read by Client condition	Number	Yes
`w.stats.BytesWriteClient`	Bytes Sent by Client condition	Number	Yes
w.stats.bytesReadSpill	Temporary Bytes Read condition	Number	Yes
w.stats.bytesWrite	Bytes Written condition	Number	Yes
w.stats.bytesWriteSpill	Temporary Bytes Written condition	Number	Yes
`w.stats.cpu`	CPU percent usage on longest compute node (`cpu_percent` in `sys.query`)	Number	Yes
`w.stats.longestWorker`	Longest compute node condition	String (UUID)	Yes
`w.stats.maxCpu`	CPU percent usage of max compute node (`cpu_percent_max` in `sys.query`)		Yes
`w.stats.memory`	Memory condition (on longest compute node, `memory_bytes` in `sys.query`)	Number	Yes
w.stats.maxMemory	Max Memory condition	Number	Yes
w.stats.rowsDeleted	Rows Deleted condition	Number	Yes
w.stats.rowsInserted	Rows Inserted condition	Number	Yes
w.stats.rowsReturned	Rows Returned condition	Number	Yes
w.statsStored	Store Execution Statistics action and condition	Boolean	Yes

Name	Description	Data Type	Read-only
`w.superUser`	Superuser condition (`superuser` column in `sys.user` and `sys.role`)	Boolean	Yes
w.tags	Set Query Tags action or condition	String	No
`w.transactionId`	Transaction ID condition (`transaction_id` in `sys.query`)	Number	Yes
w.type	Type condition (type of query)	String	Yes
w.user	User Name condition	String	Yes
`w.userId`	User ID condition	String	Yes
`w.workstation`	Workstation condition, where this value corresponds to `client_ip_address` in the `sys.log_session` and `sys.log_session` views.	String	Yes
`wlm.assignMaximumResources`	Action to assign the maximum available resources in a pool. See Default Error Recovery.	Boolean	N/A
`wlm.assignMostConcurrentPool` `wlm.assignMostConcurrentPoolWithTempSpace` `wlm.assignMostConcurrentPoolWithoutTempSpace` `wlm.assignSmallestPoolForMemoryassignLargestMemoryPool` `wlm.assignLargestMemoryPoolWithTempSpace` `wlm.assignLargestMemoryPoolWithoutTempSpace` `wlm.assignLargestTempSpacePool wlm.assignLeastBusyPool`	Actions to assign resource pools. These actions take no parameters.	N/A	N/A
wlm.throttle(max, barrierName)	Limit Concurrent Queries action (maximum number, name of "barrier," such as a user, role, or application)	Number, String	No

Recoverable Error Codes

Platforms: All platforms

Parent topic: Creating WLM Rules

The following error codes define types of Yellowbrick error conditions that are recoverable. Queries that fail with these errors can be restarted by the system, given the appropriate rule configuration. See Restarting Queries.

```
08000 ERRCODE_CONNECTION_EXCEPTION       : connection exception
08003 ERRCODE_CONNECTION_DOES_NOT_EXIST  : connection does not exist
08006 ERRCODE_CONNECTION_FAILURE         : connection failure
40P01 ERRCODE_T_R_DEADLOCK_DETECTED      : deadlock detected
53000 ERRCODE_INSUFFICIENT_RESOURCES     : insufficient resources
53200 ERRCODE_OUT_OF_MEMORY              : out of memory
EEOOM ERRCODE_QUERY_OOM                  : Query out of memory
KE001 MISSINGFILE                        : File not found
KE002 OUTOFMEMORY                        : Out of memory
KE004 QUEUEFULL                          : Queue full
KE007 MISSINGID                          : Missing id
KE017 FILESYSTEMMISSINGFILE              : Missing file
KE019 FILESYSTEMOUTOFBOUNDS              : File system out of bounds
KE022 TCPSOCKETERROR                     : TCP socket error
KE023 RPCPROTOCOLERROR                   : RPC protocol error
KE029 SPILLLIMITSEXCEEDED                : WLM assigned limits exceeded during spill
KE031 FILESYSTEMSPILLFULL                : File system space reserved for spilling full
KE032 SPILLLIMITEXCEEDED                 : Attempt to use spill space above limit
KE036 TCPSOCKETTIMEOUT                   : TCP socket timeout
KE037 RPCEXCEPTION                       : RPC processing exception
KE038 RPCCHANNELCLOSED                   : RPC channel closed
KE039 RPCCHANNELBROKEN                   : RPC channel broken
KE041 Library file write
KU003 IOVALIDATION                       : Malformed IoRequest
KU004 INVALIDIOREQUEST                   : Malformed IoRequest
KU005 INVALIDNVMEIOREQUEST               : Malformed NVMe IORequest
KU006 FBDDRIVERFAILURE                   : FakeBlockDevice driver failure
KU007 NVMEDRIVERFAILURE                  : NMVe driver failure
KU008 UNEXPECTEDTCPSOCKETERROR           : Unexpected TCP socket error
KU009 SCHEDULEFAILURE                    : Failed to schedule task
KU010 IBDRIVERFAILURE                    : IB driver failure
KU011 FILESYSTEMMISSINGPREFIX            : Could not find file with prefix for Parity
KU012 FILESYSTEMNONUNIQUEPREFIX          : Too many files with prefix for Parity
KU016 FPGADRIVERFAILURE                  : FPGA driver failure
KU017 BOUNCEBUFFERDRIVERFAILURE          : Bounce buffer driver failure
KUSEG SEGV                               : Segfault
P0004 ERRCODE_ASSERT_FAILURE             : assert failure
WL017 ERRCODE_WORKLOAD_CANCEL_CLIENT_WAIT: query cancelling and so client wait should not proceed
WM001 ERRCODE_WORKER_OFFLINE             : worker offline
WM002 ERRCODE_SYSTEM_NOT_READY           : system not ready
WM003 ERRCODE_CANCEL_FOR_RESTART         : Query restarting
YB004 OUTOFHASHMEMORY                    : Hash table too big
YB005 SPILLINGSORT                       : Attempting to spill a sort to disc
YB006 TEMPTABLEOUTOFMEMORY               : Temp table exceeded max size
YB007 MISSINGTEMPTABLE                   : Temp table cannot be found or is not readable
YB008 NOSUCHEXECUTIONID                  : ExecutionID not found
YB009 NOSUCHNODEID                       : NodeID not found
YB010 INVALIDREGISTRATION                : Invalid worker registration.
YB044 RecoverableGeneric
YU002 NOSUCHWORKERID                     : invalid worker id
```

Rule Actions

Platforms: All platforms

Parent topic: Creating WLM Rules

The following table lists the specific rule actions that you can trigger based on one or more conditions. Each action applies to one or more rule types.

Action	Description	Rule Types	Accepted Values
Set Resource Pool	Assign the query to a specific WLM resource pool. See also the other "Assign" rules in this table, which provide some flexibility in choosing resource pools.	Assemble, Compile	Select from the drop-down list.
Set Priority	Reduce or increase the CPU priority of the query. The priority of queries is set exclusively by rules. Priority starts at `Normal`, and rules may set priority to a lower (`Low`) or a higher (`High` or `Critical`) value. See Priority Levels.	Submit, Assemble, Compile, Runtime	Low, Normal, High, Critical
Set Cost	Override the cost set by the planner. This action can be used to influence the behavior of subsequent rules that are applied to the same query.	Assemble, Compile, Runtime	Any number
Set Lookup Query	Pin the query in memory because it is likely to be run frequently.	Assemble, Compile	`True` or `False`
Set Short Query Bias	A “short query” is expected to run very quickly and use resources for a very short time; therefore, it is given a temporary boost in priority to get it finished under concurrent loads.	Assemble, Compile	`True` or `False`
Set Query Tags	Set query tags that will be logged for this query. See Using Query Tags.	All types	Any string
Set Maximum Execution Time Limit	Set a limit on the maximum execution time, in seconds. Cancel the query if the threshold is reached.	Assemble, Compile	Any number
Set Maximum Row Limit	Set a row limit. A query is aborted with the following error if it attempts to return more than the specified number of rows: `WLM row limit exceeded` You can also set a maximum row limit for individual resource pools when you create them. Note that by default the system rule `global_defaultRowLimit` imposes a limit of 5,000,000 rows on non-superuser queries. Superuser queries are not subject to this limit. If necessary, you can modify (increase or decrease) this global row limit.	Assemble, Compile	Any number
Set Maximum Wait Limit	Set a limit, in seconds, for how long a query may wait in a WLM queue.	Assemble, Compile	Any number
Set Requested Memory (in megabytes) or Set Requested Memory Percent	Set the requested amount of memory, either in megabytes, or as a percentage.	Assemble, Compile	Any number

Action	Description	Rule Types	Accepted Values
Set Requested Temporary Space (in megabytes) or Set Requested Temporary Space Percent	The maximum spill size per slot is fixed by the WLM configuration and calculated up-front when the resource pool is configured. This size may not be exceeded by a query, and all queries get this value for their slot when they start. A WLM rule, however, can restrict spill size. To disable spilling, or allow queries to go out of memory earlier, you can set the value to a lower number. Setting a higher value results in a warning being logged.	Assemble, Compile	Any number
Log at `INFO`, `DEBUG`, `WARN`, `ERROR`	Log a user-defined message at the specified logging level. Logging messages at `WARN` and `ERROR` levels sends a "WLM Rule Alert" to any configured alert endpoints.	All types	Any string
Abort Query	Cancel the query and return a message to the client that states the reason.	Submit, Assemble, Compile, Runtime	Any string
Move Query to New Resource Pool	Move a running query to a different resource pool. See Moving Queries.	Compile, Runtime	Default pool for profile or profile name and pool name (select from drop-down list)
Restart Query in New Pool	Restart a query in a different resource pool. See Restarting Queries	Compile, Runtime	Select from drop-down list
Store Execution Statistics	Turn on or off statistics logging to the sys.log_query view. For example, you may decide that an application that performs well does not need any statistics recorded during execution of any of its queries.	All types	`True` or `False`
Assign Most Concurrent Resource Pool [With Temp Space, Without Temp Space]	Assign the resource pool that has the most concurrency (execution slots). There are three variations of this rule so you can take temporary space for spilling into account.	Assemble, Compile	No value required
Assign Smallest Pool for Query Estimated Memory	Assign the smallest pool that fits the specified minimum amount of memory.	Assemble, Compile	Enter the minimum memory value (in MB)
Assign Largest Memory Resource Pool [With Temp Space, Without Temp Space]	Assign the resource pool that has the largest memory allocation. There are three variations of this rule so you can take temporary space for spilling into account.	Assemble, Compile	No value required
Assign Largest Temporary Space Resource Pool	Assign the resource pool that has the largest amount of temporary space for spilling.	Assemble, Compile	No value required
Assign Least Busy Resource Pool	Assign the resource pool that is the least busy when the query comes in.	Assemble, Compile	No value required

Action	Description	Rule Types	Accepted Values
Limit Concurrent Queries	Limit the number of concurrent queries that can run in a resource pool. Optionally, specify per-application, role, or user limits.	Submit, Assemble, Compile	Number of queries and an optional throttle name. For example: `wlm.throttle(2, w.application)`

Priority Levels

Priority settings affect the behavior of queries both before they are executed and during their execution. When the system is busy and queries are queued up (waiting), queries are sent to the compute nodes for execution based on their priority (`Low` , `Normal` , `High` , and `Critical`). Resource pools maintain four different queues, one for each level. The system serves higher-priority queries before lower-priority queries, increasing the allocation of CPU time proportionally at each level by 2-3x. For example, if you have two concurrent queries, one set to `Low` and one set to `Normal` , the `Normal` query takes approximately 75% of the CPU time, and the `Low` query approximately 25% (the remainder).

During execution, priority settings are only meaningful if queries are running concurrently and incoming concurrent queries are assigned different priorities. Two queries operating at the same priority are given equal share of the system. If only one query is running, it gets 100% of the available CPU, regardless of its priority.

Rule Conditions

Platforms: All platforms

Parent topic: Creating WLM Rules

Rules consist of one or more conditions. When satisfied, these conditions trigger rule actions. You can define different kinds of conditions in Yellowbrick Manager by selecting directly from drop-down lists or by using filter operators. The following table describes these operators.

Filter Operator	Description	Examples
Is True, Not True	Boolean condition	Query is/is not a single compute node query
Equals, Not Equals	Simple equality condition	Schema does not equal `public`; Application equals `ybsql`
Greater Than, Less Than	> or <	Cost >10000
Greater Than or Equal To, Less Than or Equal To	>= or <=	Memory Usage Estimate <=100
Contains String, Not Contains String	Simple substring operations	SQL contains `union all`; User Name does not contain `bob`
Matches, Not Matches	Regular expression pattern matching. Escaping is applied only when items are selected from a drop-down list. Regular expression validation is applied when entries are typed; warnings are displayed.	SQL matches `sys\.const`; SQL matches `sum\(substr`
Matches Ignore Case, Not Matches Ignore Case	Pattern matching as above, but with respect to case	

The following table lists the specific rule conditions that you can set up, shows which rule types support each condition, and lists the accepted values for each condition.

Filter Condition	Description	Rule Types	Accepted Values
Application	Application name, as sent by the application itself, or as defined for a session with a SQL SET command: `set application_name <name>;`	All types	Any string that does or does not match an application name, such as `ybsql` or `ym`. Note that `ybload` application names have the version appended, so you cannot filter on the exact string `'ybload'`.
Average CPU Usage	You can use **Query Details** > **Statistics** > **RESOURCES** (or **UTILIZATION**) in Yellowbrick Manager to find this value for past executions of a query. Alternatively, you can query the `sys.log_query` view.	Runtime, Completion	Any percentage value

Filter Condition	Description	Rule Types	Accepted Values
Bytes Network	Bytes moved internally over the network on the cluster (manager node to compute nodes and vice versa). This number does not include data being returned to the client. Use **Query Details** > **Statistics** > **RESOURCES** in Yellowbrick Manager to find this value for past executions of a query.	Runtime, Completion	Positive whole numbers
Bytes Read	Use **Query Details** > **Statistics** > **RESOURCES** in Yellowbrick Manager to find this value for past executions of a query. Alternatively, select `io_read_bytes` from `sys.log_query`.	Runtime, Completion	Positive whole numbers
Bytes Written	Use **Query Details** > **Statistics** > **RESOURCES** in Yellowbrick Manager to find this value for past executions of a query. Alternatively, select `io_write_bytes` from `sys.log_query`.	Runtime, Completion	Positive whole numbers
Command	Filter on a SQL command: `select`, `update`, `insert`, `delete`, or `utility`.	Assemble, Compile, Restart, Runtime, Completion	Select from the drop-down list. Filter on `utility` for CREATE TABLE commands. **Note:** You can define rules for `INSERT INTO...SELECT` statements but not for `INSERT INTO...VALUES` statements. `INSERT INTO...VALUES` operations are not executed by the backend database.
Cost	A relative value that is set by the planner.	Compile, Restart, Runtime, Completion	Any number
Database	Database name	All types	Select from the drop-down list.
Error Code	Yellowbrick error code; see Error Codes and Recoverable Error Codes.	Restart, Runtime, Completion	A string that matches or does not match a specific error code.
Error Message	Error message text.	Restart, Runtime, Completion	Any string
Execution Duration (in ms)	Number of milliseconds the query spent during execution on the compute nodes.	Runtime, Completion	Any number
Hostname/IP Address	Name or IP address of client machine	All types	Any string
Max Memory (in bytes)	Maximum runtime memory. Use **Query Details** > **Statistics** > **RESOURCES** in Yellowbrick Manager to find this value for past executions of a query.	Runtime, Completion	Any number
Memory Estimate (in MB)	Estimated runtime memory. Use **Query Details** > **Statistics** > **RESOURCES** in Yellowbrick Manager to find this value for past executions of a query. Alternatively, select `memory_estimated_bytes` from `sys.log_query`.	Compile, Restart, Runtime, Completion	Any number

Filter Condition	Description	Rule Types	Accepted Values
Memory Estimate Confidence	Estimated degree of confidence in the memory estimate value.	Compile, Restart, Runtime, Completion	Unknown, None, Low, High
Priority	The CPU priority of queries starts at `Normal` and is only settable by rules. Priority is not an attribute of a profile or pool. Based on a condition, you can trigger an action to set the priority to a lower (`Low`) or a higher (`High` or `Critical`) priority. Priority settings are only meaningful when multiple queries are running concurrently. If only one query is running, it is allocated 100% of the available CPU, regardless of its priority. When queries are running concurrently, the percentage of CPU time increases 2-3x by priority level. For example, if you have concurrent queries set to `Low` and `Normal` , the `Normal` query takes approximately 75% of the CPU time, and the `Low` query approximately 25% (the remainder).	All types	Low, Normal, High, Critical
Query State	Current state of the query. See How Queries Are Executed for details about query states.	All types	Any string
Query Tags	Use query tags to classify queries, then take actions. (You can also set a tag as an action). See Using Query Tags.	All types	Any string
Query is Movable	The query may be moved to another resource pool (and continue its execution there or be restarted there). See Moving Queries.	All types except Completion	Select a pool from the drop-down list.
Query is Restartable	The query may be restarted in another resource pool. See Restarting Queries	All types except Completion	`Is True` , `Not True`
Referenced Tables	Name of a table that is referenced somewhere in a query.	Compile, Runtime, Completion	`Contains` or `Not Contains` `<table_name>` (select from list)
Resource Pool	Name of a WLM resource pool. See Setting and Matching Resource Pools in WLM Rules.	All types	Select a pool from the drop-down list.
Rows Deleted	Number of rows deleted by a DELETE operation.	Runtime, Completion	Any number
Rows Inserted	Number of rows inserted by an INSERT operation.	Runtime, Completion	Any number
Rows Returned	Number of rows returned to the client application.	Runtime, Completion	Any number
SQL	Any piece of SQL text that may be found in a query. You can also use this condition to check for a comment placed in a query, rather than filter on the SQL itself.	All types	Any string
Schema	Name of a schema.	Assemble, Compile, Runtime, Completion	Select from the drop-down list. The `search_path` for the current session determines whether the condition applies. If a named schema is in the `search_path` , that is a match.

Filter Condition	Description	Rule Types	Accepted Values
Single Worker	Queries that only involve replicated tables are marked as “single compute node” queries.	Compile, Runtime, Completion	`Is True`, `Not True`
Temporary Bytes Read	Number of bytes read from disk because spilling occurs during the query. Select `io_spill_read_bytes` from `sys.log_query` to find this value for past executions of a query.	Runtime, Completion	Any number
Temporary Bytes Written	Number of bytes written to disk because spilling occurs during the query. Select `io_spill_write_bytes` from `sys.log_query` to find this value for past executions of a query.	Runtime, Completion	Any number
Total Duration (in ms)	The total time a query took to run (end time - submit time). This includes planning, compilation, and queue time. Use **Query Details** > **Statistics** > **TIMING** in Yellowbrick Manager to find this value for past executions of a query. Alternatively, select `total_ms` from `sys.log_query`.	Runtime, Completion	Any number
Total Memory (in bytes)	The total amount of memory used across the cluster to run the query. Use **Query Details** > **Statistics** > **UTILIZATION** in Yellowbrick Manager to find this value for past executions of a query. Alternatively, select `memory_total_bytes` from `sys.log_query`.	Runtime, Completion	Any number
Type	Type of administration operation, such as `create`, `delete`, `drop`, `backup`, `restore`, `analyze`, and `ycopy`. These values correspond to the `type` values in the `sys.query` and `sys.log_query` views.	Assemble, Compile, Runtime, Completion	Select from the drop-down list.**Note:** You can define rules for `INSERT INTO...SELECT` statements but not for `INSERT INTO...VALUES` statements. `INSERT INTO...VALUES` operations are not executed by the backend database. See also Managing the Row Store.
User Name	Database user name	All types	Select from the drop-down list.
User Roles	Database roles (in which a user has membership)	All types	Select from the drop-down list.

Rule Processing

Active rules are evaluated every time a query or other operation is executed. An active rule is an *enabled* rule that is associated with the currently active profile. Remember that global rules apply across all profiles. You need to be aware which global and system-defined rules are active, as well as the rules you have created.

To see how specific rules are applied to logged and active queries, go to **Workload Management** > **Rule Processing**. This screen presents detailed information about the application of each rule to each query, including when rule processing begins and ends.

When and how rules are processed depends on the type of rule in question; for details, see Rule Types. In general, rules are *evaluated* in a cycle that follows the query lifecycle. Active `submit` rules are evaluated first when a query starts, followed by `assemble` rules, `compile` rules, `runtime` rules, and finally `completion` rules. Any active `restart` rules may also be evaluated as required.

Runtime rules are evaluated every 5 seconds, assuming that the query runs longer than that time. A runtime rule may not apply the first time it is evaluated but may apply later. Other types rules are evaluated only once and either apply at that moment or not apply at all. However, note that `restart` rules send a query back to the `assemble` state, which means that certain rules will be re-evaluated for the restarted instance of the query.

Whether a rule is *applied* or not depends on its logic. For a given query all active rules must be evaluated but only some (or none) will be applied. For example, a rule that triggers an action when a certain user runs a query is only applied when that condition is met (along with any other related conditions).

Note: When profiles and resource pools change, these changes affect rule processing. For example, if at the end of all processing, a rule action for a query specifies a resource pool that does not exist, the rule returns an error and the query runs in the default pool.

Setting and Matching Resource Pools in WLM Rules

A rule condition on the `w.resourcePool` property will not find a match unless a previous active rule in the sequence has already taken an action to *set* the resource pool explicitly. Until an active rule is evaluated and sets the resource pool to some other value, `w.resourcePool` equals `null` (throughout the entire rule evaluation and application process). The WLM rule definition language does not provide a way to read the current default pool name for the active profile.

For example, consider the following rule, which attempts to log a message when the resource pool is `small`:

```
if (w.resourcePool === 'small')
            { log.info('Executed in the small pool'); }
```

This rule will not find a match if it is not preceded by a rule that sets the resource pool to `small`. For example, the following `default_mapToSmall` rule, which runs when the `default` profile is active, contains such an action:

```
if (w.memoryEstimate <= 1024 && w.memoryEstimateConfidence === 'High')
            {w.resourcePool = 'small';}
```

If you write a rule condition on `w.resourcePool`, be sure to set up a prior rule that sets the pool name appropriately.

Rule Types

Platforms: All platforms

Parent topic: Creating WLM Rules

The following table defines the types of rules that you can create. You can apply a rule at certain points in the life cycle of a query by specifying a type that maps to the current state of the query. Rules are applied as the query enters the state defined by the rule type. See also How Queries Are Executed.

Rule Type	Description
Submit	Evaluated when the query is first registered by the front-end database.
Hinting	Evaluated during `parse` state, right after query submission. These allow for planner hints to be injected in the query. **Note:** This is only available when using SQL command CREATE WLM RULE. See Plan Hinting for all details.
Assemble	Evaluated when the query is building the artifacts required for query compilation, before it is submitted to a resource pool. A variety of actions can be taken at this stage, including imposing limits on execution time and memory, setting query priority, or selecting a resource pool.
Compile	Evaluated during the pre-execution phase for a query, when a resource pool has been selected. The same set of actions that you can apply for Assemble rules are available, along with one additional action: the ability to move the query to a different pool.
Restart on Error	Evaluated when a query encounters an error that allows it to be restarted automatically by the system. Queries are potentially restartable from the `assemble`, `compile`, `acquire resources`, and `run` states. Restartable (or "recoverable") queries may loop back into the cycle multiple times. They always restart from the `assemble` state. See Recoverable Error Codes.A small number of user-defined actions can be defined for evaluation when queries go into a restart state. These rules are mainly intended for logging purposes. Because restarted queries loop back through earlier states in the query life cycle, the rules for those states are evaluated and applied again. This behavior may cause the restarted query to acquire and run with exactly the same resources, more resources, or less resources. Queries restarted on error may result in the same error that the first attempt at execution produced. The default behavior defined by the `global_restartErrorPolicy` rule is to give each query 1 retry, running the query in the same resource pool as before. If this policy does not work well in your application, you may want to set up a rule that changes the query to use a pool with more memory, or in the case of flex pools, use the maximum memory available. See Restart Rule Examples. **Note:** If you are using CREATE WLM RULE, the SQL type for Restart on Error is `restart_for_error`.
Restart via Rule or Administrator	Evaluated when a query goes into a restart state that was triggered either by the application of a rule or by a RESTART `command run by an administrator. See also "Restart on Error."**Note:** If you are using` CREATE WLM RULE, the SQL type for Restart via Rule or Administrator is `restart_for_user`.
Runtime	Evaluated while the query is running (in the `run` state). These rules are processed for long-running queries only, at 5-second intervals. If a query runs to completion in less than 5 seconds, these rules are not processed at all. Several different actions can be taken. Long-running queries can be intercepted and aborted, the priority of queries can be reset, queries can be moved to different pools, concurrency can be throttled, and query activity can be logged.
Completion	Evaluated once after all query execution steps are completed or stopped for some reason. Completion states are `Done`, `Cancel`, and `Error`. The main actions available at this stage are related to logging.

A Simple WLM Example

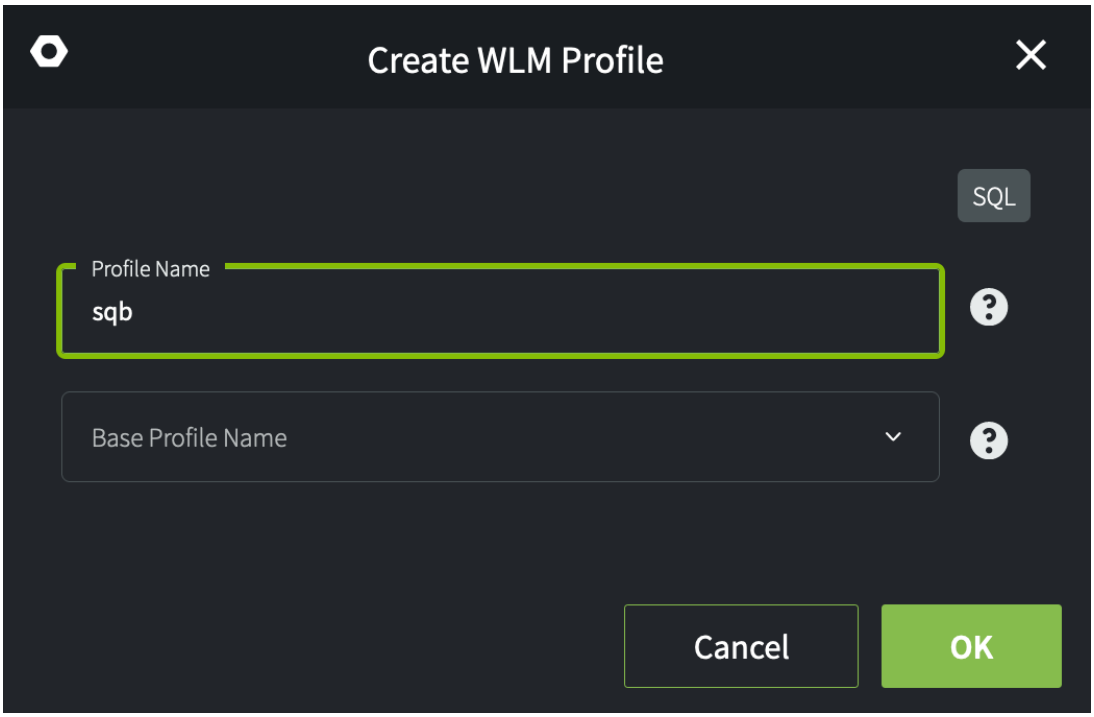
This section explains how to create a very simple set of WLM objects and implement a single rule that is applied when certain queries are run. All of the setup steps in the following example can be completed in Yellowbrick Manager (as shown) or by using SQL commands. SQL is generated at each step and can be downloaded for the profile, if needed.

To set up a simple example of workload management:

1. Log in to Yellowbrick Manager as an administrator and go to **Workload Management** > **Profiles** > **+Profile**.

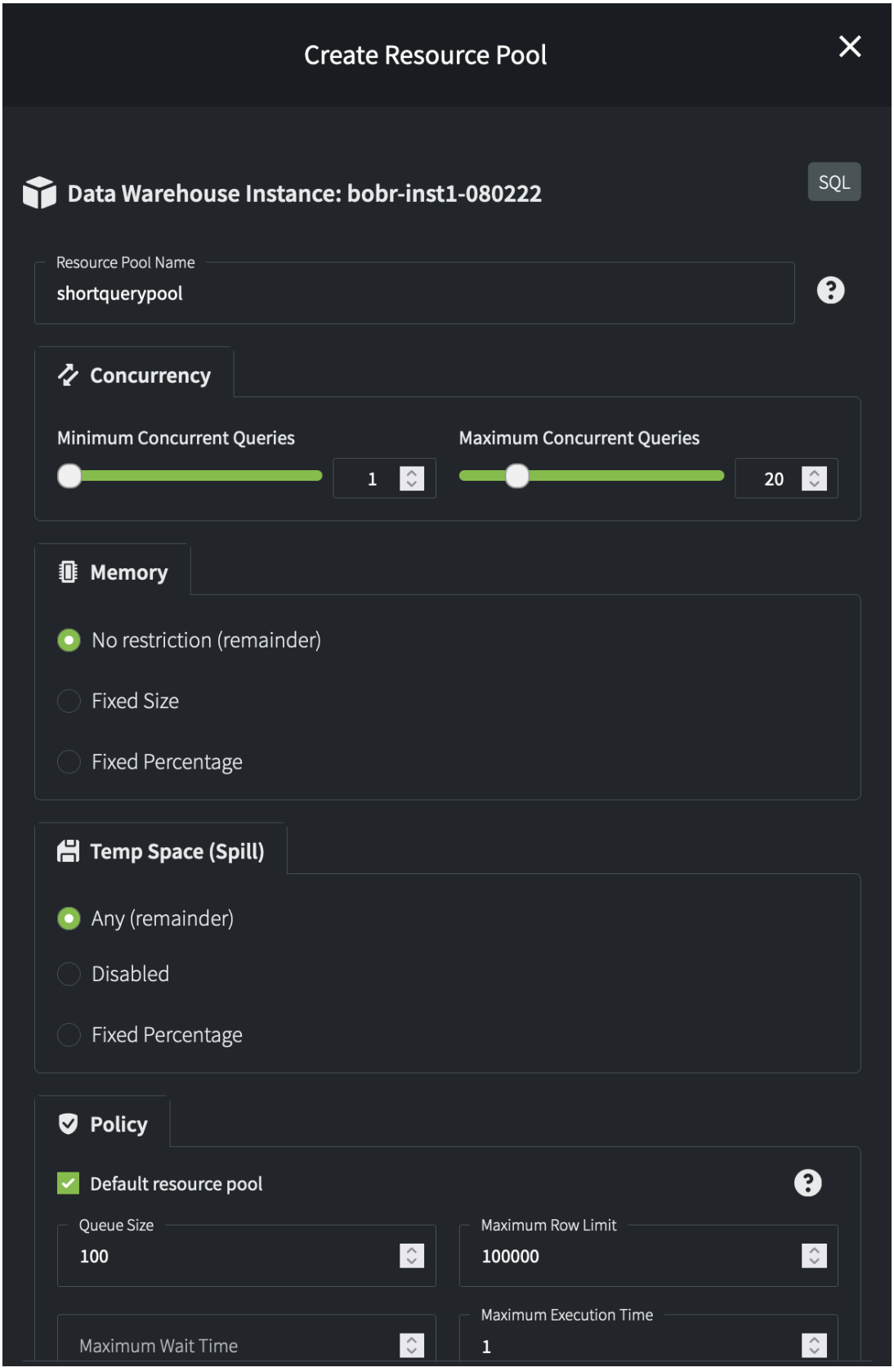
Note: Regular users cannot see the **Workload Management** tab.

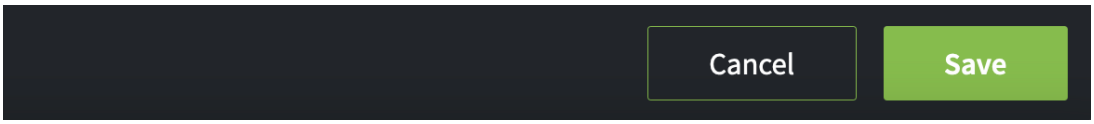
2. Create a profile and give it a meaningful name, such as `sqb` (for "shortquerybias").



By default, this new profile contains no resource pools.

3. Click **Create Resource Pool** under the `sqb` profile. Name the new pool `shortquerypool`, and give it 20 slots (maximum concurrency), a queue size of 100, a maximum row limit of 100000, and a maximum execution time of 1 second. Also make this pool the default pool for the profile.


Create Resource Pool
Data Warehouse Instance: bobr-inst1-080222
SQL
Resource Pool Name
shortquerypool
Concurrency
Minimum Concurrent Queries
1
Maximum Concurrent Queries
20
Memory
No restriction (remainder)
Fixed Size
Fixed Percentage
Temp Space (Spill)
Any (remainder)
Disabled
Fixed Percentage
Policy
Default resource pool
Queue Size
100
Maximum Row Limit
100000
Maximum Wait Time
Maximum Execution Time
1

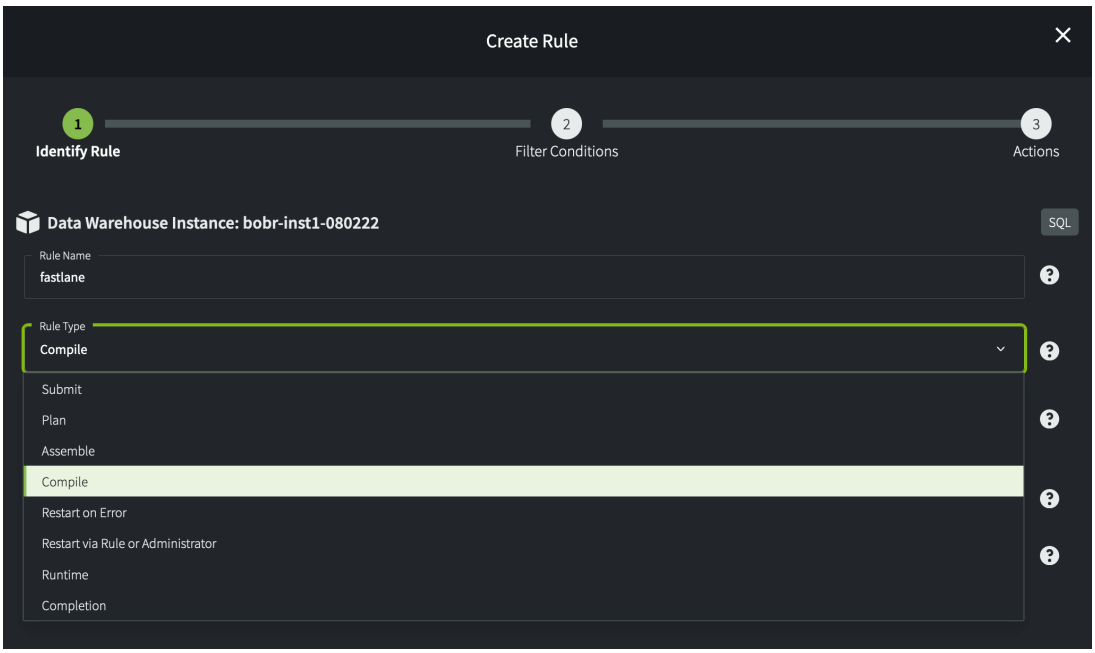


4. Create a second resource pool in the same profile. Name the new pool `longpool`, and give it 2 slots (maximum concurrency), a queue size of 100, no maximum row limit, and no maximum execution time.

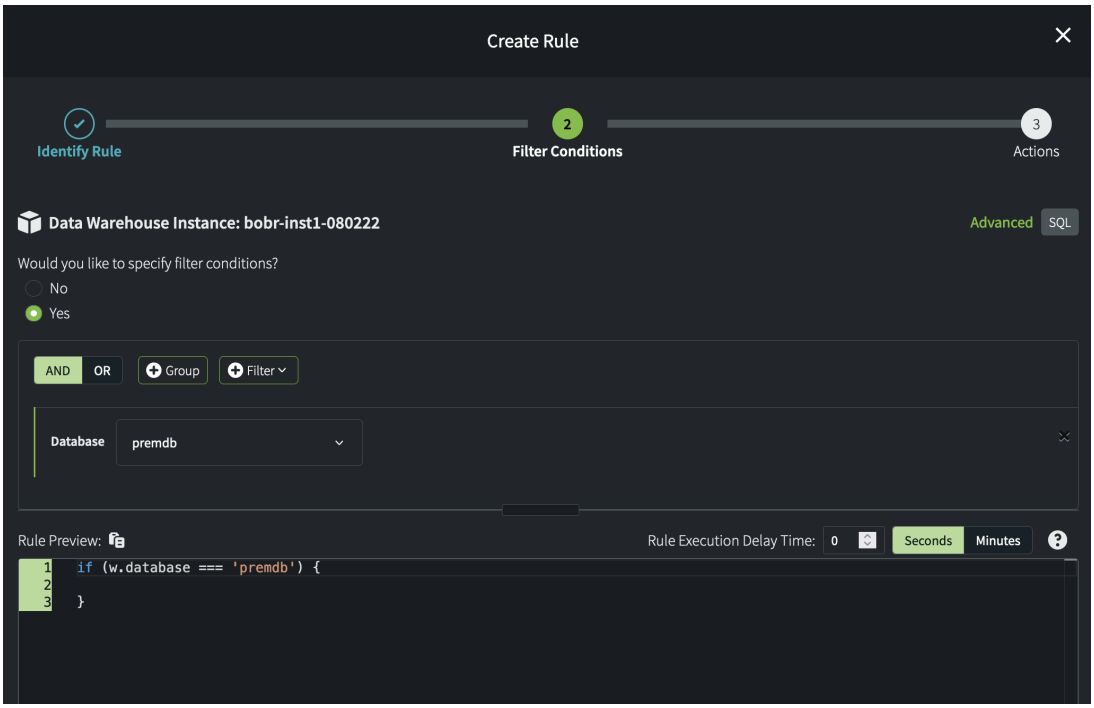
When the `sqb` profile is activated for a running cluster, it will also accrue the `system` pool, so three resource pools will be in operation.

5. Go to **Workload Management** > **Global Rules** > **+Rule**.

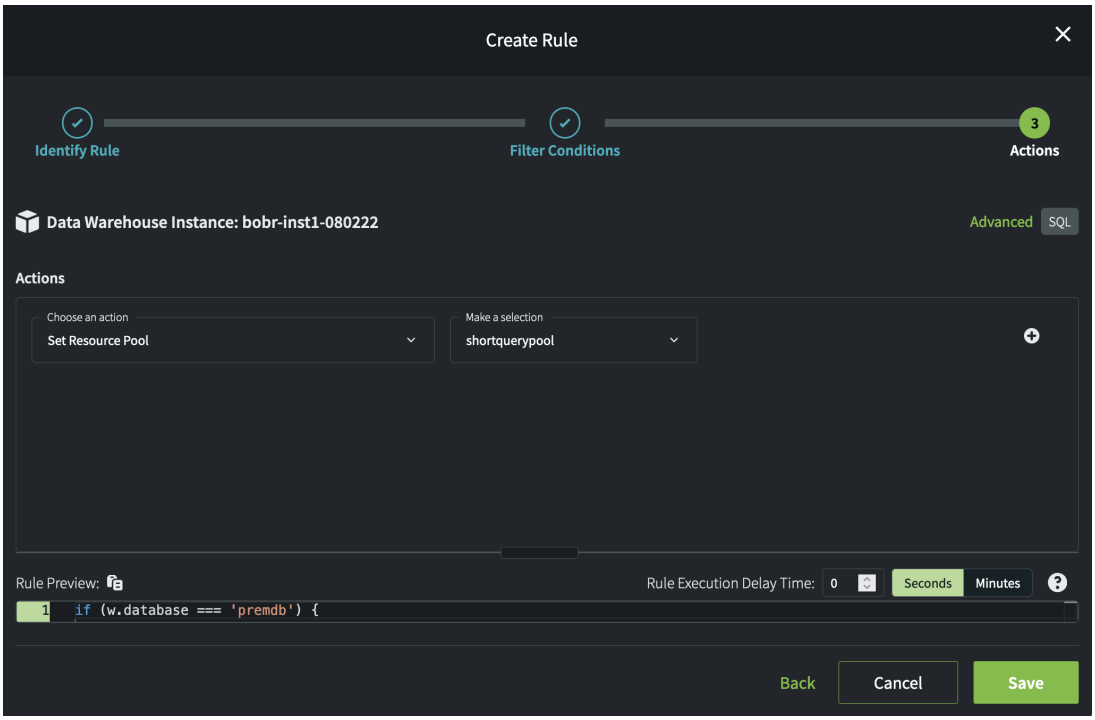
6. Create a `Compile` type rule in the `sqb` profile and name it `fastlane`.



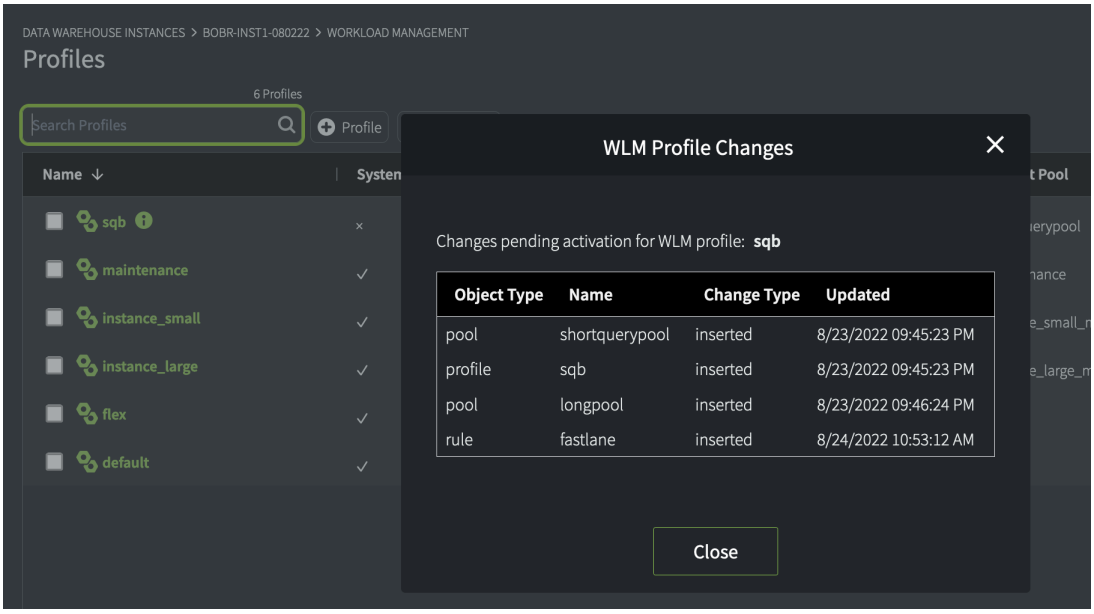
7. Click **Next**, then set a filter where `Database=premdb`.



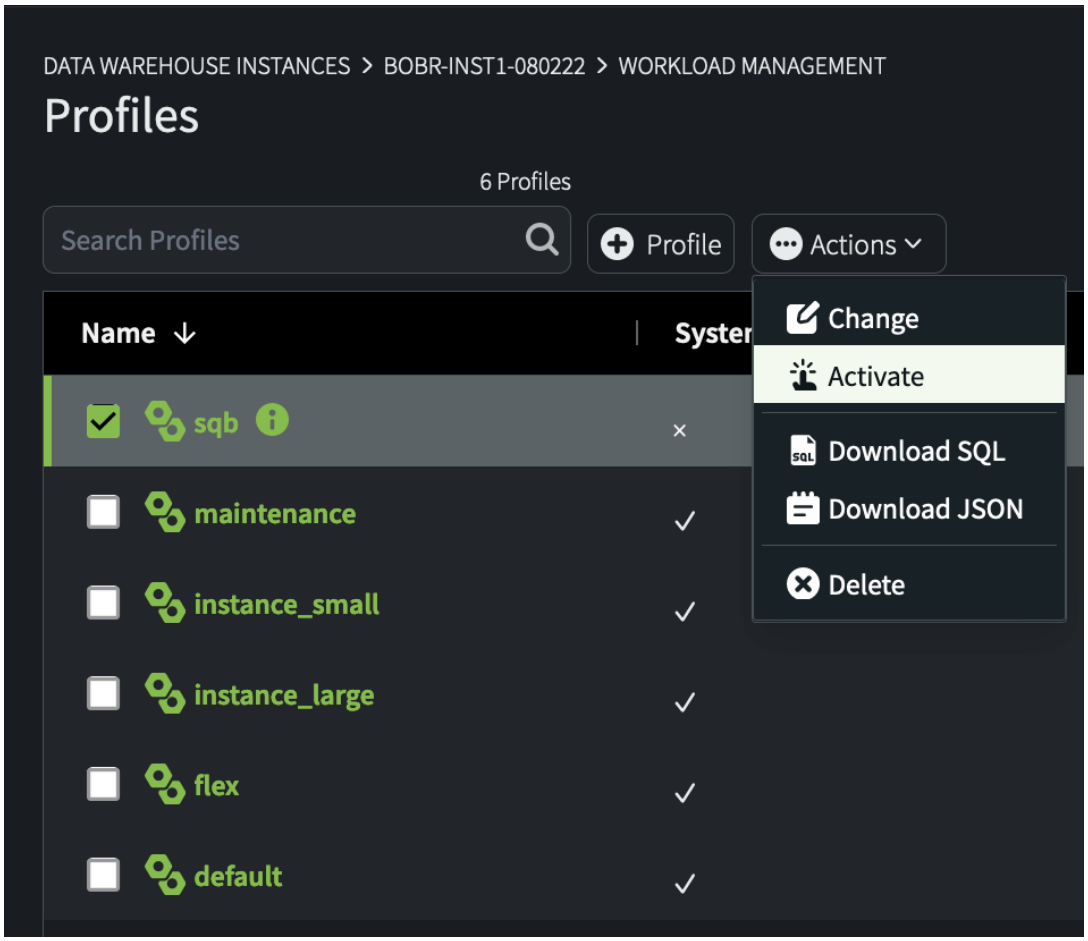
8. Click **Next**, specify an action to set the resource pool to `shortquerypool`, and click **Save**.



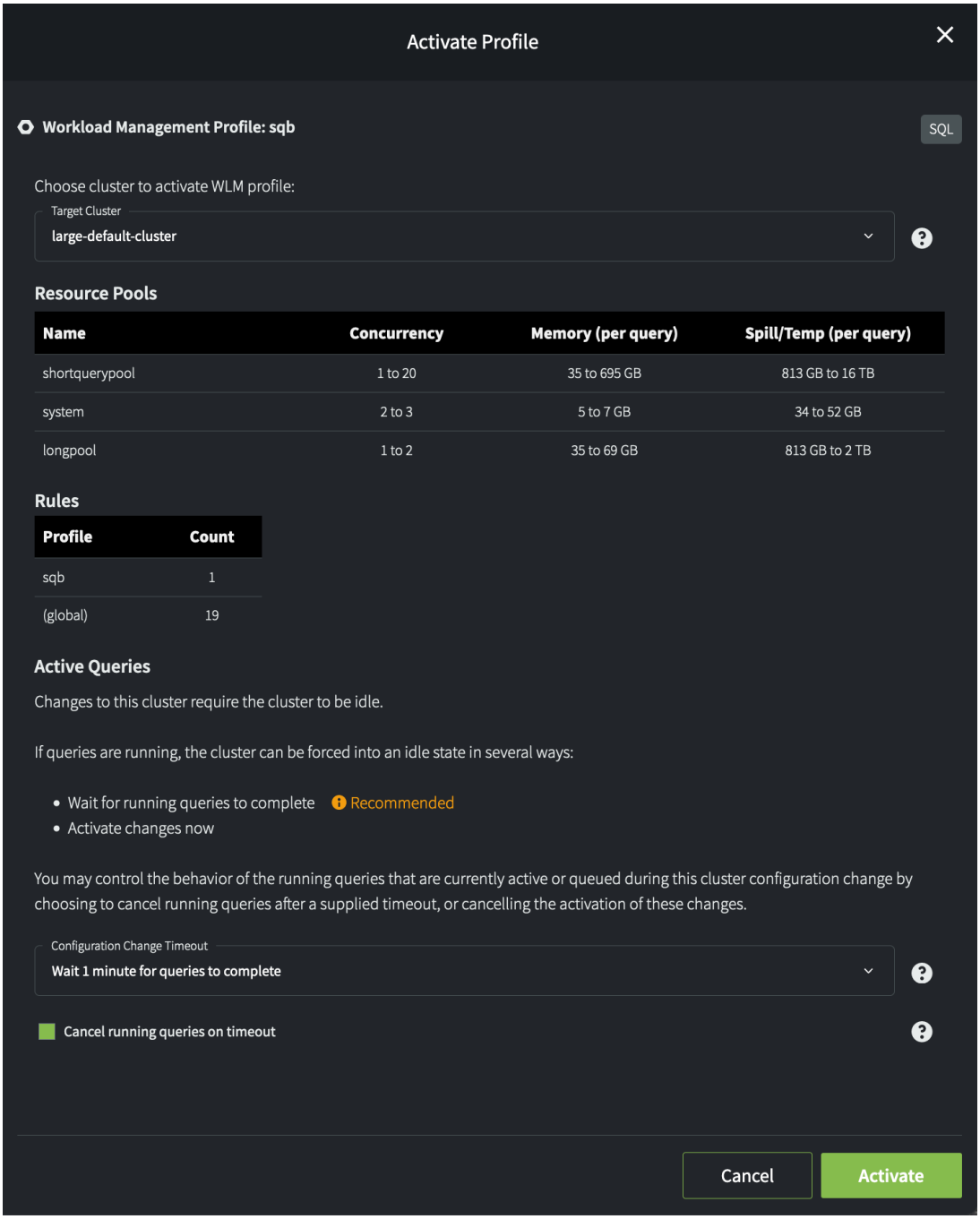
9. Go back to **Profiles**, then click the green info icon next to `sqb`, which shows the profile changes that are ready to be activated.



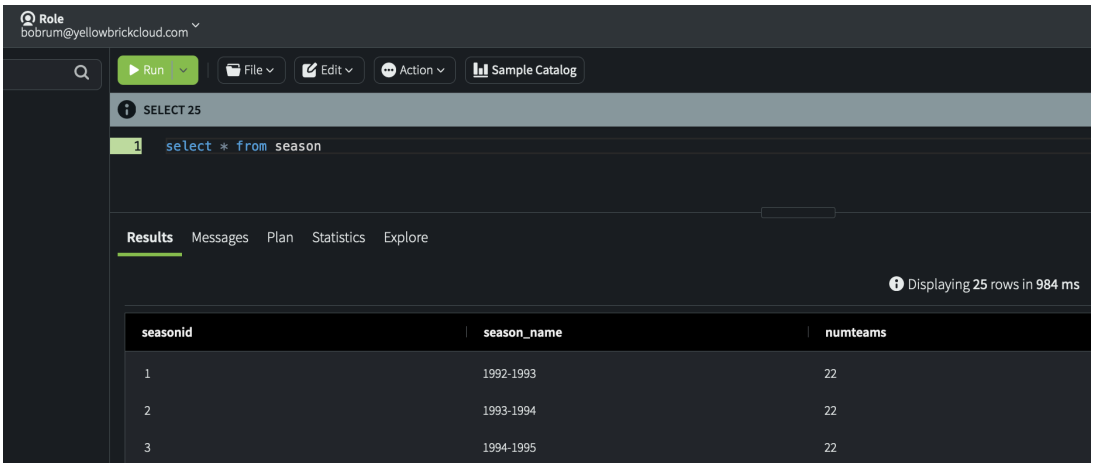
10. Prepare to activate the `sqb` profile by selecting the profile, then selecting **Activate** from the **Actions** menu.



11. Select the `large-default-cluster` target for activation, accept all the default options, and click **Activate**. Note that the `system` pool has been added to the list of resource pools, and one rule is specific to this profile.

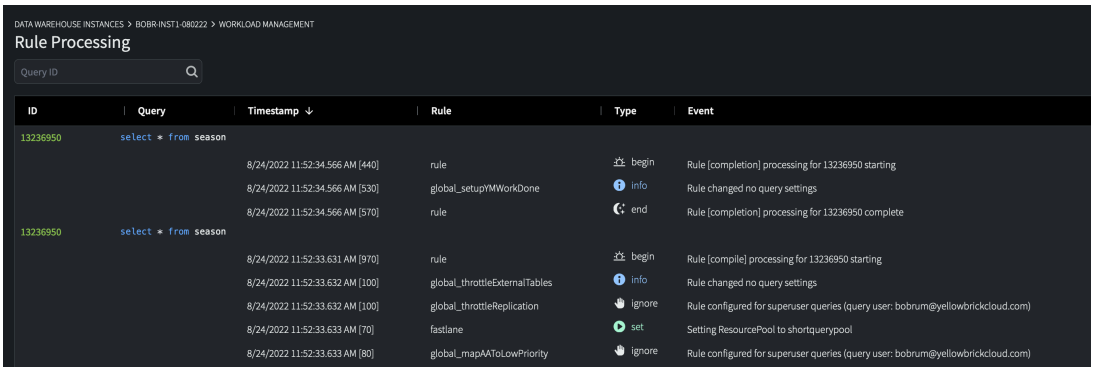


12. Go to the Query Editor and run a query as a regular user with `select` privileges on the tables in the `premdb` database (and `USAGE` granted on the `large-default-cluster`. For example:



13. Log in as an administrator so you can monitor WLM activity.

14. Go to **Workload Management** > **Rule Processing** to see which rules and pools were processed for the query.



Note that the `fastlane` rule was applied during the compile phase of the query, and the resource pool was set to `shortquerypool`.

You can also check rule-processing behavior by selecting from the `sys.log_query` view. For example:

You can download the SQL for the `sqb` profile for future use:

```
CREATE WLM PROFILE "sqb";
CREATE WLM RESOURCE POOL "shortquerypool" (
           PROFILE "sqb"
```

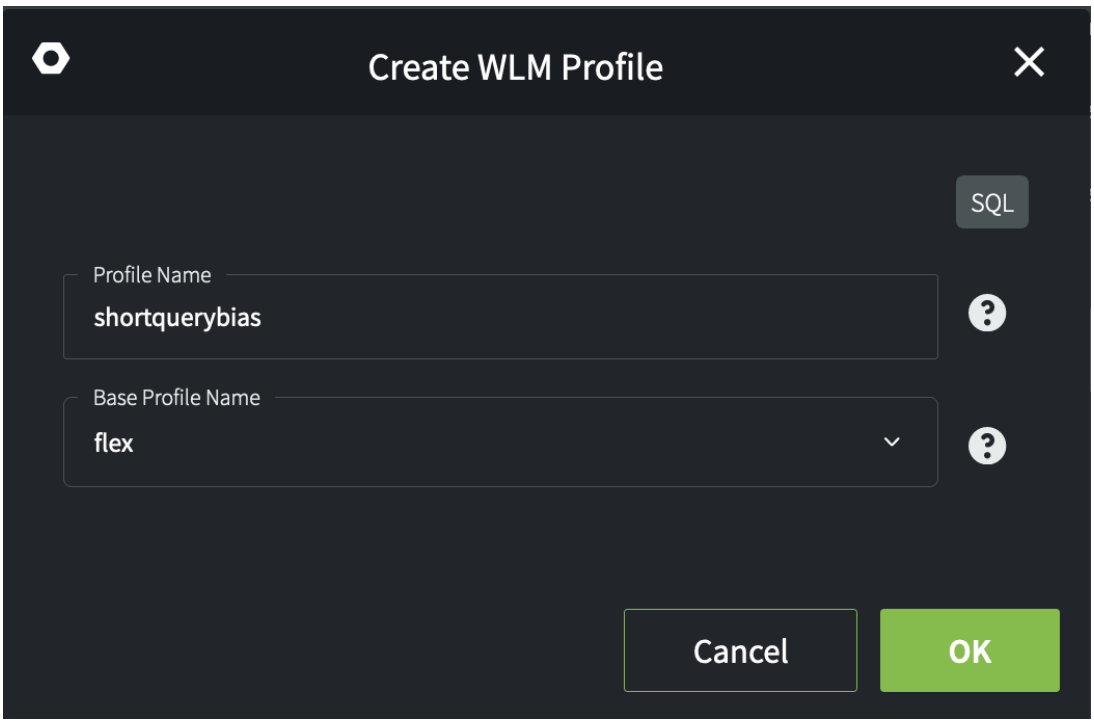
```
          , REQUESTED_MEMORY NULL
          , MAX_SPILL_PCT NULL
          , MAX_CONCURRENCY 20
          , MIN_CONCURRENCY 1
          , QUEUE_SIZE 100
 , MAXIMUM_ROW_LIMIT 100000
 , MAXIMUM_EXEC_TIME 1);
CREATE WLM RESOURCE POOL "longpool" (
            PROFILE "sqb"
          , REQUESTED_MEMORY NULL
          , MAX_SPILL_PCT NULL
          , MAX_CONCURRENCY 2
          , MIN_CONCURRENCY 1
          , QUEUE_SIZE 100);
CREATE WLM RULE "fastlane" (
            PROFILE "sqb"
          , TYPE compile
          , RULE_ORDER 1
          , SUPERUSER false
          , ENABLED true
          , JAVASCRIPT $$if (w.database === 'premdb') {
    w.resourcePool = 'shortquerypool';
}$$);
ALTER WLM PROFILE "sqb" (DEFAULT_POOL "shortquerypool");
```

Creating WLM Profiles

In terms of how you create it, a profile is simply a container for some *resource pools* and *rules*. You can create a profile in Yellowbrick Manager or by using a SQL command:

- In Yellowbrick Manager, go to **Workload Management** > **+Profile**.

You can create a new blank profile, or you can create a profile as a copy of another profile, which duplicates all of the resource pools in the base profile (copied pools are named `profile: pool`). For example:



- Use the SQL command CREATE WLM PROFILE.

To see the profiles that are currently in the system, go to **Workload Management** in Yellowbrick Manager, query the sys.wlm_active_profile view, or use the `ybsql \dwp` command, as shown here:

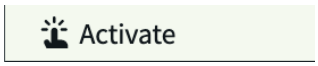
```
premdb=> \dwp
                List of WLM profiles
     Name      | Active? |    Default pool    | System?
---------------+---------+--------------------+---------
 default       | no      | large              | yes
 flex          | no      | mix                | yes
 instance_large | no      | instance_large_mix | yes
 instance_small | yes     | instance_small_mix | yes
```

```
 maintenance    | no      | maintenance        | yes
 new            | no      |                    | no
(6 rows)
```

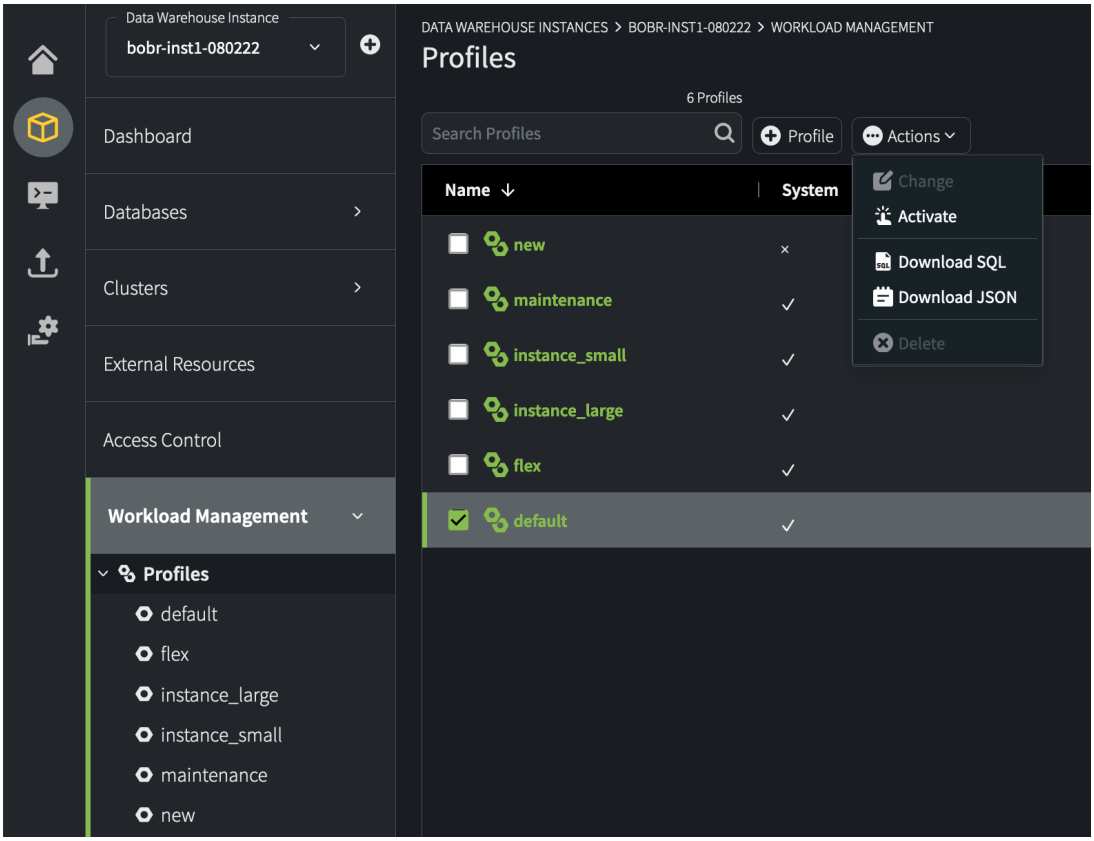
Profile Activation

The active configuration for workload management is a transient state that you set by activating a profile. The resource pools and rules associated with the profile determine the WLM behavior for all of the queries and other operations that run on the system.

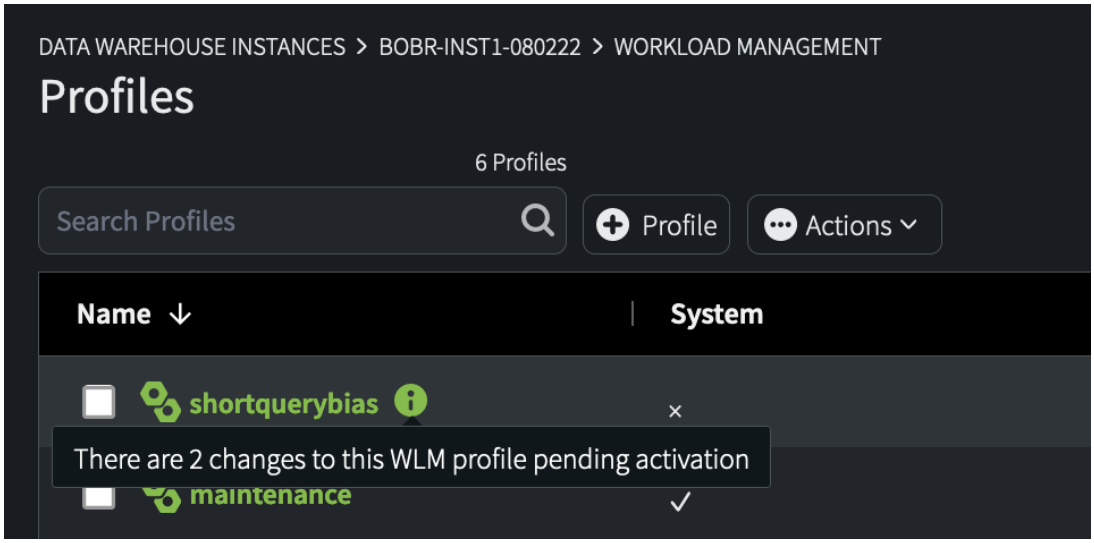
You must activate any changes to your WLM configuration before they will take effect. For example, you can add or change a rule at any time, but the new or changed rule will have no effect until you activate the changes. You can use the **Activate** button in Yellowbrick Manager:



This button becomes available on the **Profiles** screen when you select a profile or when changes are pending, such as the addition of a WLM rule:



You will also see a green "info" icon next to a profile when it has changes pending:



You can choose to activate the changes immediately or wait a specified period of time for active queries to complete. You can set the profile activation to fail or succeed if queries are still running when the timeout expires.

Note: The target of an **Activate** operation must be a compute cluster that is running when activation is requested. This operation does not persist between cluster restarts and will fail if the cluster is suspended.

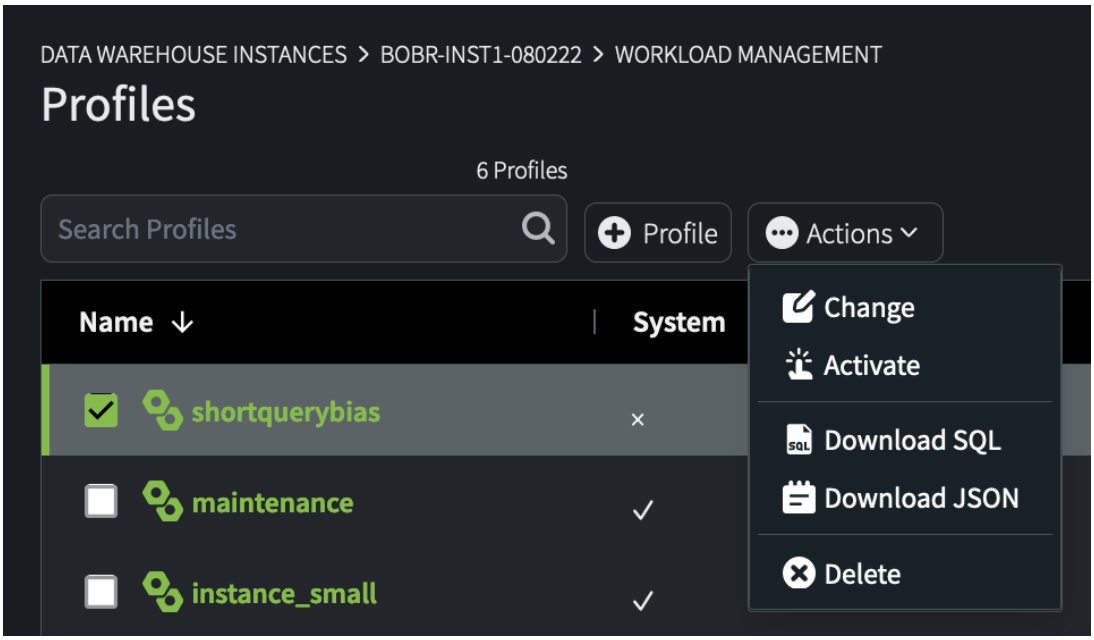
When you activate a profile, the currently active profile is deactivated. There is no specific option or command that deactivates a profile.

You can also use the ALTER WLM PROFILE command to activate profiles.

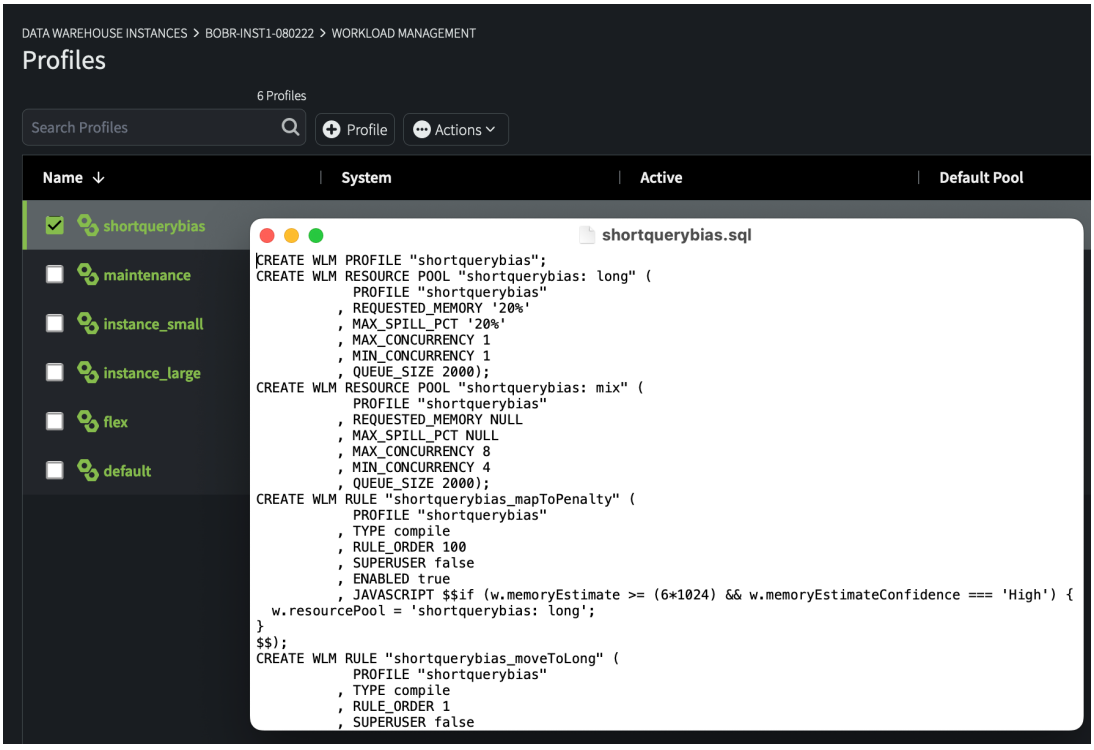
Download SQL and JSON Options

The **Actions** menu for profiles provides an option to download the definition of a profile as a SQL file or a JSON file. You can save this file for convenient reuse and reference. Select a profile first, then select **Download SQL** or **Download JSON**. You may be prompted to select an application suitable for capturing the download.

For example:



This example shows a SQL file downloaded and saved in a text editor:



Dropping Profiles

When you drop a profile, its associated resource pools and rules are dropped automatically as well. See DROP WLM PROFILE.

System-Defined Profiles

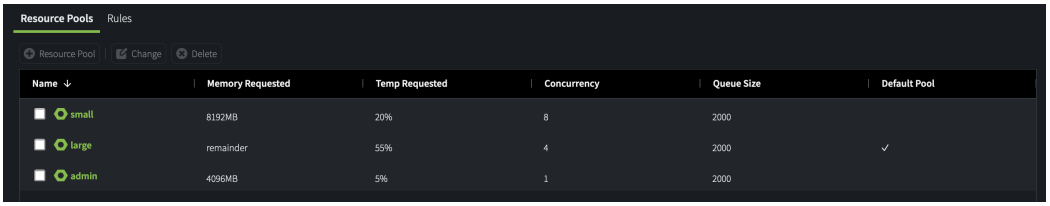
When a Yellowbrick data warehouse is installed, it contains several system-defined profiles, which you cannot remove or modify; however, they may serve as templates for your own profiles. You can use one of these profiles as a base for a new profile that you create.

The `default` profile remains active until you activate another profile, which guarantees that, in the absence of a user-defined strategy, default WLM behavior is in effect for all queries. If a user-defined profile fails to load and activate at any time, the `default` profile becomes the active profile.

Note: When any profile is activated for a cluster, it also has the `system` pool.

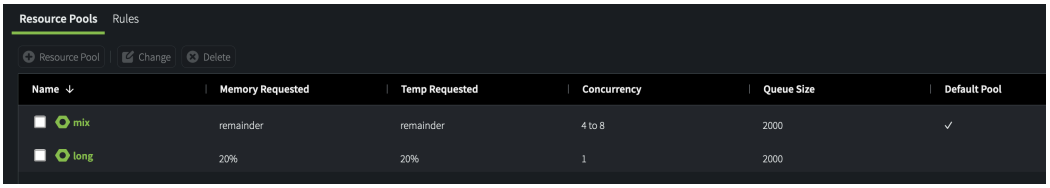
default Profile

This profile is defined with the following resource pools, which you can monitor when the profile is active. The actual memory allocation when the profile is activated varies by cluster, depending on the hardware instance type.



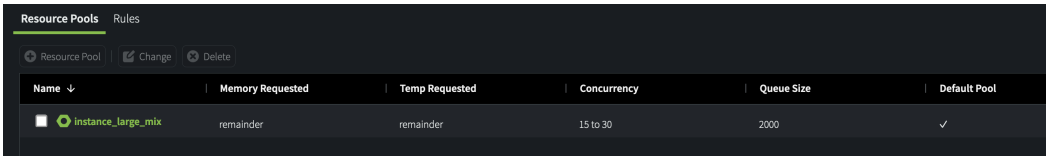
flex Profile

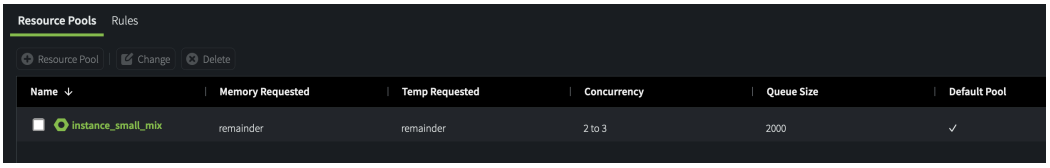
This profile contains the following resource pools, which you can monitor when the profile is active. Like the profile itself, these pools cannot be changed. This profile has a flex pool called `mix`, with a minimum concurrency of 4 and a maximum concurrency of 8.



instance_large and instance_small Profiles

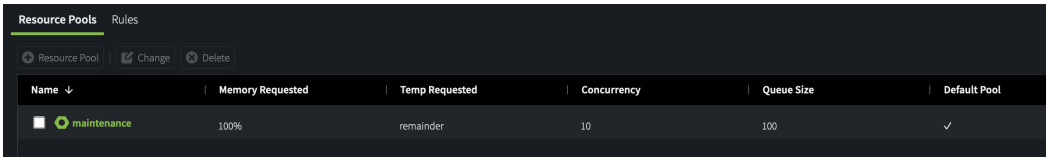
These profiles are targeted for clusters with large and small instance types. They both contain one resource pool. Both pools are flex pools with minimum and maximum concurrency settings (15/30, 2/3).





maintenance Profile

The `maintenance` profile contains a single resource pool with a fixed concurrency of 10.



Administrators can put the Yellowbrick system in *maintenance mode* by activating this profile in Yellowbrick Manager or running the following command:

```
alter wlm profile "maintenance" activate;
```

All connected sessions for regular users are closed, and their active queries are canceled. While the maintenance profile is active, regular users cannot connect to the database; they will see a `System in maintenance mode` unrecoverable error message. Any superuser can turn maintenance mode off by switching to another profile. For example:

```
alter wlm profile "default" activate;
```

Moving Queries

Administrators can *move* and *restart* queries, either by defining WLM rules that are applied conditionally or by running SQL commands while a query is in flight. A query can only be moved to a pool, or restarted in a pool, that is part of the active WLM configuration. This section explains the move (or "pool hopping") case. See also Restarting Queries

---

Moving a Query to Another Pool

Queries are sometimes assigned to a pool that turns out not to be an optimal choice for some reason. For example, a query may be estimated to be a short-running query. If this estimate is wrong, administrators may want to move the query into another pool, rather than consume resources allocated to the short-query pool and hold up genuine short queries.

Administrators can move a query from one pool to another by creating WLM compile or runtime rules with the following action:

```
w.moveToResourcePool('name')
```

A second Boolean argument to this property allows a query to be "moved" to the same pool where it started:

```
w.moveToResourcePool('name', 'true')
```

You can use Yellowbrick Manager or the CREATE WLM RULE command to define these rules.

Alternatively, instead of relying on rule processing, you can use a MOVE query command to move a query while it is executing.

A query that is explicitly moved may either continue to execute in the new pool, or it may have to be *restarted* in the new pool. If a query is moved into a pool with less memory or spill space available per slot than the original pool, the query cannot continue in the new pool and has to be restarted. The query will either start again in the new pool immediately or wait in the queue for that pool until a slot becomes available.

Movable Queries

The following conditions apply to moving queries:

- The query must be of type `SELECT`, `INSERT` (`INTO SELECT`), or `CTAS`.
- The query must be running on the compute nodes (not on the front-end database).
- The query must not have failed with a non-recoverable error.
- The query must be restartable (in case the attempt to move the query results in an attempt to restart it). This may happen when the target resource pool does not have adequate resources to continue running the query.
- The query must not be running in a transaction that has modified data (which effectively makes the query ineligible for restart).
- The query must not have started sending data to the client.

Example of a Move Rule

For example, the following rule condition states that any `CTAS` type query that is running in a given resource pool can be moved to the `ctas_pool` pool:

```
if (w.type === 'ctas') {
    w.moveToResourcePool('ctas_pool');
}
```

A SQL command to create this runtime rule would look like this:

```
premdb=# create wlm rule ctas_move
(type runtime,
javascript $$ if(w.type === 'ctas') {w.moveToResourcePool('ctas_pool')}$$
);
CREATE WLM RULE
```

For another example, see Move a Query to the "Long" Pool.

Example of a SQL MOVE command

The following SQL command moves a query to another pool.

```
premdb# move 17831 to wlm resource pool flexpool10;
MOVE
```

where `17831` is the ID of a query and `flexpool10` is the name of a resource pool in the active WLM configuration.

Restarting Queries

This section describes how WLM handles cases where the system restarts queries or an administrator (or rule) triggers a restart. See also Moving Queries.

Restartable Queries

Queries may be restarted by the system or by an administrator when they meet certain criteria. To be restartable, a query must be in one of the following states:

- `assemble`
- `compile`
- `acquire_resources`
- `run`
- `Restart On Error`

Note: Restarting a query in the `assemble`, `compile`, `acquire_resources`, or `run` states will run the query again, going through all earlier states in the query's life cycle.

However, restarting a query during `Restart On Error` will only move the query to the target resource pool.

If a query has already returned its first row to the client (`client_wait` state), it cannot be restarted.

In addition, only the following types of queries can be restarted:

- `select`
- `ctas`
- `insert` (`INTO SELECT`, but not `VALUES`)

If a query is processing an error or is already in the process of restarting, it cannot be restarted; however, a restart may be possible later. See the Default Error Recovery section.

Default Error Recovery

By default, the same query may be restarted *only once* before it is deemed to be unrecoverable. The restart policy in system-defined rules (`global_restartErrorPolicy` and `global_restartErrorPolicySuperuser`) enforces this behavior; in general, you do not need to change it. Depending on the type of error a query encounters, attempting to restart it more than once may not be of practical value. The query may error out in exactly the same way before it eventually aborts.

The system rules named `global_restartErrorCodes` and `global_restartErrorCodesSuperuser` constrain the default restart policy so that it applies only to queries that fail with the following subset of recoverable error codes:

```
KE038 RPCCHANNELCLOSED
KE039 RPCCHANNELBROKEN
KE041 Library file write
P0004 ERRCODE_ASSERT_FAILURE
WM001 ERRCODE_WORKER_OFFLINE
WM002 ERRCODE_SYSTEM_NOT_READY
WM003 ERRCODE_CANCEL_FOR_RESTART
YB044 RecoverableGeneric
```

Queries that fail with other recoverable error codes (or any other error code) are not subject to the policy defined by these rules. Again, you do not need to change this behavior, but you can create new, modified `ErrorCodes` rules that include other codes from the recoverable list. See Restart Rule Examples.

Another system-defined rule (`flex_expandResourcesErrorHandler`) defines restart behavior for queries that run in the `flex` profile:

```
log.info(w + ' is restarting for error ' + w.errorCode);
if (String(w.errorCode).match(/53200|KE002|YB004|KE032|KE029|YB006|EEOOM/)) {

  // See if we can't expand resources; if we can, lets try the query with more resources.
  if (!wlm.assignMaximumResources(w)) {
    w.errorRecoverable = false;
    log.info(w + ' cannot expand resources; marked as not recoverable');
  } else {
    w.errorRecoverable = true;
    log.info(w + ' expanded resources for restart (memory ' + w.requestedMemoryMB + ', spill ' + w.requestedSpillMB + ')');
  }
}
```

This rule logs messages for queries that restarted after failing with a specific subset of the recoverable error codes (errors that typically indicate conditions under which a query is likely to benefit from more resources). For example, if a query runs out of memory, the `flex` profile can expand or contract its resources to accommodate different levels of concurrency, effectively making more (or all) of its memory available.

If `wlm.assignMaximumResources` returns `true` , this means that resources available to the pool (memory, temp space, priority/CPU) were expanded, and the `INFO` message that is sent when the query completes logs those runtime resource values. If the resources were not expanded, the `INFO` message reports that instead. Note that the `wlm.assignMaximumResources` property is in place for the purpose of error recovery and may not be of practical use within your own WLM rules. User-defined rules are more likely to benefit from setting *requested* resources, as defined by a different set of memory and spill space properties, or by setting query priority.

Requesting Additional Resources for Restarted Queries

Administrators can restart a running query in another pool explicitly by using the RESTART query command. When you restart a query in this way, you can request specific resources (memory, priority, and spill space), which are allocated if they are available when the command is submitted.

For example:

```
premdb=# restart 347010 to wlm resource pool large
with ( priority high, memory '500MB', memory '40%', spill '10%' );
RESTART
```

The same functionality is available via WLM rules that contain a restart action. The following rule shows how to request resources when a query restarts in a pool. The rule defines two actions based on a set of criteria.

```
if ((String(w.application).indexOf('ybsql') >= 0) &&
   w.user === 'bobr' &&
   w.errorCode === 'EEOOM') {
    w.requestedMemoryPercent = 100;
    w.restartInResourcePool('max_memory_pool');
}
```

This rule requests a restart in the `max_memory_pool` pool with 100% of its memory if an OOM error occurs for user `bobr` running a `ybsql` query.

Throttling Concurrency

You can use WLM rules to set an action that limits query concurrency when a condition is met. This kind of rule provides another layer of concurrency control for query workloads, in addition to the minimum and maximum concurrency settings that you define per resource pool. Throttling is a mechanism for making sure that lower-priority workloads are queued to protect other higher-priority workloads.

To set up one of these rules, use the `Submit`, `Assemble`, or `Compile` rule type and the Limit Concurrent Queries action. The result is a `wlm.throttle` rule. A "throttle" rule is primarily used to reduce rather than increase traffic to a pool. This type of rule requires one or two attributes:

- A `count` value, which limits concurrency to some number of queries (given some condition)
- A `throttleName` attribute that defines a concurrency limit per user, role, or application (for example). In theory this name can refer to any classification of queries, but is mainly intended to provide per-user or per-role controls over concurrency.

You can use the `count` attribute by itself in a rule, or you can use both attributes.

Note: A global system-defined rule (`global_throttleConcurrentQueries`) sets a maximum query concurrency limit of `500`. This rule can be changed, but Yellowbrick recommends that you create your own rule if you want to change this limit. You can use a larger value for the rule order so that a user-defined rule overrides the global rule. Alternatively, you can disable the global rule.

As usual, you need to activate the profile after making rule changes or adding new rules, but you may also need to suspend and resume the instance for this kind of change to take effect. If you see the following error during WLM rule processing, either suspend and resume the instance or drop and re-create the rule:

```
Rule error for throttle: Cannot throttle with different count values...
```

Examples

For example, write a rule that limits user `yb100` to `1` query, effectively turning off concurrency for that user:

```
if (w.user === 'yb100') { wlm.throttle(1, w.user); }
```

If this user submits two queries in succession, the second query may not start executing until the first is finished. Now rewrite this rule to use a count of `3`:

```
if (w.user === 'yb100') { wlm.throttle(3, w.user); }
```

User `yb100` can now start three queries in succession and they can all run concurrently (assuming that slots are available).

If you do not put a condition on the user, concurrency is limited for each user on the system. The following example disables concurrency for all users who run `INSERT` queries. Each user may run one `INSERT` at a time.

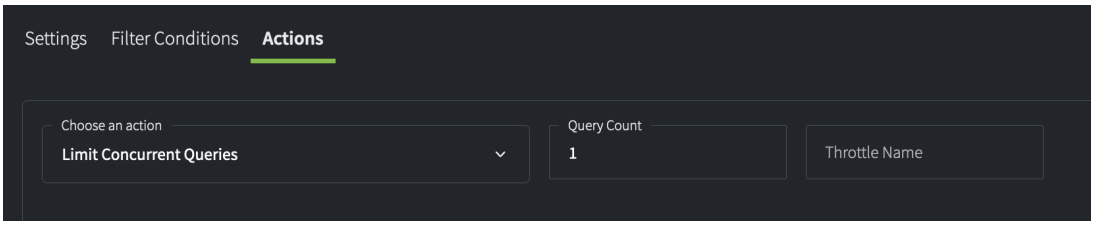
```
if (w.type === 'insert') {
    wlm.throttle(1, w.user);
```

The following example applies concurrency control at the application level. The `ybload` application is limited to using resources one load at a time. Any user with load privileges can run a single load, but two concurrent loads would be prohibited:

```
if (w.application === 'ybload') {wlm.throttle(1);}
```

The effect of this rule is similar to always assigning `ybload` operations to a resource pool with only one slot (minimum and maximum concurrency values of `1/1`).

If you create this rule in Yellowbrick Manager, it looks like this. Note that the action chosen is **Limit Concurrent Queries**, and that no throttle name is required (just the count value of `1`).



In the following example, assume that the users are developers who belong to a role called `dev`. The WLM rule in this case disables concurrency for individual users in that role but enables it at the role level:

```
if (String(w.roles).match(/dev/)) { wlm.throttle(1, w.user); }
```

That is, if 5 developers belong to this role, maximum concurrency for the pool when this rule is applied is 5.

Using Query Tags

You can define labels, known as *query tags*, to track query activity. The labels are typically used to classify workloads into a group in the absence of other identifying criteria. You can create WLM rules that define conditions based on the presence of tags. For example, based on the presence of a tag when a query runs, the query can be directed to a specific resource pool.

To define query tags in the database at the session level, use the `set ybd_query_tags` command. It's just an arbitrary string presented to the workload manager, without any special formatting. You can specify multiple items with semicolons or spaces or whatever you like. For example:

```
premdb=# set ybd_query_tags='premdb_query';
SET
```

To see the query tags that are currently available, use the `SHOW` command:

```
premdb=> show ybd_query_tags;
 ybd_query_tags
----------------
 premdb_query
(1 row)
```

You can use an ALTER ROLE command to make query tags apply to all of a user's sessions. For example:

```
premdb=# alter role bobr set ybd_query_tags='adolf ran this query';
ALTER ROLE
```

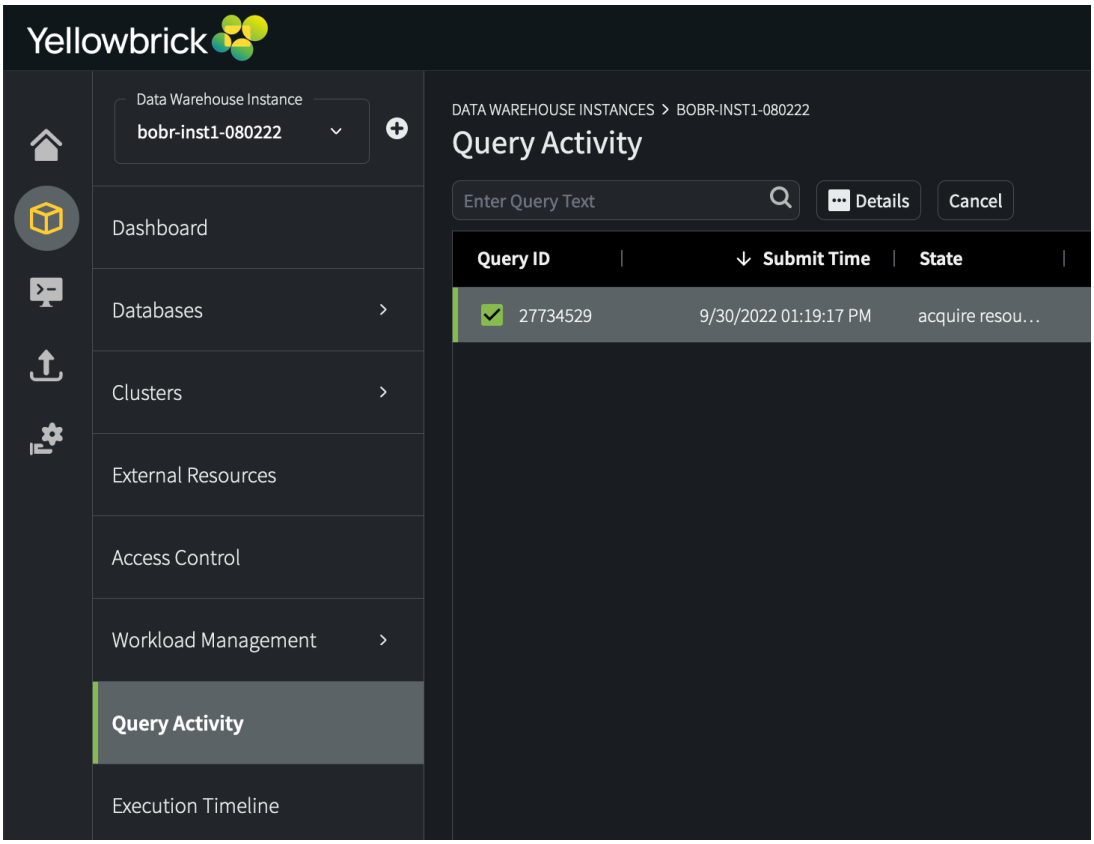
All of the queries that this user runs are logged with this string. Therefore, WLM rule conditions and actions can be defined with reference to it.

Tags associated with query execution are logged in sys.log_query.

Canceling Queries

You can cancel a running query in Yellowbrick Manager or by using the CANCEL command.

Go to **Query Activity** in Yellowbrick Manager, select the query, and click **Cancel**. For example:



Alternatively, select from the `sys.query` view to get the query ID, then use the `CANCEL` command.

To cancel an active query via SQL commands:

1. Get the `query_id` of the active query that you want to cancel. For example, if only one query is running:

```
premdb=# select query_id from sys.query;
 query_id
----------
    22669
(1 row)
```

2. Run the `CANCEL` command and supply the `query_id`:

```
premdb=# cancel 22669;
CANCEL
```

Note: If you are canceling a query from within your own session, you can use `Ctrl-C`. The `CANCEL` command is really intended for cases where another session or application runs a query that needs to be canceled by an administrator.

Aborting a Query with WLM Rules

You can cancel queries conditionally, when they are in certain states, by writing rules with an Abort Query action. See Creating WLM Rules.

Delayed Cancellation with statement_timeout

If you want to cancel queries after they have been running for a specific period of time, you can use the `statement_timeout` parameter. You can apply this parameter to a specific query, a session, or all of the sessions for a specific user. For example:

```
premdb=# set statement_timeout=1000;
SET
premdb=# insert into newmatchstats select * from newmatchstats;
ERROR:  canceling statement due to statement timeout
```

```
premdb=# alter role bobr set statement_timeout=1;
ALTER ROLE
premdb=# \c premdb bobr
Password for user bobr:
You are now connected to database "premdb" as user "bobr".
premdb=> select * from season, match where season.seasonid=match.seasonid and season_name='1999-2000';
ERROR:  canceling statement due to statement timeout
```

Creating Databases

Platforms: All platforms

Parent topic: Databases

Physical databases have some important attributes, including their name, owner, and *encoding*. See CREATE DATABASE for syntax details.

Database Encodings

The database *encoding* is critical because this attribute determines the type of data that can be loaded and queried. If you want to load and query multi-byte data, you need to create a database with the `UTF8` encoding. By default, databases are created with the `LATIN9` encoding, which does not support multi-byte data but provides better performance for query processing. If your database will not contain any multibyte data, use `LATIN9`.

`LATIN9`, (also known as `ISO 8859-15`) is based on `LATIN1` but has a few differences, including the addition of the Euro sign and replacements for some other characters. LATIN9 encodes 8-bit single-byte characters, covering character sets commonly used in Western Europe, the Americas, Oceania, and Africa.

You cannot change the encoding at runtime, so it is important to choose the right encoding when you create a database.

Note: Yellowbrick databases only support LATIN9 and UTF8 encodings in the database and for all data export (`SELECT`, `ybsql \copy to`, `ybunload`). For convenience, import paths via `ybsql \copy from` and `ybload` can read other encodings on the client side and transcode to one of the Yellowbrick-supported encodings.

- The Yellowbrick `ybsql \copy from` command supports the client encodings that Postgres supports. See the Postgres 9.5 documentation.
- Yellowbrick `ybload` operations support the client encodings that Java supports, but this list depends on the specific version of Java that is installed. For example, if you are using the default distribution of Oracle Java 8, see the Oracle documentation.

Locales

The locale for both `LATIN9` databases and `UTF8` databases is `C`. This locale applies to all database operations; no other locales are supported. You cannot change locale settings such as `lc_time`, `lc_collate`, and `lc_numeric`.

The bulk loader (`ybload`) supports different locales only for the purpose of importing data; once data is stored in a Yellowbrick database, it is always displayed with the default locale settings and sorted with the default collation order. See ybload Options for information about the `--locale` option.

Schemas

A database schema is a means of segregating tables and other objects inside a single database. For syntax details, see CREATE SCHEMA. After creating a schema, you can create tables and views that belong to it, as well as grant privileges to the objects in the schema.

The `search_path` configuration parameter defines the visibility of schemas to users.

```
premdb=# set search_path to 'bobr';
SET
premdb=# \d
       List of relations
 Schema | Name | Type  |  Owner
--------+------+-------+---------
 bobr   | new  | table | brumsby
(1 row)
```

```
premdb=# show search_path;
 search_path
-------------
 bobr
(1 row)

premdb=# reset search_path;
RESET
premdb=# show search_path;
   search_path
-----------------
 "$user", public
(1 row)
```

Alternatively, you can use a `SET SCHEMA` command, with single quotes around the schema name:

```
premdb=# set schema 'bobr';
SET
```

Cross-Database Queries

Yellowbrick databases are independent objects, but you can reference tables from multiple databases in the same query (in addition to writing queries that cross schemas in the same database). See Cross-Database Queries.

Creating Stored Procedures

Platforms: All platforms

Parent topic: Databases

You can create PL/pgSQL (`plpgsql`) language stored procedures and execute them in a Yellowbrick database. Yellowbrick stored procedures may perform actions without returning results, such as running DML operations or storing complex SQL calculations. You can also write stored procedures that return either a value or a set of rows. For details about the `plpgsql` language, see the PostgreSQL 9.5 documentation.

Stored procedures can be invoked at any time via `ybsql` or another SQL interface, and different values can be passed as parameters on each invocation.

Use the CREATE PROCEDURE command to create stored procedures. If your procedures do not return result sets (using `SETOF` syntax), execute them with the CALL command. If a procedure returns a set, you must execute it by using a simple SELECT statement that does not contain table references. For example:

```
select proc_set();
```

A procedure that returns a single value may be executed with either a CALL statement or a SELECT statement. The CALL statement does not display the returned value.

If a procedure that does not return a value is executed with a SELECT statement, the query returns a `NULL` value.

Stored procedures may have arguments that use any of the supported Yellowbrick data types. Stored procedure code may contain the following SQL commands:

- SELECT
- CREATE TABLE AS
- INSERT
- UPDATE
- DELETE
- TRUNCATE

Note: Although superusers are allowed to create stored procedures in the `sys` schema, this practice is not recommended. All users should create stored procedures in `public` or a user-defined schema.

Cross-Database Queries

You can write cross-database queries that select from tables in different physical databases, provided that these databases belong to the same data warehouse instance.

You can *run* cross-database queries from any database that belongs to the same instance as the tables listed in the `FROM` clause. The database you are connected to (the current database) does not have to contain any of the listed tables from which these queries read data.

The `FROM` clause must reference the tables unambiguously, typically using a combination of database name, schema name, and table name. (The database and schema names may be omitted from the local tables, but fully qualified table references are recommended.) For example:

```
premdb.public.awayteam
yellowbrick.public.match
test.schema1.table1
devel.schema1.table1
```

SQL Commands That Support Cross-Database Reads

Several SQL commands that do cross-database *reads* are supported. However, *writing* data across databases is not supported. The following SQL commands support cross-database reads:

- CREATE TABLE AS (CTAS): A locally created table may read some or all of its data from remote databases. You cannot create a table in a remote database.
- CREATE VIEW: A locally created view may read some or all of its data from remote databases. You cannot create a view in a remote database.
- DECLARE: You can declare cursors that reference remote tables. You can also fetch from, move, and close these cursors.
- DELETE and TRUNCATE: You can delete rows from a local table based on a query against one or more remote tables. You cannot delete (or truncate) rows from a remote table.
- EXPLAIN: You can explain queries that select from local and remote tables.
- INSERT: You can insert rows into a local table based on a query that selects from one or more remote tables. You cannot insert rows into a remote table.
- PREPARE and EXECUTE: You can prepare and execute queries that reference remote tables.
- SELECT, SELECT INTO, TABLE: You can select from any combination of tables and views in local and remote databases.
- UPDATE: You can update rows in a local table based on a query that selects from one or more remote tables. You cannot update rows in a remote table.

Restrictions

SQL statements that write to a table (`INSERT` , `UPDATE` , `DELETE` , `TRUNCATE`) must be executed from the database where the table resides. For example, the following `INSERT` fails because it attempts a cross-database write:

```
yellowbrick=# insert into premdb.public.match select * from yellowbrick.public.match;
ERROR:  cross-database modifications are not allowed: premdb.public.match
```

You cannot create, drop, or alter tables, views, and procedures remotely. You must create, drop, or alter them from the current database. For example, the following command fails because it attempts a cross-database `ALTER VIEW` :

```
yellowbrick=# alter view premdb.public.vteam rename to viewteam;
ERROR:  cross-database references are not allowed: "premdb.public.vteam"
```

`DESCRIBE`: You cannot describe remote tables. To run the DESCRIBE command on a table, connect to the database where the table was created.

`CREATE PROCEDURE`: You can create stored procedures in the current database only, and you can call stored procedures only if they were created in the current database. A stored procedure that is created with the `SETOF table` syntax must refer to a table that is created in the current database (not a remote table).

Queries and other commands cannot read from a remote `UTF8` database and write to a local `LATIN9` database. The opposite path is supported: reading from a remote `LATIN9` database and writing to a local `UTF8` database.

Note: If you create a view with cross-database table references, and one of the referenced databases is dropped and re-created or restored, the restore will not contain the view. The view was based on a reference to the original database ID, not the restored database ID. The `sys.view` system view will still contain a record of the view, but it will not be functional.

Examples

Assume that the same tables exist in both the `yellowbrick` database and the `premdb` database in the same data warehouse instance:

```
yellowbrick=# \d
         List of relations
 Schema |   Name   | Type  |  Owner
--------+----------+-------+---------
 public | awayteam | table | yb007
 public | hometeam | table | yb007
 public | match    | table | yb007
 public | season   | table | yb007
 public | team     | table | yb007
(5 rows)

yellowbrick=# \c premdb
You are now connected to database "premdb" as user "yb007".
premdb=# \d
         List of relations
 Schema |   Name   | Type  |  Owner
--------+----------+-------+---------
 public | awayteam | table | yb007
 public | hometeam | table | yb007
 public | match    | table | yb007
 public | season   | table | yb007
 public | team     | table | yb007
(5 rows)
```

In the following examples, `premdb` is the "local" database.

This example selects from one remote table:

```
premdb=# select distinct winners from yellowbrick.public.season where winners is not null;
     winners
------------------
 Manchester United
 Blackburn Rovers
 Chelsea
 Manchester City
 Leicester City
 Arsenal
(6 rows)
```

This example is a union of a local table and a remote table.

```
premdb=# select * from yellowbrick.public.hometeam union select * from premdb.public.awayteam
order by htid;
 htid |          name
------+------------------------
   2 | Arsenal
   3 | Aston Villa
   4 | Barnsley
   5 | Birmingham City
...
```

This example creates a local table by joining a local table and a remote table:

```
premdb=# create table home_and_away as
select pph.htid, ypa.atid, pph.name
from premdb.public.hometeam pph join yellowbrick.public.awayteam ypa on pph.name=ypa.name
order by pph.name;
SELECT 47
premdb=# select * from home_and_away;
 htid | atid |          name
------+------+------------------------
   2 |   51 | Arsenal
   3 |   52 | Aston Villa
   4 |   53 | Barnsley
   5 |   54 | Birmingham City
   6 |   55 | Blackburn Rovers
   7 |   56 | Blackpool
...
```

In this example, the user is connected to the `test` database, but the query joins a table in the `yellowbrick` database with a table in the `premdb` database:

```
test=# select *
from premdb.public.awayteam ppa
join yellowbrick.public.match ypm
on ypm.atid = ppa.atid
order by ypm.atid limit 3;
 atid |  name   | seasonid |      matchday       | htid | atid | ftscore | htscore
------+---------+----------+---------------------+------+------+---------+---------
   51 | Arsenal |        6 | 1997-09-27 00:00:00 |   18 |   51 | 2-2     | 0-2
   51 | Arsenal |        3 | 1995-02-25 00:00:00 |   16 |   51 | 0-3     |
   51 | Arsenal |        3 | 1994-12-28 00:00:00 |   21 |   51 | 0-2     |
(3 rows)
```

The following `INSERT` selects from the remote table and inserts into the local table.

```
premdb=# insert into public.awayteam select * from yellowbrick.public.awayteam;
INSERT 0 50
```

The following statements are not allowed because they attempt to write to the remote database:

```
premdb=# insert into yellowbrick.public.awayteam select * from public.awayteam;
ERROR:  cross-database modifications are not allowed: yellowbrick.public.awayteam
premdb=# delete from yellowbrick.public.awayteam;
ERROR:  cross-database modifications are not allowed: yellowbrick.public.awayteam
```

Database Limits

Platforms: All platforms

Parent topic: Databases

This section lists the limits that are applied to database objects in Yellowbrick instances.

Database Limit	Value	Notes
Databases per instance	1,000	The maximum number of databases includes the `yellowbrick` database, so the effective limit is 999 user-created databases.
Tables per instance	100,000	There is no limit on the number of tables per database, only a limit on the total number per data warehouse instance. This limit applies to persistent tables only.
Maximum number of nodes per compute cluster per instance	64	
Size of internal temporary tables	30MB	Temporary tables that the system creates during queries are limited to a size of 30MB. This limitation does not apply to temporary tables created by users.
Columns per table	2000	Storage limit for the maximum number of columns in a single table

Database Limit	Value	Notes

Columns per view or select	8000	The maximum number of columns that can be placed in a row. For example, the output of a `UNION` query is limited to 8000 columns. In practice, the limit applies to the number of columns that have to be propagated up the query plan tree during execution. Therefore, you may find that less than 8000 columns can be selected because of the behavior of the plan. Columns that are *not* in the select list but are needed for its computation (such as `ORDER BY` columns, `GROUP BY` columns, and temporary columns for `CASE` expressions) *do* count towards the limit.
		
		For example, a simple query that selects one column but orders by another column propagates two columns, not one:
		
		`yellowbrick=> explain(stable,verbose) select c1 from t1 order by c2;`
		`QUERY PLAN`
		`----------`
		`workers node`
		`single SELECT`
		`(t1.c1, t1.c2)`
		`distribute single`
		`all DISTRIBUTE SORT`
		`(t1.c1, t1.c2)`
		`distribute single`
		`all SORT ON (t1.c2 ASC NULLS LAST)`
		`(t1.c1, t1.c2)`
		`distribute on (t1.c1)`
		`all SCAN t1`
		`(t1.c1, t1.c2)`
		`distribute on (t1.c1)`
		
		`(14 rows)`

Rows per table	2^{64}	
Maximum width of a row in a column	64231 bytes	A single row can be 231 bytes longer than any single `VARCHAR` column. See Maximum Row Size.
Length of a `VARCHAR` column	64000 bytes	For other data type limits, see SQL Data Types.
Length of database object names	128 bytes	See SQL Identifiers.
Number of distribution columns per table	1	See Distribution Options.
Number of sort columns per table	1	See Sorted and Clustered Tables.
Number of cluster columns per table	4	See Sorted and Clustered Tables.
Number of partition columns per table	4	See Partitioning Options.
Maximum number of `IN` list items	200000	See IN.

Database Limit	Value	Notes
Maximum number of `WHEN` clauses in a `CASE` expression	8192	
Maximum number of ACLs per database object	5000	The maximum number of privileges you can grant to a database object is 5000.
User connection/session limits (`max_user_connections`)	2000	The maximum number of database user connections is 2000. (The related `max_connections` value is 2300, which allows for 300 additional system user connections; `max_user_connections` must be set to a lower value than `max_connections` .) See also Managing Idle Sessions.

Maximum Row Size

When you create a table, the maximum row limit of 64231 bytes is not enforced. However, you will not be able to insert more than 64231 bytes into a table (as an absolute maximum), and depending on the definition of the columns in the table, the actual limit on the stored size of a row may be smaller.

Additional overhead bytes per column are stored as follows:

- If a column (of any type) can be `NULL` : + 1 byte

For example, a `CHAR(10)` column that accepts `NULL` values occupies 11 bytes.

- If the declared length of a `VARCHAR` column <= 30: *declared length* + 2

For example, a `VARCHAR(25)` column always occupies 27 bytes (or 28 if it is nullable).

- `VARCHAR` columns >30: *byte length* + 2 + 8

For example, a `VARCHAR(500)` column occupies its actual length + 10 bytes. If 500 bytes are inserted, its size is 510 bytes (or 511 if it is nullable). If 100 bytes are inserted, its size is 110 bytes (or 111 if it is nullable).

Practical Limits for Numbers of Tables and Partitions

Yellowbrick does not enforce limits on either the number of tables per database or the number of partitions per table. The maximum number of tables is 70,000 *per cluster*, regardless of the number of databases you create. Ideally, the number of partitions created per table should be less than 1,000. (See Partitioning Tables for details about how the actual number of partitions is computed, based on the partitioning columns you define. The enforced maximum is approximately 250,000 partitions.)

Respecting these more practical limits will help to optimize the storage system and improve both query performance and the efficiency of background database operations. In addition to monitoring the number of tables and partitions on your system, try to avoid rapid loading of a large number of small tables with a few rows at a time. When many thousands of small write transactions are committed to the database, fragmentation of the storage media is likely to occur.

Database Objects

Platforms: All platforms

Parent topic: Databases

Yellowbrick supports the following database objects and commands to create and modify them:

- Physical databases
- Schemas
- Tables (persistent and temporary)
- Views
- Roles (roles and users are equivalent objects)
- Prepared statements
- Stored procedures
- Cursors
- Sequences

DML Commands

Yellowbrick supports several DML commands:

- INSERT
- UPDATE
- DELETE
- TRUNCATE

Transactions

Yellowbrick supports the following commands for working with transactions:

- BEGIN/START
- END/COMMIT
- ABORT/ROLLBACK
- SAVEPOINT
- RELEASE SAVEPOINT
- ROLLBACK TO SAVEPOINT

Not Supported

Yellowbrick does not support the following database objects:

- Collations
- Domains

- Indexes
- Tablespaces
- Triggers
- User-defined types

Generating DDL

You can generate DDL for a specific database by using the `ybdumpschema` and `ybdumproles` client tools. To generate DDL for a specific database object, use the DESCRIBE command with the `WITH DDL` or `ONLY DDL` option.

The `ybdumpschema` tool generates DDL for the following objects in the specified database:

- Tables
- Views
- Schemas
- Sequences
- Stored procedures
- Database ACLs

The `ybdumproles` tool generates DDL for database users and roles.

You can run these tools from the directory on the client system where the `ybtools` were installed (`/usr/bin` on Linux platforms). Use the `--help` option to see the command parameters for each tool:

```
$ ybdumpschema --help
Generate the DDL to re-create the objects in a database
Usage:
  ybdumpschema [OPTION]... [DBNAME]
  -d, --dbname=DBNAME      database to re-create
  -f, --file=FILENAME      output file name
  -h, --host=HOSTNAME      database server host
  -O, --owner              object ownership
  -p, --port=PORT          database server port number
  -U, --username=NAME      connect as specified database user
  -v, --verbose            verbose mode
  -w, --no-password        never prompt for password
  -W, --password           force password prompt
  -x, --privileges         object privileges (grant/revoke)
  -V, --version            show version information and exit
  -?, --help               show this help, then exit

$ ybdumproles --help
Generate the DDL to re-create the roles in a database
Usage:
  ybdumproles [OPTION]...
  -d, --dbname=DBNAME      database to re-create
  -f, --file=FILENAME      output file name
  -h, --host=HOSTNAME      database server host
  -p, --port=PORT          database server port number
  -U, --username=NAME      connect as specified database user
  -w, --no-password        never prompt for password
  -W, --password           force password prompt
  -V, --version            show version information and exit
  -?, --help               show this help, then exit
```

ybdumpschema and ybdumproles Options

Most of the options for these two tools are the same; `ybdumpschema` has a few additional options.

-d, --dbname=DBNAME
Specify the name of a single database on the host system. The value of the environment variable `YBDATABASE` is used if you do not specify a database. For `ybdumpschema` this is the name of the database where you want to generate DDL.

-f, --file=FILENAME
Specify a file name or a path to a file where you want to send the DDL output. Make sure you are running as a user with write permissions on that file and location. If you do not specify a file name, the output prints to `stdout`.

-h, --host=HOSTNAME
Specify the name of the host system where the database resides. See the following table.

-O, --owner
Generate DDL for object ownership (`ALTER TABLE...OWNER TO` statements). This option applies to `ybdumpschema` only.

-p, --port=PORT
See the following table.

-U, --username=NAME
Database username. See the following table.

-v, --verbose
Verbose mode, which prints messages to `stderr` as the tool processes various object types and settings.

-w, --no-password
Do not prompt for the database user's password.

-W, --password
Prompt for the database user's password. The value of the environment variable `YBPASSWORD` is read if no password is supplied at the command line. See the following table.

-x, --privileges
Generate DDL for privileges on objects (`GRANT` and `REVOKE` statements). This option applies to `ybdumpschema` only.

-?, --help
Return help text for all of the options.

Database Connection Parameters

Command-Line Options	Environment Variable	Description	Example

`-h` or `--host`	`YBHOST`	Destination server host name. Default: `localhost`	`-h test.ybsystem.io`
			
			`export YBHOST=test.ybsystem.io`

Command-Line Options	Environment Variable	Description	Example

`-p` or `--port`	`YBPORT`	Destination server port number. Default: `5432`	`--port 5433`
			
			`export YBPORT=5433`

`-U` or `--username`	`YBUSER`	Database login username. No default.	`-U bobr`
			
			`export YBUSER=bobr`

`-W` or `--password`	`YBPASSWORD`	Interactive prompt for the database user's password. No default.	`--password`
			
			`export YBPASSWORD=********`

ybdumpschema Examples

Export DDL for a database named `ybdb1` to a file named `ybdb1.ddl`:

```
$ **ybdumpschema -d ybdb1 -f /home/yb100/ybdb1.ddl -U yellowbrick
**Password:
$
```

Send DDL for the `premdb` database to `stdout`:

```
$ **ybdumpschema -d premdb -U bobr**
Password:

SET client_encoding = 'LATIN9';
SET client_min_messages = warning;

SET search_path = public, pg_catalog;

--
-- Name: awayteam; Type: TABLE; Schema: public; Owner: -
--

CREATE TABLE awayteam (
   atid smallint,
   name character varying(30)
)
DISTRIBUTE REPLICATE;
...
```

Generate DDL for object permissions (in addition to all other DDL):

```
$ **ybdumpschema -d premdb -x -U bobr**
...
--
-- Name: public; Type: ACL; Schema: -; Owner: yb100
--

REVOKE ALL ON SCHEMA public FROM PUBLIC;
REVOKE ALL ON SCHEMA public FROM yb100;
GRANT ALL ON SCHEMA public TO yb100;
GRANT ALL ON SCHEMA public TO PUBLIC;


--
-- Name: match; Type: ACL; Schema: public; Owner: yb100
--

REVOKE ALL ON TABLE match FROM PUBLIC;
REVOKE ALL ON TABLE match FROM yb100;
GRANT ALL ON TABLE match TO yb100;
GRANT SELECT ON TABLE match TO bobr;
...
```

In this example, `ybdumpschema` runs against an empty database, `yellowbrick_test`, with `stdout` redirected to `/dev/null`. The only information printed to the terminal is the verbose output, which is sent to `stderr`.

```
$ **ybdumpschema -d yellowbrick\_test -v &gt; /dev/null**
ybdumpschema: reading extensions
ybdumpschema: identifying extension members
ybdumpschema: reading schemas
ybdumpschema: reading user-defined tables
ybdumpschema: reading user-defined functions
ybdumpschema: reading user-defined types
ybdumpschema: reading procedural languages
ybdumpschema: reading user-defined aggregate functions
...
ybdumpschema: saving encoding = LATIN9
ybdumpschema: saving standard_conforming_strings = on
ybdumpschema: saving database definition
ybdumpschema: creating SCHEMA "public"
ybdumpschema: creating COMMENT "SCHEMA public"
ybdumpschema: setting owner and privileges for DATABASE "yellowbrick_test"
ybdumpschema: setting owner and privileges for SCHEMA "public"
ybdumpschema: setting owner and privileges for COMMENT "SCHEMA public"
```

ybdumproles Example

```
$ **ybdumproles --file=dumproles.txt**
$ **more dumproles.txt \| grep bobr**
CREATE ROLE bobr;
ALTER ROLE bobr WITH NOSUPERUSER INHERIT NOCREATEROLE NOCREATEDB LOGIN;

$ ybdumproles -d premdb -h yb102 -U yellowbrick
Password:
--
-- Yellowbrick database cluster dump
--

SET default_transaction_read_only = off;
```

```
SET client_encoding = 'LATIN9';
SET standard_conforming_strings = on;

--
-- Roles
--

CREATE ROLE bobr;
ALTER ROLE bobr WITH NOSUPERUSER INHERIT NOCREATEROLE NOCREATEDB LOGIN PASSWORD 'md50324e3faf22a8ae63c13ce2a97670f45';
...
```

Managing Database Users and Roles

Creating database users and roles, then managing their access to the system, is the main mechanism for enforcing database security. Users and roles are system-wide objects; they do not belong to specific databases. Users are independent accounts with login access, while roles define groups of users that accrue privileges through role membership.

You can create database users and roles by using standard SQL commands, or through Yellowbrick Manager on cloud platforms. After creating users and roles, you can use GRANT and REVOKE commands at different levels of the system to create a fine-grained security model. Objects in the system can be individually secured or secured in the context of a schema, database, or all databases.

You can grant access privileges (ACLs) on various database objects (or on the system itself) to one or more users or roles. Objects in the system can be individually secured or secured in the context of a schema, database, or all databases. For example, you can grant specific privileges so specific users can:

- Run EXPLAIN on a query but not be able to SELECT from the tables in the query
- Use INSERT INTO SELECT on a table without having the privilege to bulk load it (which requires BULK LOAD privilege on the database)
- Back up databases without being able to restore them

You also have flexibility in terms of the definition of roles and users. For example, you can create users with membership in an admin role, but then revoke specific privileges that an admin would normally hold. Conversely, a regular user (non-admin) can be granted certain privileges that would normally belong to an admin. You can create roles that have users (members), then assign privileges to roles. By default, these privileges are inherited by the users that belong to a role.

Cloud Platform Behavior

Note that on cloud platforms, behavior is slightly different, notably:

- No default end-users or additional administrators are created during deployment, but some roles are created. (See Users and Roles.)
- All new users belong to the `consumer` role, which is an empty role (no default privileges).
- Be sure to grant new users and roles usage of at least one compute cluster with `USAGE ON CLUSTER`.
- Individual users may be assigned to a default cluster where their queries always run, see ALTER USER SET DEFAULT CLUSTER for more information.

Useful Links

For more details and examples, see the following SQL commands:

- CREATE ROLE
- GRANT
- REVOKE

In `ybsql`, use the `\du` command to see the current list of database users and roles. You can also query the following system views:

- sys.user
- sys.role

A set of `HAS_*_PRIVILEGE` functions are useful for listing the privileges that have been granted to specific users and roles. See System Functions.

You can create schemas in databases and views on tables as other means of access control. For example, you can create a view on a table that projects only certain columns, then you can give roles access to the view but not the table.

For information about managing users and roles on appliance platforms via an LDAP server, see LDAP Integration.

Managing Idle Sessions

The database proactively closes both idle sessions and idle transactions within sessions when they reach a timeout limit. You can modify the default settings of the following configuration parameters in order to avoid running out of connections (the maximum number of user connections for the cluster is 2000). These parameters may also help you avoid problems where the presence of idle sessions or idle transactions within sessions holds up background maintenance operations. When these operations are blocked, database storage may reach capacity.

- `idle_session_timeout`: how long any user session may remain idle before being disconnected.
- `idle_in_transaction_session_timeout`: how long any transaction within any user session may remain idle before being disconnected.

Note: Running `show idle_session_timeout` or `show idle_in_transaction_session_timeout` as a superuser will always return unlimited (`0`), which is the superuser's setting, not the default setting for the cluster.

Both parameters accept settings with the following units: `ms` , `s` , `min` , `h` , and `d` .

Managing the Row Store

Each data warehouse instance you create contains its own predefined persistent volume that serves as its *row store*. The row store is an intermediate storage system for data that is not loaded directly to the *column store*. By default, tables that are loaded via `INSERT INTO...VALUES` statements and `ybsql \copy` operations write rows to the row store, while tables that are loaded via `ybload` operations and `INSERT INTO...SELECT` statements write rows directly to the column store (object storage accessed by the compute nodes).

The row store also serves as temporary holding area for deleted rows that are periodically "garbage-collected" (or vacuumed) by a background "GC" operation.

Default Flushing Behavior

A background service periodically flushes rows from the row store to the column store. This service scans database tables and queues them for flushing. The service issues the appropriate commands automatically, taking into account the expected number of workers where the flush operation will run. For efficient storage in the column store, data is flushed automatically when a table accrues more than 30MB of data in the row store (or when a single transaction writing to the row store exceeds 30MB of data). *You do not need to flush tables manually.*

To make sure that smaller batches of data do not remain in the row store for too long before being flushed (for example, updates to small dimension tables), a time limit of 6 hours also applies. By default, data does not remain in the row store any longer than 6 hours, regardless of the amount of data ready to be flushed for a table. Both of these limits (30MB and 6 hours) are set globally and apply to all tables.

Depending on your workload, the row store volume may become full or run the risk of becoming full. To mitigate these concerns, you can manage the row store at the table level, as described in the following sections. However, note that most workloads should run efficiently without any changes to the default behavior.

Default Action Taken When the Row Store Becomes Full

When the row store becomes full for a table (or completely full), the default behavior is to insert rows directly into the column store. This behavior ensures that data continues to load; however, performance will not be optimal. When rows start being loaded into the column store instead of the row store, you will see a warning message. For example:

```
premdb=# \copy match from '/home/brumsby/match.csv' with delimiter ',' null '';
WARNING:  file limit 1 reached for oid 16437 performing slow insert
CONTEXT:  COPY match, line 6503: "17,2008-12-13 00:00:00,25,67,0-1,0-0"
COPY 8606
```

Internally, the `\copy` or `INSERT` operation that you were running is translated to a `YCOPY` operation, which is an internally generated plan for sending rows directly to the compute nodes. The threshold for switching from updating the row store to updating the column store is 30MB of data; when a specific `\copy` or `INSERT` transaction exceeds that limit, the remaining rows are loaded via `YCOPY`. (This threshold may be overridden by a row store size limit that you set for a table; see Modifying the Row Store Size for Tables.)

Although load performance is slower when inserts are promoted to `YCOPY` operations, the related flushing and space management behavior is optimized. Therefore, the default behavior is recommended for most workloads.

The row store full action also applies to `UPDATE` statements because an `UPDATE` statement is executed as a `DELETE` followed by an `INSERT`. For example:

```
premdb_match=# update match set seasonid=29;
ERROR:  file limit 1 reached for oid 16489...
```

Independent `DELETE` statements do not respect the action setting; however, if the entire row store drive becomes full, `DELETE` statements may fail.

You cannot explicitly run a `YCOPY` operation, but you can monitor the `YCOPY` queries that the system generates in the context of workload management (WLM), and you can track `YCOPY` activity in the `sys.query` and `sys.log_query` views.

Alternative Actions: Block or Cancel

Two other actions can be taken when the row store becomes full:

- Block the insert until a flush is possible, then resume. The current transaction will wait until sufficient space is available in the row store, then continue.
- Cancel the insert and return an error.

To modify the default behavior and enforce the block or cancel option for specific tables, use an ALTER TABLE command. You cannot modify the behavior at the database or system level. For example, the following command sets the Cancel action (`'C'`) for table `match` :

```
premdb=# alter table match set rowstore_full_action 'C';
ALTER TABLE
```

To see the current action setting for tables, query the sys.table view.

Modifying the Row Store Size for Tables

In addition to setting the action that is taken when the row store becomes full, you can limit the size of the row store for specific tables. In this way, you can mitigate the effects of the following problems:

- Slow performance of queries on the table
- Inability of the system to accept writes because the table occupies too much space in the row store
- Failure of `INSERT` , `UPDATE` , and `DELETE` queries when the row store becomes full

Using an ALTER TABLE command, you can set the size limit to `32MB` or greater, which is the default size of a single file in the row store. For example, setting the size limit to `32GB` translates to a maximum of 1024 32MB files.

Monitoring Row Store Operations

You can use the following methods to monitor the behavior of writes to the row store:

- **Create WLM rules**: For example, you can create a rule that sends a log message or aborts `YCOPY` queries. You may want to manage how many `YCOPY` operations are occurring and not allow too many to run. For example, the following rule condition aborts queries that have the type `ycopy` , then logs a message.

```
if (w.type === 'ycopy') {
    w.abort('YCOPY operation aborted');
}
```

Alternatively, a rule condition can filter on the SQL text. For example:

```
if ((String(w.SQLText).indexOf('ycopy') >= 0)) {
    w.abort('YCOPY operation aborted');
}
```

For more details, see Creating WLM Rules.

- **Query system views**: sys.table, sys.query, and sys.log_query. The `sys.table` view tracks row counts and table-level settings.

Password Policies

To make deployments as secure as possible, we recommend using external authentication. See how enable single sign-on for cloud platforms or how to [set up LDAP] for appliance platforms.

If external authentication techniques cannot be used for certain users or deployments, Yellowbrick supports configurable Password Policies. The feature only applies to local database users with credentials stored in the database. The policies provide several additional controls to make sure that password complexity, expiry and login attempts meet organizational security guidelines.

Enabling Password Policies

Password Policies is a new feature that must be enabled by setting the `enable_password_policy` configuration parameter, after which a full database restart is required. If the configuration parameter is not set, running `ALTER USER` or `ALTER SYSTEM` commands will receive an error such as:

```txt
ERROR:  Password policies are not enabled

HINT:  Set enable_password_policy to true to enable this feature
```

> If you disable `enable_password_policy` (and restart the DB), all password policy checks will be disabled regardless of which options are set for users and system.

Password Policy Rules

There are two types of password policy rules:

- System level
- User level

Password policies for users override those defined for the system.

Viewing password policies

You can view current password policies by querying the system view `sys.password_policy`.

Setting a System-Level Password Policy

Use the following query to set up system-level password policies:

```sql
ALTER SYSTEM SET enable_password_policy = true;

-- Restart the database

ALTER [SYSTEM | USER | ROLE ] ALTER PASSWORD_POLICY WITH (
  <options>
)
```

```
Where <options> can be:
  [ MIN_LENGTH [=] <min length> ]
  [ MIN_LETTERS [=] <min letters> ]
  [ MIN_LOWER_CASE [=] <min lower case> ]
  [ MIN_UPPER_CASE [=] <min upper case> ]
  [ MIN_SYMBOLS [=] <min symbols> ]
  [ MIN_DIGITS [=] <min symbols> ]
  [ MAX_REUSE [=] <max reuse> ]
  [ MAX_ATTEMPTS [=] <max attempts> ]
  [ UNLOCK_AFTER_S [=] <unlock interval s> ]
  [ EXPIRE_USER_AFTER_S [=] <expire user interval s> ]
  [ EXPIRE_PASSWORD_AFTER_S [=] <expire password interval s> ]
```

The parameters are as follows:

`MIN_LENGTH`
: Minimum length of characters the password must have. If 0, passwords of any length are accepted.

`MIN_LETTERS`
: Minimum number of letters in the password. Letters can be uppercase and/or lowercase. If 0, passwords without letters are accepted.

`MIN_LOWER_CASE`
: The amount of lowercase letters required. If 0, passwords with no lowercase letters are accepted.

`MIN_UPPER_CASE`
: The amount of uppercase letters required. If 0, passwords with no uppercase letters are accepted.

`MIN_SYMBOLS`
: Minimum number of symbols (anything that is not a letter or digit). If 0, passwords with no symbols are accepted.

`MIN_DIGITS`
: Minimum number of digits that must be present in the password. If 0, passwords with no digits are accepted.

`MAX_REUSE`
: Maximum number of times the same password can be used by the user. If set to 0, there is no limit. The maximum limit for `MAX_REUSE` is 10.

`MAX_ATTEMPTS`
: Maximum number of unsuccessful login attempts in a row before the account is locked out. If 0, there are no maximum unsuccessful login attempts.

`UNLOCK_AFTER_S`
: The amount of time (in seconds) that must pass until an account is reenabled after exceeding the maximum attempts. If 0, a locked out user will not automatically unlock.

`EXPIRE_USER_AFTER_S`
: Time elapsed (in seconds) since the user last logged in before the admin must unlock the account. This can be shorter than the password expiry. If 0, users do not expire.

`EXPIRE_PASSWORD_AFTER_S`
: An interval (in seconds) formatted duration after which the password expires if it is not changed. If 0, passwords do not expire.

For example, to change the minimum length of a password for all users at the system level:

```sql
ALTER SYSTEM ALTER PASSWORD_POLICY WITH min_length = 10;
```

Setting a User-Level Password Policy

A system-level policy is not required to set user-level (or role-level) policies. Note that `ALTER USER` and `ALTER ROLE` are synonymous below. Administrators can override system-level settings using `ALTER USER` as described in the grammar above. For example:

```sql
-- Specify the minimum password length for a specific user:
ALTER USER "testuser1@yellowbrick.com" ALTER PASSWORD_POLICY WITH min_length = 20;

-- Define the number of failed login attempts for a user before they are locked out:
ALTER USER "testuser1@yellowbrick.com" ALTER PASSWORD_POLICY WITH max_attempts = 2;

-- Allow a user to automatically unlock and log back in after a certain amount of seconds.
ALTER USER "testuser1@yellowbrick.com" ALTER PASSWORD_POLICY WITH unlock_after_s = 180;
```

Unlocking Locked Out Users

A user that is locked out can be unlocked by a superuser by setting the corresponding value to 0:

```sql
ALTER USER "testuser1@yellowbrick.com" ALTER PASSWORD_POLICY WITH max_attempts = 0;
```

The user can now log back in if they exceeded the number of login attempts

Expiring Users After a Time Period

We can configure the database to Expire a user automatically after a certain number of seconds This time interval starts counting from the last login.

```sql
ALTER USER "testuser1@yellowbrick.com" ALTER PASSWORD_POLICY WITH expire_user_after_s = 300;
```

In this case, when this expired user logs out and tries to login again, the following message is received:

```txt
ybsql: FATAL:  role "testuser1@yellowbrick.com" has expired and is not permitted to log in
```

Forcing Password Expiry After a Time Interval

A user's password can be expired after a time interval, for example:

```sql
-- Expire user's password after a certain amount of seconds. This time interval begins from the moment the last password was set.
ALTER USER "testuser1@yellowbrick.com" ALTER PASSWORD_POLICY WITH EXPIRE_PASSWORD_AFTER_S = 300;
```

The time interval is measured from any of these events:

```sql
-- User creation:
CREATE USER "testuser1@yellowbrick.com" WITH PASSWORD "Password1";

-- Last password change:
ALTER USER "testuser1@yellowbrick.com" WITH PASSWORD "Password1";
```

When users log in, they will receive a friendly reminder because password expiry is set:

```txt
WARNING:  the password for role "testuser1@yellowbrick.com" will expire in less than 1 day
```

Once the password has expired, if this user logs out and attempts to log in again, they will receive the following message:

```txt
ybsql: FATAL:  the password for role "testuser1@yellowbrick.com" has expired. Please contact the administrator to unlock
```

Note: The expiration of the user account or the password can only be determined during authentication.

Password Criteria Error Messages

The section below shows a few examples of error messages that are displayed when a password does not meet the specified policy criteria.

Minimum length password criteria is not met

```
-- Specify minimum password length for all users at system level
ALTER SYSTEM ALTER PASSWORD_POLICY WITH min_length = 11;

-- Set the password with length 2
ALTER USER "testuser1@yellowbrick.com" WITH PASSWORD '89';
```

```txt
ERROR:  password is too short. Minimum length is 11
```

Minimum letters password criteria is not met

```
-- Specify minimum password letters for all users at system level:
ALTER SYSTEM ALTER PASSWORD_POLICY WITH min_letters = 6;

-- Set the password with minimum length 11 and 3 letters
ALTER USER "testuser1@yellowbrick.com" WITH PASSWORD 'new_1234567';
```

```txt
ERROR:  password does not contain enough letters. Minimum letters is 6
```

Maximum login attempts exceeded

```
-- Set the maximum login attempts for a user to 2
ALTER USER "testuser1@yellowbrick.com" ALTER PASSWORD_POLICY WITH max_attempts = 2;
```

The user enters an incorrect password two times. When they try to log in with the correct password for the third time, they receive the following message:

```txt
ybsql: FATAL:  role "testuser1@yellowbrick.com" has been locked out due to too many failed login attempts
```

Allow a user to automatically unlock and log back in after 180 seconds. Please note that this interval starts from the time of a successful login.

```
ALTER USER "testuser1@yellowbrick.com" ALTER PASSWORD_POLICY WITH unlock_after_s = 180;
```

Related Topics

See user logging at `sys.user`. See the system view `sys.password_policy`. And `enable_password_policy` for enabling the feature.

Storage Engine Filter Expressions

Storage Engine (SE) filter expressions are an advanced mechanism to reduce the amount of data read from disk when executing queries. These textual expressions are automatically generated upon query submission and are consumed by the SE during query execution.

The deeper a filter can be pushed into the query execution tree, the better. This minimizes the data read from disk, reducing the work and resources required to execute the query.

Consider the following query:

```sql
SELECT x FROM result_table WHERE x = 3;
```

The corresponding simplified query tree:

```
SELECT x
  |
FILTER (x = 3)
  |
SCAN result_table
```

Without scan-level filtering in the storage engine, every row in `result_table` would be scanned and loaded into memory. For example, if `result_table` contains 100GB of data but only a few rows satisfy `x = 3`, most of the scanning and memory usage would be wasted, leading to unnecessarily long execution times.

Rows are written to disk in "groups" called shards, which contain metadata such as the minimum and maximum values for each column. This metadata enables skipping entire shards that cannot satisfy the query conditions. For example, if all but one shard has a minimum value of 10 for column `x`, those shards can be safely skipped when querying for `x = 3`.

SE filter expressions provide more advanced scan-level filtering than previous mechanisms like bloom filters and scan restrictions. By reducing the data read from disk, these filters improve query execution time and reduce resource consumption.

Note: This document focuses on SE filter expressions and should not be confused with bloom filters or scan restrictions.

Basic Usage

The generation of SE filter expressions is controlled by the `enable_se_filter_expression_generation` configuration parameter, which accepts a boolean value. To enable filter generation, set this parameter to true.

```sql
SET enable_se_filter_expression_generation TO true;

CREATE TABLE result_table (x INT, y INT);

EXPLAIN (se_filters) SELECT x FROM result_table WHERE x = 3;

                                QUERY PLAN
------------------------------------------------------------------------------
 id   rows_planned   workers   node
  5             1       all   SELECT
```

```
  2             1       all   SCAN result_table
                                (result_table.x = $0) AND scan_constraints: (result_table.x = $0)
                                SE Filter: (c1 = i3);
```

In the output above, the SE Filter line under the SCAN node shows the generated SE filter expression. The expression filters on column `x` (the first column of the table, indicated by `c1`) for rows where `x` equals the integer value 3.

SE filter expressions are derived from a query's JOIN and WHERE clauses. The following section describes the kinds of clauses that the planner will attempt to turn into an SE filter expressions.

Supported WHERE Clauses

The following types of WHERE clauses can generate SE filter expressions:

1. Simple Comparisons

   ```
   <column> <op> <constant>
   ```

   Supported operators: >, >=, <, <=, =, !=

   Columns and constants cannot be NUMERIC/DECIMAL

2. NULL Tests

```sql
SELECT * FROM result_table WHERE x IS NULL;
```

3. Constant IN Lists

```sql
SELECT * FROM result_table WHERE x IN (1, 2, 3);
```

4. Subquery IN Lists

```sql
SELECT * FROM result_table WHERE x IN (SELECT y FROM bar);
```

The table in the subquery must be replicated.

5. AND/OR Combinations

```sql
SELECT * FROM result_table WHERE x = 3 OR y = 5;
SELECT * FROM result_table WHERE x = 3 OR y in (1, 2, 3);`
```

If a clause doesn't meet these conditions, an `UNK` term will be generated for the predicate column, which is ignored during filtering.

```sql
EXPLAIN (se_filters) SELECT * FROM result_table WHERE x = 3 OR (y + 1 > 5);

                            QUERY PLAN
------------------------------------------------------------------------
 id   rows_planned   workers   node
  5             1       all   SELECT
  2             1       all   SCAN result_table
                                ((result_table.x = $0) OR ((result_table.y + $1) > $2))
                                SE Filter: ((c1 = i3) OR UNK);
```

Inlist Queries

An inlist query filters a column based on a set of values. The following examples demonstrate two equivalent queries:

```sql
SELECT * FROM result_table WHERE x = 1 OR x = 2 OR x = 3;

SELECT * FROM result_table WHERE x IN (1, 2, 3);
```

We refer to such queries as inlist queries, regardless of whether the IN operator is explicitly used, as both check if `x` belongs to a set of values.

When executing inlist queries, the generated SE filter expression depends on:

1. The number of values in the set.
2. The value of the `inlist_threshold` configuration parameter.

Let us analyze the EXPLAIN output for a query with different `inlist_threshold` settings:

inlist_threshold Equal to the Set Size

```sql
SET inlist_threshold TO 3;

EXPLAIN (se_filters) SELECT * FROM result_table WHERE x IN (1, 2, 3);

                                QUERY PLAN
--------------------------------------------------------------------------------
 id   rows_planned   workers   node
  5              1       all   SELECT
  2              1       all   SCAN result_table
                                 ((result_table.x = $2) OR ((result_table.x = $0) OR (result_table.x = $1)))
                                 SE Filter: ((c1 = i1) OR (c1 = i2) OR (c1 = i3));
```

inlist_threshold Less Than the Set Size

```sql
SET inlist_threshold TO 2;

EXPLAIN (se_filters) SELECT * FROM result_table WHERE x IN (1, 2, 3);

                                    QUERY PLAN
-----------------------------------------------------------------------------------
 id   rows_planned   workers   node
 10             1       all   SELECT
  1             1       all   SEMI LEFT INNER HASH JOIN ON (#inlist_100#.val = result_table.x)
  4             1       all   |-SCAN result_table
                              |   result_table.x = bloom(7) AND scan_constraints: min_max(result_table.x)
                              |   SE Filter: (c1 in (build7));
  7             3       all   |-BUILD
  9             3       all     SCAN TEMP 3
```

When the number of values in the set exceeds the `inlist_threshold`, the planner will attempt to rewrite the query to use a SEMI JOIN, which improves performance for larger sets. For example, the SE filter expression `c1 in (build7)` says that the first column in `result_table`, in this case `x`, must match the values in the build node with ID 7, which references a temporary table containing the set values.

Using Subqueries in Inlist Queries

SE filter expressions can also be generated when a subquery provides the values to filter on. However, the following conditions must be met:

- The table used in the subquery must be replicated. For details on how to create a replicated table click here.
- The subquery must return only one column. Multi-column inlists (example shown below) will not generate SE filter expressions:

```sql
SELECT * FROM result_table WHERE (x, y) IN (SELECT x, y FROM bar);
```

The following is an example where an SE filter expression is generated when filtering `x` on values in another table.:

```sql
CREATE TABLE bar(y int) DISTRIBUTE REPLICATE;

EXPLAIN (se_filters) SELECT * FROM result_table WHERE x IN (SELECT y FROM bar WHERE y > 2);
                                    QUERY PLAN
-------------------------------------------------------------------------------------------
 id   rows_planned   workers   node
 13             1       all   SELECT
  1             1       all   SEMI LEFT INNER HASH JOIN ON (bar.y = result_table.x)
  4             1       all   |-SCAN result_table
                              |   result_table.x = bloom(7) AND scan_constraints: min_max(result_table.x)
                              |   SE Filter: (c1 in (build7));
  7             1       all   |-BUILD
 10             1       all     SCAN bar
                                  (bar.y > $0) AND scan_constraints: (bar.y > $0)
                                  SE Filter: (c1 > i2);
```

Here, the SE filter expression for `result_table` references a build node derived from a scan on `bar`. Additionally, `bar` has its own SE filter expression, specifying that column 1 (`c1`) of `bar` must be greater than 2.

Data Ordering

The order in which data is written to disk plays a significant role in the observed performance of SE filter expressions. Shards, the "groups" in which rows are written to disk, are the key unit to consider here. As mentioned earlier, the metadata on shards tracks information such as the minimum and maximum values for a column across the rows in that shard. For SE filter expressions to produce large speedups (by skipping as much data as possible), the range of values for a predicate column in a shard should be as narrow as possible.

Shards are immutable and do not change once written to disk - even when `UPDATE` or `DELETE` commands are issued. In both cases, the records affected by the query are marked as deleted. In the case of an `UPDATE`, the modified record is appended as a new row in another shard. Over time, these operations can cause the data of interest to be spread across more shards, reducing the performance benefits SE filter expressions can provide.

When a workload is well understood, it may be advisable to periodically resort the data based on the columns of interest.

Top-Level OR Conditions

SE filter expressions apply only to the first top-level OR condition in a query. This restriction ensures the filter application remains computationally efficient.

Examples:

Single Top-Level OR Condition

```sql
WHERE (col1 = const OR col2 = const OR col3 = const OR col1 = otherConst);
```

There is only one top-level OR here. All terms in the OR condition are included in the SE filter expression.

Nested OR and AND Conditions

```sql
WHERE (col1 = const AND col2 = const AND (col3 = const1 OR col3 = const2 OR col4 = const));
```

Still only one top-level OR is present and all terms in the resulting SE filter are applied.

Multiple Top-Level OR Conditions

```sql
WHERE (col1 = const) AND (col2 = const OR col3 = const) AND (col4 = const OR col5 = const);
```

SE filter expressions apply to columns `col1`, `col2`, and `col3`. Filters on `col4` and `col5` are processed later in the execution engine.

User Audit Log

The database can optionally retain detailed logging for key operations pertaining to user authentication attempts as well as changes to roles, users and grants. This detailed log, called the *User Audit Log*, is persisted outside the database itself in a tamper-proof fashion and can be retrieved even when the database is not running.

Since this feature is designed for auditing, any auditable events that take place in a transaction will be logged even if the transaction is rolled back or the operation fails due to lack of permission. Normal database users can't view or retrieve the audit log; it must be retrieved by copying a file from out of the underlying storage infrastructure. See instructions below on how to do so. This allows the retrieval of the audit log even if the database is offline.

Enabling the User Audit Log

User Audit Log is a new feature that must be enabled by setting the `enable_user_audit` configuration parameter, after which a full database restart is required. If the configuration parameter is not set, the other settings described here will have no effect.

Regular vs. Extended Logging

The User Audit Log can record basic operations and extended operations. Basic operations are normally enough to meet auditability requirements, but some highly regulated deployments have further requirements and can use extended logging. Note that extended logging will slightly reduce the performance of login operations and consume a little more storage space. Extended operations are enabled by enabling the configuration parameter `enable_user_audit_extended`.

The differences between basic and extended logging are summarised in the table below. The *Other user/s affected* column denotes whether an affected user is logged in the respective column also.

Type	Basic/Extended	Other user/s affected
LOGIN_FAIL	Basic	
LOCKED_OUT	Basic	
PASSWORD_CHANGE	Basic	✍
PASSWORD_POLICY_CHANGE	Basic	✍ (empty for SYSTEM)
CREATE_USER	Basic	✍
ALTER_USER	Basic	✍ (empty for ALL)
DROP_USER	Basic	✍
ALTER_SYSTEM	Basic	
GRANT_ATTEMPT	Basic	✍
GRANT_FAIL	Basic	✍
GRANT_SUCCESS	Basic	✍
REVOKE_ATTEMPT	Basic	✍

Type	Basic/Extended	Other user/s affected
REVOKE_FAIL	Basic	✍
REVOKE_SUCCESS	Basic	✍
ALTER_DEFAULT_PRIVILEGES_ATTEMPT	Basic	
ALTER_DEFAULT_PRIVILEGES_FAIL	Basic	
ALTER_DEFAULT_PRIVILEGES_SUCCESS	Basic	
LOGIN_SUCCESS	Extended	
LOGOUT_SUCCESS	Extended	
LOGOUT_TIMEOUT	Extended	
LOGOUT_KILL	Extended	✍
SET_ROLE	Extended	✍ (user being set to)

File Format

The user audit logs are stored in a CSV (comma separated value) format for easy inspection and import into external tools. No header row is present. The order of the columns is as follows:

1. Timestamp
2. Name of the event
3. Name of the user performing the action
4. Name of the user affected by the action (if applicable)
5. IP address of remote end of the connection, or "[local]" if done locally
6. Session Id for the connection that performed the action (0, if not yet available)
7. Additional details (empty, showing options or SQL)

Log File Storage, Rotation and Retrieval

The user audit log files are stored in the following directories:

- Appliance deployments: `/mnt/ybdata/ybd/postgresql/build/db/log/`
- Cloud deployments: `/mnt/ybdata/log/`

Log files are stored and rotated like other system logs: The active log file is called `user_audit_log.csv` and those compressed and rotated are named `user_audit_log.csv.N.gz` where N is numeric.

Two configuration variables control the rotation of the log files. `user_audit_max_bytes_per_file` controls how many bytes can accumulate in the active, uncompressed log file before it's compressed and archived. `user_audit_max_files` controls the maximum number of rotated files that are retained.

On appliance platforms, Log files are accessed and retrieved on the manager node using SSH tools. On cloud platforms, `kubectl` access to the `ybinst-pg` pod is used instead. Examples are as follows:

Use case	Cloud platforms	Appliance platforms

Use case	Cloud platforms	Appliance platforms
Get last 10 audited log events	`kubectl exec -n testuser-dev-ns -it ybinst-testuser-inst-0 -c ybinst-pg -- tail -n 10 /mnt/ybdata/log/user_audit_log.csv`	`sudo tail -n 10 /mnt/ybdata/ybd/postgresql/build/db/log/user_audit_log.csv`
Get all audited log events	`kubectl exec -n testuser-dev-ns -it ybinst-testuser-inst-0 -c ybinst-pg -- cat /mnt/ybdata/log/user_audit_log.csv`	`sudo cat /mnt/ybdata/ybd/postgresql/build/db/log/user_audit_log.csv`
Copy file from a path to local	`kubectl cp testuser-dev-ns/ybinst-testuser-inst-0:/mnt/ybdata/log/user_audit_log.csv -c ybinst-pg ./user_audit_log.csv`	`sudo scp /mnt/ybdata/ybd/postgresql/build/db/log/user_audit_log.csv username@dest:path`

Example Content

Examples of data stored in the CSV file are as follows (with commas replaced with tabs for readability):

```txt
2024-08-06T21:10:59.486751  LOGIN_FAIL  testuser1   testuser2   10.10.80.2  0
2024-08-06T21:10:59.508518  LOCKED_OUT  testuser1   testuser2   10.10.80.2  18532
2024-08-06T21:10:59.544257  PASSWORD_CHANGE testuser1   testuser2   10.10.80.2  18534
2024-08-06T21:10:59.526155  PASSWORD_POLICY_CHANGE  testuser1   testuser2   10.10.80.2  18533   ALTER SYSTEM ALTER PASSWORD_POLICY
2024-08-06T21:11:53.820660  CREATE_USER testuser1   testuser2   10.10.80.2  18574   options: canlogin=1, superuser=1
2024-08-06T21:12:01.165818  ALTER_USER  testuser1   testuser2   10.10.80.2  18590   options: createdb=0
2024-08-06T21:12:03.781664  DROP_USER   testuser1   testuser2   10.10.80.2  18595
2024-08-06T21:11:02.659091  ALTER_SYSTEM    testuser1   testuser2   10.10.80.2  18540   ALTER SYSTEM SET user_audit_max_bytes_per_
2024-08-06T21:11:04.234346  GRANT_ATTEMPT   testuser1   testuser2   10.10.80.2  18544   glinda_test_utf8: GRANT ALL PRIVILEGES ON
2024-08-06T21:11:04.241732  GRANT_FAIL  testuser1   testuser2   10.10.80.2  18544   glinda_test_utf8: GRANT ALL PRIVILEGES ON TABL
2024-08-06T21:11:04.235329  GRANT_SUCCESS   testuser1   testuser2   10.10.80.2  18544   glinda_test_utf8: GRANT ALL PRIVILEGES ON
2024-08-06T21:11:04.244128  REVOKE_ATTEMPT  testuser1   testuser2   10.10.80.2  18544   glinda_test_utf8: REVOKE ALL PRIVILEGES ON
2024-08-06T21:11:04.244573  REVOKE_FAIL testuser1   testuser2   10.10.80.2  18544   glinda_test_utf8: REVOKE ALL PRIVILEGES ON TAB
2024-08-06T21:11:04.764052  REVOKE_SUCCESS  testuser1   testuser2   10.10.80.2  18545   glinda_test_utf8: REVOKE sysadmin FROM "te
2024-08-06T21:11:05.297111  ALTER_DEFAULT_PRIVILEGES_ATTEMPT    testuser1   testuser2   10.10.80.2  18546   glinda_test_utf8: ALTE
2024-08-06T21:11:05.297741  ALTER_DEFAULT_PRIVILEGES_FAIL   testuser1   testuser2   10.10.80.2  18546   glinda_test_utf8: ALTER DE
2024-08-06T21:12:05.894674  ALTER_DEFAULT_PRIVILEGES_SUCCESS    testuser1   testuser2   10.10.80.2  18600   glinda_test_utf8: ALTE
2024-08-06T21:11:13.761428  LOGIN_SUCCESS   testuser1   testuser2   [local] 0
2024-08-06T21:11:13.823209  LOGOUT_SUCCESS  testuser1   testuser2   [local] 18549
2024-08-06T21:11:54.907913  LOGOUT_TIMEOUT  testuser1   testuser2   10.10.80.2  18576   idle_session_timeout exceeded
2024-08-06T21:11:58.501585  LOGOUT_KILL testuser1   testuser2   10.10.80.2  18580   PID:178698
2024-08-06T21:11:51.157413  SET_ROLE    testuser1   testuser2   [local] 18562
```

Installing Airbyte

Yellowbrick Documentation > How-to Guides > Install Airbyte

Platforms: EE: AWS cloud

Parent topic: How-to Guides

This guide details the steps required to install Airbyte in the same AWS EKS cluster that hosts Yellowbrick. While this guide is aimed at AWS, the same pattern can be followed with Azure and GCP, substituting the cloud provider-specific commands with suitable alternatives.

You will need the following packages installed locally:

- Helm command line client — `helm`
- AWS command line client — `aws`
- AWS EKS command line client — `eksctl`
- Kubernetes command line client — `kubectl`
- Command for substituting environment variables — `envsubst`. This isn't strictly necessary, but it helps for configuring the install for different environments.

You will need to be logged into the AWS account hosting Yellowbrick to be able to complete this guide. You will also need access to the Yellowbrick EKS cluster and its associated Kube config file. You can obtain this by running:

```sh
aws eks update-kubeconfig --region REGION --name CLUSTER
```

Where `REGION` is the AWS region the EKS cluster was created in, and `CLUSTER` is the name of the EKS cluster running Yellowbrick.

The process of installing Airbyte into the Yellowbrick EKS cluster involves:

- Creating a new EKS node group and scaling up a single m5.2xlarge EC2 node in the group
- Creating an RDS instance to store Airbyte state
- Creating an IAM role for Airbyte and assigning an S3 access policy to it. Airbyte will use the S3 bucket to store log files
- Deploying the Airbyte application on EKS using Helm

Step 1: Configuring the Environment

Create a file `airbyte-env.sh` and paste in the following:

```sh
#!/bin/bash

export CLUSTER_NAME="<CLUSTER NAME>"
export AIRBYTE_BUCKET_NAME="<AIRBYTE S3 BUCKET>"
export RDS_USER="airbyte"
export RDS_PASSWORD="<RDS PASSWORD>"
export NODE_TYPE="m5.2xlarge"
export NODEGROUP_NAME="yb-airbyte-nodegroup"
export REGION="<AWS REGION>"
export NAMESPACE="airbyte"
export RELEASE_NAME="airbyte"

export ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)

export VPC_ID=$(aws eks describe-cluster \
```

```
  --name "$CLUSTER_NAME" \
  --region "$REGION" \
  --query "cluster.resourcesVpcConfig.vpcId" \
  --output text)

export SUBNET_IDS=$(aws eks describe-cluster \
  --name "$CLUSTER_NAME" \
  --region "$REGION" \
  --query "cluster.resourcesVpcConfig.subnetIds" \
  --output text)

export SUBNET1=$(echo "$SUBNET_IDS" | awk '{print $1}')
export SUBNET2=$(echo "$SUBNET_IDS" | awk '{print $2}')

export AZ1=$(aws ec2 describe-subnets \
  --subnet-ids "$SUBNET1" \
  --query "Subnets[0].AvailabilityZone" \
  --output text)

export AZ2=$(aws ec2 describe-subnets \
  --subnet-ids "$SUBNET2" \
  --query "Subnets[0].AvailabilityZone" \
  --output text)

export SECURITY_GROUP=$(aws ec2 describe-security-groups \
  --region "$REGION" \
  --filters "Name=vpc-id,Values=$VPC_ID" "Name=group-name,Values=eks-cluster-sg-$CLUSTER_NAME*" \
  --query "SecurityGroups[0].GroupId" \
  --output text)

export OIDC_PROVIDER=$(aws eks describe-cluster \
  --name "$CLUSTER_NAME" \
  --region "$REGION" \
  --query "cluster.identity.oidc.issuer" \
  --output text | sed 's/^https:\/\///')
```

Modify the variables inside the `<>` placeholders and save the file. `CLUSTER_NAME` should be the name of the Kubernetes cluster that Yellowbrick is installed on. `AIRBYTE_BUCKET_NAME` is the name of the S3 bucket Airbyte will use to store logs. The installation steps below expect this bucket to exist, so create the bucket now if it doesn't exist. Also, set the password for the RDS instance that Airbyte will use to store state: `RDS PASSWORD`. These environment variables are used in the steps following.

Set up the environment by executing:

```sh
source ./airbyte-env.sh
```

Step 2: Create a New Node Group

Airbyte should not run on any of the EC2 nodes in the EKS cluster that run Yellowbrick pods. Instead, create a new node group in the EKS cluster for Airbyte's use. Create a YAML file `nodegroup.yaml.template` with the contents:

```yaml
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: ${CLUSTER_NAME}
  region: ${REGION}

vpc:
  id: ${VPC_ID}
  subnets:
    private:
      ${AZ1}:
```

```
        id: ${SUBNET1}
      ${AZ2}:
        id: ${SUBNET2}
  securityGroup: ${SECURITY_GROUP}

managedNodeGroups:
  - name: ${NODEGROUP_NAME}
    instanceType: ${NODE_TYPE}
    desiredCapacity: 0
    minSize: 0
    maxSize: 1
    labels:
      workload: "airbyte"
      cluster.yellowbrick.io/owned: "true"
    tags:
      k8s.io/cluster-autoscaler/enabled: "true"
      k8s.io/cluster-autoscaler/${CLUSTER_NAME}: "owned"

      # Important node-template tags for scale-from-zero
      k8s.io/cluster-autoscaler/node-template/label/workload: "airbyte"
    amiFamily: AmazonLinux2
    privateNetworking: true
```

The node group will be set up to launch a maximum of one `m5.2xlarge` EC2 nodes to host Airbyte. Adjust the node type and number of nodes depending on your requirements.

Create and scale the node group by executing the commands:

```sh
envsubst < nodegroup.yaml.template > nodegroup.yaml

eksctl create nodegroup -f nodegroup.yaml

eksctl scale nodegroup --cluster $CLUSTER_NAME --region $REGION --name $NODEGROUP_NAME --nodes 1
```

Step 3: Create RDS instance

To create the AWS RDS instance needed to store Airbyte state, first create an RDS subnet group. You will use the two subnets that already exist in the AWS EKS cluster that hosts Yellowbrick to form the group. Run the following:

```sh
aws rds create-db-subnet-group \
  --db-subnet-group-name airbyte-subnet-group \
  --db-subnet-group-description "Subnet group for Airbyte RDS" \
  --subnet-ids "$SUBNET1" "$SUBNET2" \
  --tags Key=Project,Value=airbyte \
  --region "$REGION"
```

Next, create the RDS instance and wait for it to be created:

```sh
aws rds create-db-instance \
  --db-instance-identifier airbyte-db \
  --db-instance-class db.t3.micro \
  --engine postgres \
  --allocated-storage 20 \
  --db-name airbyte \
  --master-username "$RDS_USER" \
  --master-user-password "$RDS_PASSWORD" \
  --vpc-security-group-ids "$SECURITY_GROUP" \
  --db-subnet-group-name airbyte-subnet-group \
  --backup-retention-period 7 \
  --no-publicly-accessible \
  --region "$REGION" \
```

```
  --no-cli-pager

  aws rds wait db-instance-available \
  --db-instance-identifier airbyte-db \
  --region "$REGION"
```

Capture the RDS host name in an environment variable for use in later steps:

```sh
export RDS_HOST=$(aws rds describe-db-instances \
  --db-instance-identifier airbyte-db \
  --query "DBInstances[0].Endpoint.Address" \
  --output text)
```

Step 4: Configure S3 Access for Airbyte

Create an IAM role and access policies that allow Airbyte to read and write from the S3 bucket specified earlier. Create the AWS policy file `airbyte-s3-policy.json.template`:

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowAirbyteS3",
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:GetObject",
        "s3:ListBucket",
        "s3:DeleteObject"
      ],
      "Resource": [
        "arn:aws:s3:::$AIRBYTE_BUCKET_NAME",
        "arn:aws:s3:::$AIRBYTE_BUCKET_NAME/*"
      ]
    }
  ]
}
```

Create and apply the policy:

```sh
envsubst < ./airbyte-s3-policy.json.template > /tmp/airbyte-s3-policy.json

aws iam create-policy \
  --policy-name AirbyteS3Access \
  --policy-document file:///tmp/airbyte-s3-policy.json \
  --region "$REGION"

export AIRBYTE_POLICY_ARN=$(aws iam list-policies --query "Policies[?PolicyName=='AirbyteS3Access'].Arn" --output text)
```

Create an IAM role for Airbyte, and attach the S3 policy by saving the following into a file `trust-policy.json.template`:

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::${ACCOUNT_ID}:oidc-provider/$OIDC_PROVIDER"
```

```
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "${OIDC_PROVIDER}:sub": "system:serviceaccount:${NAMESPACE}:airbyte-admin"
        }
      }
    }
  ]
}
```

This will allow the Airbyte pods (via an Airbyte service account) to assume the associated IAM role, by verifying their identity through the EKS cluster's OIDC identity provider.

Create the role by executing:

```sh
envsubst < ./trust-policy.json.template > /tmp/trust-policy.json

aws iam create-role \
  --role-name AirbyteAdminS3AccessRole \
  --assume-role-policy-document file:///tmp/trust-policy.json
```

Attach the S3 access policy to the role:

```sh
aws iam attach-role-policy \
  --role-name AirbyteAdminS3AccessRole \
  --policy-arn "$AIRBYTE_POLICY_ARN"
```

Step 5: Install Airbyte using Helm

Create the Kubernetes namespace in which Airbyte will be deployed. Save the following into a file called `secrets.yaml.template`:

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: airbyte-config-secrets
type: Opaque
stringData:
  database-password: ${RDS_PASSWORD}
```

Create the namespace and save the RDS password secret using:

```sh
kubectl create namespace "$NAMESPACE"

envsubst < secrets.yaml.template > secrets.yaml

kubectl apply -f secrets.yaml -n "$NAMESPACE"
```

Create the Airbyte service account and associate it with the IAM role created earlier:

```sh
kubectl create serviceaccount airbyte-admin -n "$NAMESPACE"

kubectl annotate serviceaccount airbyte-admin \
  -n "$NAMESPACE" \
  eks.amazonaws.com/role-arn=arn:aws:iam::${ACCOUNT_ID}:role/AirbyteAdminS3AccessRole \
  --overwrite
```

Create the Kubernetes role `airbyte-admin-role` and a role binding, which ties together the role with the service account `airbyte-admin` within the Airbyte namespace.

Create a file `airbyte-role.yaml.template` and insert the following YAML:

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: airbyte-admin-role
  namespace: $NAMESPACE
rules:
  - apiGroups: ["*"]
    resources: ["jobs", "pods", "pods/log", "pods/exec", "pods/attach", "secrets"]
    verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: airbyte-admin-rolebinding
  namespace: $NAMESPACE
subjects:
  - kind: ServiceAccount
    name: airbyte-admin
    namespace: $NAMESPACE
roleRef:
  kind: Role
  name: airbyte-admin-role
  apiGroup: rbac.authorization.k8s.io
```

Run the following to create the role:

```sh
envsubst < "./airbyte-role.yaml.template" | kubectl apply -f -
```

Finally, install Airbyte by first creating a `values.yaml.template` file with the content:

```yaml
serviceAccount:
  create: false
  name: airbyte-admin

postgresql:
  enabled: false

global:
  database:
    type: external
    secretName: "airbyte-config-secrets"
    host: ${RDS_HOST}
    port: 5432
    database: "airbyte"
    user: ${RDS_USER}
    passwordSecretKey: "database-password"

  storage:
    type: "S3"
    secretName: "airbyte-config-secrets"
    bucket:
      log: ${AIRBYTE_BUCKET_NAME}
      state: ${AIRBYTE_BUCKET_NAME}
      workloadOutput: ${AIRBYTE_BUCKET_NAME}
    s3:
      region: ${REGION}
      authenticationType: instanceProfile
```

Then execute the following commands:

```sh
helm repo add airbyte https://airbytehq.github.io/helm-charts

helm repo update

envsubst < ./values.yaml.template > ./values.yaml

helm install airbyte airbyte/airbyte --namespace "$NAMESPACE" --values ./values.yaml
```

After several minutes, the Airbyte service will be available. To access the Airbyte application, set up port forwarding to the service:

```sh
kubectl -n "$NAMESPACE" port-forward deployment/airbyte-webapp 8080:8080
```

Access the Airbyte console using the url:

```
http://localhost:8080/
```

Step 6: Uninstalling Airbyte and Supporting Resources

Source the environment file:

```sh
source ./airbyte-env.sh
```

Uninstall Airbyte, the namespace and the Kubernetes service account and role:

```sh
helm uninstall "$RELEASE_NAME" -n "$NAMESPACE"

kubectl delete rolebinding airbyte-admin-rolebinding -n "$NAMESPACE" --ignore-not-found

kubectl delete role airbyte-admin-role -n "$NAMESPACE" --ignore-not-found

kubectl delete serviceaccount airbyte-admin -n "$NAMESPACE" --ignore-not-found

kubectl delete namespace "$NAMESPACE"
```

Detach and delete the AWS IAM policies and role:

```sh
if aws iam get-role --role-name AirbyteAdminS3AccessRole >/dev/null 2>&1; then
  ATTACHED_POLICIES=$(aws iam list-attached-role-policies \
    --role-name AirbyteAdminS3AccessRole \
    --query "AttachedPolicies[].PolicyArn" \
    --output text)
  for POLICY_ARN in $ATTACHED_POLICIES; do
    aws iam detach-role-policy --role-name AirbyteAdminS3AccessRole --policy-arn "$POLICY_ARN"
  done
fi

aws iam delete-role \
  --role-name AirbyteAdminS3AccessRole

aws iam delete-policy \
  --policy-arn "arn:aws:iam::$(aws sts get-caller-identity --query Account --output text):policy/AirbyteS3Access"
```

Delete the RDS instance and subnet group:

```sh
aws rds delete-db-instance \
  --db-instance-identifier airbyte-db \
  --skip-final-snapshot \
  --region "$REGION" \
  --no-cli-pager

aws rds wait db-instance-deleted \
  --db-instance-identifier airbyte-db \
  --region "$REGION"

aws rds delete-db-subnet-group \
  --db-subnet-group-name airbyte-subnet-group \
  --region "$REGION"
```

Finally, delete the EKS node group:

```sh
eksctl delete nodegroup \
  --cluster "$CLUSTER_NAME" \
  --name "$NODEGROUP_NAME" \
  --region "$REGION"
```

Tutorials

Platforms: All platforms

Parent topic: Yellowbrick Documentation

In this section:

If you are new to Yellowbrick, read these documents to start building your applications and integrations.

Walkthrough Tour

This walkthrough guides you through obtaining a trial license, creating compute clusters, loading data, running SQL statements, and connecting to a third-party tool. It assumes that you have completed a public installation of Yellowbrick.

Overview

The walkthrough includes the following steps:

1. Request and install a trial license.
2. Create the following two compute clusters:
   a. One for data loading
   b. One for querying
3. Load sample data.
4. Create a role and users.
5. Run SQL statements.
6. Connect to DBeaver.

---

1. Request and Install a Trial License

1. Log in to Yellowbrick Manager. A banner indicating an unlicensed installation is displayed at the top of the page.
2. Click the banner link or navigate to **Instance Management** → **Instance Management** → **License**.

> **Note: Instance Management** is accessible by using the cube icon in the left pane.

3. Apply for a trial license. You will receive the license key through an email within a few minutes (check your Junk or Spam folder if necessary).
4. Click **Install a License**. Copy the license key from the email and paste in to the text field.
5. Upon installation, you will receive a 30-day trial license supporting up to 128 vCPUs.

---

2. Create Two Compute Clusters

Yellowbrick Manager supports managing multiple compute clusters. Initially, only the instance you named during deployment is displayed.

Perform the following steps:

a. Create the `small-query` Cluster:

1. Navigate to **Instance Management** → **Compute Clusters**. An empty grid is displayed.
2. Click **Cluster**.
3. Create a cluster named "small-query" with one small node (latest version). Default settings include the following:
   - Auto-suspend after 5 minutes of idle time.
   - Auto-resume on query execution (initial SQL statements might take a few minutes).

- Default Workload Management Profile.

4. Wait a few minutes for Kubernetes to provision the hardware.

b. Create the `small-load` Cluster:

1. Repeat the above process to create a `small-load` cluster with same settings.
2. Verify that both clusters appear in the grid.

---

3. Load Sample Data

1. Navigate to **Query Editor** (third icon "`>-`" in the left pane).
2. Create the "yellowbrick_trial" database.
3. Click **Sample Catalog** and select **Premier League Database**.
4. Choose **Setup/Load 1**.
5. Ensure **small-load** is selected in the **Cluster** dropdown and **yellowbrick_trial** is selected as the database.
6. Click **Run** to execute the pre-populated SQL statements. After completion, the `premdb` schema should contain five tables.
7. Load additional data:
   - Clear the SQL editor.
   - Select **Premier League Database** → **Setup/Load 2** from the **Sample Catalog**.
   - Run the SQL statements to populate the `newmatchstats` table with 774,540 rows.

---

4. Create a Role and Users

1. Navigate to **Instance Management** → **Access Control** → **Roles**.
2. Click **Role** to create an **analyst** role with default cluster and no additional privileges.
3. Go to the **Users** tab and click **User** to create two users:
   - "ybtools10"
   - "ybapps10"
4. Assign both users to the **analyst** role with **small-query** as their default cluster.
5. Set user passwords.

> **Note:** Use single sign-on with MFA in production environments.

Grant Permissions:

In the **Query Editor**, run the following:

```sql
GRANT USAGE ON CLUSTER "small-query" TO analyst;
ALTER USER ybtools10 SET search_path TO premdb;
ALTER USER ybapps10 SET search_path TO premdb;
```

---

5. Run SQL Statements

Perform the following steps:

1. Grant the `analyst` role access to tables:

```sql
GRANT SELECT ON ALL TABLES IN SCHEMA premdb TO analyst;
```

2. Log out and log back in as **ybtools10**.
3. Choose **yellowbrick_trial** as the Database.
4. In **Query Editor**, confirm cluster settings:

```sql
SELECT * FROM current_cluster();
SELECT default_cluster();
```

5. Choose **small-query** as the Cluster, if not pre-selected.
6. Run the following SQL statements to retrieve match statistics:

```sql
SELECT t1.season_name, t1.winners, homegoals + awaygoals AS total
FROM (
    SELECT season_name, winners, SUM(SUBSTR(ftscore, 1, 1)::int) AS homegoals
    FROM premdb.season JOIN premdb.match ON season.seasonid = match.seasonid
    GROUP BY season_name, winners
) t1
JOIN (
    SELECT season_name, winners, SUM(SUBSTR(ftscore, 3, 1)::int) AS awaygoals
    FROM premdb.season JOIN premdb.match ON season.seasonid = match.seasonid
    GROUP BY season_name, winners
) t2 ON t1.season_name = t2.season_name
ORDER BY 1, 2;
```

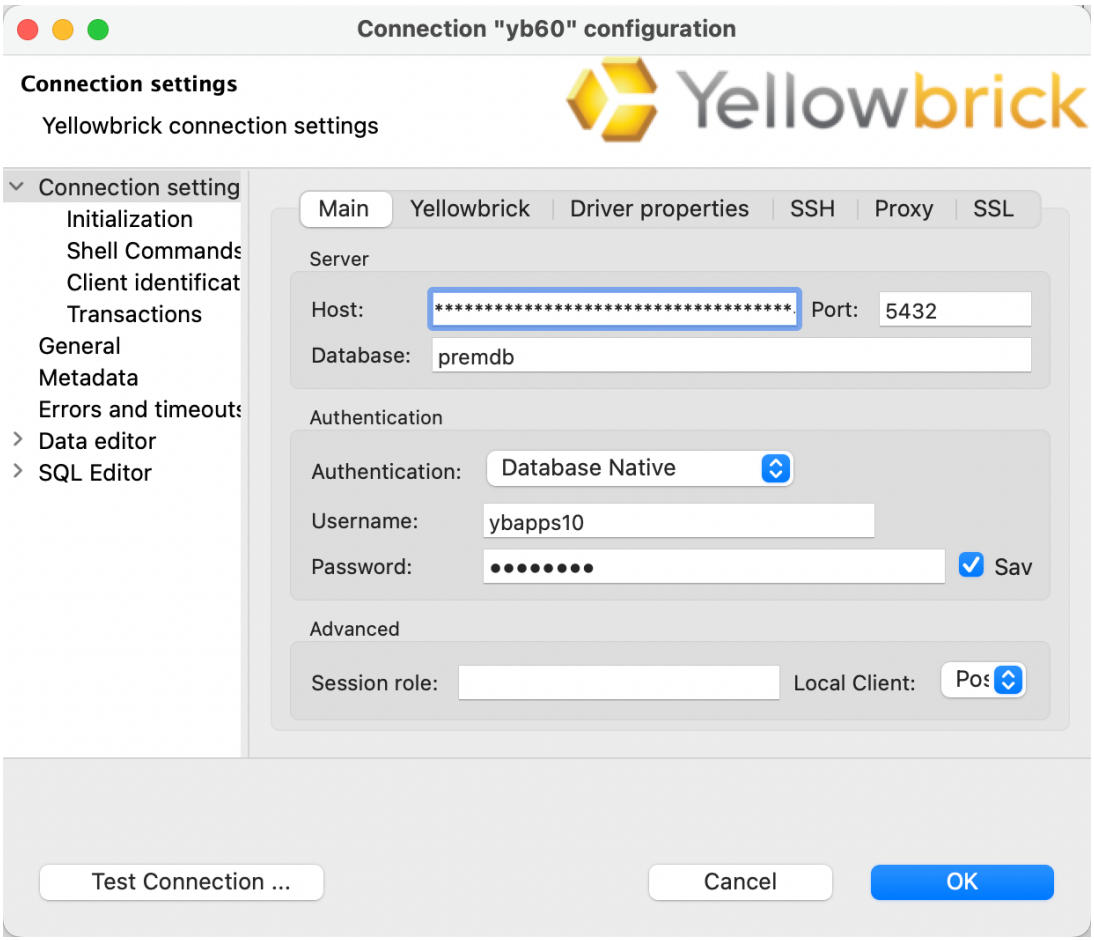
---

6. Connect to DBeaver

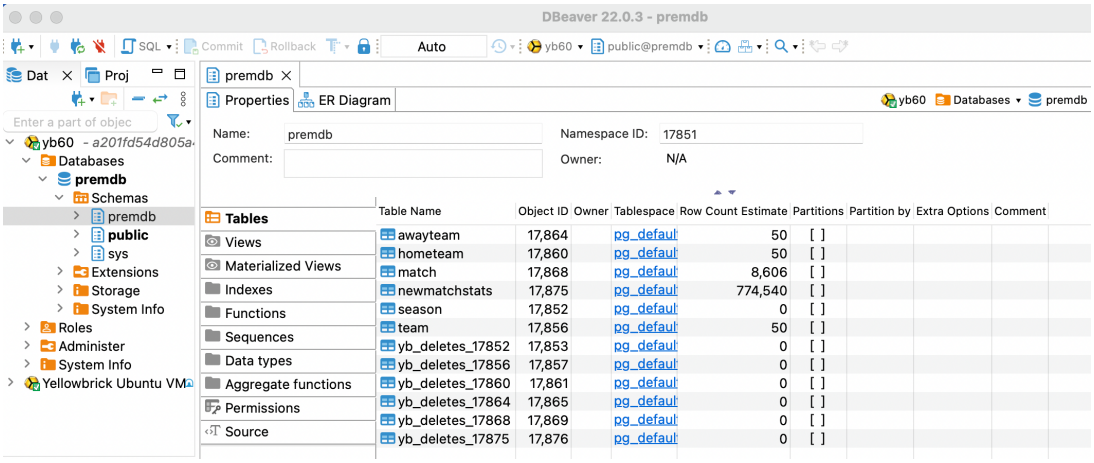
Perform the following steps:

1. Download DBeaver and install it.
2. Create a new database connection:
   - Navigate to **Database** → **New Database Connection** → **Yellowbrick**.
   - Enter the values for the following:
     - Host: Take the Host/Port from the **Instance Management** page of the Yellowbrick Manager
     - Database: "yellowbrick_trial"
     - Username: "ybapps10"
     - Password: Enter password for the Username **ybapps10**

Sample Connection Setup:



View the Database in DBeaver:



You can now run SQL statements as **ybapps10** within DBeaver.

✅ You have successfully completed the walkthrough.

Connect to DBeaver

Description

This tutorial describes how to connect Yellowbrick to the DBeaver SQL Developer tool.

DBeaver is an SQL client and database administration tool. It uses the JDBC API to interact with Yellowbrick by using a PostgreSQL JDBC driver. It provides an editor that supports code completion and syntax highlighting. It offers a plugin architecture (based on the Eclipse plugins architecture) that helps users to modify the application's behavior. This plugin architecture also provides database-specific functionality or database-independent features. This desktop application is written in Java and based on the Eclipse platform.

There are many other SQL IDEs available.

Prerequisites

To complete this tutorial, you will need:

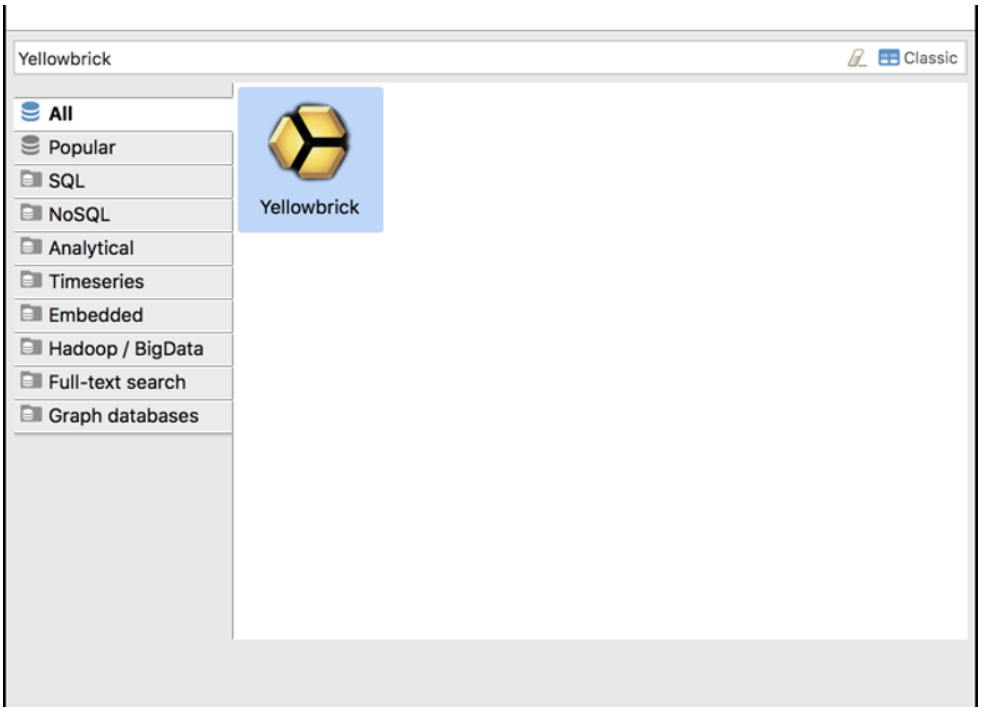
- Access to a Yellowbrick database instance and an active cluster.
- The DBeaver tool installed. The community edition (CE) of DBeaver is free and open source software distributed under the Apache License, while a closed-source enterprise edition of DBeaver is available under a commercial license.

Configure DBeaver Connection

The first time you run DBeaver, it will open with the **Connect to database** dialog box.

Enter "Yellowbrick" in the text filter, click the **All** option in the database type selection, click the **Yellowbrick** icon, and then click the **Next** button.

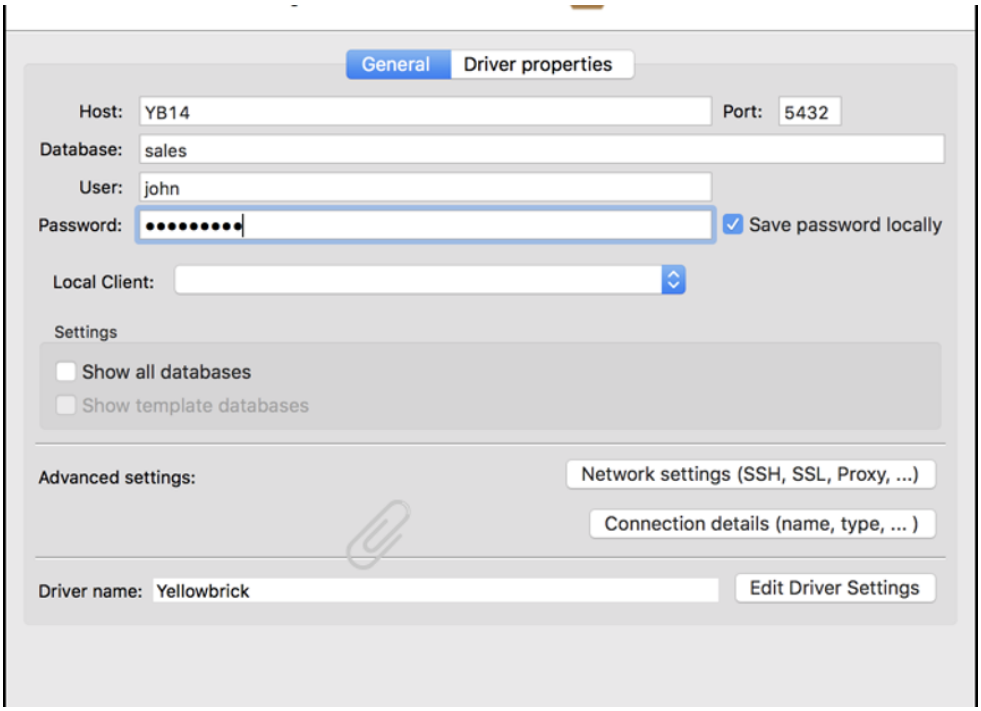
> **Note:** If Yellowbrick icon is not available, you might be running an old version of DBeaver. Please download and install the latest version.



Enter the values of **Host**, **Port**, **Database**, and **User** to connect to the Yellowbrick database. You might also enter the **Password**. Select the **Save password locally** checkbox if you want DBeaver to store the connection password. Click the **Test Connection** button.

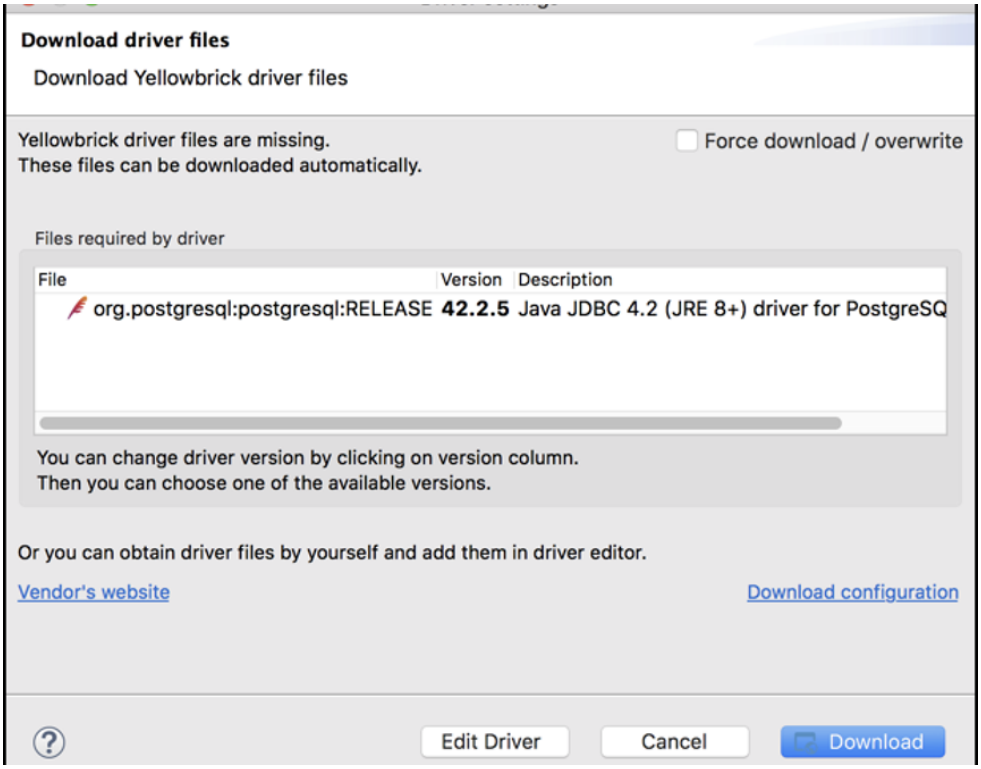
The **Host/Port** for the JDBC and ODBC connections can be found at the **Instance Management -> Summary** page of the Yellowbrick Manager.

> **Note: Instance Management** is accessible by using the cube icon in the left pane.



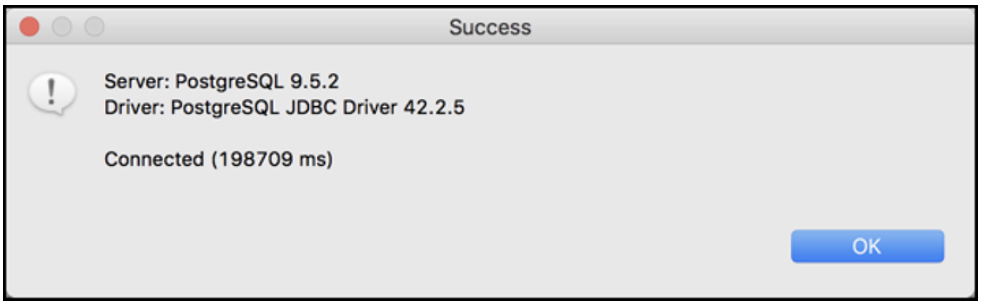
For the first time Yellowbrick connection setup, the **Driver settings** dialog box will appear with the latest PostgreSQL JDBC driver pre-selected in the **Files required by driver** table.

Click the **Download** button.



Once the driver download is successful, you will see a **Success** dialog box.

> **Note:** Assumption is that the credentials entered are correct and that network access to the Yellowbrick instance is available.



You will now see your database objects in DBeaver's object browser. Start running SQL statements from DBeaver. You can find more information about using DBeaver on the DBeaver Documentation site.

Data Science with Jupyter Notebooks on Yellowbrick

Yellowbrick Documentation > Tutorials > Connect Jupyter Notebooks

Platforms: EE: All cloud platforms

Parent topic: Tutorials

Description

Python is a popular programming language used to analyze datasets. Jupyter Notebooks provide an interactive environment to write, test, and share code. In this tutorial, we will show you how to use these popular tools to connect to Yellowbrick and analyze data.

Learning Objectives

After completing this tutorial, you will understand:

- How to configure a Jupyter Notebook environment to connect to Yellowbrick Data Warehouse.
- How to use Python statements to run queries and analyze data stored in Yellowbrick from your notebook.

Prerequisites

To complete this tutorial, you will need:

- Access to a Yellowbrick database instance and an active cluster.
- A Jupyter Notebook environment from Google Colab, another notebook service, or a local notebook installation.
- Network connectivity from your Jupyter Notebook environment to your Yellowbrick database.

Step-by-step Guide

1. Prepare Your Yellowbrick Database

1. **Log in to Yellowbrick Manager Portal**

   Go to your Yellowbrick Manager and log in with your credentials.

2. **Prepare the Database Objects for This Exercise**

   a. Open the query editor window in Yellowbrick Manager and run the SQL statements below:

```sql
drop schema if exists zillow_pi cascade;
drop schema if exists zillowpi cascade;
drop schema if exists zillohvi cascade;

drop schema if exists zillowhvi cascade;
create schema if not exists zillowhvi;

grant all on schema zillowhvi to public;
grant usage on schema zillowhvi to public;
set schema 'zillowhvi';
```

```
drop table if exists homevalueindex;
drop table if exists hvibyyear;

drop external location if exists zhvis3loc;
drop external format if exists zhvis3format;
drop external storage if exists zhvis3data cascade;

create table homevalueindex (
    idxState CHAR(24),
    idxDate date,
    idxValue decimal(8,2)
) distribute replicate;

CREATE TABLE hvibyyear (
    idxstate character(24),
    idxyear int,
    idxvalue numeric(8,2)
) DISTRIBUTE REPLICATE;

CREATE EXTERNAL FORMAT zhvis3format
    TYPE csv
    WITH (delimiter ',', linesep '\n', quote_char '"', escape_char '\', num_header_lines '1');

CREATE EXTERNAL STORAGE "zhvis3data"
    TYPE s3
    ENDPOINT 'https://s3.us-east-1.amazonaws.com'
    REGION 'us-east-1';

CREATE EXTERNAL LOCATION "zhvis3loc"
    PATH 'yb-sample-data'
    EXTERNAL STORAGE "zhvis3data"
    EXTERNAL FORMAT "zhvis3format";

LOAD TABLE "zillowhvi"."homevalueindex"
    FROM ('zhvi.csv')
    EXTERNAL LOCATION "zhvis3loc"
    EXTERNAL FORMAT "zhvis3format" WITH (read_sources_concurrently 'ALLOW');

INSERT INTO zillowhvi.hvibyyear
(
    SELECT idxstate, year(idxDate), avg(idxvalue)
    FROM homevalueindex
    GROUP BY idxstate, year(idxDate)
    ORDER BY idxstate, year(idxDate)
);
```

b. Verify that the schema `zillowhvi` and the table `hvibyyear` are present and contain data by checking the editor tree on the query editor page.

c. Run the following SQL statement in the query editor or right-click the table name in the object browser and select **Explore Data**:

```SQL
SELECT count(*) FROM zillowhvi.hvibyyear;
```

2. Connect and Run Your Python Jupyter Notebook

1. **Prepare Your Jupyter Notebook Environment**

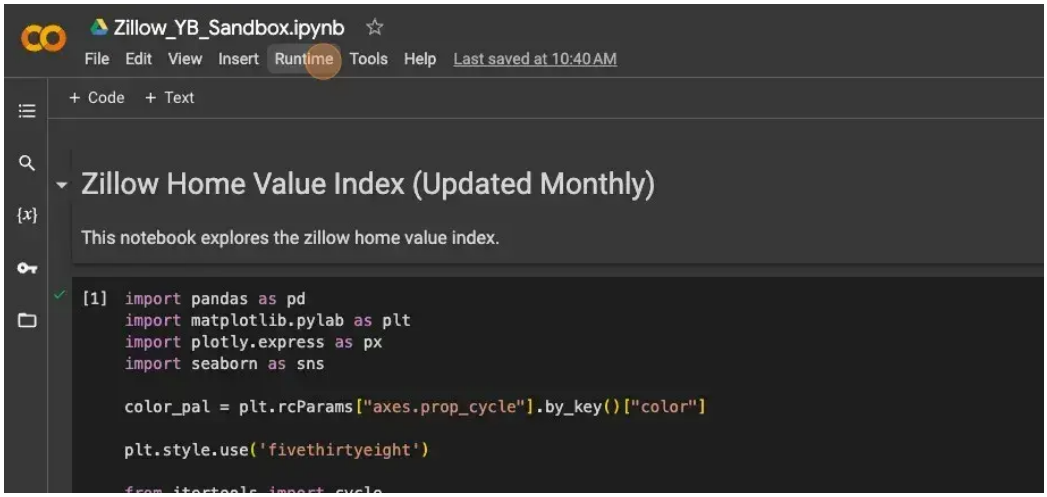
   a. Go to a Google Colab environment by using your Google account. If you have not used it before, it will automatically create an environment and use your Google Drive to store the notebooks.

   OR

   a. Use your own Jupyter environment (e.g., Amazon SageMaker Studio Lab or Visual Studio Code).

b. Use a notebook that employs Python DataFrames to analyze data stored in Yellowbrick.

c. Upload the file Zillow_YB_Sandbox.ipynb.

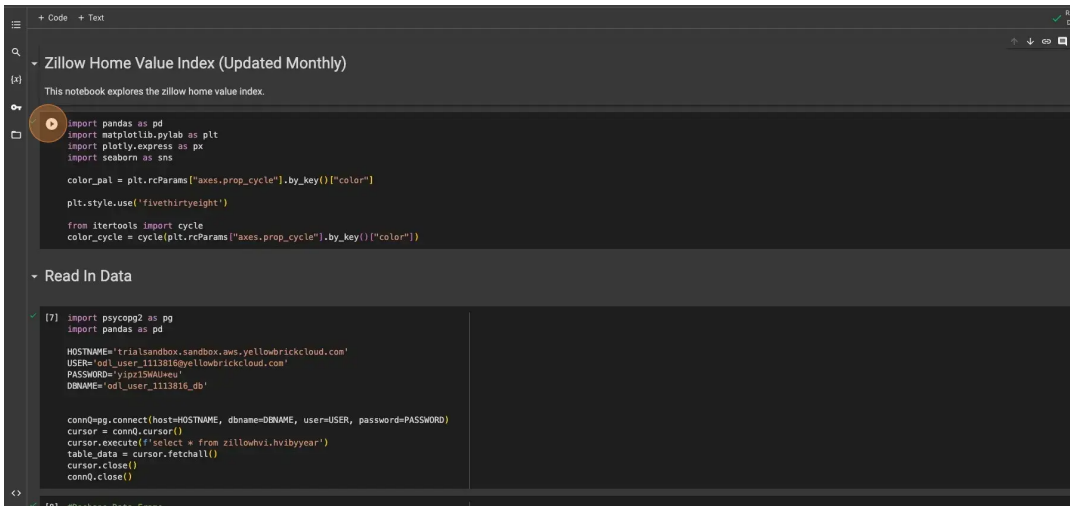


2. **Enter Your Database Credentials**

   In the Jupyter Notebook, enter the **username**, **password**, and **database name** that you are using. Also, verify the **host** in the second cell of the notebook, immediately after the **Read In Data** section.

3. **Final Step**

   You now have a Python notebook successfully connected to your Yellowbrick database environment for data science tasks. To run the code, go to the **Runtime** tab and select **Run all** to run the entire notebook, or execute each cell individually as needed.



Check an S3 Bucket with an AWS Lambda Function and Load Data into Yellowbrick

Description

This tutorial demonstrates how to automate the loading of an existing table. You will use an AWS Lambda function to check an S3 bucket and load the required data file into a Yellowbrick database. The destination table name gets determined by the folder name (for the database schema) and the file name without the extension (for the table name).

Learning Objectives

By completing this tutorial, you will understand how to do the following:

- Create an AWS Lambda function in Python with a file-based trigger.
- Write Python code in an AWS Lambda function to authenticate and connect to a Yellowbrick database.
- Run SQL statements to load data into an external table.

> **Note:** This automation pattern has high adaptability for various use cases, such as scaling database resources automatically.

Prerequisites

To complete this tutorial, ensure you have the following prerequisites:

- Access to a Yellowbrick database instance and an active cluster.
- Permissions to create and manage files in an AWS S3 bucket.
- Permissions to create an AWS Lambda function.
- Network connectivity between your AWS Lambda function and your Yellowbrick Data Warehouse.

Step-by-Step Guide

1. Prepare Your Database

1. Log in to Yellowbrick Manager.
2. In the Query Editor, run the following SQL statements to create a schema and table:

```sql
CREATE SCHEMA IF NOT EXISTS premdb;
DROP TABLE IF EXISTS premdb.match;
CREATE TABLE premdb.match (
    seasonid SMALLINT,
    matchday TIMESTAMP WITHOUT TIME ZONE,
    htid SMALLINT,
    atid SMALLINT,
    ftscore CHAR(3),
```

```
    htscore CHAR(3)
)
DISTRIBUTE ON (seasonid);
```

2. Set Up S3 Storage and Access Keys

a. Create an AWS Access Key

1. Log in to your AWS account.
2. In the search bar, type "IAM" and select the service.
3. Click your **username** under **Users**.
4. Under **Security credentials**, click **Create access key**.
5. Select **Third-party service**.
6. Add a **description (optional)** (e.g., "Access Key for Yellowbrick Lambda Loader").
7. Click **Create access key**.

> **Note:** AWS allows only two access keys per user account. Reuse existing keys, if necessary.

b. Create an S3 Bucket

1. In the AWS console, search for **S3**. In the AWS console, type "S3" in the search bar.
2. Click **Create bucket** and name it "yb-lambda-loader-v1."
3. Allow public access and acknowledge the prompt.

c. Create Folders in the S3 Bucket

1. Inside the **yb-lambda-loader-v1** bucket:
   - Click **Create folder** and name it "premdb."
   - Inside **premdb**, create another folder named "match."

Your bucket structure should be as follows:

```
yb-lambda-loader-v1/
└── premdb/
    └── match/
```

3. Configure External Storage in Yellowbrick

Run the following SQL statements, replacing placeholders with your details:

```sql
CREATE EXTERNAL FORMAT IF NOT EXISTS premdbs3format_lambda_<username>
TYPE csv WITH (
    delimiter ',',
    skip_blank_lines 'true'
);

CREATE EXTERNAL STORAGE IF NOT EXISTS premdbs3storage_lambda_<username>
TYPE s3
ENDPOINT 'https://s3.<your-region>.amazonaws.com'
REGION '<your-region>'
IDENTITY '<access_key>'
CREDENTIAL '<secret_key>';

CREATE EXTERNAL LOCATION IF NOT EXISTS premdbs3location_lambda_<username>
```

```
PATH 'yb-lambda-loader-v1'
EXTERNAL STORAGE premdbs3storage_lambda_<username>;
```

> **Note:** Replace `<your-region>` with your AWS region and ensure the `PATH` matches your bucket name.

4. Create and Configure Your AWS Lambda Function

a. Create a Lambda Function

1. In the AWS console, type "Lambda" in the search bar.
2. Click **Create function**.
3. Enter a name (e.g., "yb-lambda-loader-v1").
4. Select **Python 3.8** as the **runtime**.
5. Click **Create function**.

b. Upload Function Code

1. Download the lambda_load_table.py file.
2. Zip it into `lambda_load_table.zip` .
3. In the Lambda console, select **Upload from** -> **.zip file**.
4. Upload your zipped file.

c. Set Function Configuration

1. Go to **Configuration** -> **General configuration**.
2. Click **Edit** and set the **timeout** to **30 seconds**.
3. Click **Save**.

d. Add Environment Variables

In **Configuration** -> **Environment variables**, add the following values to their keys:

Key	Value
YBDATABASE	`<your Yellowbrick database name>`
YBUSER	`<your Yellowbrick username>`
YBPASSWORD	`<your Yellowbrick password>`
YBSCHEMA	`premdb`
YBHOST	`<host URL from Yellowbrick Manager>`
YBEXTERNALFORMAT	`premdbs3format_lambda_<username>`
YBEXTERNALLOCATION	`premdbs3location_lambda_<username>`

e. Set Permissions

1. In **Configuration** -> **Permissions**, select the role created for your Lambda function.
2. Click **Add permissions** -> **Create inline policy**.
3. Select the **JSON** tab and paste the following JSON statements:

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["s3:*"],
            "Resource": "arn:aws:s3:::*"
        }
    ]
}
```

4. Click **Next**, name the policy (e.g., "s3-lambda-access"), and create it.

f. Add an S3 Trigger

1. Navigate to **Triggers** and click **Add trigger**.
2. Select **S3** as the source.
3. Choose your bucket (`yb-lambda-loader-v1`).
4. Set **Event type** to **PUT**.
5. In **Prefix (optional)**, type "premdb/."
6. Check the acknowledgment box and click **Add**.

5. Test the Lambda Function

a. Upload a Test File

1. Download the sample match CSV file.
2. In the S3 console, navigate to `premdb/match/` .
3. Click **Upload**, select the test file, and complete the upload.

b. Verify Lambda Execution

1. In the Lambda console, go to **Monitor** -> **View CloudWatch logs**.
2. Confirm that the Lambda function triggers and loads the rows successfully.

c. Verify Data in Yellowbrick

1. In Yellowbrick Manager, open the **Query Editor**.
2. Run the following SQL statement:

```sql
SELECT COUNT(*) FROM premdb.match;
```

3. Ensure the row count matches that of the uploaded CSV file.

d. Load Additional Files

Upload new CSV files with the same structure to `premdb/match/` to trigger automatic data loading.

Conclusion

You have successfully automated data loading in Yellowbrick by using an AWS Lambda function that monitors an S3 bucket. This setup can streamline data pipelines and improve operational efficiency.

Stream Data From Apache Kafka to Yellowbrick

Platforms: EE: All cloud platforms

Parent topic: Tutorials

Description

This tutorial explains how to set up an integration between Apache Kafka and Yellowbrick. This integration enables streaming from a Kafka topic directly into a Yellowbrick table.

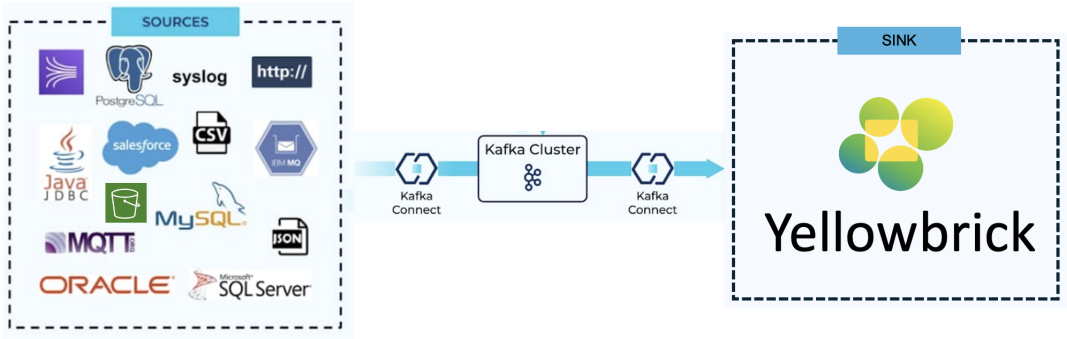
Learning Objectives

By completing this tutorial, you will understand how to do the following:

- Configure Kafka Connect for Yellowbrick.
- Simulate a real-time data stream.
- Route messages from a Kafka topic directly into a Yellowbrick table.

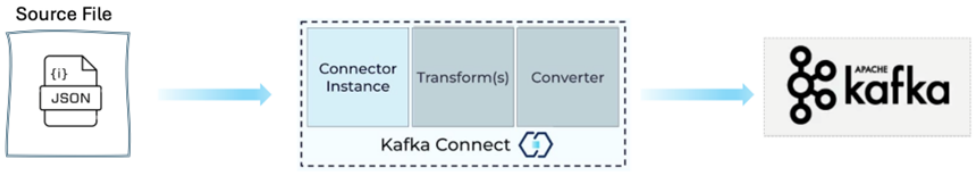
Solution Overview

Apache Kafka is an open source, distributed event streaming platform used to create high-performance data pipelines, perform streaming analytics, and integrate data for mission-critical applications.

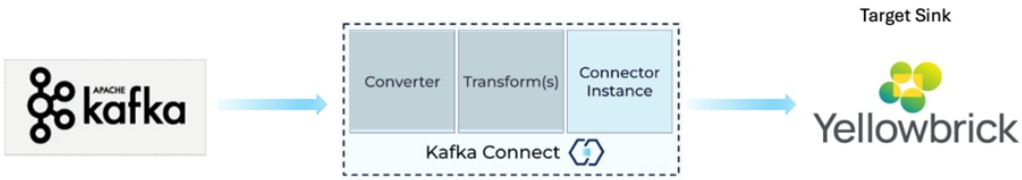


Kafka Connect includes two types of the following connectors:

- **Source Connector:** Imports data from source databases and streams CDC (Change Data Capture) updates, time-series data, and other real-time updates to Kafka topics. It can also collect metrics from application servers for low-latency stream processing.



- **Sink Connector:** Delivers data from Kafka topics into target platforms, such as Yellowbrick for downstream analytics and OLAP workloads.



Prerequisites

To complete this tutorial, ensure you have the following prerequisites:

- Apache Kafka is not available on your machine.

> **Note:** If you already have Kafka or are using a managed platform such as Confluent Cloud, please directly visit Set Up Kafka Connect to Stream Source Data to Kafka.

- A Linux environment is present.

> **Note:** If you are using Windows without the WSL (Windows Subsystem for Linux), replace the `.sh` commands with their PowerShell or `.bat` equivalents.

- Access to a Yellowbrick environment.

Step-By-Step Guide

Set Up Kafka

1. **Download Apache Kafka**

   Download the latest Kafka binaries from the Apache Kafka Downloads Page.

> **Note:** Please refer the installation directory as `$KAFKA_HOME` throughout this tutorial.

2. **Start Services**

   a. Open terminal windows, navigate to `$KAFKA_HOME`, and run the following commands:

```bash
bin/zookeeper-server-start.sh config/zookeeper.properties
bin/kafka-server-start.sh config/server.properties
```

   b. Verify that both Zookeeper and Kafka services start without error messages.

3. **Test the Kafka Service**

   Use the following bash commands in the terminal to test the Kafka service:

```bash
# Create a new test topic
bin/kafka-topics.sh --create --bootstrap-server localhost:9092 --replication-factor 1 --partitions 1 --topic test

# List topics to verify creation
bin/kafka-topics.sh --list --bootstrap-server localhost:9092

# Produce messages to the Kafka topic
bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test
Hello, World
It Is a Bright New Day
Time to Innovate and Make a Change

# Consume messages from the Kafka topic
bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic test --from-beginning
```

4. **Expected Output:**

The expected output is as follows:

```
Hello, World
It Is a Bright New Day
Time to Innovate and Make a Change
```

Once verified, you might close these terminals.

Set Up Kafka Connect to Stream Source Data to Kafka

Perform the following commands in the terminal:

1. **Configure the Source Connector**

   Edit `connect-file-source.properties` in `$KAFKA_HOME/config`:

   ```bash
   name=local-file-source
   connector.class=FileStreamSource
   tasks.max=1
   file=test.txt
   topic=connect-test
   ```

2. **Download and Set Up Configuration Files**

   Download the required files from the Yellowbrick GitHub Repository and extract them into `$KAFKA_HOME/configYB`.

3. **Verify the Plugin Path**

   In `connect-standalone.properties`, ensure the `plugin.path` points to the correct JAR file:

   ```bash
   plugin.path=libs/connect-file-3.7.0.jar
   ```

4. **Prepare the Source Data**

   Download `your-file.txt` and place it in `$KAFKA_HOME`:

   ```bash
   cat your-file.txt >> ybd-source-json.txt
   ```

5. **Start Kafka Connect**

```bash
bin/connect-standalone.sh config/connect-standalone.properties config/connect-file-source.properties
```

6. **Verify Message Reception**

```bash
bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic connect-source --from-beginning
```

Configure Kafka Connect for Yellowbrick

Perform the following commands in the terminal:

1. **Install YB Tools**

   Download and install YBTools.

   Copy the Kafka connector JAR to `$KAFKA_HOME/libs`:

```bash
cp /ybtools/integration/kafka/kafka-connect-yellowbrick-6.9.0-SNAPSHOT-shaded.jar $KAFKA_HOME/libs
```

2. **Create a Target Table in Yellowbrick**

   In the Yellowbrick SQL Editor, run:

```sql
CREATE SCHEMA kafka;
CREATE TABLE kafka.kafka_ybd_source_load (
    col1 VARCHAR(30),
    col2 VARCHAR(30),
    tstmp TIMESTAMP DEFAULT CURRENT_TIMESTAMP
) DISTRIBUTE RANDOM;
```

3. **Configure the Sink Connector**

   Edit `connect-YB-sink_t2.properties` with your Yellowbrick connection details:

```yaml
yb.hostname=myinstance.aws.yellowbrickcloud.com
yb.port=5432
yb.database=odl_user_XXXXXX_db
yb.username=odl_user_XXXXXX
yb.password=<your password>
yb.table=kafka_ybd_source_load
yb.schema=kafka
yb.columns=col1,col2
```

4. **Update Kafka Connect Port**

   To avoid port conflicts, edit `connect-standalone.properties`:

```bash
listeners=http://0.0.0.0:8085
```

5. **Start the Sink Connector**

```bash
bin/connect-standalone.sh configYB/connect-standalone_t2.properties configYB/connect-YB-sink_t2.properties
```

6. **Verify Data Ingestion**

   In the Yellowbrick SQL Editor, run:

```sql
SELECT * FROM kafka.kafka_ybd_source_load;
```

7. **Test Continuous Streaming**

   Add more data to the source file:

   ```bash
   cat your-file.txt >> ybd-source-json.txt
   ```

   Query the Yellowbrick table to confirm new data gets inserted.

Clean Up

1. End Kafka Connect processes for both source and sink pipelines.
2. Stop Kafka and Zookeeper services.
3. Remove temporary files using the following bash command:

   ```bash
   rm /tmp/connect.offsets
   rm -r /tmp/kafka-logs
   rm -r /tmp/zookeeper
   ```

✅ You have successfully demonstrated data streaming from Kafka into a Yellowbrick table.

Orchestrate Data with Apache Airflow

Yellowbrick Documentation > Tutorials > Orchestrate with Airflow

Platforms: EE: All cloud platforms

Parent topic: Tutorials

Description

This tutorial explains the following:

1. How to use Apache Airflow to source Bitcoin market data from CoinCap.io.
2. How to stage Bitcoin market data into Yellowbrick.
3. How to build summary KPIs using an ELT approach with SQL transformations.

> **Note:** Review the CoinCap Terms of Service. Data from CoinCap is for personal use only.

Learning Objectives

After completing this tutorial, you will understand how to do the following:

- Connect Apache Airflow to Yellowbrick.
- Orchestrate data pipelines to insert and transform data using SQL in Yellowbrick.

Prerequisites

Before starting, ensure you have the following prerequisites:

- Apache Airflow installed and configured either locally, using Docker, or via a managed Airflow service.
- Access to a Yellowbrick environment.

Step-by-Step Guide

1. Create Base Tables

Go to the Yellowbrick Query Editor (or use an IDE, such as DBeaver) and run the following SQL statements to create two tables:

```sql
CREATE SCHEMA IF NOT EXISTS coincap_bitcoin;

CREATE TABLE IF NOT EXISTS coincap_bitcoin.bitcoin_price_tracker (
    id VARCHAR(10),
    timestamp TIMESTAMP,
    rank INTEGER,
    symbol VARCHAR(10),
    name VARCHAR(255),
    supply NUMERIC,
    max_supply NUMERIC,
    market_cap_usd NUMERIC,
    volume_usd_24_hr NUMERIC,
    price_usd NUMERIC,
```

```
    change_percent_24_hr NUMERIC,
    vwap_24_hr NUMERIC,
    explorer VARCHAR(255)
) DISTRIBUTE ON (timestamp);

CREATE TABLE IF NOT EXISTS coincap_bitcoin.bitcoin_kpi (
    timestamp TIMESTAMP,
    current_price NUMERIC(18, 0),
    price_change_percentage NUMERIC(38, 2),
    supply NUMERIC(18, 0),
    max_supply NUMERIC(18, 0),
    percentage_of_max_supply_mined NUMERIC(38, 2),
    volatility NUMERIC(38, 2),
    moving_average_10_days NUMERIC(18, 0),
    market_dominance NUMERIC(38, 2),
    vwap_24_hr NUMERIC(18, 0),
    vwap_crossing VARCHAR(10)
) DISTRIBUTE ON (timestamp);
```

2. Configure Airflow

Add the Yellowbrick Connection

Use the Airflow UI to create a PostgreSQL connection:

> **Note:** Yellowbrick uses the PostgreSQL interface and drivers.

OR

Use the following bash statements:

```bash
airflow connections add \
    --conn_id ybconnection \
    --conn_type postgres \
    --conn_host myinstance.aws.yellowbrickcloud.com \
    --conn_login myusername \
    --conn_password mypassword \
    --conn_port 5432
```

> **Note:** Replace `myusername` and `mypassword` with your Yellowbrick credentials.

3. Install Pre-Created DAGs

Download the DAG files from this repository and place them in your Airflow `/dags` directory. The directory include the following files:

- `airflow_dag_coincap_to_yb.py`
- `airflow_dag_bitcoin_kpi.py`

DAG Overview

File 1: `coincap_to_yb_dag`

This file performs the following tasks:

- Fetches Bitcoin price data from CoinCap API.
- Uploads data to the `bitcoin_price_tracker` table.
- Triggers `bitcoin_kpi_dag` upon successful data upload.
- Schedule: Runs every 5 minutes.

File 2: `bitcoin_kpi_dag`

This file performs the following tasks:

- Calculates Bitcoin KPIs using data from Yellowbrick.
- Runs on-demand (no fixed schedule).
- Depends on `coincap_to_yb_dag` execution.

KPI Calculations

The following metrics get calculated and stored in `bitcoin_kpi`:

Column	Description
`Average Price`	Average Bitcoin price over different intervals.
`Price Volatility`	Measures volatility using standard deviation and other statistical methods.
`Price Change Percentage`	Percentage change in price over 24 hours.
`Price Momentum`	Measures price change momentum using methods, such as linear regression or moving averages.
`Trading Volume`	Bitcoin trading volume over time.
`Market Cap`	Market capitalization based on price and circulating supply.
`Dominance Ratio`	Bitcoin's market cap share relative to the entire cryptocurrency market.
`Price Correlation`	Correlation between Bitcoin and other financial assets.
`Support/Resistance Levels`	Identifies potential reversal points in price trends.
`Moving Averages`	Simple and exponential moving averages for trend analysis.
`RSI (Relative Strength Index)`	Identifies overbought or oversold market conditions.
`Price Oscillators`	Tracks indicators such as MACD and Stochastic Oscillator.

4. Activate the DAGs

Using the Airflow UI

Go to the Airflow interface and activate both DAGs.

Using the CLI

Please run the following bash commands:

```bash
airflow dags unpause coincap_to_yb_dag
airflow dags unpause bitcoin_kpi_dag
```

5. Validate Success

Use SQL statements to verify data ingestion and KPI calculations:

```sql
SELECT * FROM coincap_bitcoin.bitcoin_price_tracker;
SELECT * FROM coincap_bitcoin.bitcoin_kpi;
```

6. Cleanup

Deactivate the DAGs

Use the Airflow UI or CLI to pause the DAGs:

```bash
airflow dags pause coincap_to_yb_dag
airflow dags pause bitcoin_kpi_dag
```

Drop the Tables

Run the following SQL statement to clean up your database:

```sql
DROP SCHEMA coincap_bitcoin CASCADE;
```

✅ **Next Steps:** Explore other Yellowbrick tutorials to expand your data pipeline orchestration skills.

Building a RAG Chatbot with Spring Using Yellowbrick as a Vector Store

Overview

If you are a Java developer, you have probably used libraries from the Spring ecosystem. Java developers love Spring because it simplifies the development of complex applications. Its comprehensive ecosystem includes many specialized frameworks that reduce the complexity of integrating third-party APIs, databases, messaging systems, and more.

Spring AI is the latest project within the Spring ecosystem, designed to make it easy for Java developers to integrate AI capabilities into their applications. The framework abstracts the complexities of connecting to various AI-provider APIs by using familiar Spring abstractions.

Spring AI supports common tasks, such as implementing chatbots and virtual assistants, content generation and summarization, sentiment analysis, customer feedback processing, image and text recognition, and decision support.

In this tutorial, we will explain how to create a chatbot by using Spring AI. First, we will build a simple chatbot by using OpenAI, then enhance it with Retrieval-Augmented Generation (RAG) for improved contextual accuracy.

The RAG technique improves accuracy by augmenting the chatbot's knowledge with external data, such as a product manual, beyond the model's training data.

The Process

The process begins when the model receives a query, such as a user question. Use a pre-trained language model to encode this query into a vector (a mathematical representation).

Next, the system searches for relevant information within an external knowledge base by using the encoded query. The retrieved documents are then appended to the original query, providing additional context.

A vector database is ideal for storing these mathematical representations. While Yellowbrick does not natively support vector data types, it can store vector data efficiently for retrieval.

Spring abstracts the logic for interfacing with the vector store, allowing developers to switch databases without significant code changes. We will explain this by first using pgVector (a PostgreSQL extension) and then switching to Yellowbrick.

Create the Chatbot

All code for the following steps is available on GitHub.

Prerequisites

- JDK 17 or later
- OpenAI developer key

Getting Started

Please perform the following steps:

1. Visit start.spring.io and create a new project. Select Java 17 and Maven. Add dependencies for Spring Web and OpenAI. Decompress the generated project and import it into your IDE.

2. Add your OpenAI key to `application.properties`:

```properties
spring.ai.openai.api-key=<YOUR_OPENAI_KEY>
spring.main.web-application-type=none
```

3. Update your Spring application class:

```java
@Bean
public CommandLineRunner runner(ChatClient.Builder builder) {
    return args -> {
        ChatClient chatClient = builder.build();
        String response = chatClient.prompt("Tell me a joke").call().content();
        System.out.println(response);
    };
}
```

4. Run the application:

```bash
./mvnw spring-boot:run
```

Add pgVector as a Vector Store

1. Start a pgVector Docker container:

```bash
docker run -it --rm -p 5432:5432 -e POSTGRES_USER=postgres -e POSTGRES_PASSWORD=postgres pgvector/pgvector:pg17
```

2. Add the pgVector dependency to `pom.xml`:

```xml
<dependency>
  <groupId>org.springframework.ai</groupId>
  <artifactId>spring-ai-pgvector-store-spring-boot-starter</artifactId>
</dependency>
```

3. To read PDF files, add:

```xml
<dependency>
  <groupId>org.springframework.ai</groupId>
  <artifactId>spring-ai-pdf-document-reader</artifactId>
</dependency>
```

4. Configure the database connection in `application.properties`:

```properties
spring.application.name=springai
spring.datasource.url=jdbc:postgresql://localhost:5432/postgres
spring.datasource.username=postgres
spring.datasource.password=postgres
```

```
spring.ai.vectorstore.pgvector.initialize-schema=true
spring.ai.vectorstore.pgvector.max-document-batch-size=1000
```

Add Content to the Vector Store

We will use an Employee Handbook PDF for this example. Place it in `src/main/resources/data`.

Create a `DataLoaderService` class:

```java
@Service
public class DataLoaderService {

  @Value("classpath:/data/Employee_Handbook.pdf")
  private Resource pdfResource;

  @Autowired
  private VectorStore vectorStore;

  @PostConstruct
  public void load() {
    PagePdfDocumentReader pdfReader = new PagePdfDocumentReader(pdfResource,
        PdfDocumentReaderConfig.builder()
            .withPageExtractedTextFormatter(ExtractedTextFormatter.builder()
                .withNumberOfBottomTextLinesToDelete(3)
                .withNumberOfTopPagesToSkipBeforeDelete(1)
                .build())
            .withPagesPerDocument(1)
            .build());

    var tokenTextSplitter = new TokenTextSplitter();
    vectorStore.accept(tokenTextSplitter.apply(pdfReader.get()));
  }
}
```

Query with Context from the Vector Store

Prompts

The prompt ensures contextually correct responses. It consists of the following:

- **Context:** Text snippets from the knowledge base.
- **Query:** The user's input.

Code

Please perform the following steps:

1. Create a `ChatService`:

```java
@Service
public class ChatService {

  @Autowired
  @Qualifier("openAiChatModel")
  private ChatModel chatClient;

  @Autowired
  private VectorStore vectorStore;
```

```
  private final String PROMPT_BLUEPRINT = """
    Answer the query using the provided context:
    {context}
    Query:
    {query}
    If no context is available, respond with:
    "I'm sorry, I don't have the information you're looking for."
  """;

  public String chat(String query) {
    return chatClient.call(createPrompt(query, searchData(query)));
  }

  private String createPrompt(String query, List<Document> context) {
    PromptTemplate template = new PromptTemplate(PROMPT_BLUEPRINT);
    template.add("query", query);
    template.add("context", context);
    return template.render();
  }

  public List<Document> searchData(String query) {
    return vectorStore.similaritySearch(query);
  }
}
```

2. Add a controller:

```java
@RestController
class AIController {

  private final ChatService chatService;

  AIController(ChatService chatService) {
    this.chatService = chatService;
  }

  @GetMapping("/chat")
  public Map<String, String> chat(@RequestParam String query) {
    return Map.of("answer", chatService.chat(query));
  }
}
```

3. Register a `ChatClient` bean:

```java
@Bean
ChatClient chatClient(ChatClient.Builder builder) {
  return builder.build();
}
```

4. Remove `spring.main.web-application-type=none` from `application.properties`, then run the application.

5. Test with HTTPie, `curl`, or Postman:

```bash
http -v GET localhost:8080/chat?query='What is the vacation policy'
```

6. The following Json statement shows the expected response:

```json
{
  "answer": "Vacations are provided as a benefit..."
```

```
}
```

Switch to Yellowbrick as a Vector Store

Steps

Please perform the following steps:

1. Clone the repository:

```bash
git clone https://github.com/YellowbrickData/yellowbrick-learn
```

2. Navigate to the vector store directory:

```bash
cd yellowbrick-learn/samples/rags/vectorstore
```

3. Build the artifact:

```bash
./mvnw install
```

4. Replace the `pgVector` dependency in `pom.xml` with:

```xml
<dependency>
  <groupId>com.yellowbrick.spring</groupId>
  <artifactId>vectorstore</artifactId>
  <version>0.0.1-SNAPSHOT</version>
</dependency>
```

5. Update `application.properties`:

```properties
spring.datasource.url=jdbc:postgresql://myinstance.aws.yellowbrickcloud.com:5432/<username>
spring.datasource.username=<username>
spring.datasource.password=<password>
```

6. Restart the application. Queries will now run on Yellowbrick.

Bonus: Switch to Amazon Bedrock

Steps

Please perform the following steps:

1. Enable the Titan model in your AWS account.

2. Replace the OpenAI dependency with Bedrock in `pom.xml`:

```xml
<dependency>
  <groupId>org.springframework.ai</groupId>
  <artifactId>spring-ai-bedrock-ai-spring-boot-starter</artifactId>
</dependency>
```

3. Update `application.properties`:

```properties
spring.ai.bedrock.titan.chat.enabled=true
spring.ai.bedrock.titan.embedding.enabled=true
spring.ai.bedrock.titan.embedding.model=amazon.titan-embed-text-v2:0
spring.ai.bedrock.aws.region=us-east-1
spring.ai.bedrock.aws.secret-key=<your_secret_key>
spring.ai.bedrock.aws.access-key=<your_access_key>
```

4. Rerun the application to see similar results.

References

1. Spring AI
2. Yellowbrick as a Vector Store (Python Edition)
3. What is Retrieval-Augmented Generation
4. Sample App on GitHub

Vector Store

Yellowbrick Documentation > Tutorials > Python RAG Vector Store

Platforms: All platforms

Parent topic: Tutorials

Yellowbrick can store and search vector embeddings by using SQL through its integration with LangChain. This tutorial guides you through installing the required LangChain components. This tutorial also helps you use Yellowbrick to support a simple RAG (Retrieval-Augmented Generation) application with a chatbot.

Using Yellowbrick as the Vector Store for ChatGPT

This tutorial explains how to create a simple Python-based chatbot backed by ChatGPT using Yellowbrick as a vector store to support RAG.

Prerequisites

Ensure you have the following prerequisites:

- Access to a Yellowbrick instance
- An API key from OpenAI

Tutorial Overview

The tutorial is divided into the following five parts:

1. Create a baseline chatbot using LangChain without a vector store.
2. Create an embeddings table in Yellowbrick to represent the vector store.
3. Load documents (e.g., the Administration chapter of the Yellowbrick Manual).
4. Create vector embeddings from those documents and store them in Yellowbrick.
5. Ask questions to the improved chatbot to compare results.

Part 1: Creating a Baseline Chatbot without a Vector Store

We will use LangChain to question ChatGPT without any additional context provided by a vector store.

Perform the following steps:

1. Install the Required Libraries

```python
%pip install --upgrade --quiet langchain langchain-openai langchain-community psycopg2-binary tiktoken
```

2. Set Up the Credentials and Configuration

```python
# Replace these values with your Yellowbrick environment details and OpenAI API key
YBUSER = "[USER]"
YBPASSWORD = "[PASSWORD]"
YBDATABASE = "[DATABASE]"
```

```
YBHOST = "myinstance.aws.yellowbrickcloud.com"
OPENAI_API_KEY = "[OPENAI_API_KEY]"
```

3. Import the Libraries and Initialize ChatGPT

```python
import os
import urllib.parse as urlparse
from langchain.chains import LLMChain
from langchain_openai import ChatOpenAI
from langchain_core.prompts.chat import ChatPromptTemplate, HumanMessagePromptTemplate, SystemMessagePromptTemplate

# Set OpenAI API Key
os.environ["OPENAI_API_KEY"] = OPENAI_API_KEY

# Define the prompt template
system_template = """If you don't know the answer, make your best guess."""
messages = [
    SystemMessagePromptTemplate.from_template(system_template),
    HumanMessagePromptTemplate.from_template("{question}"),
]
prompt = ChatPromptTemplate.from_messages(messages)

# Initialize ChatGPT Model
llm = ChatOpenAI(model_name="gpt-3.5-turbo", temperature=0, max_tokens=256)
chain = LLMChain(llm=llm, prompt=prompt)

# Query the chatbot
def print_result_simple(query):
    result = chain(query)
    print(f"Question: {query}\nAnswer: {result['text']}")

# Example queries
print_result_simple("How many databases can be in a Yellowbrick Instance?")
print_result_simple("What's an easy way to add users in bulk to Yellowbrick?")
```

Part 2: Connecting to Yellowbrick and Creating the Embedding Tables

Create a UTF-8 encoded table in Yellowbrick to store document embeddings by performing the following steps:

```python
import psycopg2

# Build connection string
yellowbrick_connection_string = f"postgres://{urlparse.quote(YBUSER)}:{YBPASSWORD}@{YBHOST}:5432/{YBDATABASE}"
embedding_table = "my_embeddings"

# Create embedding table
create_table_query = f"""
CREATE TABLE IF NOT EXISTS {embedding_table} (
    doc_id UUID NOT NULL,
    embedding_id SMALLINT NOT NULL,
    embedding DOUBLE PRECISION NOT NULL
)
DISTRIBUTE ON (doc_id);
TRUNCATE TABLE {embedding_table};
"""

with psycopg2.connect(yellowbrick_connection_string) as conn:
    with conn.cursor() as cursor:
        cursor.execute(create_table_query)
        conn.commit()
        print(f"Table '{embedding_table}' created successfully!")
```

Part 3: Extracting the Documents to Index

Retrieve documents from an existing Yellowbrick table to prepare for embedding by performing the following steps:

```python
YB_DOC_DATABASE = "sample_data"
YB_DOC_TABLE = "yellowbrick_documentation"

yellowbrick_doc_connection_string = f"postgres://{urlparse.quote(YBUSER)}:{YBPASSWORD}@{YBHOST}:5432/{YB_DOC_DATABASE}"

with psycopg2.connect(yellowbrick_doc_connection_string) as conn:
    with conn.cursor() as cursor:
        cursor.execute(f"SELECT path, document FROM {YB_DOC_TABLE}")
        yellowbrick_documents = cursor.fetchall()
        print(f"Extracted {len(yellowbrick_documents)} documents successfully!")
```

Part 4: Load Documents into Yellowbrick Vector Store

Split documents into chunks, create embeddings, and insert them into Yellowbrick by performing the following steps:

```python
from langchain_core.documents import Document
from langchain_text_splitters import RecursiveCharacterTextSplitter
from langchain_openai import OpenAIEmbeddings
from langchain_community.vectorstores import Yellowbrick

DOCUMENT_BASE_URL = "https://docs.yellowbrick.com/6.7.1/"
chunk_size_limit = 2000
max_chunk_overlap = 200

# Convert fetched documents to LangChain Document format
documents = [
    Document(
        page_content=doc[1],
        metadata={"source": DOCUMENT_BASE_URL + doc[0].replace(".md", ".html")}
    )
    for doc in yellowbrick_documents
]

# Split documents
text_splitter = RecursiveCharacterTextSplitter(
    chunk_size=chunk_size_limit,
    chunk_overlap=max_chunk_overlap,
    separators=["\n## ", "\n", ",", " ", ""]
)
split_docs = text_splitter.split_documents(documents)

# Create embeddings and store in Yellowbrick
embeddings = OpenAIEmbeddings()
vector_store = Yellowbrick.from_documents(
    documents=split_docs,
    embedding=embeddings,
    connection_string=yellowbrick_connection_string,
    table=embedding_table
)

print(f"Created vector store with {len(split_docs)} document chunks.")
```

Part 5: Creating a Chatbot Using the Yellowbrick Vector Store

Enhance the chatbot by using the populated vector store to improve answers with relevant context.

Perform the following steps:

```python
from langchain.chains import RetrievalQAWithSourcesChain

# Initialize Retrieval Chain
chain = RetrievalQAWithSourcesChain.from_chain_type(
    llm=llm,
    chain_type="stuff",
    retriever=vector_store.as_retriever(search_kwargs={"k": 5}),
    return_source_documents=True,
    chain_type_kwargs={"prompt": prompt}
)

def print_result_with_sources(query):
    result = chain(query)
    print(f"""\nQuestion: {query}\nAnswer: {result['answer']}\nSources: {result['sources']}\n""")

# Example queries with vector store context
print_result_with_sources("How many databases can be in a Yellowbrick Instance?")
print_result_with_sources("What's an easy way to add users in bulk to Yellowbrick?")
```

Part 6: Introducing an Index to Improve Performance

Use LSH (Locality-Sensitive Hashing) to optimize retrieval time when dealing with large document sets.

Perform the following steps:

```python
lsh_params = Yellowbrick.IndexParams(
    Yellowbrick.IndexType.LSH, {"num_hyperplanes": 8, "hamming_distance": 2}
)
vector_store.create_index(lsh_params)

# Query with index enabled
chain = RetrievalQAWithSourcesChain.from_chain_type(
    llm=llm,
    chain_type="stuff",
    retriever=vector_store.as_retriever(search_kwargs={"index_params": lsh_params, "k": 5}),
    return_source_documents=True,
    chain_type_kwargs={"prompt": prompt}
)

# Example indexed queries
print_result_with_sources("How many databases can be in a Yellowbrick Instance?")
print_result_with_sources("What's an easy way to add users in bulk to Yellowbrick?")
```

Next Steps

- Modify the code to ask different questions or load your own documents.
- Use alternative embeddings (e.g., Hugging Face models or private LLMs, such as Meta's Llama 2).
- Explore the flexibility of LangChain to handle various document formats (i.e., HTML, PDF, etc.).

For more information, visit Yellowbrick Documentation.

Calling `ybload` from AWS Lambda

Yellowbrick Documentation > Tutorials > AWS Lambda with ybload

Platforms: EE: All cloud platforms

Parent topic: Tutorials

This advanced tutorial explains how to create an AWS Lambda function that can load data into Yellowbrick every time a file is written into an S3 bucket. The Lambda function that you will create, uses the `ybload` command line tool to perform the data load. In a page AWS Lambda Data Loader, you can learn how to load data with AWS Lambda by using the `LOAD TABLE` command. The `ybload` tool supports more options than the `LOAD TABLE` command (e.g., the ability to perform upserts), and so is a useful approach when dealing with more complex load requirements.

Prerequisites

- An AWS account with permissions to create and deploy resources to AWS using the AWS CLI.
- Credentials to connect to a running Yellowbrick instance by using the command line tools.
- The ybtools tarball that contains `ybload`. The easiest way to obtain this is by using "generic" package from the Yellowbrick Manager.
- Git installed, to be able to clone the tutorial code.
- OpenJDK installed, to be able to compile the tutorial code.
- Maven, to be able to build the Lambda function package.

Overview

In this tutorial you will accomplish the following tasks:

1. Clone the tutorial sample code from GitHub.
2. Configure the `deploy.sh` script in the sample code with your AWS and Yellowbrick credentials.
3. Run `deploy.sh` to create and deploy the AWS Lambda function along with its dependencies.
4. Create a Yellowbrick table to be able to receive data from the Lambda function.
5. Load the data through a test CSV file into Yellowbrick by copying the file to an S3 bucket.
6. Observe the AWS CloudWatch logs to see the progress of the load.
7. Upsert data into the same Yellowbrick table.
8. Understand the strengths and limitations of this approach.

Part 1: Clone the Sample Code Repository

Clone the sample code in GitHub using:

```sql
git clone git@github.com:markcusack/yellowbrick-learn.git
```

Part 2: Configure `deploy.sh`

Go to the `yellowbrick-learn/learn/lambda-ybload` folder. In a text editor, open `deploy.sh` and provide values for the unset variables below, and change others wherever needed:

```bash
YBTOOLS_TGZ_NAME="ybtools-7.1.2-66379.e33141f2.generic.noarch.tar.gz"
YBLOAD_TGZ_NAME="${YBTOOLS_TGZ_NAME/ybtools/ybload}"

# S3 buckets for Lambda zip and parquet/csv files dropped in
ZIP_BUCKET=
LANDING_BUCKET=

# Change to suit your AWS account
ACCESS_KEY_ID=
SECRET_ACCESS_KEY=
SESSION_TOKEN=
REGION=
AWS_ACCOUNT=

# Yellowbrick database settings and target table
YB_HOST=
YB_USER=
YB_PASSWORD=
YB_DATABASE=
YB_TABLE=
YBLOAD_EXTRA_ARGS="--disable-trust"

# No need to change these unless a conflict
YBLOAD_FUNCTION="YBLoadFunction"
ROLE_NAME="lambda-s3-execution-role"
LAMBDA_JAVA_HOME="/var/lang"
JAR_NAME="ybload-lambda-1.0-SNAPSHOT.jar"
ZIP_NAME="ybload-lambda.zip"
```

On the first line, you need to set or confirm the name of the ybtools tarball. This should match the name and location of the file you downloaded as a part of the prerequisites.

You must create the following two buckets in AWS S3:

- One that the Lambda function will load source data from.
- One that will store the Java classes for the Lambda function you will create.

Create these buckets by using the AWS CLI, or through the AWS Console. Set the names of the buckets (e.g., `ybload-source` and `ybload-zip`, in `deploy.sh`).

Next, you must configure the AWS IDs and secrets that will be used by the Lambda function to read from S3. Create an AWS IAM User, `ybload`, and obtain `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, and `AWS_SESSION_TOKEN` values for that user. Apply these values in `deploy.sh`. Also set the AWS region and AWS account in which you will deploy the Lambda function.

Set the Yellowbrick credentials. `YB_HOST` can be obtained from **Yellowbrick Manager** -> **Instance Management**-> **Host/Port**. Provide the username and password that `ybload` will use to connect to the Yellowbrick instance. You might need to create a new user, which can be done through **Yellowbrick Manager** -> **Instance Management** -> **Access Control** -> **User**.

Specify the database and table where data will be loaded. Use `learn_ybload` for the database name and `public.target` for the table name.

Part 3: Configure `deploy.sh`

Staying in the same folder, create your Lambda function:

```bash
./deploy.sh
```

This script will perform the following:

- Extract `ybload` and its dependencies from the ybtools tarball
- Compile the Lambda function in the file `YBLoadLambda.java`
- Upload the function and `ybload` into a zip file in the `ybload-zip` bucket
- Create an IAM role and permissions for the Lambda function
- Create and configure the function
- Set up a trigger to call the function when a file lands in the `ybload-source` bucket.

Check the output of the script for errors, and correct before proceeding.

Part 4: Configure the Yellowbrick Target Database and Table

After setting up the Lambda function, create a target database and table to receive the data loaded by the Lambda function. For this, use the Query Editor in Yellowbrick Manager, `ybsql`, or your preferred SQL editor.

```sql
CREATE DATABASE learn_ybload;
```

In the new database, create the target table:

```sql
CREATE TABLE public.target(id INT PRIMARY KEY, value VARCHAR(128));
```

Part 5: Load Some Data

Create and save the following data into the file `test.csv`:

```
1, "row 1"
2, "row 2"
3, "row 3"
```

Copy the file to your S3 source bucket to trigger a data load:

```
aws s3 cp test.csv s3://ybload-source/
```

Part 6: Examine the Output of Lambda Function in CloudWatch

The Lambda function logs data loads in AWS CloudWatch. You access the logs from the command line by using the following:

```
aws logs tail /aws/lambda/YBLoadFunction --follow
```

You should see a log output that culminates in the following:

```
[ INFO] SUCCESSFUL BULK LOAD: Loaded 3 good rows from 1 source(s)
```

This signifies the data has been successfully loaded into Yellowbrick.

Part 7: Upserting Data

A powerful feature of `ybload` is its ability to perform upserts (i.e., insert new records or update existing records, avoid duplicates). To demonstrate this, create a new CSV file `test2.csv` with the following entries:

```
1, "row 1"
2, "row 2 altered"
4, "row 4"
```

Loading this data into the table should result in the following data in the `target` table:

```
1, "row 1"
2, "row 2 altered"
3, "row 3"
4, "row 4"
```

The result of loading this file is to ignore the duplicated first record, update the second record and add the fourth record. You need to configure the Lambda function before you can enable this upsert functionality.

Edit `deploy.sh` and set the following:

```
YBLOAD_EXTRA_ARGS="--disable-trust --write-op upsert --key-field-names id"
```

This informs `ybload` to use the `id` field as the key to ensure you avoid loading duplicate records. This also informs `ybload` to perform an upsert write operation. Run `./deploy.sh` once more to update the Lambda function, then copy `test2.csv` into the `ybload-source` S3 bucket and observe the CloudWatch logs. Check that the load is completed without error.

In Yellowbrick Manager or `ybsql` run:

```sql
SELECT * FROM target ORDER BY id;
```

The contents of the table should be:

```
1, "row 1"
2, "row 2 altered"
3, "row 3"
4, "row 4"
```

You have successfully upserted data into the table.

Part 8: Understanding the Strengths and Limitations

Using AWS Lambda with `ybload` provides a convenient, scalable, and robust approach to loading data into Yellowbrick in a way that avoids loading duplicate records. There are limitations to this approach; however, bear in mind that a Lambda function has a limited runtime of 15 minutes. This means the execution of the Lambda function (and your data load) will be stopped if the loading process takes over 15 minutes, resulting in no data being loaded. As a rule, assume that the data will load from S3 at the maximum rate of 70 MB/s on a compute cluster with one small-v2 node. Given this rate, the largest file that can be processed by the function within the Lambda runtime limit is around 60 GB.

> **Note:** In practice, limit the maximum size of a single file to around 50 GB for this configuration.

Even given this limitation, it pays to exploit the horizontal scalability of AWS Lambda by landing many smaller files into the S3 source bucket more often. By doing this, a separate Lambda function will be invoked to process each file, massively increasing the load throughput and reducing the time for data to load in Yellowbrick.

> **Note:** The number of load tasks that can execute at the same time is dictated by Yellowbrick's WLM rules and the network bandwidth available to the compute cluster that performs the load.

If Lambda load tasks get queued, there is a chance that the Lambda function will be halted before `ybload` tries to execute or complete its load. Consider creating a dedicated cluster to process loads from your Lambda function with a simple WLM profile. This WLM profile should have a single pool and an equal minumum and maximum concurrency. As a guide, for a cluster with a single small-v1 compute node, a minimum and maximum WLM concurrency of 8 will saturate the IO to and fro S3. Increase the size of the cluster to increase throughput.

The Lambda function you created cannot only load CSV data, it can also load compressed CSV and Parquet files. Drop compressed CSV files with a `.csv.gz` suffix, or Parquet files with a `.parquet` suffix into the S3 bucket to try it out.

Using Airbyte with Yellowbrick

Airbyte is an open-source platform for extracting and loading data from a wide variety of sources and destinations. Yellowbrick is supported as a destination, which means you can load data from any of the supported sources into Yellowbrick. The Yellowbrick connector supports all modes of synchronization:

- Full Refresh
- Incremental Append
- Incremental Append and Deduplication

In this tutorial you will learn how to configure the Yellowbrick Airbyte connector to load data from a source into Yellowbrick.

Prerequisites

- Access to Airbyte Cloud, or to a self-managed instance of Airbyte. See the how-to guide for details on installing and running Airbyte on the AWS EKS cluster that also runs Yellowbrick.
- Read and write access to an AWS S3 bucket, which acts as the data source.
- Access to the `aws` command line client.
- AWS Access Key ID and AWS Secret Access Key for the S3 source bucket. Ensure that the credentials are associated with a user that has permission to list and read objects from the bucket.
- A Yellowbrick user that has been configured to enable the JSON type and functions and has permission to create a database.
- Yellowbrick version 7 or greater, with support for JSON/JSONB.

Overview

You will perform the following steps:

1. Copy a data file to the S3 source bucket
2. Configure the S3 source connector to read from the bucket
3. Configure the Yellowbrick destination to write to a database
4. Synchronize data across the connection
5. Append and deduplicate data from the source to Yellowbrick

---

Part 1: Preparing the Source

Save the following JSONL data to a file `part1.jsonl`:

```jsonl
{"id": "707754c0-2d1e-4e8a-abd8-0ed4b4a02a5c", "metering_hour": "2025-04-09T17:00:00", "total_vcpu_seconds": 49886}
```

This represents fictitious telemetry data for a cloud service. Copy the file into your S3 bucket:

```sh
aws s3 cp part1.jsonl s3://airbyte-demo-1234/
```

Part 2: Configuring the Airbyte S3 Source Connector

In the Airbyte UI, navigate to *Sources* and enter "S3" into the search dialog. Select the *S3* connector. Enter the S3 bucket name, in this case `airbyte-demo-1234`. Select the *Add* button next to *The list of streams to sync*. Change the format to *JSONL* and give the stream the name "telemetry". Leave the rest of the visible dialogs at their defaults and select *Optional Fields*. Complete the *AWS Access Key ID* and *AWS Secret Access Key* fields to give the connector access to your bucket. Scroll to the bottom of the page and select *Set up source*.

Part 3: Configuring the Airbyte Yellowbrick Destination Connector

On successfully setting up the S3 source connection, Airbyte prompts you to create a connection. Select *Create a Connection* and in the *Search Airbyte Connectors* dialog, enter "Yellowbrick". Select the *Marketplace* tab in the search results and select the *Yellowbrick* connector. Set the *Host* field to the address of your Yellowbrick Instance. You can obtain this from *Yellowbrick Manager → Instance Management → Host/Port*.

While in Yellowbrick Manager, create a database to receive the data from Airbyte. The database must be UTF-8 encoded:

```sql
CREATE DATABASE airbyte_test ENCODING=UTF8;
```

In the Airbyte console, continue to complete the configuration of the Yellowbrick destination connector by entering the name of the database created previously in the *DB Name* field.

Add your Yellowbrick user name in the *User* field. Then, under *Optional fields* enter the password associated with the user. Finally, select *Set up connection*.

Part 4: Synchronize Data Between Source and Destination

The Airbyte console will prompt you to select the mode of synchronization. In this instance, under the *Schema* pane, select *Full Refresh | Overwrite* from the *sync mode* dropdown. Select *Next* to proceed. Finally select *Finish & Sync*.

Airbyte will proceed to copy data from the S3 bucket into a table in the `airbyte_test` database that it will create. The name of the table will be the same as the name of our source stream in the S3 connector — `telemetry`.

Examine the data in the `telemetry` table in Yellowbrick Manager by executing `SELECT * FROM telemetry;` in the `airbyte_test` database:

id	metering_hour	total_vcpu_seconds	_ab_source_file_url	_ab_source_file_last_modified	_airbyte_raw_id	_airbyte_extrac
707754c0-2d1e-4e8a-abd8-0ed4b4a02a5c	2025-04-09T17:00:00	49886	part1.jsonl	2025-04-16T12:24:45.000000Z	5dc603de-8292-4539-a13e-bf6dff9a0bb3	2025-04-16T12:58:01.16

You will see that the JSONL source data has been flattened into a relational schema by the destination connector. You will also see that in addition to the three fields in our source, other metadata fields have been added by Airbyte.

Part 5: Appending and Deduping Data

The Yellowbrick connector supports appending new data, in addition to fully refreshing the data in the table. To configure appends, navigate to *Connections → Schema* in Airbyte. Change the sync mode to *Incremental | Append + Dedupe*. You must also set a primary key. Airbyte needs this in order to avoid adding duplicate data to the table. Select the `id` and `metering_hour` as our compound primary key. Save the changes. Select the *Sync now* at the top of the Airbyte window. After the sync has completed, you will notice that the `telemetry` still only contains a single record. In Append mode, Airbyte tracks the last file loaded from S3 and doesn't reload previously loaded files.

Create a new file `part5.jsonl` containing a new telemetry record:

```jsonl
{"id": "707754c0-2d1e-4e8a-abd8-0ed4b4a02a5c", "metering_hour": "2025-04-09T18:00:00", "total_vcpu_seconds": 49886, "num_vcpus": 1
```

Notice that, in addition to an updated metering hour, the data contains a new field `num_vcpus`. Copy this file into your S3 source bucket. In Airbyte, navigate to *Connections* → *Schema* and refresh the source schema. Select *Sync now* once more.

Once the sync has finished, examine the contents of the `telemetry` table:

id	num_vcpus	metering_hour	total_vcpu_seconds	_ab_source_file_url	_ab_source_file_last_modified	_airbyte_raw_id
707754c0-2d1e-4e8a-abd8-0ed4b4a02a5c	16	2025-04-09T18:00:00	49886	part5.jsonl	2025-04-16T14:18:02.000000Z	0087fe05-260d-4639-87c4-737642a9913b
707754c0-2d1e-4e8a-abd8-0ed4b4a02a5c		2025-04-09T17:00:00	49886	part1.jsonl	2025-04-16T12:24:45.000000Z	83f3f293-3899-4d5e-a34c-7aee40a84df0

You will see that, in addition to a second record in the table, the table also has a new column, `num_vcpus`. Airbyte has altered the schema of the destination table in Yellowbrick to match the new source column automatically.

See the Airbyte documentation for information on setting up scheduled synchronization, and connecting Yellowbrick to other Airbyte sources.

Setting up Yellowbrick as a Source in Databricks Unity Catalog

Yellowbrick Documentation > Tutorials > Databricks Unity Catalog

Platforms: EE: All cloud platforms

Parent topic: Tutorials

Overview

Databricks Unity Catalog allows users to map connections to external data sources, such as Yellowbrick. This enables queries from a Databricks notebook or SQL Warehouse into the Yellowbrick database. One of the most common uses for this mapping is for data scientists to bring cleansed and vetted "Gold" data from Yellowbrick back into Databricks to generate predictive models or run AI/ML pipelines.

Limitations

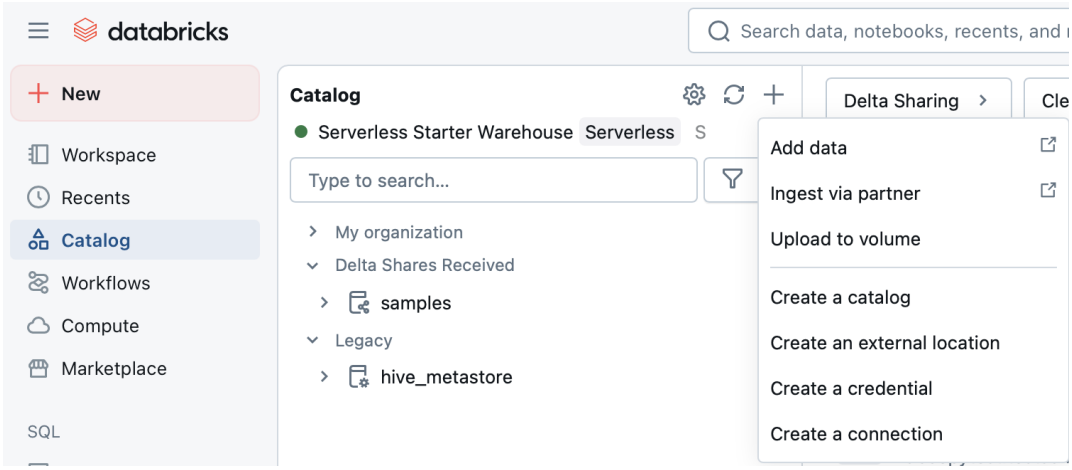
- DDL and DML statements are not permitted (e.g., `INSERT`, `UPDATE`, `DELETE`, `CREATE`, `DROP`).
- Joins are not pushed down to Yellowbrick. All data involved in joins is moved to Databricks and joined there.

Prerequisites

- If Yellowbrick is installed in a private VPC, the VPCs for Yellowbrick and Databricks must be peered. See VPC Peering for more information.

Detailed Steps

1. From the **Catalog** sidebar in the Databricks console, click the `+` button, then select **Create a Connection**.



2. On the **Connection Basics** tab:
   - **Name**: `[Name for your connection]`
   - **Connection Type**: `PostgreSQL`
   - Click **Next**.

3. On the **Authentication** tab:
   - **Host**: `[Your FQDN for Yellowbrick Instance, e.g., yb_prod.elb.us-east-1.amazonaws.com]`
   - **Port**: `5432` (or your custom port if changed)
   - **User**: `[Username]`
   - **Password**: `[Password]`
   - Click **Next**.
4. On the **Catalog Basics** tab:
   - **Catalog Name**: `[Name for the new DB Catalog representing the connection]`
   - **Database**: `[Yellowbrick Database Name]`
   - Click **Test**:
     - Choose your compute if one exists, or start a serverless compute instance.
     - Ensure the connection is successful.
   - Click **Create Catalog**.
5. On the **Access** tab:
   - Keep the defaults or modify as needed.
   - Click **Next**.
6. On the **Metadata** tab:
   - Add metadata for Databricks documentation if needed.
   - Click **Save**.

Using the New Catalog

The catalog can now be used as a read-only data source. It can be referenced via notebooks or SQL queries like any other table in the Databricks Unity Catalog. As queries hit this new catalog, you should see the corresponding queries executed against the Yellowbrick compute cluster.

Using *Ask TK* to Generate SQL

Yellowbrick Documentation > Tutorials > Generate SQL with "Ask TK"

Platforms: EE: All cloud platforms

Parent topic: Tutorials

This tutorial explains how to use the *Ask TK* function in Yellowbrick Manager to generate SQL from natural language questions. *Ask TK* uses an external large language model to interpret the question, together with additional context about the database schema, to generate a SQL expression. The purpose of this feature is to provide a SQL starting point that can be validated and built upon. It's a productivity tool, rather than a tool for generating SQL that can be blindly trusted.

Prerequisites

- Permission to connect *Ask TK* to external API endpoints. A user with the `consumeradmin` role must configure this for you. This how-to guide details the steps needed to do this.
- The URL for the **OpenAI-compatible** API endpoint to connect to. This could be a locally running service hosted by Ollama, or a commercial service such as OpenAI.
- An API key for the service. This tutorial expects you to use OpenAI as the LLM provider, but you can complete the tutorial using a different service, if required.
- The name of the LLM to connect to and the name of an embedding model to use. For this tutorial you will use the `gpt-4o` and `text-embedding-3-small` models.
- A populated database schema to ask questions against. In this tutorial you will use the NOAA sample data set.
- A running Yellowbrick compute cluster. A single compute node will suffice.

Overview

In this tutorial you will accomplish the following tasks:

1. Configure the connection to the LLM
2. Load the sample NOAA data set
3. Configure the context that you will provide with your questions
4. *Ask TK* a question about the sample data
5. Use the embedding model to find relevant tables
6. Tune the LLM prompt to add additional instructions

Part 1: Configure the LLM Connection

To configure *Ask TK* to use an external LLM service, navigate to *Hi 'User' → Profile → Ask TK* from the menu at the top right. Start by entering your API key, model name, and the URL for the API service. Be sure to specify the complete URL including the protocol, for example `https://api.openai.com`. *Ask TK* expects the endpoint to be compatible with the OpenAI API. Note, if you are using a model hosted by Ollama, then the key can be any arbitrary string, but a key must always be provided.

Decide on the number of Most Common Values to use. The default is 100. This value is used to determine the maximum number of sampled values from each column in the schema to send to the LLM to help it generate the SQL. For this tutorial, leave the value at 100. For other data sets you might decide that you don't want potentially sensitive data to be sent to an external LLM. In this case, set the value here to 0.

Part 2: Loading the Sample Data

The NOAA data set contains weather observations stretching back several hundred years. The schema consists of four main tables: `countries`, `observations`, `stations`, and `states`. You will load the sample data into these tables and then use *Ask TK* to generate SQL from natural language questions posed about the data.

Navigate to *Query Editor → Sample Catalog → NOAA data set*. Click the Setup button. If you have never loaded sample data before, Yellowbrick Manager will offer to create a database in which to load the data. You can choose to accept this if the `yellowbrick_trial` database doesn't already exist. You can also decide to load it into a different database. You will need to create this database first and configure Yellowbrick Manager to use this database before clicking the Setup button.

The Query Editor will load the setup script into a new tab. Run this script to set up the NOAA schema in the current database. Once the setup script has finished running, navigate back to *Sample Catalog → NOAA* and click the Load button. This will open a new tab in the Query Editor containing the load script. Run the load script. The NOAA data set is quite large and takes around 10 minutes to load using a compute cluster with a single SMALL compute node.

Create a new tab in the Query Editor. The SQL output from the LLM will be pasted to this tab in response to a question.

Part 3: Configuring the Prompt Context

To generate a more useful response to a question, you must provide the LLM with information about the database schema that you want to analyze. "Ask TK" has the ability to inject schema information into the prompt that will be sent to the LLM to help it generate a SQL expression. This schema information includes table names, column definitions and sample values.

To set up the context ready to ask a question, you must first ensure that the correct database and search path are set in the Query Editor. The search path must be set. For this data set, set the search path to `noaa_ghcn_pds` and then navigate to *Ask TK*. The *Ask TK* dialog enables you to submit a question, view and modify the generated prompt, and configure which tables should be included in the prompt.

To select the tables to include, navigate to the *Tables* menu at the top left of the *Ask TK* dialog and select `observations`, `states` and `stations`.

Part 4: Ask a Question

Enter the following question into the *Ask TK* dialog:

```
What was the total rainfall in CARY, NC in inches on January 10th 2024? Note that NOAA rain observations are recorded in tenths of
```

Click *Generate SQL*. After a few seconds, the LLM will paste a SQL expression into the current Query Editor tab. Close the *Ask TK* dialog to view the query:

```sql
SELECT
    SUM(o.data_value) / 254.0 AS total_rainfall_inches
FROM
    noaa_ghcn_pds.observations o
JOIN
    noaa_ghcn_pds.stations s ON o.id = s.id
WHERE
    s.name = 'CARY'
    AND s.state = 'NC'
    AND o.year_date = 20240110
    AND o.element = 'PRCP';
```

See how gpt-4o in this case has used sample values from the data to infer the correct predicates to use and the table schema to select the correct join conditions between the `observations` and `stations` table.

Run the query. We can validate the result by using the NOAA website. From this NOAA webpage, select the month of January and the year of 2024 to retrieve the actual daily weather observations for the Cary, NC weather station. You will see that the rainfall on January 10th was 1.51 inches, which matches the result of our SQL query.

Navigate to *Ask TK → View Generated Prompt* to review the prompt sent to the LLM. You will see the database schema information on the three selected tables included in the prompt. The dialog also features a prompt template. This can be modified and reset back to the default as needed. Note that each query tab has it's own *Ask TK* dialog, so you can

ask different questions and configure different prompts across query tabs.

Part 5: Automatically pick Relevant Tables

For schemas with many tables, rather than manually selecting which tables to include in the prompt, you can ask an AI model to help select what it thinks are the most relevant tables for you. *Ask TK* does this by creating vector embeddings by using the provided embedding model for the question and each table name in the schema, and then performing a cosine similarity search to determine the tables that most closely match the question.

Navigate to *Hi 'User'* → *Profile* → *Ask TK* to set the embedding model and the maximum number of relevant tables to return. For OpenAI, use the `text-embedding-3-small` model. Set the maximum number of tables to consider to 2. Generate a new SQL query and view the generated prompt. With the embedding model configured you will see that the two tables that most closely match the question have been included in the prompt instead of the three tables we manually selected in Part 3.

Part 6: Tuning the Prompt

While the default prompt provides good results generally, there might be cases where it is beneficial to provide the LLM with additional information to help it return better queries. Additional context can be added to the question, as you did in Part 4 when you told the LLM that NOAA rainfall totals are measured in tenths of a millimeter. The *Ask TK* prompt dialog can be edited to provide this additional context without having to add it to each question.

Each query tab can have a different prompt. In addition, the default prompt template can be altered for all query tabs by navigating to *Hi 'User'* → *Profile* → *Ask TK Advanced* and making modifications to the prompt template there.

Test this in a new query tab by appending the following to the prompt:

```
Note that NOAA rain observations are recorded in tenths of a millimeter.
```

and asking the question:

```
What was the total rainfall in CARY, NC in inches on January 10th 2024?
```

Ask TK should generate a SQL expression that returns the rainfall total in inches.

Conclusion

Congratulations for completing this tutorial. You have learned how to configure *Ask TK* to generate SQL expressions from natural language queries related to your databases in Yellowbrick!

Using the Geospatial Functions (Pre-release)

Yellowbrick Documentation > Tutorials > Using Geospatial Functions

Platforms: All platforms

Parent topic: Tutorials

Yellowbrick supports PostgreSQL-standard geospatial functions that enable spatial queries. This tutorial will guide you through, using some of these functions with NOAA's GHCN (Global Historical Climatology Network) weather data set. This tutorial will also specifically analyze data from observation stations within a 20 km radius of Raleigh, North Carolina, USA.

INFO

The geospatial functions are in public preview now, and so are disabled by default. The functions can be enabled with the configuration variable `enable_geospatial`, which can be applied at the system-level, session-level, user-level or query-level.

In this tutorial, you will incrementally build queries by using basic to advanced geospatial functions.

Prerequisites

- Pre-loaded NOAA data

  > **Note:** To load the NOAA data, go to page Using Ask TK to Generate SQL, Part 2: Loading the Sample Data and follow the steps to load the data using Yellowbrick Manager.

- The configuration variable `enable_geospatial` must be set to ON.

Section 1: Understanding Spatial Data in Yellowbrick

The `stations` table stores the locations of weather stations. These locations are stored in the form of a latitude and longitude field for each station. To convert the station locations into spatial data types, run the following SQL statement:

Query: Converting Station Locations to Geospatial Points

```sql
SELECT
    id,
    name,
    ST_AsText(ST_Point(longitude, latitude)) AS geom
FROM noaa_ghcn_pds.stations
WHERE state = 'NC';
```

Explanation

- `ST_Point(longitude, latitude)`: Converts longitude and latitude into a geometry point.
- Filters stations only in North Carolina (NC).

Section 2: Finding Stations Within a 20 km Radius of Raleigh, NC

Raleigh, NC has an approximate latitude and longitude of (35.7796, -78.6382).

Query: Find Nearby Weather Stations

```sql
SELECT
    st.id,
    st.name,
    st.latitude,
    st.longitude
FROM noaa_ghcn_pds.stations st
WHERE ST_DWithin(
    ST_GeogFromText('SRID=4326;POINT(-78.6382 35.7796)'),  -- Raleigh, NC
    ST_GeogFromText('SRID=4326;POINT(' || st.longitude || ' ' || st.latitude || ')'),
    20000  -- 20 km radius
);
```

Explanation

- `ST_GeogFromText('SRID=4326;POINT(-78.6382 35.7796)')`: Defines `Raleigh, NC` as a point in SRID 4326 (WGS 84).
- `ST_DWithin(A, B, 20000)`: Returns `true` if station B is within 20,000 meters (20 km) of A.

Section 3: Checking If Raleigh Covers a Weather Station

Instead of checking if a station is within a radius, you will check if Raleigh's geographical boundary covers a station.

Query: Checking If Raleigh Covers a Station

```sql
SELECT
    st.id,
    st.name,
    ST_Covers(
        ST_Buffer(ST_GeogFromText('SRID=4326;POINT(-78.6382 35.7796)'), 20000),
        ST_GeogFromText('SRID=4326;POINT(' || st.longitude || ' ' || st.latitude || ')')
    )::BOOLEAN AS is_covered
FROM noaa_ghcn_pds.stations st;
```

Explanation

- `ST_Buffer(Raleigh, 20000)`: Creates a 20 km radius boundary around Raleigh.
- `ST_Covers(A, B)`: Checks if Raleigh's buffered area accommodates the station's point.

Section 4: Finding Precipitation Observations in the Area

Now, let's find the PRCP (precipitation) observations from the stations within 20 km of Raleigh.

Query: Get PRCP Observations in the Area

```sql
SELECT
    obs.id,
    st.name,
    obs.element,
    obs.data_value / 254.0 as rainfall_inches
FROM noaa_ghcn_pds.observations obs
JOIN noaa_ghcn_pds.stations st
    ON obs.id = st.id
WHERE obs.element = 'PRCP'
AND ST_DWithin(
    ST_GeogFromText('SRID=4326;POINT(-78.6382 35.7796)'),
```

```
    ST_GeogFromText('SRID=4326;POINT(' || st.longitude || ' ' || st.latitude || ')'),
    20000
)
AND obs.year_date BETWEEN 20240101 AND 20241231;
```

Explanation

- Finds PRCP (precipitation) observations for the year 2024 (BETWEEN 20240101 AND 20241231).
- Uses `ST_DWithin(A, B, 20000)` to filter only stations within 20 km of Raleigh.

Section 5: Intersecting a Custom Polygon Boundary (Advanced)

Let's define a polygonal area around Raleigh and find stations inside it.

Query: Checking Intersections with a Polygon

```sql
SELECT
    st.id,
    st.name,
    ST_Intersects(
        ST_GeogFromText('SRID=4326;POLYGON((-78.7 35.75, -78.6 35.75, -78.6 35.80, -78.7 35.80, -78.7 35.75))'),
        ST_GeogFromText('SRID=4326;POINT(' || st.longitude || ' ' || st.latitude || ')')
    ) AS intersects_polygon
FROM noaa_ghcn_pds.stations st
WHERE intersects_polygon;
```

Explanation

- Defines a polygon boundary around Raleigh.
- Uses `ST_Intersects(A, B)`: Returns `true` if a station's point B intersects the polygon A.

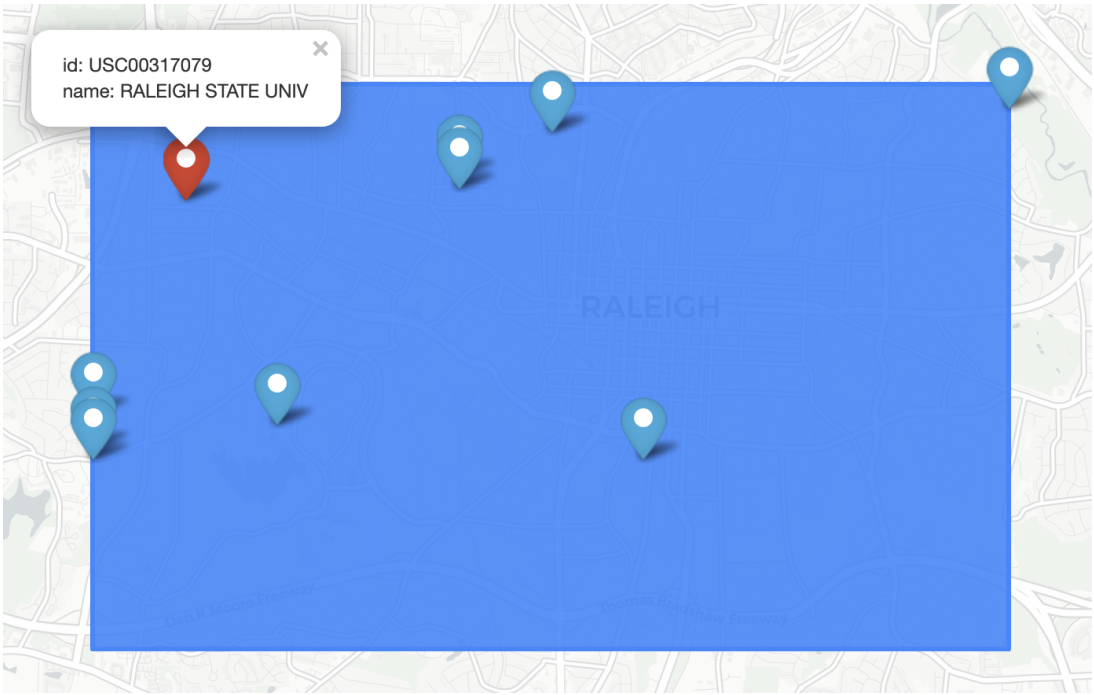
Section 6: Visualizing the Spatial Data

The prior result set can be rendered in a tool such as DBeaver Enterprise Edition or JetBrains DataGrip. However, this result set requires some modifications to the SQL statements. It is to account for the limitation that these tools only support the `GEOMETRY` spatial data type, and Yellowbrick only supports the `GEOGRAPHY` data type. In DataGrip, connect to Yellowbrick by using the PostgreSQL driver and database containing the NOAA data, and run the following SQL statement:

```sql
SELECT
    st.id,
    st.name,
    ST_AsText(ST_GeogFromText('SRID=4326;POLYGON((-78.7 35.75, -78.6 35.75, -78.6 35.80, -78.7 35.80, -78.7 35.75))')) boundary,
    ST_AsText(ST_GeogFromText('SRID=4326;POINT(' || st.longitude || ' ' || st.latitude || ')')) stations
FROM noaa_ghcn_pds.stations st
where ST_Contains(boundary::geography, stations::geography)
```

Switch to DataGrip's **Geo Viewer** to display the bounding polygon around Raleigh and the stations contained within.

> **Note:** Convert the boundary and stations `GEOGRAPHY` values into their OGC Well-Known Text (WKT) values. It allows rendering within Geo Viewer and filter on those stations contained within the boundary.



Function Description Table

Query Goal	Function Used	Description
Convert coordinates to geospatial points	`ST_Point()`	Prepares data for geospatial queries
Find stations within 20 km	`ST_DWithin()`	Radius-based filtering
Check if Raleigh's area covers a station	`ST_Covers()`	Works with buffered regions
Get PRCP observations near Raleigh	`ST_DWithin()`	Joins weather data with station locations
Find stations inside a polygon	`ST_Intersects()`	Works with custom boundaries
Render spatial data	`ST_Contains()`	Displays stations within a polygon

Using Python with Yellowbrick

In this tutorial, you will learn how to set up a connection to a Yellowbrick database from Python and pull data into a Pandas DataFrame. You will install and use PostgreSQL-compatible libraries for Python to set up the connection and query data in Yellowbrick.

Prerequisites

- Yellowbrick credentials (host, port, username, password).
- Pre-loaded NOAA data in your Yellowbrick instance (for example, in the noaa_ghcn_pds.observations table). If you need to load the NOAA sample data, see these instructions.

Overview

We will walk through the following steps:

1. Installing the required libraries.
2. Writing a Python script to connect to Yellowbrick and run SQL directly (using psycopg2).
3. Writing a Python script to query into Pandas (using pd.read_sql).

Part 1: Installing Python and Supporting Libraries

Linux (Ubuntu/Debian)

Install Python 3 and pip:

```
sudo apt update
sudo apt install python3 python3-pip
```

Install psycopg2 (PostgreSQL driver):

```
pip3 install psycopg2
```

Install Pandas and NumPy:

```
pip3 install pandas numpy
```

macOS

Install Python (via Homebrew):

```
brew install python
```

Install psycopg2:

```
pip3 install psycopg2
```

Install Pandas and NumPy:

```
pip3 install pandas numpy
```

Windows

Install Python:

Download from python.org or via the Microsoft Store. Make sure to select the “Add Python to PATH” option.

Install psycopg2:

```
pip install psycopg2
```

Install Pandas and NumPy:

```
pip install pandas numpy
```

If you already have Python, ensure you have the latest versions of psycopg2, pandas, and numpy.

Part 2: Query Yellowbrick from Python Using Pure SQL

Create a Python script, for example `test.py`, to connect to Yellowbrick and query the 10 most recent records:

```
import psycopg2

# Modify these with your actual Yellowbrick connection details
dbname = "yellowbrick_trial"   # Replace with your Yellowbrick database containing NOAA data
host = ""                      # Complete with your Yellowbrick hostname
port = 5432                    # Default Yellowbrick Postgres port
user = ""                      # Complete with your Yellowbrick username
password = ""                  # Complete with your Yellowbrick password

query = """
SELECT *
FROM noaa_ghcn_pds.observations
ORDER BY year_date DESC, id, element
LIMIT 10;
"""

# Connect to Yellowbrick using psycopg2
conn = psycopg2.connect(
    dbname=dbname,
    host=host,
    port=port,
    user=user,
    password=password
)

# Create a cursor and execute the query
cur = conn.cursor()
cur.execute(query)
```

```
# Fetch the results
rows = cur.fetchall()

# Get column names
column_names = [desc[0] for desc in cur.description]

# Print output
print("Most recent 10 records:")
print(column_names)
for row in rows:
    print(row)

# Clean up
cur.close()
conn.close()
```

How to run:

```
python test.py
```

Explanation:

- We create a connection to Yellowbrick (which speaks PostgreSQL protocol).
- We execute the SQL query ordering by year_date DESC and limiting to 10.
- We fetch the results with fetchall().
- We close the cursor and connection.

Part 3: Query Yellowbrick into a Pandas DataFrame

While the above approach works fine, you can also leverage Pandas for a simpler DataFrame workflow. Create a second script, for example `test_pandas.py`:

```
import psycopg2
import pandas as pd

# Modify these with your actual Yellowbrick connection details
dbname = "yellowbrick_trial"   # Replace with your Yellowbrick database containing NOAA data
host = ""                      # Complete with your Yellowbrick hostname
port = 5432
user = ""                      # Complete with your Yellowbrick username
password = ""                  # Complete with your Yellowbrick password

query = """
SELECT *
FROM noaa_ghcn_pds.observations
ORDER BY year_date DESC, id, element
LIMIT 10;
"""

# Connect using psycopg2
conn = psycopg2.connect(
    dbname=dbname,
    host=host,
    port=port,
    user=user,
    password=password
)

# Load directly into a Pandas DataFrame
df = pd.read_sql(query, conn)
```

```
# Print DataFrame
print("Dataframe of the 10 most recent records:")
print(df)

# Clean up connection
conn.close()
```

How to run:

```
python test_pandas.py
```

Explanation:

- `pd.read_sql` automatically creates a DataFrame from the query results.
- We see the same 10 records, but now in a Pandas DataFrame format.

Optional: sqlalchemy Approach

If you prefer, you can install sqlalchemy (`pip3 install sqlalchemy`) and use a connection URL like:

```
from sqlalchemy import create_engine
import pandas as pd

engine = create_engine(
    'postgresql://<user>:<password>@<host>:<port>/<dbname>'
)

df = pd.read_sql("SELECT ...", engine)
```

But psycopg2 + read_sql is perfectly fine for straightforward use cases.

NOTE

You must override the Yellowbrick database version to use sqlalchemy:

```
ALTER SYSTEM SET override_version='PostgreSQL 9.5.10 on x86_64-redhat-linux-gnu';
SELECT pg_reload_conf();
```

Conclusion

You've successfully:

1. Installed Python and the PostgreSQL client libraries needed for Yellowbrick.
2. Connected to Yellowbrick using psycopg2 to run a direct SQL query.
3. Queried data from Yellowbrick into a Pandas DataFrame for further analysis or visualization.

With these building blocks, you can continue exploring analyses, visualizations (e.g., using matplotlib), and more advanced data manipulations using Python and Yellowbrick! Happy coding!

Using R with Yellowbrick

Platforms: All platforms

Parent topic: Tutorials

In this tutorial, you will learn how to set up a connection to a Yellowbrick database from a script in R, and pull data into a DataFrame. You will install and use the PostgreSQL libraries for R to set up the connection and access data.

Prerequisites

- Credentials to connect to a running Yellowbrick instance by using the command line tools.
- Pre-loaded NOAA data. To load the NOAA data, go to page Using Ask TK to Generate SQL, Part 2: Loading the Sample Data and follow the steps to load the data using Yellowbrick Manager.

Overview

In this tutorial you will accomplish the following tasks:

1. Install the required libraries.
2. Write a script in R to directly query Yellowbrick using SQL.
3. Write a script in R that uses `dplyr` to query Yellowbrick.

Part 1: Installing R and Supporting Libraries

If installing R on a Linux client (e.g. Ubuntu), install the following:

- Install R: `sudo apt install r-base`
- Install libpq: `sudo apt install libpq-dev`

If installing R on MacOS, install the following:

- Install R: `brew install r`
- Install libpq: `brew install libpq`

If installing R on Windows, install the following:

- Install R: See instructions
- Install PostgreSQL CLI tools

Part 2: Query Yellowbrick from R using SQL

Create an R script, `test.r`, that will connect to your Yellowbrick instance and extract the most recent 10 records into a DataFrame:

```
options(repos = c(CRAN = "https://cran.rstudio.com/"))
install.packages("RPostgres")
install.packages("DBI")
```

```
library(DBI)
library(RPostgres)

dbname <- "yellowbrick_trial" # Replace with the database containing the NOAA data
host <- "" # Complete with your Yellowbrick hostname
port <- 5432
user <- "" # Complete with your Yellowbrick username
password <- "" # Complete with your Yellowbrick password
query <- "SELECT * FROM noaa_ghcn_pds.observations ORDER BY year_date DESC, id, element LIMIT 10;"

con <- dbConnect(Postgres(),
                 dbname = dbname,
                 host = host,
                 port = port,
                 user = user,
                 password = password,
                 sslmode = "require")

result <- dbGetQuery(con, query)

print(result)

dbDisconnect(con)
```

Run the script as follows:

```
Rscript test.r
```

> **Note:** You might have to run `sudo Rscript test.r` for the first time to download and install the packages system-wide on Linux. You can comment out the `options` and `install` lines in the script for subsequent executions.

You will see output of the form:

```
           id year_date element data_value m_flag q_flag s_flag obs_time
1  BF1CS000002  20250223    PRCP          0   <NA>   <NA>      N     <NA>
2  BF1CS000002  20250223    SNOW          0   <NA>   <NA>      N     <NA>
3  BF1FP000001  20250223    PRCP          0   <NA>   <NA>      N     <NA>
4  BF1FP000001  20250223    SNOW          0   <NA>   <NA>      N     <NA>
5  BF1NP000001  20250223    PRCP          0   <NA>   <NA>      N     <NA>
6  BF1NP000001  20250223    SNOW          0   <NA>   <NA>      N     <NA>
7  BF1NP000011  20250223    PRCP          0   <NA>   <NA>      N     <NA>
8  BF1NP000011  20250223    SNOW          0   <NA>   <NA>      N     <NA>
9  BF1NP000013  20250223    PRCP          0   <NA>   <NA>      N     <NA>
10 BF1NP000013  20250223    SNOW          0   <NA>   <NA>      N     <NA>
```

Part 3: Query Yellowbrick from R using `dplyr`

Yellowbrick also supports `dplyr` as a SQL abstraction layer in R. Create a file `test2.r` containing the following:

```
options(repos = c(CRAN = "https://cran.rstudio.com/"))
install.packages("dplyr")
install.packages("dbplyr")
install.packages("DBI")
install.packages("RPostgres")

library(DBI)
library(RPostgres)
library(dplyr)
library(dbplyr)
```

```
dbname <- "yellowbrick_trial" # Replace with the database containing the NOAA data
host <- "" # Complete with your Yellowbrick hostname
port <- 5432
user <- "" # Complete with your Yellowbrick username
password <- "" # Complete with your Yellowbrick password

con <- dbConnect(Postgres(),
                 dbname = dbname,
                 host = host,
                 port = port,
                 user = user,
                 password = password,
                 sslmode = "require")

observations <- tbl(con, in_schema("noaa_ghcn_pds", "observations"))

query_result <- observations %>%
  arrange(desc(year_date), id, element) %>%
  head(10) %>%
  collect()

print(query_result)

dbDisconnect(con)
```

You will see output in the following form:

```
   id          year_date element data_value m_flag q_flag s_flag obs_time
   <chr>           <int> <chr>        <dbl> <chr>  <chr>  <chr>  <chr>
 1 BF1CS000002  20250223 PRCP             0 NA     NA     N      NA
 2 BF1CS000002  20250223 SNOW             0 NA     NA     N      NA
 3 BF1FP000001  20250223 PRCP             0 NA     NA     N      NA
 4 BF1FP000001  20250223 SNOW             0 NA     NA     N      NA
 5 BF1NP000001  20250223 PRCP             0 NA     NA     N      NA
 6 BF1NP000001  20250223 SNOW             0 NA     NA     N      NA
 7 BF1NP000011  20250223 PRCP             0 NA     NA     N      NA
 8 BF1NP000011  20250223 SNOW             0 NA     NA     N      NA
 9 BF1NP000013  20250223 PRCP             0 NA     NA     N      NA
10 BF1NP000013  20250223 SNOW             0 NA     NA     N      NA
```

Now, the R script extracts the most recent 10 records from the Yellowbrick table, this time by using `dplyr`.

Conclusion

You have learned how to install the R environment and connect to Yellowbrick to extract data into an R DataFrame from a table in Yellowbrick.

Happy calculating!

VPC Peering Step-By-Step Tutorial

Yellowbrick Documentation > Tutorials > VPC Peering Tutorial

Platforms: EE: All cloud platforms

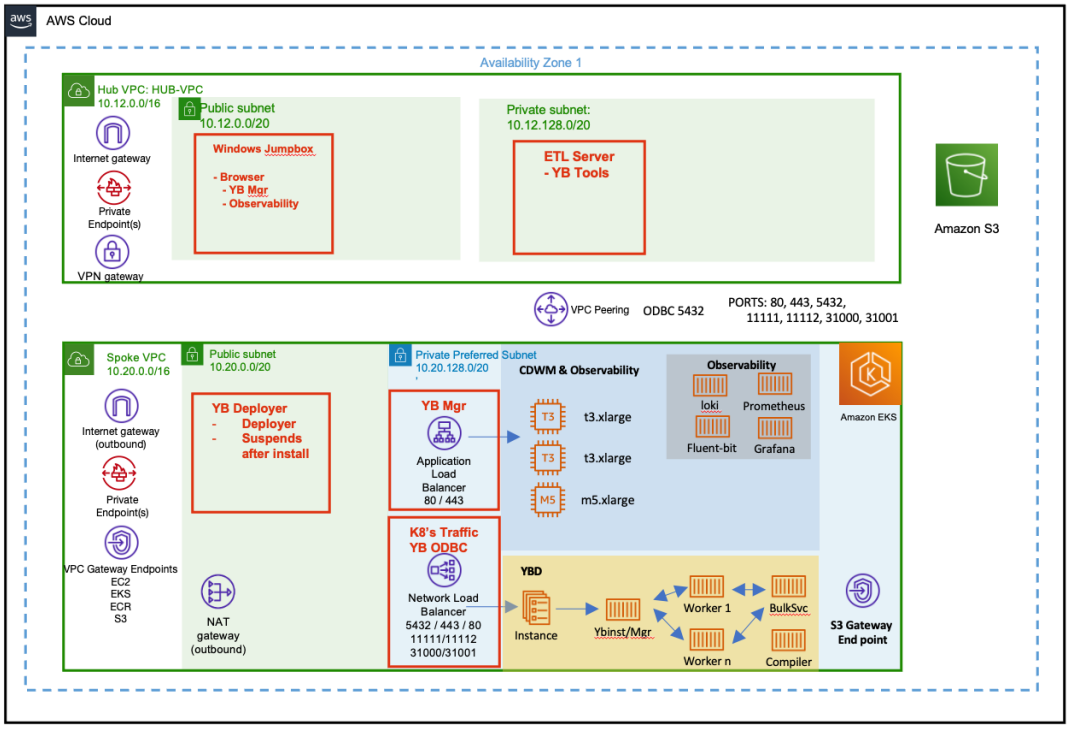
Parent topic: Tutorials

This tutorial will create the example hub and spoke architecture shown below in AWS. It will:

1. Create a spoke VPC with the Yellowbrick Deployer installed in a public subnet and the Yellowbrick Database installed in the private subnet
2. Create a Hub VPC where you will have a Windows Jumpbox in a public subnet. From here, you'll be able to access the Yellowbrick Manager
3. Create a private subnet within the Hub VPC where you can have your data application (Tableau, Microstrategy, etc) and ETL tools (YBTools)

The architecture below is what you will have after following the step-by-step tutorial.

Hub & Spoke Architecture – ETL / Data App in Hub, Yellowbrick in Spoke VPC



Prerequisites

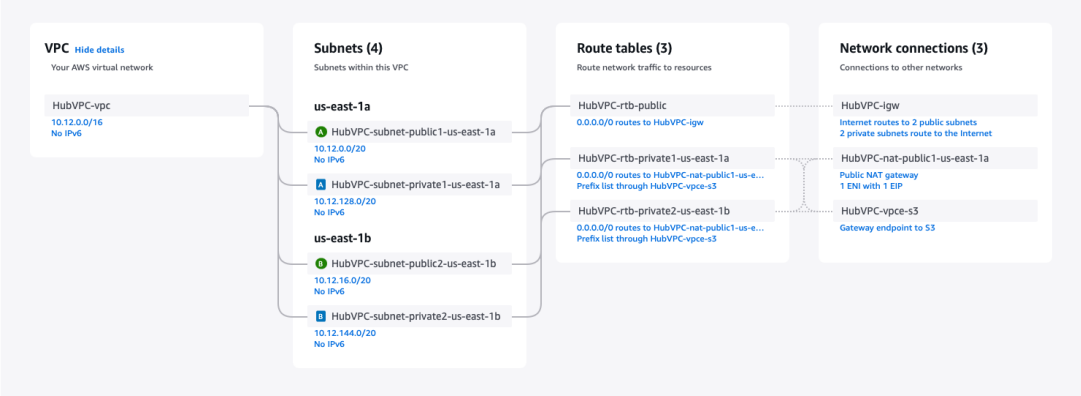
Running the Deployer on AWS requires permission to create a virtual network, a security group, an EC2 instance and custom IAM roles. Although not mandatory, it is highly recommended to use a dedicated AWS account to simplify permission setup and ensure resource isolation.

Create the VPCs

1) Create the Hub VPC

- Go to VPC in AWS, click "Create VPC". Set the following fields (everything else stays default):
- Select **VPC and More**
- **Name tag auto-generation:** `HubVPC`
  - Checkbox **Auto-generate**
- **IPv4 CIDR block:** `10.12.0.0.16`
- **NAT gateway:** `In 1 AZ`

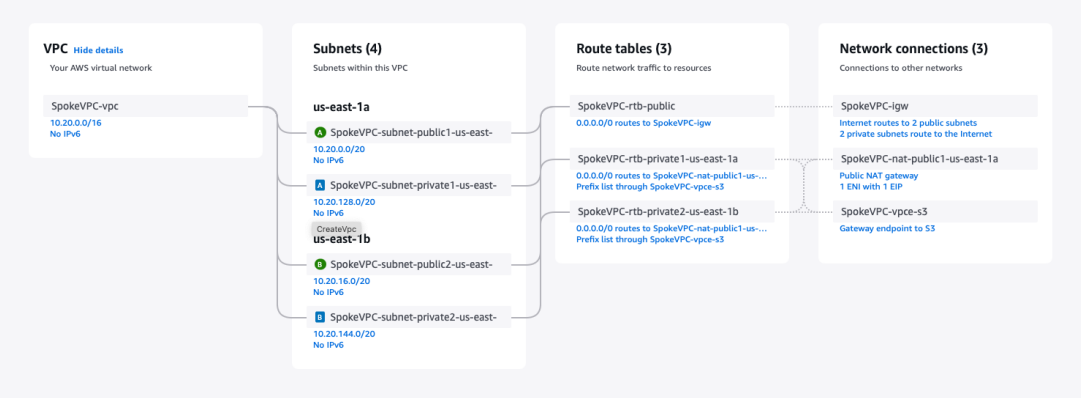
It should look like this:



2) Create the Spoke VPC

- Go to VPC in AWS, click "Create VPC". Set the following fields (everything else stays default):
- Select **VPC and More**
- **Name tag auto-generation:** `SpokeVPC`
  - Checkbox **Auto-generate**
- **IPv4 CIDR block:** `10.20.0.0.16`
- **NAT gateway:** `In 1 AZ`

It should look like this:



3) Add the "Primary" tag on the private subnet within the SpokeVPC

- In the Resource Map for the SpokeVPC-vpc VPC, click on the **SpokeVPC-subnet-private1-[REGION]** subnet
- Once in the subnet details page, click the **Tags** tab
- Hit **Manage Tags** button on far right
- Hit the **Add new tag** button
- Set the tag detail
  - **Enter Key:** `primary`
  - **Enter Value:** `true`
- Hit **Save**
- For more detail see: Private Install -> Creating VPC Network -> Subnet Configuration.

Install Yellowbrick Deployer into public subnet of SpokeVPC

1) Ensure the minimum EC2 quota levels are available

- Go to Service Quotas
- Under "AWS Services" type EC2 in the search bar
- Select "Amazon Elastic Compute Cloud (Amazon EC2)"
- In the "Search by quota name" field type "Running On-Demand Standard"
- Ensure the account-level quota value is greater than 46
  - If it isn't, click the link for "Running On-Demand Standard" and in the upper right corner, request an increase.

2) Pick Cloud Formation template

- Go to Cloud Formation and Select Stacks
- Hit the button to Create Stack (with new resources - standard)
- Keep the defaults: Choose an exiting template and AmazonS3 URL
- Copy the Deployer URL from the cloud documentation here: Cloud Install
- **Amazon S3 URL:** `[COPIED URL EXAMPLE: https://yb-installer-prod.s3.amazonaws.com/deploy/[VERSION NUMBER]/deploy-enterprise.json]`
- Hit **NEXT** Button

3) Specify stack details: (Other defaults are OK)

- **Stack name:** `yb-deployer-vpc-peering`
- **VPC:** `SpokeVPC-vpc`
- **Subnet:** `SpokeVPC-subnet-public1-[region]` where region might be us-east-1a if installed in us-east-1
- Hit **NEXT** Button
- All defaults on this page are OK.
- Checkbox "I Acknowledge..."
- Hit **NEXT** again and wait ~5 minutes for deployment

4) Open the Deployer Wep Page

- Go to the Outputs tab of the newly completed stack
- There should be three outputs: DeployerAccessKey, DeployerIP, and DeployerURL
- Copy these values down for later reference. You'll use this same deployer to teardown the instance at the end.
- Click the **DeployerURL** link to open the deployer web page.
- Ignore any "Connection is not Private" warnings and open the page.

Use Deployer to install Yellowbrick Database into the private subnet of SpokeVPC

- Once on the Yellowbrick Deployer, Click **NEXT** to go past the welcome screen
- Click "**Agree**" and **NEXT** again to go past EULA
- On the **Provider** page
  - **Instnace Name:** `YellowbrickDB`
  - **Cloud Region:** `us-east-1` (or whatever region you setup the VPCs in)
  - Hit **NEXT**
- On the **Restrict Access** page
  - Choose **Private**
  - Hit **NEXT**
- On **Network** page
  - Under **Infrastructure resrouces**, using the dropdowns:
  - **Choose VPC:** `SpokeVPC-vpc (10.20.0.0/16)`
  - **Primary Subnet:** `SpokeVPC-subnet-private1-[Region]a (10.20.128.0/20)`
  - **Secondary Subnet:** `SpokeVPC-subnet-private2-[Region]b (10.20.128.0/20)`
  - Hit **NEXT**
- On **Initial Account** page
  - Click the **Download it** link to download the username and password
    - This file should be called `yb-YellowbrickDB-credentials.txt`
  - Alternatively, show the password and record it.
  - NOTE: This is the only place that username/password will be shown to you.
  - Hit **NEXT**

- On **Storage** page
  - Keep the **Create new storage buckets** checked
  - Hit **NEXT**
- On **Install** page
  - Hit **INSTALL**
  - The install will take about 20 minutes.
  - Keep this browser window open while its deploying, we will check status later.
  - Continue below in another browser tab

Create HubVPC infrastructure

1) Create a Windows Jumpbox EC2 Instance in Public Subnet

- Go to **EC2** / **Instances**
- Hit **Launch Instance** button on upper right
- Enter the fields:
  - **Name:** `WindowsJumpbox`
  - Application and OS Images:
    - Click **Windows Microsoft**
    - `Microsoft Windows Server 2025 Base` Should be selected.
  - **Instance Type:** `t3.xlarge`
  - Key pair (login)
    - Click **Create new key pair**
    - **Enter key pair name:** `hub-key`
    - Hit **Create key pair**
    - Save file: hub-key.pem
  - Network Settings
    - **Network:** `HubVPC-vpc`
    - **Subnet:** `HubVPC-subnet-public1-[region]a`
    - Select **Create security group**
    - Checkbox **Allow RDP traffic from:**, select: `My IP`
    - Hit **Edit** in upper right of Network Settings box
    - **Auto-assign public IP address:** `Enable`
  - Hit **Launch Instance** button on right

2) Create a Linux ETL/Application Server in Private Subnet

- Go to **EC2** / **Instances**
- Hit **Launch Instance** button on upper right
- Enter the fields:
  - **Name:** `ETLTools`
  - Application and OS Images:
    - Click **Ubuntu**

- `Ubuntu Server 22.04 LTS` should be selected.
- If a dialog opens up, hit **Confirm changes**

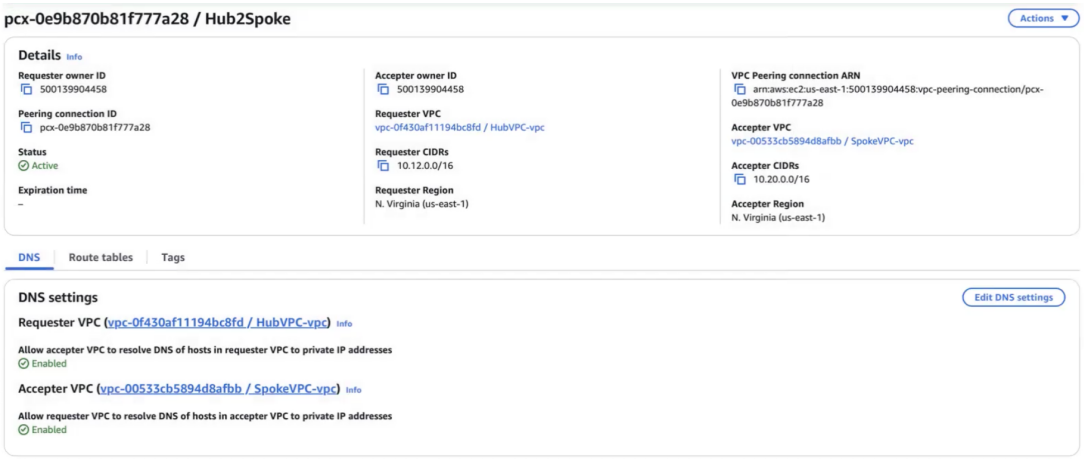
- **Instance Type:** `t2.xlarge`
- Key pair (login)
  - Select previous key pair: `hub-key`
- Network Settings
  - Hit **Edit** in upper right of Network Settings box
  - **Network:** `HubVPC-vpc`
  - **Subnet:** `HubVPC-subnet-private1-[region]a`
  - Select **Create security group**, then under **Inbound Security Rule**
    - Click **Add security group rule**
    - **Type:** `All TCP`
    - **Source type:** `Custom`
    - **Source:** `10.20.0.0/16`
- Configure Storage
  - You may want to increase storage if needed, default is 8 GB
- Hit **Launch Instance** button on right

Setup Connectivity between VPCs

1) Peer HubVPC to SpokeVPC

- Go to **Peering Connection** in AWS
- Hit **Create peering connection** in upper right
- Enter the fields:
  - **Name:** `Hub2Spoke`
  - Select a local VPC to peer with: **VPC ID (Requester):** select `HubVPC-vpc`
  - Under Select another VPC to peer with:
    - **Account:** `My Account`
    - **Region:** `This Region ([Region])`
    - **VPC ID (Accepter):** `SpokeVPC-vpc`
- Hit **Create peering connection** button
- On the next screen, click the **Actions** Button and select `Accept request`
- A Dialog opens up with request details, click **Accept request** button
- On the **DNS** tab, click **Edit DNS settings** button on the right
  - In the **Edit DNS settings** page,
    - Under **Request DNS resolution**, checkbox `Allow accepter VPC`
    - Under **Accepter DNS resolution**, checkbox `Allow requester VPC`
    - Hit **Save changes**

The VPC peering connection should look something like this:



2) Setup Routes: Hub to Spoke

- Go to **VPC**, **Your VPCs**, and follow the link on **VPC ID** on `HubVPC-vpc`
- On the **Resource map** tab, under **Route tables**, click on **HubVPC-rtb-public**
  - Click **Actions** in upper right, then select **Edit routes**
    - Click **Add route**
    - **Destination:** `10.20.0.0/16`
    - **Target:** Select `Peering connection`, then select `Hub2Spoke`
    - Hit **Save changes** button

2) Setup Routes: Spoke back to Hub (from both AZs and Public)

- Go to **VPC**, **Your VPCs**, and follow the link on **VPC ID** on `SpokeVPC-vpc`
- On the **Resource map** tab, under **Route tables**, click on **SpokeVPC-rtb-private1-[Region]a**
  - Click **Actions** in upper right, then select **Edit routes**
    - Click **Add route**
    - **Destination:** `10.12.0.0/16`
    - **Target:** Select `Peering connection`, then select `Hub2Spoke`
    - Hit **Save changes** button
- Go to **VPC**, **Your VPCs**, and follow the link on **VPC ID** on `SpokeVPC-vpc`
- On the **Resource map** tab, under **Route tables**, click on **SpokeVPC-rtb-private2-[Region]b**
  - Click **Actions** in upper right, then select **Edit routes**
    - Click **Add route**
    - **Destination:** `10.12.0.0/16`
    - **Target:** Select `Peering connection`, then select `Hub2Spoke`
    - Hit **Save changes** button
- Go to **VPC**, **Your VPCs**, and follow the link on **VPC ID** on `SpokeVPC-vpc`
- On the **Resource map** tab, under **Route tables**, click on **SpokeVPC-rtb-public**
  - Click **Actions** in upper right, then select **Edit routes**
    - Click **Add route**

- **Destination:** `10.12.0.0/16`
- **Target:** Select `Peering connection`, then select `Hub2Spoke`
- Hit **Save changes** button

Check connectivity between Hub and Spoke

1) Check install of Yellowbrick

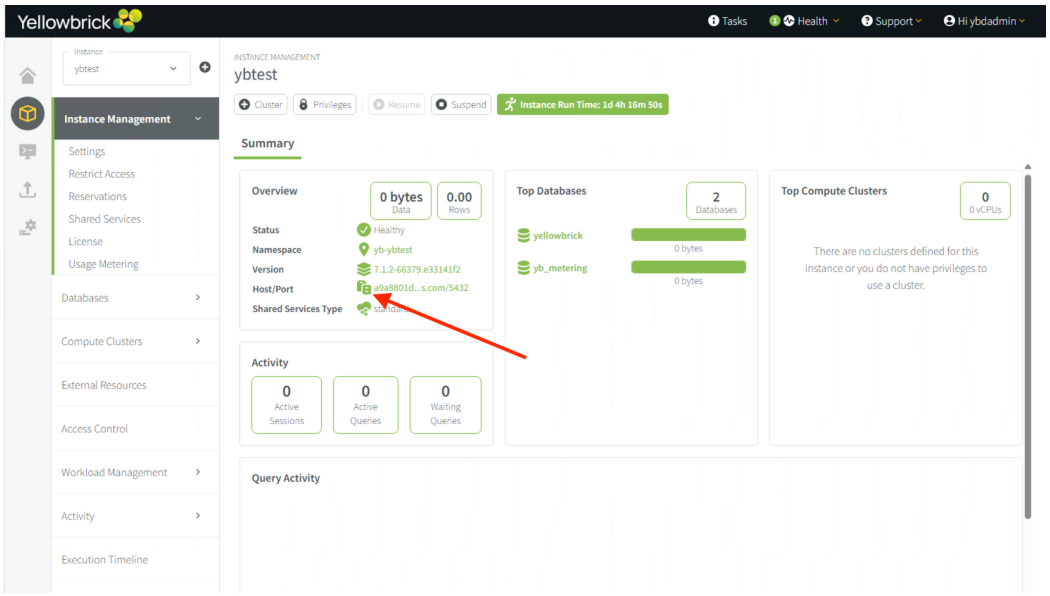
- Go back to the Yellobrick Deployer tab
- Ensure the deployer finished installing Yellowbrick
- Save the Yellowbrick Manager URL

2) Log in to Windows Jump Box

- In AWS, go to **EC2**, **Instances** and click the instance ID for your WindowsJumpBox
- Hit the **Connect** button in the upper right
- Switch to the **RDP Client** tab
- Hit the **Download remote desktop file**, save this file: `WindowsJumpbox.rdp`
- Click the **Get password** link
- Hit the **Upload private key file** button
- Select the `hub-key.pem` file we saved earlier
- Hit the **Decrypt passsword** button
- Save the **Password** somewhere as `JumpboxPassword`
- Open the `WindowsJumpbox.rdp` file
- When prompted, paste the `JumpboxPassword` into password field and hit **Continue**

3) Connect to Yellowbrick Manager

- From within the Windows Jumpbox
- Open up **Microsoft Edge** browser
- Copy the Yellowbrick Manager URL saved from the deployer in Step 1 above
- Enter the credentials stored in `yb-YellowbrickDB-credentials.txt`
- You should now have connectivity to the Yellowbrick Manager from the public subnet of the Hub VPC
- Within the Yellowbrick Manager, go to the Instance Management homepage and save the Hostname as `DBHostName`
  - Click the clipboard icon next to the tip of the red arrow below:



4) Connect to Yellowbrick Database

- From AWS, go to **EC2, Instances** and click the Instance ID for the ETLTools instance
- Copy the **Private IPv4 address** for the ETLTools instance, save this somewhere as `ETLToolsIP`
- From within the Windows Jumpbox
- Open up **Windows Powershell**
- Create a SSH .PEM file
  - Open up the `hub-key.pem` file in a text editor
  - Copy the contents into the clipboard
  - Type `@"` at the powershell prompt and hit return
  - Paste the contents of the clipboard (the hub-key.pem file) and hit return
  - Type `"@ | Out-File -FilePath ssh.pem -Encoding ascii` and hit return. This makes the ssh.pem file.
- At the powershell prompt type: `ssh -i ssh.pem ubuntu@[ETLToolsIP]`
  - If prompted, say Yes
- You should now be logged in as user `ubuntu` on ETLTools box in the private subnet. This subnet has access to the database instance.
- Type `nc -zv [DBHostName] 5432`
- This confirms connectivity between the ETLTools box and the Yellowbrick Database Instance

---

Next Steps

- Using the Yellowbrick Manager
  - Install a License
  - Setup a compute cluster
  - Load and query sample data
- Download and Install YBTools on the ETLBox and use the tools to load data into Yellowbrick
- Setup other linux servers in the private subnet to host Data Applications

Reference

Yellowbrick Documentation > Reference

Platforms: All platforms

Parent topic: Yellowbrick Documentation

In this section:

- Platform Support
- Configuration Parameters
- ybsql Reference
- SQL Reference
- Error Codes
- EULA
- Glossary
- Third-Party Licenses

Welcome to the Yellowbrick reference documentation.

Platform Support

In this section:

Yellowbrick currently has a single edition — Enterprise Edition — which targets multiple supported platforms. Although most core database documentation is applicable to all platforms, there are differences. This Platform Support Guide explains the differences and contains sections of documentation that may be pertinent to only one platform or edition.

When looking at shared documentation, at the top of each page of you'll find capsules listing the platforms that the page in question applies to.

The current set of platforms and editions are documented below.

Yellowbrick Edition

Enterprise Edition (EE)

The fully featured, scale-out version of Yellowbrick with enterprise support. Enterprise Edition runs on all Yellowbrick appliances, as well as all public and private cloud platforms. The database engine is identical between platforms, but the infrastructure and storage architecture are different, leading to some differences in system views, SQL utility statements and administration.

Yellowbrick Appliances

Tinman appliance (EE-APPLIANCE-TINMAN)

Tinman is Yellowbrick's first-generation appliance used in appliance platforms data centers. It is a blade server architecture using Intel Broadwell CPUs with integrated 56gbit active/passive dual port Inifiband backplane and integrated NVMe storage devices.

Andromeda appliance (EE-APPLIANCE-ANDROMEDA)

Andromeda is Yellowbrick's second-generation appliance used in appliance platforms data centers. It is also a blade server architecture using AMD EPYC CPUs with integrated 100gbit active/active dual port Infiniband backplane and removable NVMe storage devices.

Yellowbrick Cloud Platforms

Yellowbrick software supports numerous cloud platforms by running on top of Kubernetes. Currently supported cloud platforms are as folllows:

Amazon Web Services (EE-CLOUD-AWS)

Yellowbrick runs on top of EKS (Elastic Kubernetes Services), stores metadata and row-oriented data on EBS (Elastic Block Store) and columnar data on S3.

Microsoft Azure (EE-CLOUD-AZURE)

Yellowbrick runs on top of AKS (Azure Kubernetes Services), stores metadata and row-oriented data on Azure Disk Storage and columnar data on Azure Blob Storage.

Google Compute Platform (EE-CLOUD-GCP)

Yellowbrick runs on top of GKE (Google Kubernetes Engine), stores metadata and row-oriented data on persistent disks, and columnar data on GCS (Google Cloud Storage).

Differences Between Appliance and Cloud Platforms

Both appliance platforms and cloud platforms use the same underlying database engine, that processes data using SQL in exactly the same way. The differences are in the following areas:

Area	Appliance platforms	Cloud platforms
Installation	Appliance hardware is installed into a data center by Yellowbrick staff and includes all database and system software preinstalled.	Cloud software is installed into a cloud account using the Yellowbrick deployer.
Storage	Appliances have co-located storage and compute. Data is stored on SSDs within compute nodes and protected using parity encoding.	Cloud software makes use of separate storage and compute. The software stores data on both block storage and object storage from the cloud provider.
Compute nodes	The number of compute nodes available to Yellowbrick is dictated by how many blades are installed in the appliance.	Compute nodes are allocated and de-allocated dynamically from the cloud using Kuberentes, with the number of compute nodes limited solely by the capacity of a given availability zone.
Elastic compute clusters	Appliances have one 'default' compute cluster comprising all intalled compute blades.	Multiple compute clusters with separate storage and compute can be created dynamically. Users are assigned to clusters and intensive workloads can be load-balanced across them.
User Interface	The Yellowbrick Systems Management Console (SMC) is used for management of the appliance, along with YBCLI for management of the physical platform. Third party SQL authoring tools must be used.	Yellowbrick Manager is used for management of the instance as well as authoring and issuing ad-hoc SQL. Third party SQL authoring tools are optional.
Authentication and authorization	Database local authentication, OAuth2.0 authorization and LDAP synchronization are supported. Kerberos is also supported via editing configuration files. OAuth2.0 allows easy integration with federated identity management solutions.	Database local authentication and OAuth2.0 authorization are supported.

Cloud Users and Roles

Yellowbrick Documentation > Reference > Platform Support > Cloud Users and Roles

Platforms: EE: All cloud platforms

Parent topic: Platform Support

This document describes the types of users and roles that exist, and can be created, for Yellowbrick deployments. Administrator accounts can be set up at multiple levels, based on a set of predefined admin roles. Database user accounts can be set up with or without access to Yellowbrick Manager. Database users and roles exist separately per data warehouse instance. Different sets of users and roles may belong to each instance in the same cloud deployment. The following users and roles are created by default in each new installation:

(initial administrator)

In the Deployer, you input (or automatically generate) the initial administrator credentials This account is intended for initial login to Yellowbrick Manager, but is not intended for extensive use. The `ybdadmin` user should create new users and roles with the same or fewer privileges, as required. This user is a member of all other predefined `admin` roles we well as the consumer role:

- `clusteradmin`
- consumeradmin
- instanceadmin
- `securityadmin` : privileges to create, drop, and alter roles and users, as well as external authorization control
- `sysadmin` : broad privileges at the system, session, and database level, but no privileges on clusters
- `useradmin` : privileges to create, drop, and alter roles and users

clusteradmin role

This role grants all privileges on clusters, including the ability to create new clusters, and alter, drop, or use any cluster for a given instance.

consumeradmin role

This role is designed for administrators. It confers privileges to create and manage instances, run diagnostics, and perform upgrades. The `ybdadmin` user belongs to the `consumeradmin` role; however, a new user who is granted membership in `consumeradmin` *does not* acquire membership in the other admin roles that `ybdadmin` has.

instanceadmin role

This role grants privileges for database administration: cluster management, database creation, workload management, access control for other users, and so on. A user with membership in this role is effectively a database administrator for a given data warehouse instance. Members of this role may explicitly grant additional users membership in the `instanceadmin` role.

This role grants membership in other predefined admin roles: `clusteradmin` , `securityadmin` , `useradmin` , `sysadmin` (but *not* the `consumeradmin` role).

consumer role

All new users belong to this empty role. `ybdadmin` has membership in the `consumer` role `WITH ADMIN` .

yellowbrick

: A `yellowbrick` database superuser exists but is not intended for customer use and has no login password. It is used for system processes only and must not be tampered with.

sys_ybd_* (service accounts)

> Several system-defined no-login service roles exist in the system and are visible in all instances. Do not modify these roles. These role names begin with a `sys_ybd_*` prefix. Some of these services run background operations, such as flushing and analyzing tables.

You can create database users and roles by navigating to *Instamcen Management → Access Control* in Yellowbrick Manager or by using SQL.

See also Managing Database Users and Roles.

Cluster Access

When you create new users and roles, be sure to grant them `USAGE ON CLUSTER` to at least one cluster. See ON CLUSTER.

Individual users may be assigned a default cluster where their queries always run. See ALTER USER SET DEFAULT_CLUSTER. (The default cluster assignment does not propagate to users from roles.)

Supported Cloud Regions

Platforms: EE: All cloud platforms

Parent topic: Platform Support

The following regions are formally qualified and supported by Yellowbrick. If you want to run in another region, your mileage may vary since formal qualification may not have taken place. Please get in touch if you'd like to see additional regions on this list.

---

aws

- us-east-2
- us-east-1
- us-gov-east-1
- us-gov-west-1
- us-west-2
- us-west-1
- af-south-1
- ap-southeast-1
- ap-southeast-2
- ap-northeast-1
- ap-northeast-2
- ap-northeast-3
- ap-south-1
- eu-central-1
- eu-west-1
- eu-west-2
- eu-north-1
- eu-west-3
- ca-central-1
- sa-east-1

---

azure

- eastus
- eastus2
- southcentralus
- westus2
- australiaeast
- southeastasia
- northeurope

- uksouth
- westeurope
- centralus
- southafricanorth
- westus3
- uaenorth
- switzerlandnorth
- qatarcentral
- polandcentral
- italynorth
- israelcentral
- norwayeast
- koreacentral
- swedencentral
- centralindia
- eastasia
- japaneast
- canadacentral
- francecentral
- germanywestcentral
- brazilsouth

---

gcp

- us-central1
- europe-west1
- us-east1
- asia-southeast1
- us-east4
- europe-west4

Configuration Parameters

Yellowbrick Documentation > Reference > Configuration Parameters

Platforms: All platforms

Parent topic: Reference

In this section:

NOTE

The set of parameters documented here is incomplete; it will be extended in subsequent releases.

Configuration Parameters are variables used for controlling the behavior of the database at different levels. The behaviour is inherited directly from PostgreSQL where users are most familiar with changing behaviour by editing the `postgresql.conf` file and similar.

Parameters are set at a system level and can be overridden per user or per session. User settings take precedence over system settings, and similarly session settings take place over user settings. All parameter names are case-insensitive. Every parameter takes a value of one of five types: Boolean, integer, floating point, string or enum. Boolean values can be written as on, off, true, false, yes, no, 1, 0 (all case-insensitive) or any unambiguous prefix of these.

Some settings specify a memory or time value. Each of these has an implicit unit, which is either kilobytes, blocks (typically eight kilobytes), milliseconds, seconds, or minutes. Valid memory units are kB (kilobytes), MB (megabytes), and GB (gigabytes); valid time units are ms (milliseconds), s (seconds), min (minutes), h (hours), and d (days). Note that the multiplier for memory units is 1024, not 1000.

This section of the documentation details user-settable configuration parameters for Yellowbrick.

Changing parameters globally

Parameters can be changed globally by using the ALTER SYSTEM command and reloading the configuration afterwards.

Changing parameters for users

You can override global settings for specific users by using ALTER ROLE commands. An ALTER ROLE...SET command sets a parameter value for all sessions that are run by the specified user or role.

Changing parameters for sessions

You can override global and user-specific parameters by using SET commands. A SET command sets a parameter value for the current session only. No other sessions are affected.

See also the RESET and SHOW commands.

Compatibility Parameters

Platforms: All platforms

Parent topic: Configuration Parameters

In this section:

NOTE

The set of parameters documented here is incomplete; it will be extended in subsequent releases.

Yellowbrick implements several configuration variables to alter behaviour or increase functional surface area for compatibility with other databases. They are documented here.

enable_alternative_round (boolean)

The second argument (decimal places) of the `ROUND` function requires a constant value; column values are not allowed. Disallowing a column value for the second parameter enables optimization of function execution ahead of time, but it can break compatibility with SQL queries or workloads that require this.

By setting this parameter to `ON` you will be able to use `ROUND` with non-constant values for the second argument. Internally, the query planner will translate the function call to `ROUND_VAR`.

For example:

```txt
set enable_alternative_round to on;

select round(1.111567, seasonid), seasonid from season order by 2 limit 6;
  round   | seasonid
----------+----------
 1.100000 |        1
 1.110000 |        2
 1.112000 |        3
 1.111600 |        4
 1.111570 |        5
 1.111567 |        6
(6 rows)
```

Note that when you do this, `ROUND` will produce the same return type that `ROUND_VAR` produces.

enable_safe_compat_functions_with_nz

Platforms: All platforms

Parent topic: Compatibility Parameters

This configuration parameter enables type-safe casting functions for compatibility with IBM Netezza and other databases.

For example, to enable the functions system-wide:

```
alter system set enable_safe_compat_functions_with_nz to true;
WARNING:  Will be effective after the next server configuration reload, or after the next server restart in the case of parameters

select pg_reload_conf();
 pg_reload_conf
----------------
 t
(1 row)
```

override_version (string)

This allows the version string returned by the database to be changed to match that required by drivers or third-party applications. For example, a client may not recognize Yellowbrick as a supported database, but will function fine with PostgreSQL. Similarly, perhaps a newer version of PostgreSQL needs to be declared for compatibility.

For example:

```sql
SET override_version='PostgreSQL 9.5.10 on x86_64-redhat-linux-gnu'
```

To set the version system-wide and commit the change:

```sql
ALTER SYSTEM SET override_version='PostgreSQL 9.5.10 on x86_64-redhat-linux-gnu';
SELECT pg_reload_conf();
```

The version can also be overridden for just a particular role or session.

Data Formatting and Behavioral Parameters

Platforms: All platforms

Parent topic: Configuration Parameters

In this section:

NOTE

The set of parameters documented here is incomplete; it will be extended in subsequent releases.

Yellowbrick implements several configuration variables to adapt the behaviour of the database to match other databases or tweak output formatting.

bytea_output (VARBINARY)

Platforms: All platforms

Parent topic: Data Processing and Formatting

- Default value (as of this release): `hex`

This configuration parameter determines the output format of `bytea` data type values. It can have two possible values: `hex` or `escape`.

datestyle (string)

This parameter sets the display format for date and time values, as well as the rules for interpreting ambiguous date input values. For historical reasons, this variable contains two independent components: the output format specification (`ISO` , `Postgres` , `SQL` , or `German`) and the input/output specification for year/month/day ordering (`DMY` , `MDY` , or `YMD`).

These can be set separately or together. The keywords `Euro` and `European` are synonyms for DMY; the keywords `US` , `NonEuro` , and `NonEuropean` are synonyms for `MDY` .

The built-in default is `ISO, MDY` .

See also ybload Date Formats and DATE.

enable_silent_coerce (boolean)

Platforms: All platforms

Parent topic: Data Processing and Formatting

This parameter controls whether or not to downcast and truncate VARCHAR strings, potentially allowing them to be inserted into a column that is too small to contain them (for example, allowing a 3-character string to be inserted into a 2-character column).

See also Implicit Casting.

enable_subquery_remove_useless_order_by (boolean)

Yellowbrick by default ignores `ORDER BY` statements in CTEs and subqueries without a `LIMIT` clause. This causes unordered data to be returned by users of the `STRING_AGG` function. This optimization can be disabled:

```sql
ALTER SYSTEM set enable_subquery_remove_useless_order_by to 'off';
```

extra_float_digits (integer)

This parameter adjusts the number of digits displayed for floating-point values. The parameter value is added to the standard number of digits. The value can be set as high as 3, to include partially-significant digits; this is especially useful for dumping float data that needs to be restored exactly. Or it can be set negative to suppress unwanted digits.

For example, note the difference in query results against a `REAL` column when `extra_float_digits` is set to `3` compared to `0`:

```
reset extra_float_digits;

create table extra_float as select avg_att::real from team where avg_att>0;
SELECT 20
select * from extra_float;
 avg_att
---------
  35.776
  20.594
  24.631
  34.91
  59.944
  33.69
  11.189
  ...
(20 rows)

set extra_float_digits to 3;
SET
create table extra_float_3 as select avg_att::real from team where avg_att>0;
SELECT 20
select * from extra_float_3;
  avg_att
------------
 35.776001
 20.5939999
 24.6310005
 34.9099998
 59.9440002
 33.6899986
 11.1890001
 ...
(20 rows)
```

See also real data types.

json_missing_element_default (string)

Platforms: All platforms

Parent topic: Data Processing and Formatting

This configuration parameter is used to change the default behavior of `ERROR ON ERROR` when accessing missing elements in JSON documents. It accepts two values:

- "null" (the default setting)
- "error"

```sql
set json_missing_element_default to 'null';
select '{"answer":42}':question;
-- <null>
```

```sql
set json_missing_element_default to 'error';
select '{"answer":42}':question;
--ERROR:  No key found at path: 'strict $."question"' for '{"answer":42}'
```

max_levenshtein_distance (integer)

This configuration parameter affects the behaviour of the `LE_DST` and `DLE_DST` functions.

These functions are very compute-intensive. The maximum distance (or "threshold") expected in a Levenshtein distance calculation can significantly impact runtime, enabling early termination, pruned search space and reduced memory usage.

This parameter changes the maximum distance for which you expect an accurate answer. The default value is `-1` which means there is no limit.

For example, you may only want to check if the distance between two strings is greater than 5 characters; you don't need to know what the real distance value is. In this case, set `max_levenshtein_distance` to `5`. If the real distance between the two strings is less than or equal to 5, the function returns the real value. Otherwise, it returns an answer for the first comparison that is greater than 5 and stops processing.

See also `max_levenshtein_string_size`.

max_levenshtein_string_size (integer)

This configuration parameter affects the behaviour of the `LE_DST` and `DLE_DST` functions.

These functions are very compute-intensive. Longer strings require more comparisons, increasing both CPU time and memory usage.

This parameter changes the maximum size (in bytes) of the two input arguments to the function. The default is `255`.

See also `max_levenshtein_distance`.

timezone (string)

This parameter sets the time zone for displaying and interpreting time stamps. The default is `UTC`. See also TIMESTAMP WITH TIME ZONE and ybload Timestamp Formats.

Feature Enablement Parameters

Platforms: All platforms

Parent topic: Configuration Parameters

In this section:

NOTE

The set of parameters documented here is incomplete; it will be extended in subsequent releases.

Yellowbrick implements several configuration variables to enable new features also known as *feature flags*. These configuration variables are defaulted to `off` in the release in which they are introduced; defaulted to `on` in a subsequent release; and then finally removed (with the feature always enabled) in a subsequent release.

enable_full_bytea (boolean)

- Default value (as of this release): `off`

This configuration parameter enables the use of Binary data type .

enable_full_funcscan (boolean)

- Default value (as of this release): `off`

This configuration parameter enables the use of Set Returning Functions.

enable_full_json (boolean)

- Default value (as of this release): `off`

This configuration parameter enables:

- The full use of the JSON and JSONB types.

enable_full_lateral_join (boolean)

- Default value (as of this release): `off`

This configuration parameter enables the use of Set Returning Functions.

enable_full_samplescan (boolean)

- Default value (as of this release): `off`

This configuration parameter enables the use of TABLESAMPLE.

enable_geospatial (boolean)

- Default value (as of this release): `off`

This configuration parameter enables the use of Geography data type and Geospatial functions.

enable_password_policy (boolean)

- Default value (as of this release): `off`

This configuration parameter enables configurable password policies. The feature adds additional controls to further secure locally authenticated database users.

enable_query_hint_injection (boolean)

- Default value (as of this release): `off`

This server configuration parameter enables the use of Plan Hinting.

enable_se_filter_expression_generation (boolean)

- Default value (as of this release): `false`

This configuration parameter enables the Advanced Storage Engine Filters feature.

```sql
-- Generate engine filter expressions by setting this configuration parameter to true
SET enable_se_filter_expression_generation = true;
```

enable_sql_unload (boolean)

- Default value (as of this release): `off`

This configuration parameter enables the full use of SQL Unload.

enable_user_audit (boolean)

- Default value (as of this release): `off`

This configuration parameter enables detailed authentication and authorization logging. The feature stores detailed logs of authentication attempts and changes to roles, users and grants outside the database.

enable_wlm_query_hint_injection (boolean)

- Default value (as of this release): `off`

This server configuration parameter enables the use of Plan Hinting through WLM rules.

query_hint_report_level (enum)

- Default value (as of this release): `log`

This configuration parameter adjusts the level of received messages related to Plan Hinting.

Available values are: `debug5` , `debug4` , `debug3` , `debug2` , `debug1` , `log` , `notice` , `warning` , `error`

ybd_allow_udf_creation (boolean)

- Default value (as of this release): `off`

This configuration parameter enables SQL user defined functions (UDFs). See the SQL UDF Create Function documentation for more information.

General Parameters

Yellowbrick Documentation > Reference > Configuration Parameters > General

Platforms: All platforms

Parent topic: Configuration Parameters

In this section:

NOTE

The set of parameters documented here is incomplete; it will be extended in subsequent releases.

This section outlines Configuration Parameters not classified in other areas of the documentation.

application_name (string)

This configuration variable sets the application name (typically the name of a connecting client tool) reported in statistics and logs. It's typically set the first time an application connects to the database.

Only printable ASCII characters may be used; any others will be replaced with question marks.

See also sys.session and sys.log_query.

client_encoding (string)

Platforms: All platforms

Parent topic: General

This configuration parameter sets the client-side encoding (character set). The default is to use the database encoding. For more information, see Creating Databases.

client_min_messages (enum)

Platforms: All platforms

Parent topic: General

This configuration parameter controls the verbosity of errors returned to the client. Valid values are `debug5`, `debug4`, `debug3`, `debug2`, `debug1`, `log`, `notice`, `warning`, `info`, `error`.

The default value is `warning`.

cluster_name (string)

Sets the name of the instance, which is included in the executable process title. Not to be confused with compute clusters. For example:

```sql
ALTER SYSTEM set cluster_name TO 'prod_y2b100;
```

The change will take effect once the database is restarted.

compute_blade_filesystem_utilization_critical_threshold (integer)

- Default value (as of this release): `95`
- Valid range: `1` to `100`. `-1` may be used to disable this feature.

This setting determines the **filesystem utilization threshold** for a blade, expressed as a percentage. When the utilization exceeds this threshold, a **critical alert** is triggered. Use -1 to disable. Only members of the `sysadmin` role or superusers can set this parameter. It is recommended to set this parameter globally instead of setting it per session (refer to this page to set the parameters globally).

enable_geography_client_compat_mode (bool)

If `true`, change the type OID of the geography column in the extended protocol metadata from `geography` to `text`. This allows some clients to better parse WKT geography results.

Default is `false`.

enable_rowpacket_compression_in_distribution (boolean)

- Default value (as of this release): `off` (appliance platforms), `on` (cloud platforms)

This configuration parameter enables compression of data sent between distribute nodes in queries to reduce the amount of network bandwidth used. It commonly increases the performance of data distribution-bound queries at the expense of consuming more CPU, on platforms without zero-copy networking.

enable_user_audit_extended (boolean)

Platforms: All platforms

Parent topic: General

This configuration parameter enables extended user audit logging. See the detailed authentication and authorization logging feature documentation to understand the difference between basic and extended logging.

idle_in_transaction_session_timeout (integer)

This configuration parameter controls how long any transaction within any user session may remain idle before being disconnected. Any session will time out if it has an open transaction that has not run a statement for the specified length of time. The default is 10 minutes (600000 ms). Connections terminated due to `idle_in_transaction_session_timeout` will result in a `sys.log_session` state value of `25P03`.

This parameter accepts settings with the following units: `ms`, `s`, `min`, `h`, and `d`.

Note: Running `show idle_in_transaction_session_timeout` as a superuser will always return unlimited (`0`), which is the superuser's setting.

See also: `idle_session_timeout`.

idle_session_timeout (integer)

Yellowbrick Documentation > Reference > Configuration Parameters > General > idle_session_timeout

Platforms: All platforms

Parent topic: General

This configuration parameter controls how long any user session may remain idle before being disconnected. Any session will time out if it has not run a statement for the specified length of time (regardless of open transactions). The default is 60 minutes (3600000 ms). Connections terminated due to `idle_session_timeout` will result in a `sys.log_session` state value of `25P04`.

This parameter accepts settings with the following units: `ms`, `s`, `min`, `h`, and `d`.

Note: Running `show idle_session_timeout` as a superuser will always return unlimited (`0`), which is the superuser's setting.

You can modify this parameter with the following commands:

- ALTER SYSTEM: Only `sysadmin` users can change these settings with this command, which applies to all sessions for all users. A configuration reload or database restart is required for the new values to take effect. For example:

```
alter system set idle_session_timeout to '2h';

show idle_session_timeout;
 idle_session_timeout
----------------------
 1h
(1 row)
```

- SET: Any user can change these settings with this command. The new value is applied only to the current user session. The next session for that user resets to the default value. For example:

```
set idle_session_timeout to 999999;

show idle_session_timeout;
 idle_session_timeout
----------------------
 999999ms
(1 row)
```

- ALTER ROLE: changes the settings only for the specified role. Restarting the database is not required, and the new settings are applied to all new sessions for that role. You do not have to be a `sysadmin` user to run this command, but you must have permissions to alter the role in question.

```
alter role bobr set idle_session_timeout to 999999;

show idle_session_timeout;
 idle_session_timeout
----------------------
 1h
(1 row)
```

See also: `idle_in_transaction_session_timeout`.

json_null_byte (string)

Yellowbrick Documentation > Reference > Configuration Parameters > General > json_null_byte

Platforms: All platforms

Parent topic: General

This configuration parameter is used to change the default behavior when handling null characters during JSONB -> VARCHAR conversions. It accepts two values:

- "error" (the default setting) - errors out the query
- "replace" - replaces the null character with the Unicode replacement character U+FFFD

```sql
SET json_null_byte TO 'error';
SELECT '"\u0000\u0000\u0000\u0000"'::JSONB::VARCHAR;
--ERROR:  null byte casting is disabled, see https://docs.yellowbrick.com/latest/gucs/guc_json_null_byte.html
```

```sql
SET json_null_byte TO 'replace';
SELECT '"\u0000\u0000\u0000\u0000"'::JSONB::VARCHAR;
-- ````
```

listen_addresses (string)

This parameter specifies the TCP/IP address(es) on which the server is to listen for connections from client applications. The value takes the form of a comma-separated list of host names and/or numeric IP addresses. The special entry * corresponds to all available network interfaces. The entry `0.0.0.0` allows listening for all IPv4 addresses and `::` allows listening for all IPv6 addresses. The default value is `*`.

Changing this parameter requires restarting the database.

log_timezone (string)

Platforms: All platforms

Parent topic: General

This parameter sets the time zone used for timestamps written in the server logs. The built-in default is UTC. This parameter can only be set at a system level.

max_connections (integer)

Platforms: All platforms

Parent topic: General

This parameter sets the maximum number of connections to the database, including external users and internal system connections. The default is `2300`. Changing this parameter requires restarting the database and requires consideration of available compute and memory.

See also `max_user_connections` and Database Limits.

max_quota_objects (integer)

The database has a default maximum of 500k disk usage objects. You can change the maximum to a value from 500000 to 20000000 using an ALTER SYSTEM command. For example, if you wanted to increase the maximum from 500000 objects to 1000000:

```
alter system set max_quota_objects to 1000000;
WARNING: Will be effective after the next server configuration reload, or after the next server restart in the case of parameters
```

max_user_connections (integer)

Platforms: All platforms

Parent topic: General

This parameter sets the maximum number of external user connections to the database, excluding internal system connections. The default is `2000`. The value must be less than `max_connections`. Changing this parameter requires restarting the database and requires consideration of available compute and memory.

See also `max_connections` and Database Limits.

ybd_secrets_manager (string)

This configuration variable controls where the database looks up secrets to retrieve keys used for encrypting and decrypting sensitive data. The valid values are:

- `none` : Encryption functions will not operate
- `k8ss` : Use the Kubernetes secrets manager
- `vault` : Use the HashiCorp Vault running on an appliance

The value should not need to be changed from the default. It must be set at a system level and configuration reloaded for it to take effect.

port (integer)

Platforms: All platforms

Parent topic: General

This parameter shows the TCP port the database server listens on. The default is `5432`.

rowstore_block_threshold_percent (integer)

- Default value (as of this release): `90`
- Valid range: `1` to `99`

This configuration parameter defines the percentage of occupied space at which all operations are blocked. Reload this parameter for the changes to take effect. Only members of the `sysadmin` role or superusers can set this parameter. It is recommended to set this parameter globally instead of setting it per session (refer to this page to set the parameters globally).

rowstore_read_only_threshold_percent (integer)

- Default value (as of this release): `80`
- Valid range: `1` to `99` (<= `rowstore_block_threshold_percent`)

This configuration parameter defines the occupied space percentage at which only READ operations are allowed. Reload this parameter for the changes to take effect. Only members of the `sysadmin` role or superusers can set this parameter. It is recommended to set this parameter globally instead of setting it per session (refer to this page to set the parameters globally).

rowstore_warning_threshold_percent (integer)

- Default value (as of this release): `70`
- Valid range: `1` to `99` (<= `rowstore_read_only_threshold_percent`)

This configuration parameter defines the percentage of occupied space at which users are warned about limited disk space. Reload this parameter for the changes to take effect. Only members of the `sysadmin` role or superusers can set this parameter. It is recommended to set this parameter globally instead of setting it per session (refer to this page to set the parameters globally).

search_path (string)

The `search_path` determines the order in which schemas are searched when a schema object (like a table or view) is referenced by a query without a schema qualifier. Writing fully qualified object name prefixed by a schema in queries can become tedious, and changing this parameter allows you to omit them.

By default, the database searches through a few specific schemas. It starts with a schema named after the current user, followed by the public schema:

For example:

```txt
show search_path;
search_path
---------------
"$user", public
(1 row)
```

The first element specifies that a schema with the same name as the current user is to be searched. If no such schema exists, the entry is ignored. The second element refers to the public schema that we have seen already.

The first schema in the search path that exists is the default location for creating new objects. That is the reason that by default objects are created in the public schema. When objects are referenced in any other context without schema qualification (table modification, data modification, or query commands) the search path is traversed until a matching object is found. Therefore, in the default configuration, any unqualified access again can only refer to the public schema.

To put an additional schema `neilschema` on the search path, run an example such as:

```txt
set search_path to neilschema,sys;
SET

show search_path;
 search_path
-------------
neilschema, sys
(1 row)
```

See also `CURRENT_SCHEMA` and the PostgreSQL documentation.

statement_timeout (integer)

Platforms: All platforms

Parent topic: General

This parameter sets the maximum allowed duration, in milliseconds, of any statement. See Canceling Queries.

tcp_keepalives_count (integer)

Platforms: All platforms

Parent topic: General

Specifies the number of TCP keepalive messages that can be lost before the database server's connection to the client is considered dead. A value of 0 (the default) selects the platform default

tcp_keepalives_idle (integer)

Platforms: All platforms

Parent topic: General

Specifies the amount of time with no network activity after which the platform should send a TCP keepalive message to the client. If this value is specified without units, it is taken as seconds. A value of 0 (the default) selects the platform default.

tcp_keepalives_interval (integer)

Platforms: All platforms

Parent topic: General

Specifies the amount of time after which a TCP keepalive message that has not been acknowledged by the client should be retransmitted. If this value is specified without units, it is taken as seconds. A value of 0 (the default) selects the platform default.

user_audit_max_bytes_per_file (integer)

Platforms: All platforms

Parent topic: General

This configuration parameter configures detailed authentication and authorization logging. It controls how many bytes are written to the active, uncompressed log file. Once this size is exceeded, the rotation and compression of the file is triggered. Valid range is 1024 to 2147483647.

user_audit_max_files (integer)

Platforms: All platforms

Parent topic: General

This configuration parameter configures detailed authentication and authorization logging. It controls how many rotated/compressed log files to retain before deleting the oldest file.

For example, with this parameter set to 10, you will have 11 files: one active or uncompressed file, and 10 rotated or compressed files. Valid range is 10 to 100.

yb_last_execid (string)

Platforms: All platforms

Parent topic: General

This parameter show the `query_id` of the last query ran in the session. It is read-only. This identifier will match that shown in `sys.log_query`.

yb_server_version (string)

Platforms: All platforms

Parent topic: General

Returns a string representing the version of the Yellowbrick database in use, for example `7.1.0`.

See also `yb_server_version_num`.

yb_server_version_num (integer)

Platforms: All platforms

Parent topic: General

Returns an integer representing the version of the Yellowbrick database in use, for example `70100`.

See also `yb_server_version`.

ybd_query_tags (string)

Platforms: All platforms

Parent topic: General

This parameter sets a label on a query which is logged and accessible to the workload manager for classifying workloads for resource allocation.

See Using Query Tags.

Tuning Parameters

Platforms: All platforms

Parent topic: Configuration Parameters

In this section:

NOTE

The set of parameters documented here is incomplete; it will be extended in subsequent releases.

This section outlines Configuration Parameters used for tuning the query planner and general optimizations.

enable_implied_pushdown (boolean)

- Default value (as of this release): `off`

This configuration parameter enables transitive pushdown of qualifiers. When the planner can determine that two columns must be the same value, it will attempt to apply any qualifiers specified for one column to the other column. It then pushes any new qualifiers as far down in the query plan as possible.

In the example below, we can see that `A.x` and `B.y` must have the same value. This allows the qualifier `y > 3` to be applied to the output of the join. Furthermore, because this is an `INNER JOIN`, the qualifier can be pushed down to the scan, resulting in even better performance. This effectively adds the clause `B.y > 3` to the query during execution.

```sql
CREATE TABLE A (x int);
CREATE TABLE B (y int);

SET enable_implied_pushdown TO true;

EXPLAIN SELECT * FROM A INNER JOIN B ON A.x = B.y WHERE A.x > 3;

                                                QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
 id   rows_planned   workers   node
 13             1       all   SELECT
  1             1       all   INNER HASH JOIN ON (b.y = a.x)
  4             1       all   |-SCAN a
                              |   (a.x > $0) AND a.x = bloom(7) AND scan_constraints: min_max(a.x) AND (a.x > $0)
  7             1       all   |-BUILD
 10             1       all     SCAN b
                                  (b.y > $2) AND scan_constraints: (b.y > $2)  <-- filter applied to A.x is pushed down to B.y as
```

enable_query_trace (boolean)

This parameter enables full query statistics to be recorded for queries executed. Full statistics contain data from each node and each thread. Setting this parameter to `on` will add extra information to `EXPLAIN (ANALYZE)` queries. Performance will be substantially degraded by enabling this option globally; we recommend just using a session at a time or when necessary for diagnosing performance issues.

Example:

```SQL
-- Turn on tracing for all future queries (use RESET to disable)
SET enable_query_trace TO ON;

-- Enable tracing for future and all running queries of a current session
ALTER SESSION <session_id> SET enable_query_trace TO ON;
```

See also EXPLAIN.

inlist_threshold (integer)

- Default value (as of this release): `10`

This configuration parameter specifies the threshold on the number of elements in an IN-list before that IN-list is considered for a semi-join rewrite.

How It Works

Consider a table named `result_table` that contains a single integer column named `x`.

```sql
create table result_table(x int);

insert into result_table(x) values (1), (2), (3), (4), (5), (6), (7), (8), (9), (10), (11);
```

If the number of elements in the IN-list exceeds the value of `guc_inlist_threshold`, the query planner may rewrite the query into a semi-join, which can optimize performance for larger datasets.

```sql
EXPLAIN SELECT * FROM result_table WHERE x IN (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11);

                                QUERY PLAN
-------------------------------------------------------------------------------------------------------
id   rows_planned   workers   node
10             11       all   SELECT
 1             11       all   SEMI LEFT INNER HASH JOIN ON (#inlist_100#.val = result_table.x)
 4             11       all   |-SCAN result_table
                              |   result_table.x = bloom(7) AND scan_constraints: min_max(result_table.x)
 7             11       all   |-BUILD
 9             11       all     SCAN TEMP 3
```

For smaller IN-lists (below the threshold), the query is typically processed as-is without rewriting.

```sql
EXPLAIN SELECT * FROM result_table WHERE x IN (1, 2, 3, 4, 5, 6, 7, 8, 9);

                                QUERY PLAN
-------------------------------------------------------------------------------
id   rows_planned   workers   node
 5              6       all   SELECT
 2              6       all   SCAN result_table
                                ((result_table.x = $8) OR ((result_table.x = $7) OR ((result_table.x = $6) OR ((result_table.x = $
```

se_tablescan_lookahead_kb (integer)

Memory adjustment can be done by setting up this configuration parameter. Effective setting for this configuration parameter generally range from 10 to 100 megabytes.

- Recommended setting for cloud environments:

```sql
-- Set the additional memory. It will be used during query scan/query skip to improve performance.
SET se_tablescan_lookahead_kb = 128000;
```

- Recommended setting for appliance platforms:

```sql
SET se_tablescan_lookahead_kb = 1;
```

ybd_analyze_after_loads (boolean)

Analyze operations are used to keep statistics up to date, as described in Auto-Analyzing Tables.

If your workload consists of a large number of continuous loads using `ybload` or the `LOAD TABLE` command, you may need to disable the subsequent ANALYZE operations that take place after each operation by setting this configuration parameter. This parameter defaults to `ON`.

For example, to set the parameter for a role performing bulk loads:

```sql
create role ybload_user;
alter role ybload_user set ybd_analyze_after_loads to off;
```

After changing this parameter, you have to reload the system configuration (or restart the database) before the new behavior takes effect.

See also `ybd_analyze_after_writes`.

ybd_analyze_after_writes (boolean)

Analyze operations are used to keep statistics up to date, as described in Auto-Analyzing Tables.

If your workload requires frequent small INSERT INTO...SELECT operations into a large fact table and you want to optimize performance, you can disable the immediate ANALYZE operations that happen after each INSERT. To disable these ANALYZE operations, set this configuration parameter to `OFF`. This parameter defaults to `ON`.

For example, to set the parameter for a role performing large data loads:

```sql
create role insert_select;
alter role insert_select set ybd_analyze_after_writes to off;
```

After a configuration reload, immediate ANALYZE operations will be disabled for all sessions run when the `insert_select` user runs INSERT INTO...SELECT operations. The advantage to disabling the parameter in this way is that it will remain enabled for other write operations. Setting it at a system level is discouraged since it can cause statistics to drift resulting in bad query plans.

After changing this parameter, you have to reload the system configuration (or restart the database) before the new behavior takes effect.

See also `ybd_analyze_after_loads`.

Yellowbrick Row Store (YRS) Alerting Parameters

Platforms: All platforms

Parent topic: Configuration Parameters

In this section:

NOTE

Advanced usage: Please contact Customer Support if you need to change the default values for these configuration parameters.

Yellowbrick implements several configuration parameters for Yellowbrick Row Store (YRS) alerting, which monitor the following:

- Number of data files
- Commit records
- Delete records
- Unused Files

These parameters can trigger alerts with the following severity levels:

- `Informational`
- `Minor`
- `Major`
- `Critical`

Note: For all the alert types:

- The `Minor` threshold cannot be greater than the `Major` and `Critical` thresholds.
- The `Major` threshold cannot be smaller than the `Critical` threshold.

yrs_commit_records_count_critical_threshold (integer)

Advanced usage

- Default value (as of this release): `-1` (disabled).
- Valid range: `-1` to `2147483647`. `-1` or `0` disable the alert.

This configuration parameter triggers a `Critical` severity alert named `Yellowbrick Row Store Commit Records` when the number of commit records in the YRS exceeds the configuration threshold. Use the following command to retrieve the count of commit records:

```sql
select count(*) from yb_yrs_commit_files();
```

yrs_commit_records_count_major_threshold (integer)

Advanced usage

- Default value (as of this release): `-1` (disabled).
- Valid range: `-1` to `2147483647`. `-1` or `0` disable the alert.

This configuration parameter triggers a `Major` severity alert named `Yellowbrick Row Store Commit Records` when the number of commit records in the YRS exceeds the configuration threshold. Use the following command to retrieve the count of commit records:

```sql
select count(*) from yb_yrs_commit_files();
```

yrs_commit_records_count_minor_threshold (integer)

Advanced usage

- Default value (as of this release): `-1` (disabled).
- Valid range: `-1` to `2147483647`. `-1` or `0` disable the alert.

This configuration parameter triggers a `Minor` severity alert named `Yellowbrick Row Store Commit Records` when the number of commit records in the YRS exceeds the configuration threshold. Use the following command to retrieve the count of commit records:

```sql
select count(*) from yb_yrs_commit_files();
```

yrs_data_files_count_critical_threshold (integer)

Advanced usage

- Default value (as of this release): `-1` (disabled).
- Valid range: `-1` to `2147483647`. `-1` or `0` disable the alert.

This configuration parameter triggers a `Critical` severity alert named `Yellowbrick Row Store Data Files` when the number of files in the YRS exceeds the configuration threshold. Use the following command to retrieve the count of data files:

```sql
select count(*) from yb_yrs_data_files();
```

yrs_data_files_count_major_threshold (integer)

Platforms: All platforms

Parent topic: Yellowbrick Row Store (YRS) Alerting Parameters

Advanced usage

- Default value (as of this release): `-1` (disabled).
- Valid range: `-1` to `2147483647`. `-1` or `0` disable the alert.

This configuration parameter triggers a `Major` severity alert named `Yellowbrick Row Store Data Files` when the number of files in the YRS exceeds the configuration threshold. Use the following command to retrieve the count of data files:

```sql
select count(*) from yb_yrs_data_files();
```

yrs_data_files_count_minor_threshold (integer)

Platforms: All platforms

Parent topic: Yellowbrick Row Store (YRS) Alerting Parameters

Advanced usage

- Default value (as of this release): `-1` (disabled).
- Valid range: `-1` to `2147483647`. `-1` or `0` disable the alert.

This configuration parameter triggers a `Minor` severity alert named `Yellowbrick Row Store Data Files` when the number of files in the YRS exceeds the configuration threshold. Use the following command to retrieve the count of data files:

```sql
select count(*) from yb_yrs_data_files();
```

yrs_delete_records_count_critical_threshold (integer)

Advanced usage

- Default value (as of this release): `-1` (disabled).
- Valid range: `-1` to `2147483647`. `-1` or `0` disable the alert.

This configuration parameter triggers a `Critical` alert named `Yellowbrick Row Store Delete Records` when the number of deleted records in the YRS exceeds the configuration threshold. Use the following command to retrieve the count of deleted records:

```sql
select count(*) from yb_yrs_delete_files();
```

yrs_delete_records_count_major_threshold (integer)

Advanced usage

- Default value (as of this release): `-1` (disabled).
- Valid range: `-1` to `2147483647`. `-1` or `0` disable the alert.

This configuration parameter triggers a `Major` severity alert named `Yellowbrick Row Store Delete Records` when the number of deleted records in the YRS exceeds the configuration threshold. Use the following command to retrieve the count of deleted records:

```sql
select count(*) from yb_yrs_delete_files();
```

yrs_delete_records_count_minor_threshold (integer)

Advanced usage

- Default value (as of this release): `-1` (disabled).
- Valid range: `-1` to `2147483647`. `-1` or `0` disable the alert.

This configuration parameter triggers a `Minor` severity alert named `Yellowbrick Row Store Delete Records` when the number of deleted records in the YRS exceeds the configuration threshold. Use the following command to retrieve the count of deleted records:

```sql
select count(*) from yb_yrs_delete_files();
```

yrs_unused_files_count_critical_threshold (integer)

Advanced usage

- Default value (as of this release): `-1` (disabled).
- Valid range: `-1` to `2147483647`. `-1` or `0` disable the alert.

This configuration parameter triggers a `Critical` severity alert named `Yellowbrick Row Store Unused Files` when the number of unused files in the YRS exceeds the configuration threshold. Use the following command to retrieve the count of unused files:

```sql
select count(*) from yb_yrs_unused_files();
```

yrs_unused_files_count_major_threshold (integer)

Advanced usage

- Default value (as of this release): `-1` (disabled).
- Valid range: `-1` to `2147483647`. `-1` or `0` disable the alert.

This configuration parameter triggers a `Major` severity alert named `Yellowbrick Row Store Unused Files` when the number of unused files in the YRS exceeds the configuration threshold. Use the following command to retrieve the count of unused files:

```sql
select count(*) from yb_yrs_unused_files();
```

yrs_unused_files_count_minor_threshold (integer)

Advanced usage

- Default value (as of this release): `-1` (disabled).
- Valid range: `-1` to `2147483647`. `-1` or `0` disable the alert.

This configuration parameter triggers a `Minor` severity alert named `Yellowbrick Row Store Unused Files` when the number of unused files in the YRS exceeds the configuration threshold. Use the following command to retrieve the count of unused files:

```sql
select count(*) from yb_yrs_unused_files();
```

Non-functional Parameters

Yellowbrick Documentation > Reference > Configuration Parameters > Non-functional

Platforms: All platforms

Parent topic: Configuration Parameters

NOTE

The set of parameters documented here is incomplete; it will be extended in subsequent releases.

The following parameters are visible in the database but either aren't settable or have no effect whatsoever. They are present for compatibility with PostgreSQL ecosystem tools and clients.

Parameter Name	Default Value	Original semantic
`block_size`	`32768`	Size of a catalog block on disk.
`bytea_output`	`hex`	Required for pgAdmin compatibility; controls how the `bytea` type is sent to the client.
`lc_collate`	`C`	Locale configuration..
`lc_ctype`	`C`	Locale configuration.
`lc_time`	`C`	Locale configuration.
`max_index_keys`	`32`	Ignored maximum number of index keys.
`max_wal_size`	`16GB`	Write-ahead log settings. Not configurable.
`server_version`	`9.5.2`	Reports a server version string. Use `yb_server_version` instead.
`server_version_num`	`90502`	Reports a server version string. Use `yb_server_version_num` instead.
`ssl`	`ON`	Whether SSL connections are enabled.
`transaction_isolation`	`READ COMMITTED`	Transaction isolation level, only READ COMMITTED is supported.
`wal_block_size`	`8192`	Write-ahead log settings. Not configurable.
`wal_level`	`minimal`	Write-ahead log settings. Not configurable.

ybsql Reference

Platforms: All platforms

Parent topic: Reference

In this section:

The `ybsql` client tool is a command-line interface for running SQL queries and SQL commands. You can use `ybsql` to create, drop, and alter database objects, insert, update, and delete rows in tables, and run commands that manage the database. See SQL Commands.

You can download `ybsql` from the Yellowbrick Manager as part of the Yellowbrick client tools package (`ybtools`). Go to **Support** > **Drivers and Tools** > **Client Tools**.

This section describes the commands and options that are available in `ybsql` . See also Common Options in ybtools.

ybsql \copy Command

Yellowbrick Documentation > Reference > ybsql Reference > ybsql \copy Command

Platforms: All platforms

Parent topic: ybsql Reference

In this section:

The `ybsql \copy` command runs a client-based operation that inserts records from a file into a table, or sends rows from a table to a file. You can also use `stdin` and `stdout` for input and output.

Note: The SQL `COPY` command is not supported.

Yellowbrick recommends using the bulk loader (`ybload`) to load large tables into the database. The `\copy` command may be used to load smaller tables or to export table data to files.

```
\copy { table [ ( column_list ) ] | ( query ) }
{ from | to } { 'filename' | stdin | stdout }
[ [ with ] ( option [, ...] ) ]

where option is:

   FORMAT text | csv | binary
   DELIMITER 'delimiter_character'
   NULL 'null_string'
   HEADER [ 'true' | 'false' ]
   QUOTE 'quote_character'
   ESCAPE 'escape_character'
   FORCE_QUOTE { ( column_name [, ...] ) | * }
   FORCE_NULL ( column_name [, ...] )
   FORCE_NOT_NULL ( column_name [, ...] )
   ENCODING 'encoding_name'
```

Note: The options list is enclosed in parentheses and separated by commas. The `WITH` keyword is not required.

The quotes around the file name are optional.

The `ybsql` client reads or writes the file and routes the data between the server and the local file system. Therefore, file accessibility and privileges apply to the local user, not the server. For example, the user running the following command only requires `INSERT` privileges on the `team` table:

```
premdb=# \copy team from newteam.csv with (delimiter ',');
COPY 50
```

\copy Examples

The following example uses the `ENCODING`, `FORMAT`, `DELIMITER`, and `HEADER` options:

```
premdb=# \copy season from '/home/premdata/season.csv'
with (encoding 'latin9',format text, delimiter ',',header 'false')
COPY 25
```

Copy the `season` table to a `csv` file:

```
premdb=# \copy season to '/home/premdata/seasonout.csv' with (format csv)
COPY 25
```

Copy the results of a query to a file:

```
premdb=# premdb=# \copy (select * from team where teamid>25 order by teamid) to '/home/premdata/team25plus' with (delimiter ',');
COPY 25
premdb=# \q
$ more team25plus
26,27,76,Middlesbrough,Boro,Middlesbrough,Riverside Stadium,34742,0.000
27,28,77,Newcastle United,Magpies,Newcastle,St. James Park,52405,49.754
28,29,78,Norwich City,Canaries,Norwich,Carrow Road,27244,26.972
29,30,79,Nottingham Forest,Forest,Nottingham,City Ground,30445,0.000
30,32,80,Oldham Athletic,Latics,Oldham,Boundary Park,13309,0.000
...
```

The following example copies the `match` table to and from a `binary` file:

```
premdb=# \copy(select * from match) to '/home/premdata/matchout' with(format binary);
COPY 8606
premdb=# delete from match;
DELETE 8606
premdb=# \copy match from '/home/premdata/matchout' with(format binary);
COPY 8606
```

The following example copies quoted output to a `csv` file:

```
premdb=# \copy(select * from team) to '/home/premdata/teamout' with(format csv, force_quote *);
COPY 50
premdb=# \q
$ more teamout
"1","2","51","Arsenal","Gunners","London","Emirates Stadium","60260","59.944"
"2","3","52","Aston Villa","Villains","Birmingham","Villa Park","42785","33.690"
"3","4","53","Barnsley","Tykes","Barnsley","Oakwell Stadium","23009","0.000"
...
```

Assume that the source file `season_with_nulls.csv` contains this line, where the character string `null` is in the `winners` field:

```
25,2016-2017,20,null
```

The following `\copy` command loads the `season` table, setting `null` as the null string and forcing the `null` value to be loaded:

```
premdb=# \copy season from '/home/brumsby/season_with_nulls.csv' with(format csv, null 'null', force_null(winners));
COPY 25
```

The following query returns the row in the table that now contains `null`:

```
premdb=# select * from season where winners is null;
 seasonid | season_name | numteams | winners
----------+-------------+----------+---------
       25 | 2016-2017   |       20 |
```

\copy Parameters

FROM

You can copy data from either local source files or `STDIN` and load it into tables or columns in tables.

TO

You can copy data to either local output files or `STDOUT`, using a query, a table, or table columns as the source. The default output format is `text`.

FORMAT

The format of the input or output can be `text`, `csv`, or `binary`.

DELIMITER

A single one-byte character that represents the field delimiter, such as a pipe (|) or a comma (,).

NULL

The string that represents `NULL` values, such as `''` (empty string).

HEADER

Whether the source (or destination file) contains a header row: `'true'` or `'false'`. If you are copying from a file and this option is set to `'true'`, the first line in the file will be skipped. This option applies to `csv` format only. If you specify the option `header` by itself, this also means `'true'`.

QUOTE

A single one-byte character that is used when a data value is quoted. The default value is `"`. This option applies only to `csv` format.

ESCAPE

A single one-byte character that should appear before a data character that matches the `QUOTE` value. The default is the same as the `QUOTE` value (so that the quote character is doubled if it appears in the data). This option applies only to `csv` format.

FORCE_QUOTE

Adds quotes to output column values that are not `NULL`. `NULL` values are never quoted. You can quote specific columns in the table or all columns (`*`). This option applies only to `\copy to` commands that use `csv` format.

FORCE_NULL (column_list), FORCE_NOT_NULL (column_list)

Either match or do not match the values in the specified list of columns against the null string. Both options apply only to `\copy from` commands that use `csv` format.

`FORCE_NOT_NULL`: If a match is found, do not set the value to `NULL`. If the null string is empty (the default), empty values will be read and inserted as zero-length strings, not as `NULL` values, whether or not they are quoted.

`FORCE_NULL`: If a match is found, set the value to `NULL`. If the null string is empty, a quoted empty string is converted to `NULL`.

ENCODING

Typically set to `'utf8'` or `'latin9'`. For `\copy from` commands, see also the discussion of encodings in Creating Databases.

ybsql Properties and Variables

Platforms: All platforms

Parent topic: ybsql Reference

In this section:

The following sections explain how to use predefined `ybsql` properties and user-defined variables in `ybsql` sessions. Properties and variables are useful for setting various aspects of behavior, including your session history and display prompts, and for interpolating specific values in SQL statements and queries.

Examples with ybsql Variables

This section contains some examples of best practices for setting user variables in `ybsql`.

Handling space characters within \set command values

If the value for a variable does not start one space after the variable name in the `\pset` command, or if the value contains spaces, enclose the value in single quotes. For example, the value `Arsenal` below works fine without quotes (value = one word, no spaces):

```
premdb=> \set team Arsenal
premdb=> select nickname from team where name = :'team';
nickname
----------
Gunners
(1 row)
premdb=> \echo :team
Arsenal
```

However, look what happens when the value contains spaces:

```
premdb=> \set team Stoke City
premdb=> \echo :team
StokeCity
```

The space between the words in the value is skipped, causing the variable to be set in an unexpected way. To solve this problem, use single quotes for the value:

```
premdb=# \set team 'Stoke City'
premdb=# \echo :team
Stoke City
```

Escaping embedded single quotes

You can escape literal single quotes that appear inside variable values by using a pair of single quotes within the single-quoted string. For example, compare the results of the following `\pset` commands:

```
premdb=# \set message Don't run this query
unterminated quoted string
premdb=# \set message Don''t run this query
premdb=# \echo :message
Dontrunthisquery
premdb=# \set message 'Don''t run this query'
premdb=# \echo :message
Don't run this query
```

Using concatenated variables to create and populate a table

This example begins by setting up two variables: `db` and `start_time`. This is done in two steps, by running a query that names `db` and `start_time` as column aliases, then using the `\gset` command. The `\echo` command returns the current values of the two variables.

```
premdb=# select current_database() as db, date_trunc('secs',current_timestamp::timestamp) as start_time;
   db   |     start_time
--------+---------------------
 premdb | 2020-07-24 17:28:29
(1 row)

premdb=# \gset
premdb=# \echo Started at :start_time in database :db
Started at 2020-07-24 17:29:45 in database premdb
```

Now create a table with columns that you can populate with the current values of the variables:

```
premdb=# create table job_log (start_time timestamp, db_name varchar(128));
CREATE TABLE
premdb=# \echo Started at :start_time in database :"db"
Started at 2020-07-24 17:35:14 in database "premdb"

premdb=# insert into job_log values (:'start_time',:'db');
INSERT 0 1

premdb=# select * from job_log;
    start_time      | db_name
--------------------+---------
 2020-07-24 17:35:14 | premdb
(1 row)
```

Using backticks to set variable values on Linux clients

If you are running `ybsql` on a Linux client system, you can take advantage of the backtick character (`), as shown in the following examples. You can use backticks to assign the output of a shell command to a variable.

This example sets the shell language based on the results of the shell `echo` command:

```
premdb=> \set shell_lang `echo $LANG`
premdb=> \echo :shell_lang
C.UTF-8
```

This example uses a variable to insert the contents of a file into a table column. Load the file into a variable, then refer to the variable as a quoted string. For example:

```
premdb=# \set content `cat /home/premdata/seasonid.txt`
premdb=# insert into season(seasonid) values(:'content');
INSERT 0 1
```

ybsql Display Properties

The following settings modify the display of `ybsql` output. You can set these display options in two different ways:

- When starting a session:

```
ybsql --pset=NAME[=VALUE]
```

- Inside a session:

```
\pset NAME [VALUE] inside ybsql
```

border

In HTML format, a number that defines the `border=` attribute. In most other formats, only values `0` (no border), `1` (internal dividing lines), and `2` (table frame) make sense; values above 2 will be treated the same as `border=2`. In the `latex` and `latex-longtable` formats, a value of `3` adds dividing lines between data rows.

columns

Target width for `wrapped` format, as well as the width limit for determining whether output is wide enough to require the pager or to switch to the vertical display in expanded auto mode. If set to zero (the default), the environment variable `COLUMNS` controls the target width (or the detected screen width controls it if `COLUMNS` is not set). If `COLUMNS` is zero, the wrapped format only affects screen output. If `COLUMNS` is non-zero, file and pipe output is wrapped to that width as well.

expanded (or x)

Set to `on` or `off`, which enables or disables expanded mode, or `auto`. If no value is set, the `\pset expanded` toggles between `on` and `off`. In `expanded` mode, query results are displayed in two columns, with the column name on the left and the data on the right. This mode is useful if the data would not fit on the screen in the normal "horizontal" mode. When `auto` is set, the `expanded` mode is used whenever the query output is wider than the screen; otherwise, the regular mode is used. The `auto` setting is only effective for `aligned` and `wrapped` formats. For other formats, `auto` behaves as if the expanded mode is `off`.

fieldsep

Field separator to be used in `unaligned` output format so you can create tab- or comma-separated output, for example, which other programs might prefer. To set a tab as the field separator, type `\pset fieldsep '\t'`. The default field separator is `'|'` (a vertical bar).

fieldsep_zero

Zero-byte field separator for use in `unaligned` output format.

footer

Set to `on` or `off`, which enables or disables display of the table footer (row count). If no value is specified, `\pset footer` toggles between `on` and `off`.

format

Output format:

- `unaligned` : all columns of a row on one line, separated by the current field separator. Useful for output intended to be read in by other programs (for example, tab-separated or comma-separated format).
- `aligned` : standard, human-readable, well-formatted text output (the default).
- `wrapped` : similar to `aligned` but wraps wide data values across lines so the output fits in the target column width. The target width is determined as described under the `columns` option. `ybsql` does not attempt to wrap column header titles; therefore, `wrapped` format behaves the same as `aligned` if the total width needed for column headers exceeds the target.
- `html` , `asciidoc` , `latex` , `latex-longtable` , `troff-ms` : table output intended to be included in documents that use the respective mark-up language.

linestyle

Border line drawing style: `ascii` (the default) or `unicode` . This option only applies to the `aligned` and `wrapped` formats.

- `ascii` : plain ASCII characters. Newlines in data are shown by a `+` symbol in the right margin. When the `wrapped` format wraps data from one line to the next without a newline character, a dot (`.`) is shown in the right margin of the first line, and again in the left margin of the following line.
- `unicode` : Unicode box-drawing characters. Newlines in data are shown using a carriage return symbol in the right margin. When the data is wrapped from one line to the next without a newline, an ellipsis is shown in the right margin of the first line, and again in the left margin of the following line.

When the `border` setting is greater than zero, the `linestyle` option also determines the characters with which the border lines are drawn.

null

String to print when a null value is displayed. The default is to print nothing, which is easily mistaken for an empty string. For example: `\ pset null '(null)'`

pager

Control use of a pager program for query and `ybsql` help output. If the environment variable `PAGER` is set, the output is piped to the specified program. Otherwise a platform-dependent default (such as `more`) is used.

- `off` : do not use the pager program.
- `on` : use the pager when the output is to a terminal and does not fit on the screen.
- `always` : use the pager for all terminal output regardless of whether it fits on the screen.
- `\pset pager` without a value toggles pager use on and off.

pager_min_lines

If set a number greater than the page height, the pager program is not called unless there are at least this many lines of output to show. The default setting is `0` .

recordsep

Record (line) separator to use for `unaligned` format. The default is a newline character.

recordsep_zero

Zero-byte record separator for `unaligned` format.

tableattr (or T)

In `HTML` format, set attributes to be placed inside the table tag, such as `cellpadding` or `bgcolor` . (Note that border can be set with `\pset border`). If no value is given, the table attributes are unset.

In `latex-longtable` format, control the proportional width of each column that contains a left-aligned data type. Specified as a whitespace-separated list of values; for example: `'0.2 0.2 0.6'` . Unspecified output columns use the last specified value.

title

Descriptive table title for subsequent table output. If no value is specified, the title is unset.

tuples_only

`on` or `off`, which enables or disables tuples-only mode. If no value is specified, the command toggles between regular and tuples-only output. Regular output includes column headers, titles, and footers. Tuples-only mode shows only the table data (query results).

unicode_border_linestyle, unicode_column_linestyle, unicode_header_linestyle

`single` or `double` for Unicode border lines, column lines, and headers.

Examples

For readability, change the default `aligned` format for query output to `unaligned` so that all of the columns in a row appear on one line. You can use this command in combination with `\pset recordsep '\n\n'` to add an extra newline between each row. For example:

```
Output format is unaligned.
premdb=# \pset recordsep '\n\n'
Record separator is "

".
premdb=# SELECT query_id, query_text FROM sys.log_query limit 3;
query_id|query_text

19340|ANALYZE HLL public.awayteam

19328|insert into awayteam select atid,name from team;

19359|ANALYZE HLL public.hometeam

(3 rows)
```

Set an output string for `NULL` values. For example:

```
premdb=# \pset null NULL
Null display is "NULL".
premdb=#
premdb=# select * from season where winners is null;
 seasonid | season_name | numteams | winners
----------+-------------+----------+---------
       25 | 2016-2017   |       20 | NULL
(1 row)
```

ybsql Prompt Variables

You can use the following syntax to customize your `ybsql` prompts (`PROMPT1` and `PROMPT2` variables). The default value for both prompts is `'%/%R%# '`. To place a percentage sign in your prompt, use `%%`.

Substitution	Description
`%M`	Full host name of the database server; `[local]` if the connection is over a Linux domain socket.
`%m`	Host name of the database server, truncated at the first dot; `[local]` if the connection is over a Linux domain socket.
`%>`	Port number where the database server is listening.
`%n`	The database session user name. (The prompt may change during a session if a `SET SESSION AUTHORIZATION` command is run.)
`%/`	Name of the current database.
%~	Like `%/`, but the output is `~` (tilde) if the database is your default database.
`%#`	`#` if the session user is a database superuser; otherwise, `>`. (The prompt may change during a session if a `SET SESSION AUTHORIZATION` command is run.)
`%R`	`PROMPT1`: `=`, `^` if in single-line mode, `!` if the session is disconnected from the database (if `\connect` fails). `PROMPT2`: `%R` is replaced by one of the following characters: - `-` if the command was not terminated - `*` if there is an unfinished `/* ... */` comment - `'` if there is an unfinished quoted string - `"` if there is an unfinished quoted identifier - `$` if there is an unfinished dollar-quoted string - `(` if there is an unmatched left parenthesis
`%x`	Transaction status: - An empty string when not in a transaction block - `*` when in a transaction block - `!` when in a failed transaction block - `?` when the transaction state is unknown (for example, if there is no connection)
`%l`	Line number inside the current statement, starting from `1`.
`%digits`	Character with the specified octal code is substituted. For example: `%044%040` (`$` character, then a space character)
`%:name`	Value of the `ybsql` variable name. See ybsql User Variables. For example: `%:ENCODING`
`%` command``	Output of the specified `command` (similar to ordinary "back-tick" substitution). For example: `%` whoami``
`%[ ... %]`	Prompts can contain terminal control characters that, change the color, background, or style of the prompt text, or the title of the terminal window, for example. Multiple pairs of `%[` and `%]` can occur within the prompt.

Examples

```
premdb=# \set PROMPT1 'myprompt '
myprompt
```

```
premdb=# \set PROMPT1 '%/%R%# '
premdb=#
```

```
premdb=# \set PROMPT1 %044%040
$
```

```
$ \set PROMPT1 %n%100%/%044%040
ybuser1@premdb$
```

```
ybuser1@premdb$ \set PROMPT1 %M%044%040
[local]$
```

```
[local]$ \c premdb bobr
Password for user bobr:
You are now connected to database "premdb" as user "bobr".
premdb=>  \set PROMPT1 %n%044%R%040
bobr$=
```

```
bobr$=  \set PROMPT2 %n%044%R%040
bobr$= select * from team
bobr$- ;
...
```

```
ybuser1@ybuser1:/usr/bin$ ./ybsql premdb bobr --set=PROMPT1='bobr_yb '
Password for user bobr:

bobr_yb
```

ybsql Properties

Yellowbrick Documentation > Reference > ybsql Reference > ybsql Properties and Variables > ybsql Properties

Platforms: All platforms

Parent topic: ybsql Properties and Variables

The following properties are predefined options that you can set at run time or in your `ybsql` startup file (`.ybsqlrc`). By convention, these properties have uppercase names; avoid using the same names in your user-defined variables. For more information about each property, see Property Details.

Property Name	Short Description
AUTOCOMMIT on | off	Whether individual commands commit when they complete.
DBNAME	Current database name.
ECHO all | queries | errors	Display setting.
ENCODING	Current encoding.
FETCH_COUNT number	Number of rows to fetch and display.
HISTCONTROL, HISTFILE, HISTSIZE	History file settings.
HOST	Current database server host.
IGNOREEOF	Behavior when EOFs are sent.
ON_ERROR_ROLLBACK, ON_ERROR_STOP	Behavior when errors are returned.
PORT	Current database server port.
QUIET	Equivalent to `-q` .
SINGLELINE, SINGLESTEP	Equivalent to the `-S` and `-s` .
USER	Current database user.
VERBOSITY default | terse | verbose	Verbosity of error messages.

To set a property or a variable, use the `\set` command. To see the its value, use the `\echo` command with a colon in front of the property or variable name. For example, set the `HISTSIZE` property and return its value:

```
premdb=> \set HISTSIZE 100
premdb=> \echo :HISTSIZE
100
```

Property Details

AUTOCOMMIT

Values are `on` (the default) and `off` . `AUTOCOMMIT on` means that each SQL command is automatically committed when it completes. `AUTOCOMMIT off` means that SQL commands are not committed until a `COMMIT` or `END` statement is run explicitly.

For example:

```
premdb=# \set AUTOCOMMIT off
premdb=# update season set seasonid=30 where winners is null;
UPDATE 1
premdb=# end;
COMMIT
```

When `AUTOCOMMIT` is `off`, you must explicitly abandon any transaction that fails by using an `ABORT` or `ROLLBACK` command. If you exit from a session without committing a transaction, all of the changes are lost.

Warning: Yellowbrick Data recommends never setting `AUTOCOMMIT` to `off`. Leaving transactions open can have a serious impact on the entire system, such as preventing locks from being released and causing excessive resource consumption.

DBNAME

Name of the currently connected database, which is set on every connection but can be unset.

ECHO

Set to `all`, `queries`, or `errors`. These values are equivalent to the `-a`, `-e`, and `-b` options, respectively. If unset, or set to `none`, no queries are displayed.

ENCODING

The current client character set encoding.

FETCH_COUNT

An integer value that defines how many query result rows to fetch and display at a time. By default, `ybsql` fetches the entire result set before displaying it.

HISTCONTROL

`ignorespace`: lines that begin with a space are not saved in the history list. `ignoredups`: lines that match the previous history line are not saved. `ignoreboth` is a combination of the other two options. If this variable is unset or set to `none`, all lines read in interactive mode are saved.

HISTFILE

Name of the file where the history list is stored. The default name is `.ybsql_history` (or `ybsql_history` on Windows). The file is found in the user's home directory on Linux clients and in `%APPDATA%\postgresql` on Windows clients.

HISTSIZE

The number of commands to store in the command history. The default value is 500.

HOST

Database server host for the current connection, which is set on every connection but can be unset.

IGNOREEOF

Unset: sending an `EOF` character (`Control+D`) to an interactive `ybsql` session causes the application to quit. Set to a numeric value: number of `EOF` characters that are ignored before `ybsql` quits. If the variable is set but has no numeric value, the default is `10`. In the following example, the user enters `Control+D` three times before `ybsql` quits:

```
premdb=# \set IGNOREEOF 3
premdb=# Use "\q" to leave ybsql.
premdb=# Use "\q" to leave ybsql.
premdb=# \q
ybuser1@ybuser1:/usr/bin$
```

ON_ERROR_ROLLBACK

`on`: if a statement in a transaction block generates an error, ignore the error and continue. `interactive`: ignore errors only in interactive sessions, not when reading scripts. Unset or `off`: a statement in a transaction block that generates an error aborts the whole transaction.

ON_ERROR_STOP

By default, command processing continues when an error occurs. `on` or `true`: processing stops immediately. In interactive mode, `ybsql` returns to the command prompt; otherwise, `ybsql` exits, returning error code 3 to distinguish this case from unrecoverable error conditions, which are reported using error code 1. In either case, any running scripts (the top-level script and any other scripts that it may have invoked) terminate immediately. If the top-level command string contained multiple SQL commands, processing stops with the current command.

For example, if the user's startup file (`.ybsqlrc`) contains `\set ON_ERROR_STOP true` and a SQL script returns an error because a table specified in a command does not exist, the script stops running (and returns error code 3):

```
yellowbrick=# \i premdb.ddl
DROP DATABASE
CREATE DATABASE
You are now connected to database "premdb" as user "yb100".
ybsql:premdb.ddl:7: ERROR:  table "season" does not exist
premdb=# \q
```

If `ON_ERROR_STOP` is not set or set to `false`, the script reports errors but keeps running:

```
yellowbrick=# \i premdb.ddl
DROP DATABASE
CREATE DATABASE
You are now connected to database "premdb" as user "yb100".
ybsql:premdb.ddl:7: ERROR:  table "season" does not exist
ybsql:premdb.ddl:8: ERROR:  table "team" does not exist
ybsql:premdb.ddl:9: ERROR:  table "hometeam" does not exist
ybsql:premdb.ddl:10: ERROR:  table "awayteam" does not exist
ybsql:premdb.ddl:11: ERROR:  table "match" does not exist
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
...
```

PORT

Database server port for the current connection, which is set on every connection but can be unset.

PROMPT1, PROMPT2, PROMPT3

Strings that define the appearance of the `ybsql` prompt. `PROMPT1` is the normal prompt that is issued when `ybsql` starts or is ready for a new command.

`PROMPT2` is returned when a command is incomplete: for example, because it does not end with a semicolon or is missing a closing quote. Almost any text string enclosed by single quotes that you enter for the prompt value will be displayed as is. However, you can use several `%` escape sequences to plug in specific values for the current user, host, database, and so on.

`PROMPT3` is returned during `COPY...FROM STDIN` operations.

See ybsql Prompt Variables for examples and details about customizing prompts.

QUIET

`on` is equivalent to the `-q` option.

SINGLELINE

`on` is equivalent to the `-S` option.

SINGLESTEP

`on` is equivalent to the `-s` option.

USER

Database user for the current connection, which is set on every connection but can be unset.

VERBOSITY

`default`, `terse`, or `verbose` error messages. For example:

```
premdb=# \set VERBOSITY verbose
premdb=# select * from not_a_table;
ERROR:  42P01: relation "not_a_table" does not exist
LINE 1: select * from not_a_table;
                      ^
premdb=# \set VERBOSITY terse
premdb=# select * from not_a_table;
ERROR:  relation "not_a_table" does not exist at character 15
```

ybsql User Variables

You can set `ybsql` properties and user-defined variables in your startup file or during a `ybsql` session. Properties and variables are name/value pairs, where the value can be any string of any length. The name must consist of letters, digits, and underscores.

To set a variable, use the `\set` command. To see the its value, use the `\echo` command with a colon in front of the variable name. To interpolate a value in a SQL statement, again enter a colon in front of the name.

The value of a `ybsql` variable or property is copied literally and may contain unbalanced quotes, or even backslash commands. Therefore, you need to place variables and properties carefully in the context of other SQL syntax. When you want a variable value to be used as an SQL literal or identifier, quoting is recommended. To quote the value of a variable as an SQL literal, put the variable name in single quotes. To quote the value as an SQL identifier, put the variable name in double quotes.

For example, create (or set) a user variable `t1` and reference it in a query.

```
premdb=# \set t1 'match'
premdb=# select * from :"t1" limit 2;
 seasonid |      matchday       | htid | atid | ftscore | htscore
----------+---------------------+------+------+---------+---------
       21 | 2012-12-22 00:00:00 |   44 |   78 | 2-1     | 1-1
       21 | 2012-10-06 00:00:00 |   44 |   82 | 3-2     | 2-1
(2 rows)
```

If you do not specify a value in the `\set` command, the property or variable is set to an empty string.

To unset (or delete) a variable or property, use the `\unset` command. To show the values of all properties and variables, run `\set` without any arguments. For example:

```
premdb=# \set
AUTOCOMMIT = 'on'
...
DBNAME = 'premdb'
USER = 'yb100'
PORT = '5432'
ENCODING = 'UTF8'
HISTFILE = '/home/yb100/newybsqlhist'
...
db2 = 'premdb2'
```

See also `\gset` and `\prompt` in ybsql Slash (\) Commands.

ybsql Command-Line Options

In addition to the connection options, you can specify several other options on the `ybsql` command line. See also ybsql Examples.

-a, --echo-all
Print all input lines to standard output as they are read. This does not apply to empty lines or lines read interactively. This option has the same effect as setting the `ECHO` variable to `all`.

-A, --no-align
Switch the output to unaligned mode. The default output mode is aligned.

-b, --echo-errors
Print failed SQL commands to standard error output. This option has the same effect as setting the `ECHO` variable to `errors`.

-c command --command=command
Execute the contents of one command string, then exit. The command must be either a command string that does not contain any syntax specific to `ybsql` or a single backslash command (such as `\d`). If the command string contains multiple SQL commands, they are processed in a single transaction (except when the string includes explicit `BEGIN/COMMIT` commands).

-e, --echo-queries
Copy all SQL commands to standard output. This option has the same effect as setting the `ECHO` variable to `queries`.

-f filename, --file=filename
Read commands from a named file instead of interactively, then exit (similar to using the `\i` command). If the specified file is `-` (a hyphen), `ybsql` reads from standard input until an `EOF` or a `\q` command is found.

-F separator, --field-separator=separator
Use the specified separator between fields for unaligned output (`-A`). This option has the same effect as `\pset fieldsep` or `\f`.

-H, --html
Return HTML tabular output. This option has the same effect as `\pset format html` or `\H`.

--help
Return a summary of the `ybsql` command-line options.

-l, --list
List all available databases, then exit (similar to `\list`). Other non-connection options are ignored.

-L filename --log-file=filename
Send all query output to the specified file, in addition to the normal output destination.

-o filename --output=filename
Send all query output to the specified file only (no normal output, unlike `-L`). This option is equivalent to the `\o` command.

-P assignment, --pset=assignment
Set printing options (as with the `\pset` command). You must separate the name and value with an equal sign, not with a space. For example, to set the output format to LaTeX: `-P format=latex`.

-q --quiet
Suppress welcome messages and other informational output (useful when running a command with the `-c` option). This option has the same effect as setting the variable `QUIET` to `on`.

-R separator --record-separator=separator

Use the specified separator between records for unaligned output (`-A`). This option has the same effect as the `\pset recordsep` command.

--set=NAME=VALUE
Set a `ybsql` variable. See ybsql Properties and Variables.

-s --single-step
Run in single-step mode, which means that `ybsql` prompts the user before sending each command and provides an option to cancel execution. Use this mode to debug scripts.

-S --single-line
Run in single-line mode, which means that a newline terminates an SQL command (as a semicolon does). Use this option carefully (for example, when reading commands from files, where a single SQL command may be split into multiple lines).

-t --tuples-only
Do not print column names in query results or row counts at the end of the result set. This option has the same effect as the `\t` command.

-T table_options --table-attr=table_options
Specify attributes to use within the table tag for HTML output (`-H`), such as `cellpadding` or `bgcolor` .

-V, --version
Print the `ybsql` version and exit.

-z, --field-separator-zero
Set the field separator for unaligned output (`-A`) to a zero byte.

-x, --expanded
Use expanded mode for table formatting. This option has the same effect as the `\x` command.

-X, --no-ybsqlrc
Do not read the user's startup file (`~/.ybsqlrc`), if one exists in the user's home directory. See ybsql Startup File.

-0, --record-separator-zero
Set the record separator for unaligned output (`-A`) to a zero byte.

-1, --single-transaction
Execute a script in `ybsql` as a single transaction by wrapping the script with `BEGIN` and `COMMIT` commands. Either all the commands complete successfully, or no changes are applied. If the script itself uses `BEGIN` , `COMMIT` , or `ROLLBACK` , this option will not work. Also, if the script contains any command that cannot be executed inside a transaction block, this option will cause that command (and the whole transaction) to fail.

ybsql Connections

This section describes options for connecting to a database with `ybsql`. See also Setting Up a Database Connection.

Syntax Summary

```
ybsql [ connection_options ] [ other options ]
```

The order of the general options and connection options may be reversed in the command.

Connection Options

```
[ -h | --host hostname ]
[ -p | --port portnumber ]
[ -d | --dbname dbname ]
[ --cluster clustername ]
[ -U | --username username ]
[ -W | --password ]
[ -w | --no-password ]
```

-h, --host

Host name of the data warehouse instance.

-p, --port

Port number for the connection (`5432` by default).

-d, --dbname

Name of the database for the connection. Alternatively, you can specify `dbname` by itself as the first argument on the command line. For example, all of the following syntax variations are valid, where `premdb` is the database name:

```
ybsql premdb
ybsql -d premdb
ybsql --dbname premdb
```

--cluster

Name of the compute cluster for the connection.

-U, --username

Name of the database user for the connection. Alternatively, you can specify `username` by itself if the database name precedes it. For example, all of the following syntax variations are valid, where `bobr` is the username:

```
ybsql -U bobr premdb
ybsql --username bobr premdb
ybsql premdb bobr
```

If you do not specify a user, the default user is the current OS user on the client system.

-W, --password

Prompt for the user's password before connecting. Regardless of this option, `ybsql` prompts for the password automatically when the server requires password authentication. This option remains in effect for the entire session and affects the use of any `\connect` commands.

-w, --no-password

Do not prompt for the user's password. If the server requires password authentication and a password is not set with the `YBPASSWORD` environment variable, the connection fails. This option may be useful for non-interactive batch jobs and scripts. It remains in effect for the entire session and affects the use of any `\connect` commands.

Security Options

See also the discussion of SSL modes in Secure Connections for ODBC/JDBC Clients and ybsql.

--cacert STRING

Customize trust with secured communication. Enter the file name of a custom PEM-encoded certificate or the file name and password for a Java KeyStore (JKS).

For PEM format, the file must be named with a `.pem`, `.cert`, `.cer`, `.crt`, or `.key` extension. For example:

```
--cacert cacert.pem
```

For JKS format, files are always password-protected. Use the following format:

```
--cacert yellowbrick.jks:changeit
```

where the `:` character separates the file name from the password of the keystore.

See also Verifying SSL/TLS Encryption.

--disable-trust

Disable SSL/TLS trust when using secured communications. Trust is enabled by default. See also Verifying SSL/TLS Encryption.

Important: This option is not supported for use on production systems and is only recommended for testing purposes. It may be useful to disable trust during testing, then enable it when a formal signed certificate is installed on the cluster.

External Authentication Options

--auth-browser

Authenticate the connection by opening a browser window on the client machine and entering a password. `-U username` is required when you use this option. You may need to make sure that your client system is configured to open a default browser. If a browser is not available (ie. restricted console, ssh), fallback to device login if the authorization endpoint supports it.

--auth-device

Authenticate the connection by device login flow, utilizing a browser URL. You may use this option on a restricted console, copy the URL given and code, and proceed to authenticate with a separate browser or device that has a browser.

--auth-direct

Authenticate the connection directly from the `ybtools` command (as opposed to opening a browser). The `-U username` option and `YBPASSWORD`, `--password`, or `-W` are required when you use direct authentication.

--auth-issuer

Specify the URL of the OAuth/OIDC issuer. This URL belongs to the identity provider (IDP) that issues JWT tokens for user connections. The IDP checks that the user exists in the specified realm, then returns a JWT token, which is a short-lived password for authentication for that user. Alternatively, you can set the `YBAUTHISSUER` environment variable.

This option must be specified for any kind of OAuth/OIDC connection.

For example:

```
--auth-issuer https://cn.beta.yellowbrickcloud.com/auth/realms/mycompany.com
```

where `cn.beta.yellowbrickcloud.com/auth` is the Yellowbrick IDP, which has some number of realms, such as `mycompany.com`.

--auth-scopes

Supply override for scopes requested when authenticating. Some IDPs such as Azure AD require this to be specified as <client_id>/.default, which returns an access token that can be validated by an external resource owner, such as Yellowbrick Database. Alternatively, you can set the `YBAUTHSCOPES` environment variable.

--auth-token AUTH_TOKEN

Supply a JWT string in the command line explicitly. Alternatively, you can set the `YBAUTHTOKEN` environment variable.

--client-id

Specify the Yellowbrick Data Warehouse client ID, which is `ybauth`. Alternatively, you can set the `YBCLIENTID` environment variable.

This option is required for OAuth/OIDC authentication.

Connecting with a Secure Password

You can use the `ybsql \password` command to reset a password for a user securely. Passwords set or changed with this command are not exposed in plain text or saved to any log files. For example:

```
premdb=# create user jamesbond;
CREATE ROLE
premdb=# \password jamesbond
Enter new password:
Enter it again:
...
premdb=# \c premdb jamesbond
Password for user jamesbond:
You are now connected to database "premdb" as user "jamesbond".
...
```

CAUTION:

If you create and update unencrypted passwords with CREATE ROLE and ALTER ROLE commands, passwords are transmitted in plain text and may appear in log messages.

ybsql URI Syntax

`ybsql` supports an alternative PostgreSQL connection method that uses a connection string of the following form:

```
ybsql postgresql://password:user@host/dbname
```

The keywords `postgresql` and `postgres` are both supported at the beginning of the connection string.

For example:

```
% ybsql postgres://bobrum:trebor@ybbob-azure.testue1.az.yellowbrickcloud.com/premdb
ybsql (6.1.12)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)
Type:  \h for help with SQL commands
       \? for help with ybsql commands
       \g or terminate with semicolon to execute query
       \q to quit
```

In this example:

- User = `bobrum`
- Password = `trebor`
- Host = `ybbob-azure.testue1.az.yellowbrickcloud.com` (where `ybbob-azure` is the data warehouse instance name)
- Database name= `premdb`

You can use this connection string in combination with environment variables that are already set. For example, if you have `YBPASSWORD` set:

```
% ybsql postgresql://bobrum@ybbob-azure.testue1.az.yellowbrickcloud.com/premdb
```

For example, if you have `YBPASSWORD` and `YBUSER` set:

```
% ybsql postgresql://ybbob-azure.testue1.az.yellowbrickcloud.com/premdb
```

Configuration parameter variables can be set at the end of the connection string. Depending on your shell, you may need to use quotes around the connection string when you specify variables. For example:

```
% ybsql 'postgresql://bobrum:bobrum@yb-demo-azure.produe1.az.yellowbrickcloud.com/premdb?connect_timeout=5&application_name=YBSQLM
```

Additional keywords may be specified, as documented for PostgreSQL (`psql`); however, some of these keywords may not be meaningful for a `ybsql` connection.

Keyword/Value Connection Strings

A similar connection method to the URI syntax uses key/value pairs. The same keywords may be used. For example:

```
% ./ybsql 'host=ybbob-azure.testue1.az.yellowbrickcloud.com user=bobrum password=bobrum dbname=premdb'
```

ybsql Environment Variables

You can set the following environment variables in two ways:

- When starting a session:

```
NAME=VALUE [NAME=VALUE] ybsql
```

- Inside a session:

```
\setenv NAME [VALUE]
```

COLUMNS

If `\pset` columns is zero, controls the width for the `wrapped` format, and also the width for determining if wide output requires the pager or should be switched to the vertical format in expanded mode.

PAGER

Name of an external pager command. If query results do not fit on the screen, they are piped through this command (for example, `more` or `less`). The default is platform-dependent. You can disable the pager by setting `PAGER` to empty or by using pager-related options of the `\pset` command.

SHELL

Shell used by the `\!` command.

TMPDIR

Directory for temporary files. The default is `/tmp`.

YBAPPNAME

Equivalent to the `application_name` connection parameter. The Yellowbrick tools set appropriate names for themselves, such as `ybsql`. The value of `YBAPPNAME` may be used as the name for a custom application that does not set its own default name.

YBAUTHISSUER

URL of the OAuth/OIDC issuer. See Common Environment Variables in ybtools.

YBCLIENTID

Yellowbrick Data Warehouse client ID (`ybauth`). See Common Environment Variables in ybtools.

YBCLUSTER

Name of the compute cluster. See Common Environment Variables in ybtools.

YBDATABASE

Name of the destination database. Default: `yellowbrick`. See Common Environment Variables in ybtools.

YBHOST

Name of the destination server host. Default: `localhost`. See Common Environment Variables in ybtools.

YBPASSFILE

Name of a file where passwords are stored. The default file is in the user's home directory on Linux platforms and is named `.ybpass`. On Windows platforms, the file is:

```
%APPDATA%\yellowbrick\ybpass.conf
```

where `%APPDATA%` refers to the Application Data subdirectory in the user's profile.

The file itself should contain entries like this:

```
hostname:port:database:username:password
```

YBPASSWORD

Interactive prompt for the database user's password. No default. See Common Environment Variables in ybtools.

YBPORT

The destination server port number. Default: `5432`. See Common Environment Variables in ybtools.

YBSSLMODE

SSL/TLS mode used for the connection. See Secure Connections for ODBC/JDBC Clients and ybsql

YBSQL_EDITOR, EDITOR, VISUAL

Editor used by the `\e` command. Default editors are `vi` on Linux and `notepad.exe` on Windows.

YBSQL_EDITOR_LINENUMBER_ARG

The command-line argument that passes the starting line number to the editor when the `\e` command specifies a line number. The value of this variable must be a valid argument for the editor being used. For editors such as `Emacs` or `vi`, this value is a plus sign. Include a trailing space in the value of the variable if a space is required between the option name and the line number.

YBSQL_HISTORY

Alternative location for the command history file.

YBSQLRC

Alternative location for the user's `.ybsqlrc` or `ybsqlrc.conf` file. See ybsql Startup File.

YBUSER

The database login username. The default user is the current OS user on the client system. See Common Environment Variables in ybtools.

Examples

```
$ YBDATABASE=premdb YBUSER=bobr SHELL=bash ./ybsql
Password:
Null display is "[NULL]".
Expanded display is used automatically.
ybsql (1.2.1.5007)
Type "help" for help.
```

```
premdb=>
```

```
premdb=> \setenv YBUSER bobr
premdb=> \! env
...
COLUMNS=136
YBUSER=bobr
SHELL=bash
PWD=/usr/bin
YBDATABASE=premdb
premdb=>
```

```
premdb=# \setenv YBSQL_EDITOR 'emacs'
\setenv YBSQL_EDITOR_LINENUMBER_ARG '+'
premdb=# \e /home/premdb/match.csv 8500
...
```

ybsql Examples

This section presents examples of some of the `ybsql` command-line options.

Connection Options

To connect to database `premdb` on host `yb007` as user `jamesbond`:

```
me@yb100:~$ ybsql -h yb007 premdb -U jamesbond
Password for user jamesbond:
ybsql (1.2.2)
Type:  \h for help with SQL commands
       \? for help with ybsql commands
       \g or terminate with semicolon to execute query
       \q to quit
```

-c Option

Use the `-c` option to run a `ybsql` backslash command, such as `\d`, or to run a query:

```
me@yb100:~$ ybsql premdb bobr -c "\d"
Password for user bobr:
         List of relations
 Schema |   Name   | Type  |  Owner
--------+----------+-------+---------
 public | awayteam | table | brumsby
 public | hometeam | table | brumsby
 public | match    | table | brumsby
 public | season   | table | brumsby
 public | team     | table | brumsby
(5 rows)
```

```
me@yb100:~$ ybsql premdb bobr -c 'select count(*) from match;'
Password for user bobr:
 count
-------
  8606
(1 row)
```

-e Option

Use the `-e` option to print SQL query statements as part of the output:

```
me@yb100:~$ ybsql premdb -e
ybsql (1.2.1)
```

```
Type "help" for help.

premdb=# select * from team where teamid=21;
select * from team where teamid=21;
 teamid | htid | atid |     name     | nickname | city  |   stadium   | capacity | avg_att
--------+------+------+--------------+----------+-------+-------------+----------+---------
    21 |   22 |   71 | Leeds United | Whites   | Leeds | Elland Road |    39460 |   0.000
(1 row)
```

-A Option

Use the `-A` option to return unaligned output:

```
me@yb100:~$ ybsql premdb -A
ybsql (1.2.2)
Type:  \h for help with SQL commands
      \? for help with ybsql commands
      \g or terminate with semicolon to execute query
      \q to quit

premdb=# select * from team where teamid=21;
teamid|htid|atid|name|nickname|city|stadium|capacity|avg_att
21|22|71|Leeds United|Whites|Leeds|Elland Road|39460|0.000
(1 row)
```

Combination of -A, -F, -e, and -f options

Use this combination of options to run a query from a named file, echo the file in the output, return unaligned output, and set a field separator:

```
me@yb100:~$ ybsql premdb -A -F ':' -e -f goals_per_season.sql
select t1.season_name, t1.winners, homegoals+awaygoals as total
from
(select season_name, winners, sum(substr(ftscore,1,1)::int) homegoals
from season, match
where season.seasonid=match.seasonid
group by season_name, winners) t1,
(select season_name, winners, sum(substr(ftscore,3,1)::int) awaygoals
from season, match
where season.seasonid=match.seasonid
group by season_name, winners) t2
where t1.season_name=t2.season_name
order by 1,2;
season_name:winners:total
1992-1993:Manchester United:1222
1993-1994:Manchester United:1195
1994-1995:Blackburn Rovers:1195
1995-1996:Manchester United:988
1996-1997:Manchester United:970
1997-1998:Arsenal:1019
1998-1999:Manchester United:959
1999-2000:Manchester United:1060
2000-2001:Manchester United:992
2001-2002:Arsenal:1001
2002-2003:Manchester United:1000
2003-2004:Arsenal:1012
2004-2005:Chelsea:975
2005-2006:Chelsea:944
2006-2007:Manchester United:931
2007-2008:Manchester United:1002
2008-2009:Manchester United:942
2009-2010:Chelsea:1053
```

```
2010-2011:Manchester United:1063
2011-2012:Manchester City:1066
2012-2013:Manchester United:1063
2013-2014:Manchester City:1052
(22 rows)
```

-H option

Use the `-H` option to return the result set in HTML format:

```
me@yb100:~$ ybsql premdb -H -f total_goals_per_season_avg_per_match.sql
<table border="1">
  <tr>
   <th align="center">season_name</th>
   <th align="center">numteams</th>
   <th align="center">total_goals</th>
   <th align="center">goals_per_match</th>
  </tr>
  <tr valign="top">
   <td align="left">1995-1996</td>
   <td align="right">20</td>
   <td align="right">988</td>
   <td align="right">2.60</td>
  </tr>
  <tr valign="top">
   <td align="left">1996-1997</td>
   <td align="right">20</td>
   <td align="right">970</td>
   <td align="right">2.55</td>
...
```

-H and -T Options

Display the output in HTML format and use `-T` to set one or more HTML attributes:

```
me@yb100:~$ ybsql premdb -f total_goals_per_season_avg_per_match.sql -H -T bgcolor=red
<table border="1" bgcolor=red>
  <tr>
   <th align="center">season_name</th>
   <th align="center">numteams</th>
   <th align="center">total_goals</th>
   <th align="center">goals_per_match</th>
  </tr>
...
```

-o Option

Use the `-o` option to send query results to a file:

```
me@yb100:~$ ybsql premdb -f total_goals_per_season_avg_per_match.sql -o gps.txt
brumsby@brumsby:~$ more gps.txt
 season_name | numteams | total_goals | goals_per_match
-------------+----------+-------------+-----------------
 1995-1996   |       20 |         988 |            2.60
 1996-1997   |       20 |         970 |            2.55
 1997-1998   |       20 |        1019 |            2.68
```

```
 1998-1999   |       20 |         959 |           2.52
 ...
```

-L Option

Use the `-L` option to send query results and the SQL statement itself to a file. Also return the query results to the screen.

```
me@yb100:~$ ybsql premdb -f total_goals_per_season_avg_per_match.sql -L gps.txt
 season_name | numteams | total_goals | goals_per_match
-------------+----------+-------------+-----------------
 1995-1996   |       20 |         988 |           2.60
 1996-1997   |       20 |         970 |           2.55
 1997-1998   |       20 |        1019 |           2.68
 1998-1999   |       20 |         959 |           2.52
 1999-2000   |       20 |        1060 |           2.79
 2000-2001   |       20 |         992 |           2.61
 2001-2002   |       20 |        1001 |           2.63
 2002-2003   |       20 |        1000 |           2.63
 2003-2004   |       20 |        1012 |           2.66
 2004-2005   |       20 |         975 |           2.57
 2005-2006   |       20 |         944 |           2.48
 2006-2007   |       20 |         931 |           2.45
 2007-2008   |       20 |        1002 |           2.64
 2008-2009   |       20 |         942 |           2.48
 2009-2010   |       20 |        1053 |           2.77
 2010-2011   |       20 |        1063 |           2.80
 2011-2012   |       20 |        1066 |           2.81
 2012-2013   |       20 |        1063 |           2.80
 2013-2014   |       20 |        1052 |           2.77
(19 rows)

me@yb100:~$ more gps.txt
********* QUERY **********
select *, (total_goals/380.00)::dec(3,2) as goals_per_match
from (
select season_name, numteams,
sum(substr(ftscore,1,1)::int)+sum(substr(ftscore,3,1)::int) total_goals
from season, match
where season.seasonid=match.seasonid and season.seasonid>=4
group by season_name,numteams
) t1
order by 1;
**************************

 season_name | numteams | total_goals | goals_per_match
-------------+----------+-------------+-----------------
 1995-1996   |       20 |         988 |           2.60
 1996-1997   |       20 |         970 |           2.55
 1997-1998   |       20 |        1019 |           2.68
 1998-1999   |       20 |         959 |           2.52
 1999-2000   |       20 |        1060 |           2.79
...
```

-P Option

Use the `-P` option to set the text format of the query output:

```
me@yb100:~$ ybsql premdb -f total_goals_per_season_avg_per_match.sql -P format=latex
\begin{tabular}{l | r | r | r}
\textit{season\_name} & \textit{numteams} & \textit{total\_goals} & \textit{goals\_per\_match} \\
\hline
```

```
1995-1996 & 20 & 988 & 2.60 \\
1996-1997 & 20 & 970 & 2.55 \\
1997-1998 & 20 & 1019 & 2.68 \\
1998-1999 & 20 & 959 & 2.52 \\
1999-2000 & 20 & 1060 & 2.79 \\
2000-2001 & 20 & 992 & 2.61 \\
2001-2002 & 20 & 1001 & 2.63 \\
2002-2003 & 20 & 1000 & 2.63 \\
2003-2004 & 20 & 1012 & 2.66 \\
2004-2005 & 20 & 975 & 2.57 \\
2005-2006 & 20 & 944 & 2.48 \\
2006-2007 & 20 & 931 & 2.45 \\
2007-2008 & 20 & 1002 & 2.64 \\
2008-2009 & 20 & 942 & 2.48 \\
2009-2010 & 20 & 1053 & 2.77 \\
2010-2011 & 20 & 1063 & 2.80 \\
2011-2012 & 20 & 1066 & 2.81 \\
2012-2013 & 20 & 1063 & 2.80 \\
2013-2014 & 20 & 1052 & 2.77 \\
\end{tabular}

\noindent (19 rows) \\
```

-A and -R Options

Use the `-A` and `-R` options to set unaligned output with a special record separator:

```
me@yb100:~$ ybsql premdb -f total_goals_per_season_avg_per_match.sql -A -R '--'
season_name|numteams|total_goals|goals_per_match--1995-1996|20|988|2.60--1996-1997|20|970|2.55--1997-1998|20|1019|2.68--1998-1999|
```

-s Option

Use the `-s` option to process query input in single-step mode:

```
me@yb100:~$ ybsql premdb -f total_goals_per_season_avg_per_match.sql -s
***(Single step mode: verify command)****************************************
select *, (total_goals/380.00)::dec(3,2) as goals_per_match
from (
select season_name, numteams,
sum(substr(ftscore,1,1)::int)+sum(substr(ftscore,3,1)::int) total_goals
from season, match
where season.seasonid=match.seasonid and season.seasonid>=4
group by season_name,numteams
) t1
order by 1;
***(press return to proceed or enter x and return to cancel)*******************
...
```

-t Option

Remove header and footer information from result sets with the `-t` option:

```
me@yb100:~$ ybsql premdb -f total_goals_per_season_avg_per_match.sql -t
 1995-1996   |       20 |         988 |           2.60
 1996-1997   |       20 |         970 |           2.55
 1997-1998   |       20 |        1019 |           2.68
 1998-1999   |       20 |         959 |           2.52
```

```
 1999-2000   |       20 |        1060 |           2.79
 2000-2001   |       20 |         992 |           2.61
 2001-2002   |       20 |        1001 |           2.63
 2002-2003   |       20 |        1000 |           2.63
 2003-2004   |       20 |        1012 |           2.66
 2004-2005   |       20 |         975 |           2.57
 2005-2006   |       20 |         944 |           2.48
 2006-2007   |       20 |         931 |           2.45
 2007-2008   |       20 |        1002 |           2.64
 2008-2009   |       20 |         942 |           2.48
 2009-2010   |       20 |        1053 |           2.77
 2010-2011   |       20 |        1063 |           2.80
 2011-2012   |       20 |        1066 |           2.81
 2012-2013   |       20 |        1063 |           2.80
 2013-2014   |       20 |        1052 |           2.77
```

--expanded Option

Use the `--expanded` table format for query output:

```
me@yb100:~$ ybsql premdb -f total_goals_per_season_avg_per_match.sql --expanded
-[ RECORD 1 ]---+----------
season_name     | 1995-1996
numteams        | 20
total_goals     | 988
goals_per_match | 2.60
-[ RECORD 2 ]---+----------
season_name     | 1996-1997
numteams        | 20
total_goals     | 970
goals_per_match | 2.55
-[ RECORD 3 ]---+----------
season_name     | 1997-1998
numteams        | 20
total_goals     | 1019
goals_per_match | 2.68
...
```

-z Option

Display output in unaligned format (`-A`), suppress header/footer information (`-t`), and use the `-z` option (zero-byte separator):

```
me@yb100:~$ ybsql premdb -f total_goals_per_season_avg_per_match.sql -A -z -t
1995-1996209882.60
1996-1997209702.55
1997-19982010192.68
1998-1999209592.52
1999-20002010602.79
2000-2001209922.61
...
```

-X and -a Options

Turn off the `.ybsqlrc` startup file with `-X` and use `-a` to echo all of the commands in a script as they run.

```
me@yb100:~$ ybsql yellowbrick -X -a -f /home/premdb/premdb.ddl
drop database premdb;
```

```
DROP DATABASE
create database premdb;
CREATE DATABASE
\c premdb
You are now connected to database "premdb" as user "premdb1".
drop table season;
ybsql:/home/premdb/premdb.ddl:7: ERROR:  table "season" does not exist
drop table team;
ybsql:/home/premdb/premdb.ddl:8: ERROR:  table "team" does not exist
drop table hometeam;
ybsql:/home/premdb/premdb.ddl:9: ERROR:  table "hometeam" does not exist
drop table awayteam;
ybsql:/home/premdb/premdb.ddl:10: ERROR:  table "awayteam" does not exist
drop table match;
ybsql:/home/premdb/premdb.ddl:11: ERROR:  table "match" does not exist
create table season(seasonid smallint, season_name character(9), numteams smallint, winners varchar(30)) distribute replicate;
CREATE TABLE
create table team(teamid smallint, htid smallint, atid smallint, name varchar(30), nickname varchar(20), city varchar(20), stadium
CREATE TABLE
create table hometeam(htid smallint, name varchar(30)) distribute replicate;
CREATE TABLE
create table awayteam(atid smallint, name varchar(30)) distribute replicate;
CREATE TABLE
create table match(seasonid smallint, matchday timestamp, htid smallint, atid smallint, ftscore char(3), htscore char(3)) distribu
CREATE TABLE
...
```

ybsql Slash (\) Commands

Within a `ybsql` session, you can use "backslash commands" (which begin with `\`) to access, format, and return data. Many of these commands have equivalent command-line options. See also ybsql Command-Line Options and ybsql Examples.

\a

Switch the output format between aligned and unaligned. If the format is unaligned, switch to aligned. If it is aligned, switch to unaligned.

\c, \connect [dbname [username] [host] [port]]

Connect to another database.

```
yellowbrick=> \c premdb ybsql10
Password for user ybsql10:
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)
You are now connected to database "premdb" as user "ybsql10".
```

\C title

Use the specified title for the results of queries. To unset a title, enter `\C` by itself. This command is equivalent to `\pset title title`.

\cd [directory]

Change the current working directory to the specified directory (without leaving the `ybsql` session). Entering `\cd` without a directory changes to the current user's home directory. To display your current working directory, enter `\! pwd`.

\conninfo

Return information about the current database connection.

\copy

See ybsql \copy Command.

\d [table | view | sequence]

List tables, views, and sequences. The `\d` output for a specific table or view includes attributes such as the distribution type, column constraints, sort or cluster columns, and partitioning columns.

For example:

```
premdb=# \d matchstats
           Table "public.matchstats"
  Column  |           Type           | Modifiers
----------+--------------------------+-----------
 matchid  | bigint                   | not null
 matchday | timestamp with time zone | not null
 htid     | smallint                 |
 atid     | smallint                 |
 moment   | character varying(5)     |
```

```
Distribution: Hash (matchid)
Clustered Columns: (htid, atid)
Columns:
    matchid UNIQUE
```

Use `\d+` to return more details about tables and other objects that have `\d` commands.

Note: The `\d` command does not return a persistent table if a temporary table with the same name exists. The command returns only the temporary table.

See also Patterns in \d Commands.

\ddp

List default access privileges that have been defined for objects owned by roles and users that belong to those roles. See ALTER DEFAULT PRIVILEGES. For each role, this command returns a string of one-letter abbreviations like this:

```
role=arwdD
```

The letters correspond to the following privileges:

- `a` : INSERT
- `c` : CONNECT
- `C` : CREATE
- `d` : DELETE
- `D` : TRUNCATE
- `r` : SELECT
- `T` : TEMPORARY
- `U` : USAGE
- `w` : UPDATE
- `x` : REFERENCES

For example:

```
yellowbrick=> \ddp
                              Default access privileges
             Owner             | Schema | Type  |           Access privileges
-------------------------------+--------+-------+---------------------------------------
 bobr                          | public | table | ybsql10=r/bobr
 ybdadmin@yellowbrickcloud.com | public | table | bobr=a/"ybdadmin@yellowbrickcloud.com"
(2 rows)
```

The `\ddp` command returns no rows if no default access privileges have been defined.

Access privileges are logged in sys.database.

\dfp [procedure]

List stored procedures. (Use `\dfp+` to return the source code for the procedure.)

\dg [role] (same as \du)

List roles (users).

\dn [schema]

List schemas.

\dp [table | view]

List privileges on tables, views, and sequences. See also `\ddp`.

\ds [sequence]

List sequences.

\dS [table | view]

List system tables and views.

\dt

List tables only.

\du [role]

List roles (users). (Same as `\dg`.)

\dv [view]

List views.

\dvS [view]

List system views.

\dwp [profile]

List WLM profiles.

\dwr [rule]

List WLM rules. For example:

```
premdb=# \dwr slowlane
                            List of WLM rules
   Name   | Order |  Type   | Enabled? | Superuser? |  Profile   | Version
----------+-------+---------+----------+------------+------------+---------
 slowlane |   100 | prepare | yes      | yes        | shortquery |
(1 row)
```

Use `\dwr+` to return the JavaScript definition of each rule.

\dwrp [pool]

List WLM resource pools.

\e, \edit [filename] [line_number]

Open the default editor (`vi` on Linux, `notepad.exe` on Windows). If you specify a file to edit, when the editor exits, its content is copied into the query buffer. If you do not specify a file, the current query buffer is copied to a temporary file that you can edit. The query buffer is parsed as a single line, so you should use `\i` if you want to edit a script. If the query in the buffer ends with, or contains, a semicolon, `ybsql` runs the query immediately. If not, it waits in the query buffer. You can type a semicolon or `\g` to run it, or `\r` to cancel.

If you specify a line number, `ybsql` places the cursor on the specified line of the file or query buffer. For example, go to line `8500` of the file `match.csv`:

```
premdb-# \e /home/premdb/match.csv 8500
...
```

\echo text [...]

Print the specified text. If you specify multiple text entries, they are separated in the output by one space, with a newline at the end. This command may be useful for including information within the output of a script. If you use the `\o` command to redirect query output, consider using `\qecho` instead of this command.

\encoding [encoding]

Return or set the client character set encoding. For example:

```
premdb=# \encoding latin9
premdb=# \encoding
LATIN9
```

\f [string]

Define the field separator for unaligned query output (by default: `|`). Use `\f` by itself to display the current separator.

\g [filename], \g [| command]

Execute the current query input buffer. Optionally, store the query's output in the specified file or pipe the output to the specified shell command. `ybsql` writes to the file or command only when the query returns zero or more result rows, and not if the query fails or is not a SQL command that returns data. Entering `\g` by itself is equivalent to entering a semicolon. Using the `\g` command with a file or command name is similar to using the `\o` command but for one operation only.

For example:

```
premdb=# select * from match
premdb-# \g | grep "| 8-"
        8 | 1999-09-19 00:00:00 |   28 |   85 | 8-0     | 4-0
       16 | 2008-05-11 00:00:00 |   27 |   74 | 8-1     | 2-0
       18 | 2010-05-09 00:00:00 |   14 |   95 | 8-0     | 2-0
       20 | 2011-08-28 00:00:00 |   26 |   51 | 8-2     | 3-1
       21 | 2012-12-23 00:00:00 |   14 |   52 | 8-0     | 3-0
```

\gset [prefix]

Send the query input buffer to the server and store the query output into `ybsql` variables. The query must return exactly one row. Each column of the row is stored into a separate variable, which has the same name as the column. For example:

```
premdb=# select htid as var1, atid as var2 from team limit 1
premdb-# \gset
premdb=# \echo :var1 :var2
2 51
```

If you enter a prefix, it is prepended to the column names to create the variable names:

```
premdb=# select htid as var1, atid as var2 from team limit 1
premdb-# \gset bobr
premdb=# \echo :bobrvar1 :bobrvar2
2 51
```

If a column returns `NULL`, the variable is unset. If the query fails or does not return one row, no variables are changed.

\h [command]

Return a list of supported SQL commands, or a short description of a specific SQL command and a summary of its syntax. For example:

```
yellowbrick=# \h commit
Command:     COMMIT
Description: Commit the current transaction block.
Syntax:
COMMIT [ WORK | TRANSACTION ]
```

\H, \html

Display output in HTML format. If the format is already HTML, this command returns it to the default format, which is aligned text.

\i filename, \include filename

Read and execute the contents of the specified file. If the specified file is `-` (a hyphen), `ybsql` reads from standard input until an `EOF` or a `\q` command is found. To display the lines in the file as they are read, set the `ECHO` variable to `all`.

\ir filename, include_relative filename

Read and execute the contents of the specified file. When `ybsql` is invoked from a script, interpret file names relative to the directory in which the script is located, rather than the current working directory. When `ybsql` is executing in interactive mode, `\ir` works exactly the same as `\i`.

\l, \list

List databases. Use `\l+` to return descriptive comments for each database.

See also Patterns in \d Commands.

\o filename, \o | command, \out filename, \out | command

Send query output (not including error messages) to a file or pipe the output into a shell command. For example:

```
premdb=# \o | grep 2010-03-06
premdb=# select * from match;
       18 | 2010-03-06 00:00:00 |    2 |   60 | 3-1     | 1-0
       18 | 2010-03-06 00:00:00 |   45 |   57 | 1-2     | 0-2
       18 | 2010-03-06 00:00:00 |   48 |   75 | 0-1     | 0-0
```

\p, \print

Print the current query buffer (if any) to standard output. For example:

```
premdb=# select * from match where seasonid=18 and htid=2 and atid=60;
       18 | 2010-03-06 00:00:00 |    2 |   60 | 3-1     | 1-0
premdb=# \p
select * from match where seasonid=18 and htid=2 and atid=60;
```

\password [user]

Change the named user's password (or the current user's password by default). See Connecting with a Secure Password.

\prompt [text] name

Prompt the user to enter text, which is assigned to the specified variable (`name`). Optionally, you can specify the prompt string (`text`). For example:

```
premdb=# \prompt YBUSER
Current ybsql user
premdb=# \echo :YBUSER
Current ybsql user
```

If the prompt string consists of multiple words, enclose it with single quotes.

\pset [option [value]]

Set options that affect the display of query results. For some options, if you do not specify a value, the option is toggled or unset. In other cases, the current setting is displayed. For details about each option, see ybsql Display Properties. Entering the `\pset` command without any arguments displays the current value of all display options.

\q, \quit

Exit the `ybsql` session.

\qecho text [...]

Print the specified text to the current output channel, as defined by the `\o` command.

\r, \reset

Clear the query buffer.

\s [filename]

Display the `ybsql` command history or save the history to a file.

\set [name [value [...]]]

Set a variable to a single value or to the concatenation of multiple values. For example:

```
premdb=# \set var1 2 3 4 5
premdb=# \echo :var1
2345
```

`\set` by itself returns a list of all current variable values. `\set` with a name but no value sets the variable to an empty value. Variable names are case-sensitive and may can contain letters, digits, and underscores. `ybsql` treats several variables with uppercase names as special; see ybsql Properties and Variables.

\setenv name [value]

Set an environment variable. For example:

```
\setenv YBUSER bobr
```

You can unset an environment variable by not entering a value. See ybsql Environment Variables.

\t

Do not print column names in query results or row counts at the end of the result set. If `\t` is already on, this command turns it off. This command is equivalent to `\pset tuples_only`.

\T table_options

Specify attributes to use within the table tag for HTML output (`-H`), such as `cellpadding` or `bgcolor`. This command is equivalent to `\pset tableattr table_options`.

\timing [on | off]

Display the duration, in milliseconds, of each SQL statement. If timing is already on, `\timing` turns it off.

\unset name

Unset (delete) the named `ybsql` variable. For example:

```
premdb=# \set YBPASSWORD bobr
premdb=# \echo :YBPASSWORD
bobr
premdb=# \unset YBPASSWORD
premdb=# \echo :YBPASSWORD
:YBPASSWORD
```

\w filename, \write filename, \w | command, \write | command

Write the current query buffer to the specified file or pipe it to the specified shell command.

\watch [seconds]

Execute the current query buffer every 2 seconds (by default) or at a specified interval. Execution repeats until it is interrupted or the query fails. For example:

```
premdb=# \watch 5
                    Watch every 5s Tue Mar 20 18:14:21 2018

 teamid | htid | atid |  name  | nickname |  city  | stadium | capacity | avg_att
--------+------+------+--------+----------+--------+---------+----------+---------
     48 |   49 |   98 | [NULL] | [NULL]   | [NULL] | [NULL]  |        0 |   0.000
     49 |   50 |   99 | [NULL] | [NULL]   | [NULL] | [NULL]  |        0 |   0.000
     50 |   51 |  100 | [NULL] | [NULL]   | [NULL] | [NULL]  |        0 |   0.000
(3 rows)

                    Watch every 5s Tue Mar 20 18:14:26 2018

 teamid | htid | atid |  name  | nickname |  city  | stadium | capacity | avg_att
--------+------+------+--------+----------+--------+---------+----------+---------
     48 |   49 |   98 | [NULL] | [NULL]   | [NULL] | [NULL]  |        0 |   0.000
     49 |   50 |   99 | [NULL] | [NULL]   | [NULL] | [NULL]  |        0 |   0.000
     50 |   51 |  100 | [NULL] | [NULL]   | [NULL] | [NULL]  |        0 |   0.000
(3 rows)
...
```

\x [on | off | auto]

Set or toggle expanded table formatting. This command is equivalent to `\pset expanded`.

\z

Alias for `\dp`.

\! [command]

Escape to a separate shell, or execute the specified shell command then return to the `ybsql` prompt.

\?

Shows help information for `ybsql` commands and options.

Patterns in \d Commands

The `ybsql \d` commands (`\d` , `\du` , `\dv` , and others) and `\l` accept patterns that filter the display of objects. By default, these commands display all objects that are visible to the current user in the current search path.

Note the following rules for using patterns:

- The characters in a pattern are folded to lowercase. For example, `\ dt TEAM` will display the table named `team` . To preserve any uppercase letters, use double quotes around all or part of a pattern. For example, `\dt new"MATCH"stats` will display the table named `newMATCHstats` .
- To use an actual double quotation mark character in a pattern, write it as a pair of double quotes within a double-quote sequence. For example, `\dt "MATCH""STATS"` will display the table named `MATCH"STATS` .
- `*` matches any sequence of characters (including no characters). For example, `\dt match*` displays tables with names that begin with `match` .
- `?` matches any single character.
- Within double quotes, `*` , `?` , and `.` are all matched literally.
- A dot (`.`) is interpreted as a separator for a schema name followed by an object name. For example, `\dt pub*.*team` displays tables that have names that end in `team` and schema names that start with `pub` .
- Advanced users can specify patterns with regular-expression notations such as character classes (for example `[0-9]` to match any digit).

ybsql Startup File

Startup files are useful for preserving settings and shortcuts that users want to enable every time they log in to `ybsql`. You can use the `YBSQLRC` environment variable to set the location of startup files for users.

When you start up `ybsql`, the startup file is read automatically. For example, here the `NULL` display and expanded display settings are both in effect:

```
$ ybsql premdb
Null display is "[NULL]".
Expanded display is used automatically.
ybsql (5.0.0)
Type:  \h for help with SQL commands
       \? for help with ybsql commands
       \g or terminate with semicolon to execute query
       \q to quit
premdb=#
```

To start a session without reading the startup file, use the `-X` option. For example:

```
$ ybsql premdb -X
ybsql (5.0.0)
Type:  \h for help with SQL commands
       \? for help with ybsql commands
       \g or terminate with semicolon to execute query
       \q to quit
premdb=#
```

.ybsqlrc File on Linux

You can create a `.ybsqlrc` startup file in the client user's home directory. For example, the file could have contents like this:

```
user1@user1:~$ more .ybsqlrc
-- By default, NULL displays as an empty space. Is it actually an empty
-- string, or is it null? This makes that distinction visible.
\pset null '[NULL]'
-- Use table format (with headers across the top) by default, but switch to
-- expanded table format when there's a lot of data, which makes it much
-- easier to read.
\x auto
...
```

---

ybsqlrc.conf File on Windows

On Windows clients, the `ybsqlrc.conf` file must be under the default PostgreSQL directory for the user:

```
%APPDATA%\postgresql
```

If this directory does not exist, users can create it as follows:

```
mkdir %APPDATA%\postgresql
```

This command creates a directory like this:

```
C:\Users\your_username\AppData\Roaming\postgresql\
```

The `psqlrc.conf` file is not recognized, even if it exists.

SQL Reference

Platforms: All platforms

Parent topic: Reference

In this section:

This section describes all of the SQL data types, commands, and functions that Yellowbrick supports. This section also provides a reference for the SQL client, `ybsql`.

SQL Commands

Platforms: All platforms

Parent topic: SQL Reference

In this section:

CREATE REMOTE SERVER
CREATE ROLE
CREATE SCHEMA
CREATE SEQUENCE
CREATE TABLE AS
CREATE USER
CREATE VIEW
CREATE WLM PROFILE
CREATE WLM RESOURCE POOL
CREATE WLM RULE
DEALLOCATE
DECLARE
DELETE
DESCRIBE
DROP BACKUP CHAIN
DROP CLUSTER
DROP DATABASE
DROP EXTERNAL AUTHENTICATION
DROP EXTERNAL FORMAT
DROP EXTERNAL LOCATION
DROP EXTERNAL MOUNT
DROP EXTERNAL STORAGE
DROP KEY
DROP OWNED BY
DROP PROCEDURE
DROP REMOTE SERVER
DROP ROLE
DROP SCHEMA
DROP SEQUENCE
DROP TABLE
DROP USER
DROP VIEW
DROP WLM PROFILE
DROP WLM RESOURCE POOL
DROP WLM RULE
END
EXECUTE
EXPLAIN
FETCH
HELP
IMPORT SSL TRUST
INSERT
LIST BUCKETS
LIST OBJECTS
LOAD TABLE
LOCK SEQUENCE
MOVE cursor
MOVE query
PREPARE
REASSIGN OWNED BY
RELEASE SAVEPOINT
RESET
RESTART query
REVOKE
REVOKE SSL TRUST
ROLLBACK
ROLLBACK DATABASE TO SNAPSHOT

Yellowbrick supports the SQL commands that are described in this section. For background information, see Database Administration.

CREATE EXTERNAL FORMAT

In this section:

Create an external file format that you can reference when you load a table from external storage. This format defines the type of source file and a set of load configuration options.

```
CREATE EXTERNAL FORMAT [ IF NOT EXISTS ] name
TYPE [mode] file_type
WITH (option value [,option value] ...)
```

IF NOT EXISTS

Create the object if it does not already exist. If it does exist, do not create it and do not return an error.

TYPE mode

Specify the usage of this format. This guarantees the options are valid at the time of performing the load / unload operation.

- `load` : to be used in LOAD TABLE commands
- `unload` : to be used in UNLOAD TABLE commands

The default mode is `load` if unspecified.

TYPE file_type

Specify the file type, which defines the formatting style of the data files. For flat files, the type defines how field delimiters are protected in the data.

- `CSV` : Delimiters in field values were protected by wrapping the field values in quotes. For example: `"2012, Mini Cooper S, ALL4"`
- `TEXT` : Delimiters in field values were protected by preceding them with a backslash escape character. For example: `2012\, Mini Cooper S\, ALL4`
- `BCP` : Delimiters in fields were not protected (for compatibility with the Microsoft SQL Server `bcp` tool). For example: `2012, Mini Cooper S, ALL4`
- `PARQUET` : a structured, columnar storage format (binary).

The default type is `CSV`.

WITH

Each name-value pair must consist of a valid load option and a valid setting for that option. For example:

```
with (skip_blank_lines true, delimiter '|')
```

Note that these options do not exactly match the `ybload` / `ybunload` client tool options, and not all of those options are supported. Do not use the `--` prefix for options in the `WITH` clause. See External Format Load Options and External Format Unload Options.

You must have the correct privileges to run this command. See ON EXTERNAL object.

Examples

Create an external format for CSV files with two load options:

```
premdb=> create external format premdbcsv
type csv with (delimiter ',', skip_blank_lines 'true');
CREATE EXTERNAL FORMAT
```

Create an external format for parquet data with two load options:

```
premdb=> create external format premdbparquet
type parquet
with (ignore_unsupported_schema 'true', serialize_nested_as_json 'true');
CREATE EXTERNAL FORMAT
```

Create an external format for CSV files with two unload options:

```
premdb=> create external format unloadcsv
type unload csv with (delimiter ',', linesep '\n');
CREATE EXTERNAL FORMAT
```

External Format Load Options

Platforms: EE: All cloud platforms, CE

Parent topic: CREATE EXTERNAL FORMAT

This section contains a list of the bulk load options that are supported in the `WITH` clause for the `CREATE EXTERNAL FORMAT` command:

See also LOAD TABLE Options and `WITH ADVANCED` in CREATE EXTERNAL STORAGE.

These options are a subset of the options supported by the `ybload` client tool and in general have the same function. See "ybload Field Options" in the main Yellowbrick documentation.

Note: Unlike their `ybload` client counterparts, these options do not have leading `--` characters, and they use underscores instead of hyphens (`-`) in their names. For example, `trim_white` is equivalent to `--trim-white`.

Conventions

Values for load options specified in SQL statements follow these conventions:

- Numeric values, including year values, require quotes. For example:

```
y2base '1990'
```

- Boolean values do not require quotes:

```
trim_white true
```

- Strings require quotes:

```
delimiter '|'
```

- JSON formatting: use single quotes (if embedded single quotes appear inside the JSON, escape them):

```
'{json text}'
```

CREATE EXTERNAL FORMAT Options

The `WITH` clause in the `CREATE EXTERNAL FORMAT` statement supports flat file record format options and field-parsing options. These options are listed alphabetically.

comment_char ASCII_CHARACTER

Define the comment character that is used in source files. The default value is the pound sign (#). The value must be a single ASCII character or a valid escape sequence. If `skip_comment_lines` is set, commented rows in the source file are *skipped*, not rejected. The comment character may be a single-byte or multi-byte character.

convert_ascii_control true | false

Allow the caret control ASCII character `^@` to be parsed as a single-byte representation of `null` within a character string. The default is `false`.

convert_c_escape true | false

Convert (or do not convert) C-style escape sequences when they appear in `CHAR` and `VARCHAR` fields. For example, the two-character sequence `\t` can be converted into a single tab character (`0x09`) or it can be loaded unchanged.

This option does not apply to fields with data types other than `CHAR` and `VARCHAR`. For example, in an `INTEGER` field, the value `4\x35`, where `\x3` is the C-style escape sequence for the number 5, returns an error: `'\' is invalid digit`. (However, the `ybsql \copy` command will convert C-style escape sequences in all fields.)

If `type text` is used, `convert_c_escape true` is the default behavior. If `type csv` or `type bcp` is used, `convert_c_escape false` is the default.

date_style YMD | DMY | MDY | MONDY | DMONY | Y2MD | DMY2 | MDY2 | MONDY2 | DMONY2

Define the date format in terms of the order and style of the date parts, to avoid any ambiguity in parsing dates. For example, `date_style 'MDY'` means accept date values such as `08-13-2016`.

If you specify one of the `Y2` values, such as `DMY2`, you must also specify a `y2base` value. For example:

```
y2base 1990, date_style 'DMY2'
```

You can only specify the `date_style` option once in a single command.

If `date_style` is not specified, the defaults derive from `date_field_options`.

default_field_options {JSON formatted <FieldOptions>}

Specify the field options to use for fields that do not have their own per-field options or per-type options. See the main Yellowbrick documentation for the `ybload` client.

delimiter SPECIAL_CHARACTER

Define the special character that the source file uses as its field delimiter. All of the following are supported:

- A single Unicode character (may be multi-byte)
- A hex value that corresponds to any ASCII control code (such as `0x1f`)
- A valid character escape sequence

If you do not specify a field delimiter, `ybload` auto-detects it from among the following characters:

- `,`
- `|`
- `\t`
- `\us`
- `\uFFFA`

emptymarker STRING_WITH_ESCAPES

Define a marker that matches the character used to represent an empty string in your source file (an empty `CHAR` or `VARCHAR` field). The value may be a string, a character, or a valid escape sequence.

For example: `emptymarker '\t'`.

Note: To use `"` as the `emptymarker` character, you have to quote strings with a different character (`"` is the default). For example:

```
emptymarker '\"', quote-char '|'
```

If null bytes exist within strings, use `on_zero_char remove`.

encoding ENCODING_NAME

Specify the source file's encoding (character set: `UTF8`, `LATIN9`, `ISO-8859-1`, `UTF16`). If no encoding is specified, the encoding of the source file is assumed to match the database encoding. If the encoding of the source file does not match the encoding of the destination database, `ybload` will use Java transcoding to "translate" the source data to the encoding of the destination database.

Check the documentation for the version of Java you are using on the `ybload` client system. For example, if you are using the Oracle JVM, check this list of supported character sets. Names from either of the first two columns can be used with the `--encoding` option.

For the fastest load performance under `ybload`, export the data from the source system in the character set of the destination database. For example, load exported `UTF8` data into a `UTF8` database. Any required transcoding is likely to have an impact on load performance.

Note: To load UTF-16 data, create a `UTF8` database and set the `encoding` option to `UTF16`.

escape_char SPECIAL_CHARACTER

Specify an escape character: any single Unicode character (may be multi-byte). The behavior of this option depends on the `TYPE`:

- `CSV`: used to escape embedded quote characters inside quoted fields. The default escape character is `"`.
- `TEXT`: used to escape embedded field delimiters and line separators. The default escape character is `\`.
- `BCP`: disallowed with `BCP` format, which does not use escape characters.

*_field_options

Options specific to individual data types. See "ybload Field Options" in the main Yellowbrick documentation for the `ybload` client.

ignore_emptymarker_case true | false

The first option supports case-insensitive empty-marker comparisons. If you use `--ignore-emptymarker-case`, values of `EMPTY`, `Empty`, and `empty` are all recognized as empty values when `--emptymarker` is set to `NONE`. These options apply globally; you cannot specify them per data type or per field. They are applied globally regardless of how the `--emptymarker` option was specified.

ignore_nullmarker_case true | false

Set to `true`, this option supports case-insensitive null-marker comparisons. Values of `NULL`, `Null`, and `null` are all recognized as null values when `nullmarker` is set to `NULL`. This option applies globally; you cannot specify it per data type or per field. It is applied globally regardless of the `nullmarker` setting.

linesep LINE_SEPARATOR

Define the line separator (or row separator) that is used in source files: any single Unicode character (may be multi-byte) or the `\r\n` escape sequence.

If you do not specify a line separator, `ybload` auto-detects it from among the following characters:

- `\n`
- `\r\n`
- `\rs`
- `\uFFFB`

locale LOCALE_NAME

The name of the locale to use for parsing dates, timestamps, and so on. If the locale is not specified, the database locale is assumed to be `C`.

Locale names must be of the following form:

```
<language code>[_<country code>[_<variant code>]]
```

For example:

```
locale en
locale en_US
locale zh_CN
```

Variant codes are rarely used; for details, see the Java documentation.

nullmarker STRING

Define a string that matches the string used to represent `null` in your source file. If this option is unspecified or set to an empty string, adjacent delimiters without text between them are parsed as `NULL` values. This option supports valid escape sequences.

You cannot load data in which more than one value in the same column is intended to be parsed as `NULL`. However, you can load data in which different columns have different values for `NULL`.

num_header_lines [NUMBER]

Ignore one or more header lines at the top of the source file. (The first or only header line in a file is typically a list of field names.) The default is `0`. The maximum number is `5`. If you specify multiple source files, the first line is skipped in all of them.

on_extra_field [remove | error]

Specify `remove` to allow rows to be loaded when the source file contains more fields than the table has columns. The removed fields must be at the end of the line in the source file. Specify `error` to reject rows with extra fields. `error` is the default.

on_invalid_char [replace | error]

Specify the action to take when the source file contains characters that cannot be represented in the database (or characters that are invalid in the source file itself):

- The replacement character for a LATIN9 database is the question mark: `0x3F` (`?`).
- The replacement character for a UTF8 database is the Unicode replacement character: `U+FFFD` (a question mark in a diamond).

The default is `replace`.

on_missing_field [supplynull | error]

Allow rows to be loaded when the source file contains fewer fields than the table has columns. Specify `supplynull` to load the missing columns at the end of the row with `null` values. Specify `error` to reject rows with missing fields. `error` is the default.

on_string_too_long truncate | error

Truncate character strings that are longer than the specified column width (or return an error). The default is `ERROR`.

on_unescaped_embedded_quote [preserve | error]

Preserve or return an error when unescaped quotes are found inside quoted strings. The default is `error`.

on_zero_char [remove | error]

Remove null bytes (`0x00` characters) that appear within strings in `CHAR` and `VARCHAR` fields (or return an error). The default is `ERROR`.

per_field_options {JSON Object}

Specify parsing options for individual fields. See "ybload Field Options" in the main Yellowbrick documentation for the `ybload` client.

quote_char SPECIAL_CHARACTER

Define the character that is used in source files to quote field values that contain embedded delimiters. Specify any single Unicode character (may be multi-byte). The default is `"`. This option only applies when you are loading from `TYPE CSV` source files.

skip_blank_lines true | false

Skip blank lines in the source file (`true`) or detect blank rows as bad rows (`false`). The default value is `true` .

skip_comment_lines true | false

Skip lines that are commented out in the source file or detect them as bad rows. The default value is `false` (do not skip).

Note: If you use `skip_comment_lines` , make sure the data does not contain any lines that begin with the comment character (which defaults to `#` but may bet set to a different character with the `comment_char` option).

trim_white true | false

Trim or retain leading and trailing whitespace characters in each field. With `type csv` , whitespace characters inside of quotes are always preserved. The default value is `false` (preserved).

TYPE

The file type must be specified in the `TYPE` clause of the CREATE EXTERNAL FORMAT statement, not inside the `WITH` clause.

y2base YEAR

Define the pivot year (such as `1970`) for two-digit year values (such as `97` and `16`). For example, if `--y2base` is set to `1970` , two-digit years of `70` and later are assumed to be in the 1900s. Values of `69` and earlier are assumed to be in the 2000s.

If you want to specify one of the `Y2` values for `date_style` , such as `DMY2` , you must also specify a `y2base` value.

External Format Unload Options

This section contains a list of the bulk unload options that are supported in the `WITH` clause for the `CREATE EXTERNAL FORMAT` command:

File Processing

compress NONE | GZIP_FAST | GZIP | GZIP_MORE | GZIP_BEST | GZIP_STREAM_FAST | GZIP_STREAM | GZIP_STREAM_MORE | GZIP_STREAM_BEST

The default is `NONE`. The `GZIP_` options write data as GZIP compressed files in two different compression modes: "block mode" and "stream mode." The `GZIP*` options run in block mode, and the `GZIP_STREAM*` run in stream mode. See ybunload Output Files.

`GZIP` and `GZIP_FAST` are synonyms. `GZIP_MORE` provides better compression, but slower unload performance, and `GZIP_BEST` provides the best compression but much slower performance.

`GZIP_STREAM` and `GZIP_STREAM_FAST` are synonyms. `GZIP_STREAM_MORE` provides better compression, but slower unload performance, and `GZIP_STREAM_BEST` provides the best compression but much slower performance. The `GZIP_STREAM_*` options are intended to be used *only* if your downstream workflow tools cannot handle `gzip` files containing multiple compression blocks. Additionally, the `GZIP_STREAM_*` options consume significantly more network connections than the `GZIP*` options, meaning many network routers won't be able to handle the increased number of connections reliably.

file_extension STRING

Specify an extension to append to each unload file name. The defaults are:

- CSV type: `csv`
- TEXT type: `txt`
- compressed file: `gz`

file_prefix STRING

Specify a prefix to prepend to each unload file name. The default is `unload`. For example, if you use the prefix `04-30-18`, the files are numbered consecutively in the following format:

```
04-30-18_1_1.csv
04-30-18_1_2.csv
04-30-18_1_3.csv
04-30-18_1_4.csv
04-30-18_1_5.csv
...
```

The convention for naming files is as follows:

```
<file_prefix>_<worker_id>_<partnumber>.<file_extension>
```

- `file_prefix`: As defined, or `unload` by default.

- `worker_id` : A logical number assigned to each compute node.
- `partnumber` : An incrementing number starting from `1` for each compute node output file.
- `file_extension` : File type, such as `.csv` or `.gz` .

max_file_size

Specify the maximum size of each output file. For example: `100MB` or `100Mib` . Do not use fractional values. The default maximum is `100Mib` . The minimum file size is `5Mib` .

parallel

Enable parallel processing on all compute nodes for the final sort operation that occurs when an unload query contains an `ORDER BY` clause. By default, the final `ORDER BY` sort runs on a single compute node. If the unload query does not have an explicit `ORDER BY` clause, this option has no effect.

When you use the `parallel` option, files are unloaded in streams from each compute node, and the rows are guaranteed to be sorted within each file and within each stream. However, the complete set of files from all the compute nodes will not be unloaded in order.

If you intend to reload data that was unloaded with the `--parallel` option, it is recommended that you create the target table with a `SORT ON` column. The presence of a `SORT ON` column causes `ybload` source files to be loaded in fully sorted order. If you need to stitch the unload files together for loading or for use in other applications, you may need to write a script that checks the first and last lines of each unloaded file.

CSV Format

delimiter CHARACTER

Define the field delimiter that will separate columns of data in output files. The delimiter cannot be the null byte (`0x00`).

The default is a tab character `\t` in `text` format, and a `\n` in `csv` format.

escape CHARACTER

Specify the character to use for escaping quotes.

The default is the same as `quote` in `csv` format, and not allowed in other format.

header

Include a header row with column names for each output file. Only applies to `csv` and `text` format.

The default is `false` .

linesep CHARACTER

Specify a row separator.

The default is `\n` for `csv` and `text` formats.

nullmarker STRING

String to use as a null marker in the unloaded files. For example:

```
nullmarker '[NULL]'
```

For `text` type, the default null marker is `\N` .

For other formats, no default null marker is used. In the output, `NULL` values will appear as adjacent delimiters with no text between them.

quote CHARACTER

Specify the character to use for quoting.

The default is `"` in CSV, and not allowed in other format.

CREATE EXTERNAL TABLE

Yellowbrick Documentation > Reference > SQL Reference > SQL Commands > CREATE EXTERNAL TABLE

Platforms: EE: All appliance platforms

Parent topic: SQL Commands

In this section:

External Table Examples

EXTERNAL TABLE USING Options

Create an NFS-mounted external table.

```
CREATE EXTERNAL TABLE 'reference_name/path_to_file'
USING (options) AS query
```

Important: Do not try to use this kind of external table for object storage. See the Yellowbrick Test Drive documentation for information about external loads from AWS S3.

Parameters

'reference_name/path_to_file'

Specify the name of a writable mount point defined with a CREATE EXTERNAL MOUNT command, followed by the location of a specific destination file on that mount point.

The destination file does not have to exist when the `CREATE EXTERNAL TABLE` command is run. If it does exist, by default its contents are truncated and overwritten with the results of the specified query. You can change this behavior by setting `notruncate true` in the `USING` clause.

USING

Specify the data format and other load or unload options. See USING Options.

query

Specify any valid Yellowbrick query (a SELECT statement). The column list for an external table is derived from this query.

See also External Tables.

External Table Examples

Platforms: EE: All appliance platforms

Parent topic: CREATE EXTERNAL TABLE

The following examples show to create and select from external tables. These examples assume that an external mount exists with the reference name `/ext_db/`

Create an external table:

Create an external table by writing (unloading) the results of a query to a file:

```
premdb=# create external table '/ext_db/ext_season.csv'
using(format text) as select * from season;
SELECT 25
```

Query the external table:

Subsequent queries can read from the external table file. Select data from this kind of table by using `FROM EXTERNAL` syntax:

```
premdb=# select * from external '/ext_db/season.csv'
(seasonid int, season_name char(9), numteams int, winners varchar(40))
using (format csv);
 seasonid | season_name | numteams |      winners
----------+-------------+----------+-------------------
       1 | 1992-1993   |       22 | Manchester United
       2 | 1993-1994   |       22 | Manchester United
       3 | 1994-1995   |       22 | Blackburn Rovers
       4 | 1995-1996   |       20 | Manchester United
       5 | 1996-1997   |       20 | Manchester United
...
```

Create an external table with additional options:

Create an external table with several options in the `USING` clause:

- Format the output as CSV.
- Compress the output as GZIP (using the `GZIP_BEST` unload compression type).
- Specify a writable log directory relative to the mount point. (This is useful for keeping track of log files that are tied to specific uses of external tables. For example, you can direct the logs for load and unload operations to different log directories.)
- Include a header line in the output file.
- Use a pipe character as the field delimiter in the file that contains the exported data.

```
premdb=# create external table '/ext_db/newmatchstats.gz'
using(format csv, compress 'GZIP_BEST', logdir '/ext_db/logs', includeheader true, delim '|')
as select * from newmatchstats;
SELECT 774540
```

You can check that the option settings were respected by unzipping and inspecting the `newmatchstats.gz` file.

Now select some rows from this external table by using `SELECT...FROM EXTERNAL` syntax. Note the use of the `skiprows` option to ignore the header line in the file, the `delim` option to make sure the columns are read correctly from the file, and the `logdir` option:

```
premdb=# select distinct seasonid, matchday
from external '/ext_db/newmatchstats'
  (seasonid int, matchday date)
  using(format csv, delim '|', skiprows 1, logdir '/ext_db/query_logs')
where seasonid=20
order by 2;
 seasonid |  matchday
----------+------------
       20 | 2011-08-13
       20 | 2011-08-14
       20 | 2011-08-15
       20 | 2011-08-20
       20 | 2011-08-21
       20 | 2011-08-22
       20 | 2011-08-27
       20 | 2011-08-28
...
```

Note: You must specify the correct number and order of columns in the column list when selecting from external tables. For example, the following query fails because it attempts to select three columns but only specifies two in the column list for the external table reference:

```
premdb=# select distinct seasonid, matchday, winners
from external '/ext_db/newmatchstats'
(seasonid int, matchday date)
using(format csv, skiprows 1)
where seasonid=20
order by 2;
ERROR:  column "winners" does not exist
LINE 1: select distinct seasonid, matchday, winners from external 'e...
                                            ^
```

Check the ybload and ybunload log files:

You can check the log files for `ybunload` and `ybload` operations that create and select from external tables.

These logs are useful for troubleshooting problems and as a record of when and how external table operations were performed. If you do not set the `logdir` option in the `USING` clause, the logs are written to the same directory as the external files. Log files are named `<queryid>-UNLOAD.log` or `<queryid>-LOAD.log`.

For example, the following `ybunload` log was written when an external table was created:

```
$ more 118173-UNLOAD.log
2019-06-05 13:58:49.484 [ INFO]
...
   app.name_and_version = "external version 3.0.0-13800"
   java.home            = "/usr/lib/jvm/java-8-oracle/jre"
   java.version         = "1.8.0_101"
   jvm.memory           = "875.00 MB (max=6.00 GB)"
   jvm.name_and_version = "Java HotSpot(TM) 64-Bit Server VM (build 25.101-b13, mixed mode)"
   jvm.options          = "-Xms512m, -Xmx6g, -XX:+UseG1GC, -Dlogfile=/home/brumsby/ybd/external/build/bin/external.log, -Dapp.name
app.pid=21892, -Dapp.repo=/home/brumsby/ybd/external/build/lib, -Dapp.home=/home/brumsby/ybd/external/build, -Dbasedir=/home/brums
l/build"
   jvm.vendor           = "Oracle Corporation"
   os.name_and_version  = "Linux 4.4.0-31-generic (amd64)"
```

```
2019-06-05 13:58:49.484 [ INFO] <unload-4>  Beginning unload to /home/brumsby/ext_db
2019-06-05 13:58:50.756 [ INFO] <pool-6-thread-1>  state: RUNNING - Network BW: 5.57 MB/s Disk BW: 1.99 MB/s
2019-06-05 13:58:51.147 [ INFO] <unload-4>  Network I/O Complete. Waiting on file I/O
2019-06-05 13:58:51.147 [ INFO] <unload-4>  Finalizing...
2019-06-05 13:58:51.151 [ INFO] <unload-4>  Transfer complete
2019-06-05 13:58:51.151 [ INFO] <unload-4>  Transferred: 2661174.00 B Avg Network BW: 5.89 MB/s Avg Disk write rate: 1.89 MB/s
```

Here is an example of a `ybload` log file, which was written when a user selected from an external table:

```
$ more 118288-LOAD.log
2019-06-05 13:59:10.456 [ INFO]
...
   app.name_and_version = "external version 3.0.0-13800"
   java.home            = "/usr/lib/jvm/java-8-oracle/jre"
   java.version         = "1.8.0_101"
   jvm.memory           = "875.00 MB (max=6.00 GB)"
   jvm.name_and_version = "Java HotSpot(TM) 64-Bit Server VM (build 25.101-b13, mixed mode)"
   jvm.options          = "-Xms512m, -Xmx6g, -XX:+UseG1GC, -Dlogfile=/home/brumsby/ybd/external/build/bin/external.log, -Dapp.name
app.pid=21892, -Dapp.repo=/home/brumsby/ybd/external/build/lib, -Dapp.home=/home/brumsby/ybd/external/build, -Dbasedir=/home/brums
l/build"
   jvm.vendor           = "Oracle Corporation"
   os.name_and_version  = "Linux 4.4.0-31-generic (amd64)"

2019-06-05 13:59:10.458 [ INFO] <load-5>  Assuming source encoding matches database server encoding: LATIN9
2019-06-05 13:59:10.459 [ INFO] <LFSourceContext>  Gathering metadata on input files
2019-06-05 13:59:10.460 [ INFO] <load-5>  --num-header-lines: 1
2019-06-05 13:59:10.464 [ INFO] <load-5>  Session Key = h7TVdFVK4c0fehPzOk8R4fQUPd9F16Obi1fgeaivU8vwwK61mWqYbByciMXNIpdH
2019-06-05 13:59:10.570 [ INFO] <load-5>  Starting 1 source PreReaders
2019-06-05 13:59:10.571 [ INFO] <load-5>  Using database locale: C
2019-06-05 13:59:10.572 [ INFO] <PreReader #0>  Auto-detected line separator = '\n'
2019-06-05 13:59:10.572 [ INFO] <LFScanContext>  Configuration (record/field separation):
   --format                          : CSV
   --delimiter                       : |
   --linesep                         : \n
   --quote-char                      : "
   --escape-char                     : "
   --trim-white                      : false
   --skip-blank-lines                : true
   --on-missing-field                : ERROR
   --on-extra-field                  : REMOVE
   --on-unescaped-embedded-quote     : ERROR
   Internal scanner                  : RecordScannerQuote_1_1
2019-06-05 13:59:10.573 [ INFO] <PreReader #0>  Reading from 75 for /home/brumsby/ext_db/newmatchstats
2019-06-05 13:59:10.596 [ INFO] <load-5>  Configuration (pre-parsing):
2019-06-05 13:59:10.597 [ INFO] <load-5>  Configuration (session):
   tableName       : "yellowbrick"."public"."external_table"
   keepAliveSeconds: 60
   maxBadRows      : 0
2019-06-05 13:59:10.598 [ INFO] <load-5>  Bad rows will be written to file:/home/brumsby/ext_db/query_logs/118288-LOAD.badrow
2019-06-05 13:59:10.598 [ INFO] <load-5>  Starting import of 1 files
2019-06-05 13:59:10.772 [ INFO] <load-5>  Starting 1 segment reader
2019-06-05 13:59:11.135 [ INFO] <load-5>  Flushing last 774540 rows (of 774540 total) for transaction 1
2019-06-05 13:59:11.152 [ INFO] <sender 1.00>  SUCCESSFUL BULK LOAD: Loaded 774540 good rows in   0:00:00 (READ: 27.64MB/s WRITE:
```

If an external table operation fails, you can use the log files to troubleshoot. For example, a `ybload` operation that generates bad rows will write a `badrow` file to the same log directory:

```
$ more 1882713-LOAD.badrow

#error: lineByteRange(75-99)
#reason: Too few fields (1 < 2)
1|1992-08-01|2|52|01:00
```

Join an external table to a regular database table:

The following example shows how to select from an external table (`newmatchstats.csv` aliased as `ex`) and join the results to a regular database table (`match` aliased as `db`):

```
premdb=# select distinct db.seasonid, db.matchday
from
  external '/ext_db/newmatchstats.csv' (seasonid int, matchday date)
  using(format csv, delim '|', skiprows 1, logdir '/ext_db/query_logs') ex
join
  match db on db.seasonid=ex.seasonid
where db.seasonid in(10,11,12)
order by 1,2;
 seasonid |      matchday
----------+---------------------
       10 | 2001-08-18 00:00:00
       10 | 2001-08-19 00:00:00
       10 | 2001-08-20 00:00:00
       10 | 2001-08-21 00:00:00
       10 | 2001-08-22 00:00:00
       10 | 2001-08-25 00:00:00
       10 | 2001-08-26 00:00:00
       10 | 2001-08-27 00:00:00
       10 | 2001-09-08 00:00:00
       10 | 2001-09-09 00:00:00
       10 | 2001-09-15 00:00:00
...
```

Join two external tables

This query joins two external tables (aliased `nms` and `esn`). Note that you must specify the `EXTERNAL` keyword, a column list, and a `USING` clause each time you reference an external table in a query:

```
premdb=# select distinct nms.seasonid, nms.matchday
from
external '/ext_db/newmatchstats' (seasonid int, matchday date) using(format csv, delim '|', skiprows 1, logdir '/ext_db/logs_may22
nms join
external '/ext_db/ext_season_new' (seasonid int) using(format csv, logdir '/ext_db/logs_may22')
esn on nms.seasonid=esn.seasonid
order by nms.seasonid;
 seasonid |  matchday
----------+------------
        1 | 1992-08-01
        2 | 1993-09-01
        2 | 1993-08-31
        2 | 1993-08-15
        2 | 1994-05-02
...
```

Insert rows into a table by selecting from an external table:

To read data from an external table and write it into a regular table, use an `INSERT INTO SELECT...FROM EXTERNAL` statement. You do not have to declare the column list for the file in this case; it is optional. The `USING` clause is required. For example:

```
premdb=# insert into season
select * from external '/ext_db/season.csv'
```

```
using(format csv);
INSERT 0 25
```

The external table in this statement uses the schema of the target table (`season`). The columns in the external data file must be in the same order as they are in the target table, and every column in the target table must also exist in the external file.

EXTERNAL TABLE USING Options

Yellowbrick Documentation > Reference > SQL Reference > SQL Commands > CREATE EXTERNAL TABLE > EXTERNAL TABLE USING Options

Platforms: EE: All appliance platforms

Parent topic: CREATE EXTERNAL TABLE

The following options are available in the `USING` clause when you run:

- `SELECT FROM EXTERNAL` queries
- `CREATE EXTERNAL TABLE` commands

Some of these options apply only to `SELECT FROM EXTERNAL` or `CREATE EXTERNAL TABLE` operations, as stated in the descriptions. See also External Table Examples.

Note: When you select from external tables, a background `ybload` operation is run to read the data. When you create external tables, a background `ybunload` operation is run to write the data. You can find more information about some of the `USING` options by consulting the equivalent ybload `and` ybunload sections.

- Options are listed in alphabetical order for quick reference.
- Option names are shown in lowercase, but they are not case-sensitive.
- Variables for option values, such as `CHARACTER`, are shown in uppercase.
- Option values that are keywords (such as `true` and `false`) are shown in lowercase. These keywords are not case-sensitive.
- Option values that are string literals (such as a path to a directory in `logdir PATH`) are case-sensitive and should be protected with quotes when specified. For example: `logdir '/ext_db/QueryLogs'`
- Quoted option values follow standard SQL quoting rules.

Option name	Accepted values	Default value	Data type	Description
`boolstyle`	`1_0, T_F, Y_N, YES_NO, TRUE_FALSE`	`1_0`	String	Specify which pair of values to use for Boolean data types in unloaded data: `1` or `0`, `T` or `F`, `Y` or `N`, `YES` or `NO`, `TRUE` or `FALSE`. This option is ignored when you select from external tables; `ybload` automatically supports all of these styles.
`compress`	`true, false, TYPE`	Auto-detected (based on file suffix)	Boolean	Define the compression type for input and output data. - If this option is specified without a value or with `true`, `GZIP` compression is applied for `SELECT` operations. - If this option is not included or is specified with `false`, compression is auto-detected based on the file suffix for `SELECT` operations. - You can explicitly specify the compression type, but the supported type values differ for `ybload` and `ybunload`; see the `ybload [--source-compression](#source-compression)` and `ybunload [--compress](#compress)` options.
`datedelim`	`CHARACTER`	Auto-detected	String	Specify the delimiter for datetime values. Equivalent to `delim` in `ybload --date-field-options`.
`datestyle`	`FORMAT`	`YMD`	String	Specify the format for datetime values. See ybload Date Formats. Any of the supported date styles, such as `'YMD'` or `'DMY'`, are accepted. Only one date style can be specified in the `USING` clause.

Option name	Accepted values	Default value	Data type	Description
`delim` or `delimiter`	`SPECIAL_CHARACTER`	Auto-detected	String	Specify the field delimiter (a column separator, equivalent to the --delimiter option in `ybload`). See also Character Escape Sequences.
`encoding`	`ENCODING_NAME`	Server encoding	String	Equivalent to `ybload [--encoding](#encoding)`. This option does not apply to `CREATE EXTERNAL TABLE` operations.
`escapechar`	`SPECIAL_CHARACTER`	Depends on the `format` option	String	Use this character as the escape character. Equivalent to the --escape-char option in `ybload`.
`fillrecord`	`true`, `false`	`false` (`ERROR`)	Boolean	Define the behavior when fields are missing from source files: supply a `null` value or return an error. `fillrecord true` is equivalent to `ybload [--on-missing-field](#onmissingfield) SUPPLYNULL`. This option does not apply to `CREATE EXTERNAL TABLE` operations.
`format`	`csv`, `text`, `bcp`	`csv`	String	Define the format of the data being read or written: `CSV`, `TEXT`, or `BCP` (see the `ybload [--format](#format)` option for details). If this option is not specified, the default is `CSV`. If this option is specified without a value or with an empty string, the command returns an error. This option is required for `CREATE EXTERNAL TABLE` operations.
`ignorezero`	`true`, `false`	`false` (`ERROR`)	Boolean	Remove null bytes (`0x00` characters) or return an error when they appear within strings in `CHAR` and `VARCHAR` fields. `ignorezero true` is equivalent to `ybload [--on-zero-char](#onzerochar) REMOVE`. This option does not apply to `CREATE EXTERNAL TABLE` operations.
`includeheader`	`true`, `false`	`false`	Boolean	`TRUE`: Write a single header line at the beginning of each output file. `FALSE`: Do not include a header line (the default behavior). This option applies to `CREATE EXTERNAL TABLE` operations only.
`logdir`	`PATH`	Same location as external files	String	Set a path to a writable log directory, relative to an active NFS mount point (see CREATE EXTERNAL MOUNT). For example: `logdir '/qumulo/external_tables/select_from_external_logs/'` If this option is not specified, log files and any bad row files are written to the same directory as the data file. You cannot use a path to a read-only NFS mount for the log directory.
`maxerrors`	`NUMBER`	0	Integer	Set the maximum number of bad rows to tolerate. Equivalent to `ybload [--max-bad-rows](#maxbadrows) NUMBER`. **Note:** `maxerrors 32` means 32 bad rows are allowed; the load will fail on the 33rd bad row. This option does not apply to `CREATE EXTERNAL TABLE` operations.
`notruncate`	`true`, `false`	`false`	Boolean	Preserve or overwrite external table files. `TRUE`: if an external table file exists with the specified name, do not overwrite the file; return an error. `FALSE` is the default, which allows existing files to be overwritten. This option applies to `CREATE EXTERNAL TABLE` operations only.
`nullvalue`	`STRING`	`NULL`	String	Specify the string that will represent `NULL` in the result set (equivalent to the --nullmarker option).

Option name	Accepted values	Default value	Data type	Description
`quotedvalue`	`yes`, `no`, `single`, `double`, `SPECIAL_CHARACTER`	"	String	Use this character as the quote character when the format is `CSV`. Equivalent to the --quote-char `and` --escape-char options in `ybload`. **Note:** The values `yes` and `single` specify a single quotation mark (`'`). The values `no` and `double` specify a double quotation mark (`"`), which is the same as the default behavior when this option is not specified.
`recorddelim`	`SPECIAL_CHARACTER`	Auto-detected	String	Define the record delimiter (or row separator) that is used in external source files. Equivalent to `ybload [--linesep](#linesep)`. This option does not apply to `CREATE EXTERNAL TABLE` operations.
`skiprows`	`NUMBER`	0	Integer	Skip the specified number of rows when reading data in from the external file. This option is useful when the file contains one or more header lines. This option does not apply to `CREATE EXTERNAL TABLE` operations.
`timedelim`	`CHARACTER`	:	String	Specify the delimiter character for the time components of a datetime value. The default is the colon character (`:`). This option must be specified in combination with `timestyle`. Equivalent to `delim` in `ybload --time-field-options` and `--timestamp-field-options`. (See ybload Field Options.)
`timestyle`	`'24HOUR'`, `'12HOUR'`	`'24HOUR'`	String	Specify the time format used in the data file. The default is `'24HOUR'`. This option is ignored when you select from external tables; `ybload` automatically supports both formats.
`truncstring`	`true`, `false`	`false`	Boolean	Truncate strings that are longer than the specified column width or return an error. Equivalent to `ybload [--on-string-too-long](#onstringtoo)`. `true` is equivalent to `truncate`; `false` is equivalent to `error`. This option does not apply to `CREATE EXTERNAL TABLE` operations.
`y2base`	`YEAR` (`0-9999`)	No default	integer	Equivalent to the `ybload [--y2base](#y2base)` option. This option does not apply to `CREATE EXTERNAL TABLE` operations.

CREATE TABLE

In this section:

Create a new, empty table in a database.

The table is owned by the user who runs the `CREATE TABLE` command. If you specify a schema name (`schema1.table1`), the table is created in that schema; otherwise, it belongs to the current schema. See the parameter descriptions for more details.

The optional `CONSTRAINT` clauses specify constraints that new or updated rows must satisfy. For example, if a column is declared `NOT NULL`, an attempt to insert a `NULL` value into that column will fail. A constraint defines the set of valid values that one or more columns may contain. You can define constraints on tables or columns. A column constraint is part of a single column definition, but a table constraint may apply to more than one column. See the parameter descriptions for more details.

Syntax

```
CREATE [ { TEMPORARY | TEMP } ] TABLE [ IF NOT EXISTS ] table_name
( [
  { column_name data_type
   [ column_constraint [ ... ] ]
   | table_constraint }
  [, ... ]
] )
[ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
[ DISTRIBUTE { ON (column) | [ ON ] REPLICATE | [ ON ] RANDOM } ]
{ [ SORT ON (column) ] |
[ CLUSTER ON (column [, ... ] }
[ PARTITION BY ( { range_partition_spec | hash_partition_spec } [ , ... ] )

where column_constraint is:

[ CONSTRAINT constraint_name ]
{ NOT NULL |
  NULL |
  DEFAULT expression |
  UNIQUE |
  PRIMARY KEY |
  REFERENCES reftable [ ( refcolumn ) |
  ENCRYPTED WITH (COLUMN_ENCRYPTION_KEY=key, ENCRYPTION_TYPE={ DETERMINISTIC | RANDOMIZED }, ALGORITHM='algorithm') }

and table_constraint is:

[ CONSTRAINT constraint_name ]
{ UNIQUE ( column_name [, ... ] ) |
  PRIMARY KEY ( column_name [, ... ] ) |
  FOREIGN KEY ( column_name [, ... ] ) REFERENCES reftable [ ( refcolumn [, ... ] ) ] }

and range_partition_spec is:

RANGE ( column BETWEEN value AND value EACH value [, OUTSIDE RANGE] [, IS NULL] )
```

```
and hash_partition_spec is:

HASH ( column WITH number PARTITIONS [, IS NULL] )
```

Parameters

TEMPORARY | TEMP

Create a "local" temporary table that is automatically dropped at the end of the session. Temporary tables belong to a temporary schema. You cannot qualify a temporary table name with a user-defined schema name.

Temporary tables are visible only to the user who created the table and within the session where the table was created. You cannot create "global" temporary tables that are visible across sessions and users.

Temporary tables that the system creates during queries are limited to a size of 30MB. This limitation does not apply to temporary tables created by users.

IF NOT EXISTS

Create the table if it does not already exist. If it does exist, do not create it and do not return an error.

table_name

Give the table a name that is distinct from the name of any other table, sequence, or view in the same schema. The maximum length of a table name is 128 bytes; longer names are automatically truncated. See SQL Identifiers.

Optionally, qualify the name of a persistent table with the schema name. You can also qualify table names with the name of the current database, but you cannot create a table in a different database; the table must belong to the current database.

You can create temporary and persistent tables with the same name. However, a persistent table with the same name as an existing temporary table is only visible to the current session if you reference it by using its schema-qualified name.

In this example, the `books` table belongs to the `public` schema and the current `premdb` database:

```
premdb=# create table premdb.public.books(isbn varchar(20));
CREATE TABLE
```

column_name

Name each column uniquely. See SQL Identifiers.

Tables have a limit of 2,000 user-defined columns.

data_type

Define the data type for each column. See SQL Data Types.

column_constraint

Define constraints for a specific column and optionally provide a constraint name. Column constraints are a convenient way of defining single-column attributes.

- `NOT NULL`: do not allow null values in the column.
- `NULL`: allow null values in the column (the default behavior).
- `DEFAULT`: use an expression to set a default value for the column. Make sure the expression is compatible with the data type of the column.

Note: If you use ALTER TABLE to add a column with a `DEFAULT` constraint, the default value is applied only for new rows that are inserted after the column was added. Existing rows in the table will contain a `NULL` value for that column.

- `UNIQUE`: declare that all values in the specified column should be distinct.
- `PRIMARY KEY`: declare the column as the primary key of the table. A `PRIMARY KEY` constraint implies that the column is both `NOT NULL` and `UNIQUE`. For example:

```
seasonid smallint primary key
```

- `REFERENCES`: declare the column as a foreign key by naming the referenced table and column, which must be `UNIQUE`. For example:

```
playerid bigint references player(playerid)
```

`FOREIGN KEY` constraints are also created implicitly when a `PRIMARY KEY` constraint references a column in another table. The referenced column must be `UNIQUE`. For example:

```
seasonid smallint primary key references season(seasonid)
```

If you want to define a `PRIMARY KEY`, `FOREIGN KEY`, or `UNIQUE` constraint that involves multiple columns, use a table constraint.

Important: Although `PRIMARY KEY`, `FOREIGN KEY`, and `UNIQUE` constraints may be declared, they are not enforced.

If defined, these constraints may be used as hints during query planning, so you should declare them if an ETL process or some other process in your application enforces their integrity. For example, the query planner sometimes uses primary and foreign keys in numeric computations, subquery decorrelation, join ordering, and join elimination. The planner assumes that all keys in Yellowbrick tables are valid as loaded. If your application allows invalid keys, queries may return wrong results. As a rule, do not define key constraints for your tables if you doubt their validity, but always declare them when you know that they are valid.

`NOT NULL` column constraints *are* enforced.

If necessary, you can drop constraints by using ALTER TABLE commands.

table_constraint

Define constraints that apply to one or more columns in the table.

Note: `PRIMARY KEY`, `FOREIGN KEY`, and `UNIQUE` constraints may be declared, but they are not enforced. See also column_constraint.

- `UNIQUE`: declare that all values in the specified columns should be distinct.
- `PRIMARY KEY`: declare a set of columns as the primary key of the table; implies that these columns are both `NOT NULL` and `UNIQUE`.
- `FOREIGN KEY`: declare columns as foreign keys by naming the referenced table and columns. If no `refcolumn` list is specified, the primary key of the `reftable` is used.

ENCRYPTED WITH

Encrypt a `VARCHAR` column with an encryption key that is created with a CREATE KEY statement. (The key does not have to exist before the CREATE TABLE statement is run; it can be created later.) Data inserted or bulk-loaded into an encrypted column will be protected, using the ENCRYPT_KS function. See also Encrypting Sensitive Data.

You must specify an encryption key, type, and algorithm in the `WITH` clause. These parameters do not have defaults:

```
ENCRYPTED WITH (COLUMN_ENCRYPTION_KEY=key_name, ENCRYPTION_TYPE={ DETERMINISTIC | RANDOMIZED }, ALGORITHM='algorithm')
```

`DETERMINISTIC`: guaranteed to encrypt the same string with the same binary result every time, which may offer less protection, especially for low-cardinality columns. This option may be a good choice for high-cardinality columns.

`RANDOMIZED`: binary results for the same string are randomized, providing more protection. This option is safer for low-cardinality columns.

`ALGORITHM`: All of the encryption algorithms use Output Feedback Mode (OFB). Valid values are `AES_128_OFB`, `AES_192_OFB`, and `AES_256_OFB`.

The following combinations of encryption types and algorithms are supported. (The third column in the table shows the equivalent numeric parameter that you would use to emulate this combination when calling the `ENCRYPT_KS` function explicitly. Parameters 1-3 for `ENCRYPT_KS` are not supported in CREATE TABLE.) For details about how the algorithms work, see Encryption and Decryption Algorithms.

Note: You cannot create keys in one database and reference them in a table that you create in another database.

Encryption Type	Algorithm	ENCRYPT_KS Function Parameter
`DETERMINISTIC`	`AES_128_OFB`	`4`
`DETERMINISTIC`	`AES_192_OFB`	`5`
`DETERMINISTIC`	`AES_256_OFB`	`6`
`RANDOMIZED`	`AES_128_OFB`	`7`
`RANDOMIZED`	`AES_192_OFB`	`8`
`RANDOMIZED`	`AES_256_OFB`	`9`

Note: You cannot modify the definition of encrypted columns, but you can add encrypted columns to a table, using ALTER TABLE ADD COLUMN.

Encrypted columns do not support certain CREATE TABLE options, as shown in the following table:

Option	DETERMINISTIC	RANDOMIZED
Hash distribution (`DISTRIBUTE ON column`)	Supported	Not supported
Hash partitioning (`HASH column`)	Supported	Not supported
Range partitioning (`RANGE column`)	Not supported	Not supported
`SORT ON column`	Not supported	Not supported
`CLUSTER ON column`	Not supported	Not supported

ON COMMIT

Set the behavior for temporary tables when a transaction commits.

- `PRESERVE ROWS`: do not delete rows from temporary tables when transactions commit.
- `DELETE ROWS`: delete rows from temporary tables when transactions commit.
- `DROP`: drop temporary tables when transactions commit.

DISTRIBUTE

Define the data distribution scheme for the table (how the rows are distributed across the cluster). See Distribution Options.

Note: You cannot distribute a table on a floating-point column (`FLOAT`, `FLOAT4`, `FLOAT8`).

SORT ON

Define a sort column for the table (how the data is sorted when it is loaded). The `SORT ON` and `CLUSTER ON` options are mutually exclusive. See Sorted and Clustered Tables. Tables may be partitioned and sorted on the same column.

CLUSTER ON

Define up to four columns as cluster columns. The `SORT ON` and `CLUSTER ON` options are mutually exclusive. Tables may be partitioned and clustered on the same column. See Sorted and Clustered Tables.

Note: REAL and DOUBLE PRECISION columns cannot be defined as cluster columns.

PARTITION BY

Partition the table by one or more columns or expressions. See Partitioning Options. Tables may be partitioned and sorted (or clustered) on the same column. You cannot partition replicated tables.

Note: If used, the following clauses at the end of the `CREATE TABLE` statement must appear in the order shown in the syntax diagram:

1. `ON COMMIT`
2. `DISTRIBUTE`
3. `SORT ON` or `CLUSTER ON`
4. `PARTITION BY`

CREATE TABLE Examples

The following examples show different options for CREATE TABLE statements.

Table with replicated distribution and a primary key

Create a replicated table with a column defined as the primary key.

```
premdb=# create table season
(seasonid smallint primary key, season_name character(9), numteams smallint, winners varchar(30))
distribute replicate;
CREATE TABLE
premdb=# \d season
             Table "brumsby.season"
   Column    |         Type          | Modifiers
-------------+-----------------------+-----------
 seasonid    | smallint              | not null
 season_name | character(9)          |
 numteams    | smallint              |
 winners     | character varying(30) |

Distribution: Replicated
```

Table with distribution and sort keys

Create a table with the same column defined for data distribution and sorting:

```
premdb=# create table match
(seasonid smallint, matchday timestamp, htid smallint, atid smallint, ftscore char(3), htscore char(3))
distribute on(seasonid) sort on(seasonid);
CREATE TABLE
premdb=# \d match
              Table "brumsby.match"
  Column  |            Type             | Modifiers
----------+-----------------------------+-----------
 seasonid | smallint                    |
 matchday | timestamp without time zone |
 htid     | smallint                    |
 atid     | smallint                    |
 ftscore  | character(3)                |
 htscore  | character(3)                |

Distribution: Hash (seasonid)
Sort Column: (seasonid)
```

Temporary table with ON COMMIT DELETE ROWS

Create a temporary table that is truncated when a transaction commits:

```
premdb=# create temp table team
(teamid int, team_name varchar(30))
on commit delete rows;
CREATE TABLE
premdb=# \d team
          Table "pg_temp_5.team"
  Column   |         Type          | Modifiers
-----------+-----------------------+-----------
 teamid    | integer               |
 team_name | character varying(30) |

Distribution: Hash (teamid)
```

Table that uses a sequence to generate column values

Create a table with a column named `MATCHKEY` that uses the `NEXTVAL` function as its default value.

```
premdb=# create table matchstats(matchkey bigint default nextval('matchid'),
seasonid smallint, matchday date, htid smallint, atid smallint, monent varchar(5));
CREATE TABLE
premdb=# \d matchstats
                    Table "public.matchstats"
  Column  |         Type         |              Modifiers
----------+----------------------+-------------------------------------
 matchkey | bigint               | default nextval('matchid'::regclass)
 seasonid | smallint             |
 matchday | date                 |
 htid     | smallint             |
 atid     | smallint             |
 monent   | character varying(5) |

Distribution: Hash (matchkey)
```

Table with primary and foreign keys

Create a table with a primary key that references another table. (This example requires that the referenced column `season.seasonid` is declared as `UNIQUE`.) The primary key declaration results in an implicit foreign-key constraint.

```
premdb=# create table match
(seasonid smallint primary key references season(seasonid),
matchday timestamp, htid smallint, atid smallint, ftscore char(3), htscore char(3))
distribute on(seasonid)
sort on(seasonid);
CREATE TABLE
premdb=# \d match
              Table "public.match"
  Column  |            Type             | Modifiers
----------+-----------------------------+-----------
 seasonid | smallint                    | not null
 matchday | timestamp without time zone |
 htid     | smallint                    |
 atid     | smallint                    |
 ftscore  | character(3)                |
 htscore  | character(3)                |

Distribution: Hash (seasonid)
Sort Column: (seasonid)
Columns:
   seasonid PRIMARY KEY
```

```
Foreign-key constraints:
   "match_seasonid_fkey" FOREIGN KEY (seasonid) REFERENCES season(seasonid)
```

You can drop the foreign-key constraint if you want to break the relationship between the two tables. For example:

```
premdb=# alter table match drop constraint match_seasonid_fkey;
ALTER TABLE
```

Table with a two-column primary key and an explicit foreign-key constraint

Create a table with a primary key that spans two columns, and an explicit foreign-key reference to another table. (This example requires that the referenced column `season.seasonid` is declared as `UNIQUE`.)

```
premdb=# create table player(playerid bigint, teamid smallint, seasonid smallint, firstname varchar(30), lastname varchar(30), pos
primary key (playerid, teamid),
foreign key (seasonid) references season(seasonid));
CREATE TABLE
premdb=# \d player
            Table "public.player"
  Column   |         Type          | Modifiers
-----------+-----------------------+-----------
 playerid  | bigint                | not null
 teamid    | smallint              | not null
 seasonid  | smallint              |
 firstname | character varying(30) |
 lastname  | character varying(30) |
 position  | character(1)          |
 dob       | date                  |
 cob       | character varying(30) |

Distribution: Hash (playerid)
Columns:
   playerid, teamid PRIMARY KEY
Foreign-key constraints:
   "player_seasonid_fkey" FOREIGN KEY (seasonid) REFERENCES season(seasonid)
Primary-key constraints:
   "player_pkey" PRIMARY KEY (playerid, teamid)
```

Table with encrypted columns

Create a table with three encrypted columns. When the table is loaded, the source strings in these columns will be encrypted.

```
premdb=# create table player(
playerid bigint not null,
teamid smallint not null,
seasonid smallint not null,
firstname varchar(30),
lastname varchar(30),
position char(1),
dob varchar(100) encrypted with(COLUMN_ENCRYPTION_KEY=playerkey,ENCRYPTION_TYPE=RANDOMIZED,ALGORITHM = 'AES_256_OFB'),
weekly_wages varchar(100) encrypted with(COLUMN_ENCRYPTION_KEY=playerkey,ENCRYPTION_TYPE=DETERMINISTIC,ALGORITHM = 'AES_128_OFB'),
avg_mins_per_match double precision,
matches_played real,
cob varchar(100) encrypted with(COLUMN_ENCRYPTION_KEY=playerkey,ENCRYPTION_TYPE=RANDOMIZED,ALGORITHM = 'AES_192_OFB'));
CREATE TABLE
```

Load some rows into this table, then grant user `yb100` `SELECT` on the `player` table, but do not grant `DECRYPT` on the key `playerkey`. When `yb100` selects from `player`, the encrypted values are returned.

```
premdb=# create user yb100 password 'yb100';
CREATE ROLE
premdb=# grant select on player to yb100;
GRANT
premdb=# \c premdb yb100
You are now connected to database "premdb" as user "yb100".
premdb=> select * from player;
-[ RECORD 1 ]------+------------------------------------------------------------
playerid           | 2
teamid             | 41
seasonid           | 27
firstname          | Harry
lastname           | Kane
position           | F
dob                | CTc5ZT+iDT7Z80aYy5OTtV1Qk2k0zFci7MgDWUAJzTrMnq0a5iLVm5uEcA==
weekly_wages       | BP9GFdcy5ND0OKXgLeHiMlfMMd0j06Y=
avg_mins_per_match | 84.1567
matches_played     | 36.1
cob                | CDv/nAiOq+vvQzLMRvJQmfD501x6DLXsI6unLmFZ/Ds=
-[ RECORD 2 ]------+------------------------------------------------------------
playerid           | 3
teamid             | 41
seasonid           | 27
firstname          | Harry
lastname           | Winks
position           | M
dob                | CfvZTq+DCl/ahcZW0gpXL2puITwPhiBbX/0WByBvVVxP/MpSTlalxX97Zg==
weekly_wages       | BLwKffvCFHQhuneexpn6pG+1NdtdLA==
avg_mins_per_match | 72.3412
matches_played     | 31.2
cob                | CDq0xiP8GZkYwCYPwMJULx/5EmXVfy5INpB/H+J/quk=
...
```

Now grant `DECRYPT` to `yb100`. When `yb100` queries the table again, the decrypted values are returned.

```
premdb=# grant decrypt on key playerkey to yb100;
GRANT
premdb=# \c premdb yb100
You are now connected to database "premdb" as user "yb100".
premdb=> select * from player;
 playerid | teamid | seasonid | firstname | lastname  | position |    dob     | weekly_wages | avg_mins_per_match | matches_played
----------+--------+----------+-----------+-----------+----------+------------+--------------+--------------------+---------------
        2 |     41 |       27 | Harry     | Kane      | F        | 07-28-1993 | 250000       |            84.1567 |           36.1
        3 |     41 |       27 | Harry     | Winks     | M        | 02-02-1996 | 40000        |            72.3412 |           31.2
...
```

If you want users to see plain text for some encrypted columns but not others, use different keys in the `CREATE TABLE` statement, then grant privileges on the keys accordingly.

Two tables joined on encrypted columns

Regardless of permissions on encryption keys, you can use encrypted columns in equality joins. In this example, a user without `DECRYPT` privileges runs a query that joins on the `weekly_wages` column, which is encrypted with the same attributes in both tables:

```
premdb=> select p.firstname, p.lastname, p.weekly_wages
from player p, newplayer np where p.weekly_wages=np.weekly_wages and p.playerid=np.playerid;
 firstname | lastname  |           weekly_wages
-----------+-----------+----------------------------------
```

```
  Paul     | Pogba     | BD3kn8ThZr6fFfyR09jKDrHswd7HQZA=
  Alexis   | Sanchez   | BGLAs08top1lIoTRupU3hXTtCgi4r/g=
  Mesut    | Ozil      | BMyXnoYtbTT/+7XBZXGmAAwyNBkoatA=
  Romelu   | Lukaku    | BPGCDY4vTeIcj01zghWRYsIcnneUC2c=
  David    | Silva     | BPGCDY4vTeIcj01zghWRYsIcnneUC2c=
  Jamie    | Vardy     | BDgVHSxmbPe4/OOU+4Q7bfiobM0mVw==
  Marcus   | Rashford  | BPGCDY4vTeIcj01zghWRYsIcnneUC2c=
  Harry    | Kane      | BPGCDY4vTeIcj01zghWRYsIcnneUC2c=
  Harry    | Winks     | BOAXiWS24OEegpbVtFHhsoamN3fsoQ==
  Kevin    | De Bruyne | BB4wPtcyHyViIsXZKLHFCWYn6VVJM0w=
  Gonzalo  | Higuain   | BAhlSXor3krmxTdy9T1DpF9Ds3bw2Po=
  Eden     | Hazard    | BFUsmeTmHAClz25OLCxle7Zjz6PA3Lk=
  Sergio   | Aguero    | BBZMIb5nJ15KinMuUpRstRtpwrng4wA=
  David    | de Gea    | BFUsmeTmHAClz25OLCxle7Zjz6PA3Lk=
  Mo       | Salah     | BFUsmeTmHAClz25OLCxle7Zjz6PA3Lk=
  Riyad    | Mahrez    | BFUsmeTmHAClz25OLCxle7Zjz6PA3Lk=
  Virgil   | Van Dijk  | BA4du6oGxpJ6y29t2DgMI4ax0r/+6Ro=
  Dele     | Alli      | BDYd5WqZEK6YDXC53sC/iYh0TPo3LxQ=
(18 rows)
```

Note that because the encryption values in this example are deterministic, the user without `DECRYPT` privilege can still see which players receive the *same* weekly wages, so there is some of loss of protection in these query results.

Distribution Options

Platforms: All platforms

Parent topic: CREATE TABLE

Within a `CREATE TABLE` statement, you can define a single-column distribution key, or you can set the distribution to `RANDOM` or `REPLICATE`:

DISTRIBUTE ON (column)

Hash distribution across the compute nodes based on the values in the specified column. The `ON` keyword is required. This option is recommended for most tables. For example:

```
create table team
(teamid smallint, htid smallint, atid smallint, name varchar(30), nickname varchar(20), city varchar(20), stadium varchar(50)
distribute on (teamid);
```

Note: You cannot distribute a table on:

- Floating-point columns (`FLOAT` , `FLOAT4` , `FLOAT8`)
- `RANDOMIZED` encrypted columns

DISTRIBUTE [ON] REPLICATE

Replication of the entire table across all blades. The `ON` keyword is optional. This option is intended for smaller tables. For example:

```
create table season
(seasonid smallint, season_name character(9), numteams smallint, winners varchar(30))
distribute replicate;
```

DISTRIBUTE [ON] RANDOM

Random distribution of table rows. The `ON` keyword is optional.

If you do not specify the `DISTRIBUTE` clause, the table is hash-distributed on the first-named column in the table. If you do not specify the `DISTRIBUTE` clause and the first column in the table is a floating-point column, `RANDOM` distribution is used.

Partitioning Options

Platforms: All platforms

Parent topic: CREATE TABLE

This section describes the PARTITION BY syntax in detail.

---

range_partition_spec

```
RANGE ( column BETWEEN start AND end EACH interval [, OUTSIDE RANGE] [, IS NULL] )
```

RANGE

Define partitions based on ranges of values bounded by an interval.

column

A column that exists in the partitioned table. You can define a maximum of four partition columns per table. You cannot specify multiple partitions on the same column. You cannot use expressions to define partition columns.

BETWEEN start AND end EACH interval

Specify the endpoints of the range, then specify the interval that marks the boundary between partitions. Use literal values with compatible data types. For example:

```
between date '2017-01-01' and date '2017-06-30' each interval '1 day'
```

```
between timestamp '2010-01-01 00:00:00' and timestamp '2016-12-31 23:59:59' each interval '1 year'
```

```
between 1 and 10000 each 1
```

Make sure that the number of partitions derived from the specification does not exceed 32767. For example, the following range is not allowed:

```
between 1 and 1000000 each 1
```

The following range is valid:

```
between 1 and 1000000 each 100
```

OUTSIDE RANGE

Optionally, create two separate partitions specifically for values that fall outside (above or below) the specified range. One of these partitions will contain values below the range, and one will contain values above the range.

For example, the following range specification creates 13 partitions, including two `OUTSIDE RANGE` partitions and an `IS NULL` partition:

```
range (range_col between 1 and 10 each 1, outside range, is null)
```

If you query the `sys.table_partition` view for information about a table that has 13 partitions, you will see results like this:

```
premdb=# select * from sys.table_partition;
 table_id | n1 | n2 | n3 | n4 | count
----------+----+----+----+----+-------
    16422 |  0 |  0 |  0 |  0 |    21
    16422 |  1 |  0 |  0 |  0 |   462
    16422 |  2 |  0 |  0 |  0 |   462
    16422 |  3 |  0 |  0 |  0 |   462
    16422 |  4 |  0 |  0 |  0 |   380
    16422 |  5 |  0 |  0 |  0 |   380
    16422 |  6 |  0 |  0 |  0 |   380
    16422 |  7 |  0 |  0 |  0 |   380
    16422 |  8 |  0 |  0 |  0 |   380
    16422 |  9 |  0 |  0 |  0 |   380
    16422 | 10 |  0 |  0 |  0 |   380
    16422 | 11 |  0 |  0 |  0 |   510
    16422 | 12 |  0 |  0 |  0 |     0
(13 rows)
```

In this result set:

- Row `n1=0` counts values that fall below the range
- Rows `n1=1` through `n1=10` count values in the range
- Row `n1=11` counts values that fall above the range
- Row `n1=12` counts null values

`OUTSIDE RANGE` and `IS NULL` can be specified in any order.

IS NULL

Optionally, create a separate partition that holds any `NULL` values. `OUTSIDE RANGE` and `IS NULL` can be specified in any order.

Note: If the partition column may contain `NULL` values, be sure to specify `IS NULL`, or you will be unable to create the table.

---

hash_partition_spec

```
HASH ( column WITH number PARTITIONS [, IS NULL] )
```

HASH

Define partitions based on hashing the values in a column.

column

A column that exists in the partitioned table. You can define a maximum of four partitioned columns per table. You cannot specify multiple partitions on the same column.

WITH number PARTITIONS

Enter a number between 2 and 32767. If an IS NULL partition is specified, the minimum number is 1 and the maximum number is 32766. For example:

```
with 300 partitions
```

IS NULL

Optionally, create a separate partition that holds any `NULL` values. `IS NULL` is optional for hash-partitioned tables, but `OUTSIDE RANGE` is not meaningful and not required.

Note: If the partition column may contain `NULL` values, be sure to specify `IS NULL`, or you will be unable to create the table.

Supported Data Types

The following table shows which column data types are supported for range partitioning and hash partitioning.

Data Type	Range Partitioning Supported	Hash Partitioning Supported
BOOLEAN	No	No
SMALLINT	Yes	Yes
INTEGER	Yes	Yes
BIGINT	Yes	Yes
DECIMAL	No	No
REAL	No	No
DOUBLE PRECISION	No	No
UUID	No	No
VARCHAR	No	Yes*
CHAR	No	Yes
DATE	Yes	Yes
TIME	No	Yes
TIMESTAMP	Yes	Yes
TIMESTAMPTZ	Yes	Yes
IPV4, IPV6	No	Yes
MACADDR, MACADDR8	No	Yes

*You cannot hash partition `RANDOMIZED` encryption columns.

General Restrictions on Partitioning Tables

- Up to four columns in a single table can be partitioned.
- The number of partitions produced by a single specification must be no greater than 32767.
- The product of the number of partitions produced by all specifications for the same table must be no greater than 250000. For example, the following table cannot be created:

```
premdb=# create table hashtable(a int not null, b int not null)
partition by (hash(a with 3000 partitions), hash(b with 1000 partitions));
ERROR:  the product of the number of partitions produced by all specifications results in 3000000 partitions, 250000 allowed
```

- Replicated tables cannot be partitioned.
- Partitioning schemes cannot be altered after a table is created. To change partitioning, you have to re-create the table.

Examples

For example, the following table has two range-based partition columns:

- A DATE column, with the partitions separated by an interval of 1 year within a given date range.
- An INTEGER column, using an interval of 1 for the partitions within a given range of numbers. This column also specifies an optional `OUTSIDE RANGE` partition that captures any values outside of the defined range and an optional `IS NULL` partition that captures `NULL` values.

```
premdb=# create table newmatchstats(seasonid smallint not null, matchday date, htid smallint, atid smallint, moment varchar(5))
distribute on (seasonid)
partition by (
range (matchday between date '8/1/1992' and date '5/31/2017' each interval '1 year', is null),
range (htid between 2 and 51 each 1, outside range, is null)
);
CREATE TABLE
premdb=# \d newmatchstats
       Table "public.newmatchstats"
  Column  |         Type         | Modifiers
----------+----------------------+-----------
 seasonid | smallint             | not null
 matchday | date                 |
 htid     | smallint             |
 atid     | smallint             |
 moment   | character varying(5) |

Distribution: Hash (seasonid)
Partition columns:
   "matchday"    RANGE (BETWEEN date '1992-08-01' AND date '2017-05-31' EACH interval '1 year')
   "htid"    RANGE (BETWEEN 2 AND 51 EACH 1, IS NULL, OUTSIDE RANGE)
```

The following version of the `newmatchstats` table has 100 hash partitions based on values in the `matchday` column:

```
premdb=# create table newmatchstats(seasonid smallint not null, matchday date, htid smallint, atid smallint, moment varchar(5))
distribute on (seasonid)
partition by (hash(matchday with 100 partitions, is null));
CREATE TABLE
premdb=# \d newmatchstats
       Table "public.newmatchstats"
  Column  |         Type         | Modifiers
----------+----------------------+-----------
 seasonid | smallint             | not null
 matchday | date                 |
 htid     | smallint             |
 atid     | smallint             |
 moment   | character varying(5) |

Distribution: Hash (seasonid)
Partition Columns:
   "matchday"    HASH (WITH 100 PARTITIONS)
```

```
premdb=# create table partition_test (c1 int, c2 int) partition by (range (c1 between 1 and 10 each 1, is null, outside range));
CREATE TABLE
premdb=# \d partition_test
Table "public.partition_test"
 Column |  Type   | Modifiers
--------+---------+-----------
 c1     | integer |
 c2     | integer |

Distribution: Hash (c1)
Partition Columns:
```

```
    "c1"  RANGE (BETWEEN 1 AND 10 EACH 1, IS NULL, OUTSIDE RANGE)

premdb=# insert into partition_test(c1) select seasonid from match;
INSERT 0 8606
premdb=# select * from sys.table_partition;
 table_id | n1 | n2 | n3 | n4 | count
----------+----+----+----+----+-------
   16422 |  0 |  0 |  0 |  0 |     0
   16422 |  1 |  0 |  0 |  0 |   462
   16422 |  2 |  0 |  0 |  0 |   462
   16422 |  3 |  0 |  0 |  0 |   462
   16422 |  4 |  0 |  0 |  0 |   380
   16422 |  5 |  0 |  0 |  0 |   380
   16422 |  6 |  0 |  0 |  0 |   380
   16422 |  7 |  0 |  0 |  0 |   380
   16422 |  8 |  0 |  0 |  0 |   380
   16422 |  9 |  0 |  0 |  0 |   380
   16422 | 10 |  0 |  0 |  0 |   380
   16422 | 11 |  0 |  0 |  0 |  4560
   16422 | 12 |  0 |  0 |  0 |     0
(13 rows)
```

Sorted and Clustered Tables

A Yellowbrick table may be *sorted* on one column or *clustered* on multiple columns (up to four). You cannot specify a sort column and cluster columns on the same table. After specifying one of these options, you cannot change the specification without dropping and re-creating the table.

A table that is sorted on one column facilitates the skipping of blocks when tables are scanned and the sorted column is restricted in the query.

A table that is clustered on multiple columns facilitates the skipping of blocks when tables are scanned and two or more cluster columns are restricted in the query.

You cannot sort or cluster on encrypted columns.

Note: Capacity expansion (adding blades to the cluster) results in table rows being rewritten and redistributed on the storage system. If any *manual ordering* was applied to the data before the expansion, there is no guarantee that the data will remain sorted in that order. In turn any query performance advantage associated with that order will be lost (such as skipping blocks during scans).

Sort Column

You can define a sort column for a Yellowbrick table. The `SORT ON (column)` clause names a single column to sort on before data is stored on the media. For example:

```
create table team
(teamid smallint, htid smallint, atid smallint, name varchar(30), nickname varchar(20), city varchar(20), stadium varchar(50), cap
distribute on (teamid)
sort on (city);
```

Clustered Tables

You can define up to four cluster columns for a Yellowbrick table. The `CLUSTER ON (column, column, ...)` clause names multiple columns to cluster on before data is stored on the media. For example:

```
create table team
(teamid smallint, htid smallint, atid smallint, name varchar(30), nickname varchar(20), city varchar(20), stadium varchar(50), cap
distribute on (teamid)
cluster on (name, nickname, city);
```

A clustered table maintains an internal map that organizes data as a set of points across the dimensions that are defined as cluster columns. This map optimizes query performance because the optimizer can skip the blocks of data that do not qualify for multiple restrictions. One pass over the data is sufficient to find the intersection of rows that meet the `WHERE` clause criteria, and a large amount of data in the base table can be eliminated very early in the query plan.

GRANT

In this section:

Grant privileges on database objects to specific roles and grant membership in roles.

You can grant privileges at the instance level (`ON SYSTEM`) and on the following lower-level objects. (`ON SYSTEM` means that the privileges are granted to users and roles within a specific data warehouse instance.)

- Virtual compute clusters (`ON CLUSTER`)
- Databases (`ON DATABASE`)
- Schemas (`ON SCHEMA`)
- Tables, views, and columns in tables (`ON TABLE`)
- Sequences (`ON SEQUENCE`)
- Stored procedures (`ON PROCEDURE`)
- Encryption keys (`ON KEY`)
- Users and roles (`ON ROLE`)
- External storage, location, and format objects (`ON EXTERNAL`)

Note that newly granted privileges are added to any that have already been granted.

Access privileges (ACLs) are logged in sys.database. For a complete list of ACLs and their abbreviations, see Abbreviations for ACLs.

Here is a complete syntax summary for the GRANT command, followed by definitions of some common parameters. See the subsequent sections for details about each `ON` variant.

```
GRANT { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES }
   [, ...] | ALL [ PRIVILEGES ] }
   ON { [ TABLE ] table_name [, ...]
        | ALL TABLES IN SCHEMA schema_name [, ...] }
   TO role_specification [, ...] [ WITH GRANT OPTION ]

GRANT { { SELECT | INSERT | UPDATE | REFERENCES } ( column_name [, ...] )
   [, ...] | ALL [ PRIVILEGES ] ( column_name [, ...] ) }
   ON [ TABLE | VIEW ] table_name | view_name [, ...]
```

```
   TO role_specification [, ...] [ WITH GRANT OPTION ]

GRANT { { CREATE | CONNECT | TEMPORARY | TEMP |
   EXPLAIN QUERY | TRACE QUERY | VIEW QUERY TEXT |
   ALTER ANY SCHEMA | DROP ANY SCHEMA |
   BACKUP | RESTORE | BULK LOAD | CONTROL }
   [, ...] | ALL [ PRIVILEGES ] }
   ON DATABASE database_name [, ...]
   TO role_specification [, ...] [ WITH GRANT OPTION ]

GRANT { EXECUTE | ALL [ PRIVILEGES ] }
   ON { PROCEDURE procedure_name ( [ [ argmode ] [ arg_name ] arg_type [, ...] ] )
   [, ...] | ALL PROCEDURES IN SCHEMA schema_name [, ...] }
   TO role_specification [, ...] [ WITH GRANT OPTION ]

GRANT { { CREATE | USAGE } [, ...] | ALL [ PRIVILEGES ] }
   ON SCHEMA schema_name [, ...]
   TO role_specification [, ...] [ WITH GRANT OPTION ]

GRANT { { USAGE | SELECT | UPDATE }
   [, ...] | ALL [ PRIVILEGES ] }
   ON { SEQUENCE sequence_name [, ...] | ALL SEQUENCES IN SCHEMA schema_name [, ...] }
   TO role_specification [, ...] [ WITH GRANT OPTION ]

GRANT { { ENCRYPT | DECRYPT | HMAC }
   [, ...] | ALL [ PRIVILEGES ] }
   ON KEY key_name
   TO { role_specification } [, ...] [ WITH GRANT OPTION ]

GRANT { { CREATE ROLE | ALTER ANY ROLE | DROP ANY ROLE | VIEW ROLE |
   CREATE DATABASE | ALTER ANY DATABASE | DROP ANY DATABASE | BACKUP ANY DATABASE | RESTORE ANY DATABASE |
   TRACE QUERY | VIEW QUERY TEXT | EXPLAIN QUERY |
   CONTROL ANY SESSION | CONTROL LDAP }
   [, ...] | ALL [ PRIVILEGES ] }
   ON SYSTEM
   TO { role_specification } [, ...] [ WITH GRANT OPTION ]

GRANT { { ALTER ROLE | DROP ROLE | CONTROL }
   [, ...] | ALL [ PRIVILEGES ] }
   ON ROLE role_specification [, ...]
   TO role_specification [, ...] [ WITH GRANT OPTION ]

where role_specification can be:

   role_name
  | PUBLIC
  | CURRENT_USER
  | SESSION_USER

GRANT role_name [, ...] TO role_name [, ...] [ WITH ADMIN OPTION ]
```

Common Parameters (All GRANT Commands)

ALL PRIVILEGES

Grant all of the available privileges on the object. Individual privileges may be later revoked from a user with `ALL PRIVILEGES`. `CONTROL` (`ON DATABASE` or `ROLE`) is not part of `ALL`.

Note: `GRANT ALL` does not equate to superuser or table owner privileges.

See also Granting Schema-Level Privileges to Members of a Role.

role_specification

Name of an existing role or one of the following:

- `PUBLIC` : an implicit group that always includes all roles. `PUBLIC` means that privileges are granted to both existing roles and new roles when they are created. A given role accrues all of the following privileges:
  - Privileges granted directly
  - Privileges granted to roles that the role in question is a member of
  - Privileges granted to `PUBLIC` Yellowbrick grants default `CONNECT` and `CREATE TEMP TABLE` privileges for databases to `PUBLIC`. No default privileges are granted to `PUBLIC` on tables, columns, or schemas.
- CURRENT_USER
- SESSION_USER

WITH GRANT OPTION

`WITH GRANT OPTION` means that the receiving role can grant the same privilege to other roles. You cannot grant options to `PUBLIC`. Granting privileges to the owner of a table, schema, or database (the owner is typically the user who created the object) is not necessary. Owners accrue all privileges by default (but may revoke some of them for security reasons).

WITH ADMIN OPTION

Privilege to grant and revoke membership in the role to others.

Object Owners and Creators

The ability to drop and alter objects is not subject to `GRANT` actions. Owners have these privileges by default, and you cannot grant or revoke them. Conversely, you can grant or revoke membership in the role that owns an object. Also note that the owner of an object implicitly has all grant options for that object.

An object owner can revoke both default and granted privileges. If you want to revoke privileges, for security purposes it is a best practice to do so within the same transaction that creates the object. In this way, you do not allow a window of time in which another user can access the object.

Note that object *ownership* can be transferred, but the *creator* of an object is persistent.

Abbreviations for ACLs

Platforms: All platforms

Parent topic: GRANT

The following table is a reference for the ACL abbreviations that are used in `sys.database` and the output of the `ybsql \ddp` command. Some abbreviations are used to identify multiple ACLs on different objects.

Abbreviation	ON SYSTEM	ON DATABASE	SCHEMAS	TABLES/VIEWS	COLUMNS	SEQUENCES	STORED PROCEDURES
a (append)		BULK LOAD		INSERT	INSERT		
r (read)	VIEW ROLE			SELECT	SELECT	SELECT	
w (write)		ALTER ANY DATABASE	ALTER ANY SCHEMA	UPDATE	UPDATE	UPDATE	
d		DROP ANY DATABASE	DROP ANY SCHEMA	DELETE (tables)			
D				TRUNCATE (tables)			
x				REFERENCES (create a foreign-key constraint)	REFERENCES		
t							
X	CONTROL LDAP						EXECUTE
U	CONTROL ANY SESSION		USAGE			USAGE	
C	CREATE ROLE	CREATE	CREATE				
T		CREATE TEMPORARY					
c		CONNECT					
e							
E							

Abbreviation	ON SYSTEM	ON DATABASE	SCHEMAS	TABLES/VIEWS	COLUMNS	SEQUENCES	STORED PROCEDURES
h							
b	CREATE DATABASE						
p	EXPLAIN QUERY	EXPLAIN QUERY					
q	VIEW QUERY TEXT	VIEW QUERY TEXT					
Q	TRACE QUERY	TRACE QUERY					
A	ALTER ANY ROLE						
B	DROP ANY ROLE						
u	BACKUP ANY DATABASE	BACKUP					
O	RESTORE ANY DATABASE	RESTORE					
Z		CONTROL					

ON CLUSTER

Platforms: EE: All cloud platforms

Parent topic: GRANT

Syntax

```
GRANT USAGE
   ON CLUSTER cluster_name [, ...]
   TO role_specification [, ...] [ WITH GRANT OPTION ]
```

Privilege to use one or more clusters in an instance for queries and other database operations. This privilege is checked when the `USE CLUSTER` statement is run, and whenever a query plan is sent to the Yellowbrick database (backend). Users must be granted this privilege in order to connect to a data warehouse instance and run queries or database commands.

Example

Grant `USAGE` on a specific cluster to a specific user:

```
premdb=# grant usage on cluster "bobr-rc5-april4-cluster" to jamesbond;
GRANT
```

ON DATABASE

Platforms: All platforms

Parent topic: GRANT

Syntax

```
GRANT {
   { CREATE | CONNECT | TEMPORARY | TEMP |
   EXPLAIN QUERY | TRACE QUERY | VIEW QUERY TEXT |
   ALTER ANY SCHEMA | DROP ANY SCHEMA |
   BACKUP | RESTORE | BULK LOAD | CONTROL }
   [, ...] | ALL [ PRIVILEGES ] }
   ON DATABASE database_name [, ...]
   TO role_specification [, ...] [ WITH GRANT OPTION ]
```

Parameters

ON DATABASE

Grant privileges on a physical database. You can use one statement to grant privileges on several databases.

CREATE

Privilege to create new schemas in that database.

CONNECT

Privilege to connect to the specified database. By default, new users have this privilege on all databases.

TEMP or TEMPORARY

Privilege to create temporary tables while using the named database or databases.

EXPLAIN QUERY

Privilege to run the EXPLAIN command for queries against the named database or databases. Use this privilege when SELECT privilege is not being granted. Users and roles with SELECT privilege have implicit EXPLAIN QUERY privilege.

TRACE QUERY

Privilege to query a set of system views for the purpose of tracing execution details for queries that the named users and roles did not run themselves. `TRACE QUERY` privilege is implied for users' own queries and does not need to be granted. `TRACE QUERY` privilege provides access to the following system views:

- `sys.query`
- `sys.log_query`
- `sys.session`
- `sys.log_session`
- `sys.query_explain`

- `sys.query_analyze`

VIEW QUERY TEXT

Privilege to select from the following system views for the purpose of viewing SQL text for *any query*, including queries that the named users and roles run themselves:

- `sys.query`
- `sys.log_query`

Note that users who do not have this privilege will not be able to access the SQL text of their own queries.

ALTER ANY SCHEMA

Privilege to alter the name or owner of any schema in the specified database. See ALTER SCHEMA.

DROP ANY SCHEMA

Privilege to drop any schema in the specified database. See DROP SCHEMA.

BACKUP

Privilege to back up a database (using `ybbackup`).

RESTORE

Privilege to restore a database (using `ybrestore`).

BULK LOAD

Privilege to bulk load tables in the specified database (using `ybload`).

Important: The role must also have INSERT permissions on the table (but does not have to own the table).

CONTROL

Privilege to run all database-level operations. This is similar to `ALL` privileges, but `CONTROL` includes any `ON DATABASE` privileges that may be added in a future release. You cannot revoke individual `ON DATABASE` privileges from a user with `CONTROL`, and `CONTROL` is not part of `ALL`.

See also HAS_DATABASE_PRIVILEGE.

Example: CONNECT, CREATE, and TEMP Privileges WITH GRANT OPTION

Grant three privileges on the `testdb` database to user `yb100`. Include `WITH GRANT OPTION` so that this user can grant the same privilege to other roles.

```
premdb=# grant create, connect, temp on database testdb to yb100 with grant option;
GRANT
```

Note that access privileges will be listed in sys.database as `yb100=C*T*c*`, where the asterisk refers to `WITH GRANT OPTION` having been granted.

Example: BACKUP and RESTORE Privileges for a Role

For example, a fresh database has no access privileges granted to regular users. You can grant all privileges on the database to the members of a specific role (existing or new). Then you can revoke specific privileges that you don't want that role to have, such as backup and restore privileges on the database. This kind of setup can be achieved quickly with a few SQL commands.

For example, when database `premdb` is first created, access privileges for users and roles are set to `NULL`:

```
premdb=# select * from sys.database where name='premdb';
-[ RECORD 1 ]------+--------
database_id        | 16479
name               | premdb
owner_id           | 16007
encoding           | LATIN9
collation          | C
is_readonly        | f
readonly_reason    | [NULL]
is_hot_standby     | f
access_privileges  | [NULL]
table_count        | 5
rows_columnstore   | 100
rows_rowstore      | 8681
compressed_bytes   | 8388608
uncompressed_bytes | 6628
```

Create a role named `bar`, then create some user accounts that will belong to `bar`:

```
create role bar;
create user backup1 password '*******';
create user backup2 password '*******';
create user restore1 password '*******';
create user restore2 password '*******';
```

Now grant membership in `bar` to the four new users:

```
grant bar to backup1,backup2,restore1,restore2;
```

Now grant `BACKUP` and `RESTORE` privileges only, on `premdb` only, to `bar`:

```
grant backup, restore on database premdb to bar;
```

Finally, query `sys.database` to check the access privileges for `bar`:

```
premdb=# select name, access_privileges from sys.database where name='premdb';
  name  |    access_privileges
--------+--------------------------
 premdb | =Tc/ybrick1             +
        | ybrick1=CTcpqQuO/ybrick1+
        | bobr=CTcpqQuO/ybrick1   +
        | bar=uO/ybrick1          +
        | yb007=C/ybrick1
(1 row)
```

Note that `bar=u0`, where `u=BACKUP` privilege and `0=RESTORE` privilege. (For details about the abbreviations used for access privileges, see sys.database.)

At this point all four users in the bar role would be able to run `ybbackup` client operations on the `premdb` database.

Example: EXPLAIN QUERY Privilege

Grant the privilege to explain queries on a database for users who do not have `SELECT` privilege on the tables in that database:

```
premdb=# grant explain query on database premdb to bobr;
GRANT
```

```
premdb=# \c premdb bobr
You are now connected to database "premdb" as user "bobr".
premdb=> explain select * from season;
                 QUERY PLAN
----------------------------------------------
 id   rows_planned   workers   node
  0             25    single   SELECT
  2             25    single   SCAN season
 Database: premdb
 Version: 4.1.0-23947
 Hostname: yb007

(7 rows)
premdb=> select * from season;
ERROR:  permission denied for relation season
```

A user who needs to explain cross-database queries would need privileges on multiple databases. For example:

```
premdb=# grant explain query on database yellowbrick, premdb to bobr;
GRANT
premdb=# \c premdb bobr
You are now connected to database "premdb" as user "bobr".
premdb=> explain select * from premdb.public.team union select * from yellowbrick.public.team;
                                                                      QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------
 id   rows_planned   workers   node
  0            150    single   SELECT
  2            150    single   GROUP BY (union.[36], union.[37], union.[38], union.[39], union.[40], union.[41], union.[42], union
  3            150    single   APPEND
  6             50    single   |-SCAN premdb.public.team
 11            100    single   |-SCAN yellowbrick.public.team
 Database: premdb
 Version: 4.1.0-23947
 Hostname: brumsby

(10 rows)
```

Example: VIEW QUERY TEXT Privilege

User `bobr` has `SELECT` privilege on the `player` table and can run this query:

```
premdb=> \c premdb bobr
You are now connected to database "premdb" as user "bobr".
premdb=> select * from team where avg_att is not null order by avg_att desc limit 3;
 teamid | htid | atid |       name        |  nickname  |    city    |     stadium      | capacity | avg_att
--------+------+------+-------------------+------------+------------+------------------+----------+---------
     25 |   26 |   75 | Manchester United | Red Devils | Manchester | Old Trafford     |    75635 |  75.286
      1 |    2 |   51 | Arsenal           | Gunners    | London     | Emirates Stadium |    60260 |  59.944
     24 |   25 |   74 | Manchester City   | Citizens   | Manchester | Etihad Stadium   |    55097 |  54.041
(3 rows)
```

However, by default `bobr` may not see the query text for this query:

```
premdb=> select query_text from sys.log_query where query_text like 'select * from team where avg_att%';
 query_text
------------
(0 rows)
```

`VIEW QUERY TEXT` privilege must be granted to `bobr` on the `premdb` database, which contains the `player` table:

```
premdb=> \c premdb yellowbrick
You are now connected to database "premdb" as user "yellowbrick".
premdb=# grant view query text on database premdb to bobr;
GRANT
premdb=# \c premdb bobr
You are now connected to database "premdb" as user "bobr".
premdb=> select query_text from sys.log_query where query_text like 'select * from team where avg_att%';
                              query_text
------------------------------------------------------------------------
 select * from team where avg_att is not null order by avg_att desc limit 3;
(1 row)
```

Example: BULK LOAD Privilege

User `bobr` is granted privilege to run `ybload` operations on the `premdb` database:

```
yellowbrick=# grant bulk load on database premdb to bobr;
GRANT
```

This user will also need `INSERT` privilege on the specific tables being loaded.

ON EXTERNAL object

Platforms: EE: All cloud platforms, CE

Parent topic: GRANT

Syntax

```
GRANT {
   { ALTER EXTERNAL FORMAT | DROP EXTERNAL FORMAT | USAGE EXTERNAL FORMAT }
   [, ...] | ALL [ PRIVILEGES ] }
   ON EXTERNAL FORMAT format_name [, ...]
   TO role_specification [, ...] [ WITH GRANT OPTION ]
```

```
GRANT {
   { ALTER EXTERNAL LOCATION | DROP EXTERNAL LOCATION | USAGE EXTERNAL LOCATION }
   [, ...] | ALL [ PRIVILEGES ] }
   ON EXTERNAL LOCATION location_name [, ...]
   TO role_specification [, ...] [ WITH GRANT OPTION ]
```

```
GRANT {
   { ALTER EXTERNAL STORAGE | DROP EXTERNAL STORAGE | USAGE EXTERNAL STORAGE }
   [, ...] | ALL [ PRIVILEGES ] }
   ON EXTERNAL STORAGE storage_name [, ...]
   TO role_specification [, ...] [ WITH GRANT OPTION ]
```

Parameters

These privileges apply to a single named external format, external location, or external storage object.

ALTER EXTERNAL FORMAT
Privilege to alter a single external format object.

DROP EXTERNAL FORMAT
Privilege to drop a specific external format object.

USAGE EXTERNAL FORMAT
Privilege to use a specific external format object.

ALTER EXTERNAL LOCATION
Privilege to alter a single external location object.

DROP EXTERNAL LOCATION
Privilege to drop a specific external location object.

USAGE EXTERNAL LOCATION
Privilege to use a single external location object.

ALTER EXTERNAL STORAGE
Privilege to alter a single external storage object.

DROP EXTERNAL STORAGE

Privilege to drop a single external storage object.

USAGE EXTERNAL STORAGE

Privilege to use a single external storage object.

Examples

```
GRANT ALTER EXTERNAL LOCATION ON EXTERNAL LOCATION "premdbs3data" TO "ybsql10";
GRANT DROP EXTERNAL LOCATION ON EXTERNAL LOCATION "premdbs3data" TO "ybsql10";
GRANT USAGE EXTERNAL LOCATION ON EXTERNAL LOCATION "premdbs3data" TO "ybsql10";
```

```
GRANT ALL ON EXTERNAL STORAGE "premdbs3" TO "ybsql10";
```

ON KEY

Platforms: All platforms

Parent topic: GRANT

Syntax

```
GRANT { ENCRYPT | DECRYPT | HMAC | ALL [ PRIVILEGES ] }
   ON KEY key_name
   TO { role_specification } [, ...] [ WITH GRANT OPTION ]
```

Parameters

ON KEY
Grant privileges on an encryption key created with the CREATE KEY command.

ENCRYPT
Privilege to encrypt data, using a specific key.

DECRYPT
Privilege to decrypt data, using a specific key.

HMAC
Privilege to run the `HMAC_KS` function with a specific key as input to the function.

Examples

Grant `ENCRYPT` privilege on key `yb100key` to user `yb100`:

```
premdb=# grant encrypt on key yb100key to yb100;
GRANT
```

Grant both `ENCRYPT` and `DECRYPT` privileges on key `yb100key` to user `yb100`:

```
premdb=# grant all privileges on key yb100key to yb100;
GRANT
```

Grant `HMAC` privilege on a key:

```
premdb=# grant hmac on key playerkey to bobr;
GRANT
```

ON PROCEDURE

Platforms: All platforms

Parent topic: GRANT

Syntax

```
GRANT { EXECUTE | ALL [ PRIVILEGES ] }
   ON { PROCEDURE procedure_name ( [ [ argmode ] [ arg_name ] arg_type [, ...] ] ) [, ...]
        | ALL PROCEDURES IN SCHEMA schema_name [, ...] }
   TO role_specification [, ...] [ WITH GRANT OPTION ]
```

Parameters

ON PROCEDURE

Grant privileges on a stored procedure.

Note: A role cannot execute a procedure unless `EXECUTE` is granted to the role on the procedure and `USAGE` or `ALL PRIVILEGES` is granted to the role on the owning schema.

EXECUTE

Privilege to execute (run) a procedure.

ALL PROCEDURES IN SCHEMA

Grant the privilege to all the procedures in the named schema.

Examples

Grant `EXECUTE` privilege on a stored procedure to the `ybd` user.

```
premdb=# grant execute on procedure proc1() to ybd;
GRANT
```

Grant `EXECUTE` privilege on all stored procedures in the `public` schema to the `ybd` user:

```
premdb=# grant execute on all procedures in schema public to ybd;
GRANT
```

ON ROLE

Platforms: All platforms

Parent topic: GRANT

Syntax

```
GRANT { { ALTER ROLE | DROP ROLE | CONTROL }
   [, ...] | ALL [ PRIVILEGES ] }
   ON ROLE role_specification [, ...]
   TO role_specification [, ...] [ WITH GRANT OPTION ]
```

Parameters

ALTER ROLE

Privilege to make other users and roles members of the specified role and to change passwords and password policies for the role. Only superusers can alter other superusers.

DROP ROLE

Privilege to drop the specified role. Only superusers can drop other superusers.

CONTROL

All privileges on the specified role, including any privileges that may be defined as `GRANT` privileges on roles in future releases. You cannot revoke individual `ON ROLE` privileges from a user with `CONTROL`, and `CONTROL` is not part of `ALL`.

See also HAS_ROLE_PRIVILEGE and the separate GRANT ROLE command.

Example

Grant `CONTROL` privilege on four users to role `bar`:

```
premdb=# grant control on role backup1, backup2, restore1, restore2 to bar;
GRANT
```

ON SCHEMA

Platforms: All platforms

Parent topic: GRANT

Syntax

```
GRANT { { CREATE | USAGE } [, ...] | ALL [ PRIVILEGES ] }
   ON SCHEMA schema_name [, ...]
   TO role_specification [, ...] [ WITH GRANT OPTION ]
```

Parameters

ON SCHEMA

Grant privileges on a schema in a database. You can use one statement to grant privileges on several schemas.

CREATE

Privilege to create new tables and views in that schema. To rename an object, you must both own the object and have `CREATE` privilege for the schema that contains the object.

USAGE

Privilege to access objects in the specified schema (assuming that privileges specific to those objects are met). This privilege allows the role to see objects within the schema, but does not grant `SELECT` on the tables in the schema. See also Granting Schema-Level Privileges to Members of a Role.

Examples

Grant `CREATE` and `USAGE` privileges on schema `jbdb` to two users:

```
premdb=# grant all privileges on schema jbdb to yb007,yb008;
GRANT
```

Grant `USAGE` privileges on schema `jbdb` to the role `bar`:

```
premdb=# grant usage on schema jbdb to bar;
GRANT
```

See also Granting Schema-Level Privileges to Members of a Role.

ON SEQUENCE

Platforms: All platforms

Parent topic: GRANT

Syntax

```
GRANT { { USAGE | SELECT | UPDATE }
   [, ...] | ALL [ PRIVILEGES ] }
   ON { SEQUENCE sequence_name [, ...]
        | ALL SEQUENCES IN SCHEMA schema_name [, ...] }
   TO role_specification [, ...] [ WITH GRANT OPTION ]
```

Parameters

USAGE

Privilege to access sequences in the specified schema. This privilege allows the role to see sequences, but does not grant `SELECT` or `UPDATE` privileges on them.

SELECT

Privilege to select from the sequence.

UPDATE

Privilege to alter the sequence.

ON SEQUENCE

Grant privileges on specific sequence objects.

ALL SEQUENCES IN SCHEMA

Grant the privilege to all the sequences in the named schema.

Examples

Grant privileges on sequences:

```
premdb=# grant usage on matchid to bobr;
GRANT
```

For example:

```
premdb=# grant select on all sequences in schema public to bobr with grant option;
GRANT
```

ON SYSTEM

Platforms: All platforms

Parent topic: GRANT

Syntax

```
GRANT {
     { CREATE ROLE | ALTER ANY ROLE | DROP ANY ROLE | VIEW ROLE |
     CREATE DATABASE | ALTER ANY DATABASE | DROP ANY DATABASE | BACKUP ANY DATABASE | RESTORE ANY DATABASE |
     TRACE QUERY | VIEW QUERY TEXT | EXPLAIN QUERY |
     CONTROL ANY SESSION | CONTROL WLM |
     ALTER ANY CLUSTER | CREATE CLUSTER | DROP ANY CLUSTER | USAGE ANY CLUSTER
     CREATE EXTERNAL { FORMAT | LOCATION | STORAGE }
     DROP ANY EXTERNAL { FORMAT | LOCATION | STORAGE }
     USAGE ANY EXTERNAL { FORMAT | LOCATION | STORAGE }
     }
     [, ...] | ALL [ PRIVILEGES ] }
     ON SYSTEM
     TO role_specification [, ...] [ WITH GRANT OPTION ]
```

Parameters

ON SYSTEM

Grant privileges to users and roles within a data warehouse instance.

CREATE ROLE

Privilege to create roles.

ALTER ANY ROLE

Privilege to alter any role.

DROP ANY ROLE

Privilege to drop any role.

VIEW ROLE

Privilege to see all rows in the `sys.user` and `sys.role` views.

CREATE DATABASE

Privilege to create databases.

ALTER ANY DATABASE

Privilege to run ALTER DATABASE commands on any database in the system.

Note: This privilege does not include permission to add, drop, or alter replicas for database replication.

DROP ANY DATABASE

Privilege to drop any database (except `yellowbrick` and other system databases).

BACKUP ANY DATABASE

Privilege to back up any database (using `ybbackup`).

RESTORE ANY DATABASE

Privilege to restore any database (using `ybrestore`).

TRACE QUERY

Privilege to query a set of system views for the purpose of tracing execution details for queries that the named users and roles did not run themselves. `TRACE QUERY` privilege is implied for users' own queries and does not need to be granted. `TRACE QUERY` privilege provides access to the following system views:

- `sys.query`
- `sys.log_query`
- `sys.session`
- `sys.log_session`
- `sys.query_explain`
- `sys.query_analyze`

VIEW QUERY TEXT

Privilege to query the following system views:

- `sys.query`
- `sys.log_query`

EXPLAIN QUERY

Privilege to run `EXPLAIN` for queries that contain tables and views on which the role does not have existing privileges. Roles with existing privileges on those objects do not need this privilege.

CONTROL ANY SESSION

Privilege to terminate user connections by running the yb_terminate_session function. This privilege also includes permission to view everything in the sys.session and sys.log_session views.

CONTROL WLM

Privilege to create, alter, and drop workload management (WLM) profiles, resource pools, and rules.

ALTER ANY CLUSTER

Privilege to alter any compute cluster that belongs to an instance.

CREATE CLUSTER

Privilege to create new compute clusters for an instance.

DROP ANY CLUSTER

Privilege to drop any cluster that belongs to an instance.

USAGE ANY CLUSTER

Privilege to use any cluster that belongs to an instance.

CREATE EXTERNAL FORMAT | LOCATION | STORAGE

Privilege to create external format, location, and storage objects.

DROP ANY EXTERNAL FORMAT | LOCATION | STORAGE

Privilege to drop any external format, location, or storage object.

USAGE ANY EXTERNAL FORMAT | LOCATION | STORAGE

Privilege to use any external , location, or storage object that belongs to an instance.

See also HAS_SYSTEM_PRIVILEGE.

Examples

Grant role privileges to user `yb100`:

```
premdb=# grant create role, alter any role, drop any role on system to yb100;
GRANT
```

Grant the privilege to back up or restore any database to two users:

```
premdb=# grant backup any database, restore any database on system to restore1, restore2;
GRANT
```

This example shows that the user `dm1` needs two different privileges in order to be able to run `CREATE EXTERNAL LOCATION`:

```
premdb=# create user dm1 password 'dm1';
CREATE ROLE
premdb=# grant usage any external storage on system to dm1;
GRANT
grant create external location on system to dm1;
GRANT

--Now connect as user "dm1"--

premdb=> create external location dm1loc path 'nyc-tlc' external storage nyc_taxi_storage;
CREATE EXTERNAL LOCATION
```

Allow user `dbeaver1` to drop any external storage object:

```
premdb=# grant drop any external storage on system to dbeaver1;
```

Grant user `bobr` privileges to alter a set of three external objects:

```
premdb=# grant alter external storage on external storage premdbs3 to bobr;
GRANT
premdb=# grant alter external format on external format premdbs3format to bobr;
GRANT
premdb=# grant alter external location on external location premdbs3data to bobr;
GRANT
```

Create a new user and grant privileges to create, alter, and drop clusters:

```
premdb=# create user cluster2 password 'cluster2';
CREATE ROLE
premdb=# grant drop any cluster on system to cluster2;
GRANT
premdb=# grant alter any cluster on system to cluster2;
GRANT
premdb=# grant create cluster on system to cluster2;
GRANT
```

ON TABLE

Platforms: All platforms

Parent topic: GRANT

Syntax

```
GRANT { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES  }
   [, ...] | ALL [ PRIVILEGES ] }
   ON { [ TABLE ] table_name [, ...]
        | ALL TABLES IN SCHEMA schema_name [, ...] }
   TO role_specification [, ...] [ WITH GRANT OPTION ]

GRANT { { SELECT | INSERT | UPDATE | REFERENCES } ( column_name [, ...] )
   [, ...] | ALL [ PRIVILEGES ] ( column_name [, ...] ) }
   ON [ TABLE | VIEW ] table_name | view_name [, ...]
   TO role_specification [, ...] [ WITH GRANT OPTION ]
```

Parameters

ON TABLE

Grant privileges on a table, a view, or specific columns in a table. You can use one statement to grant privileges on several tables or all of the tables in a schema ("all tables" includes views).

SELECT

Privilege to select from the specified table or view, or specified columns in a table or view. This privilege applies to both temporary tables and persistent tables. This privilege also permits references to existing column values in `UPDATE` and `DELETE` statements.

The `TABLE` keyword is supported for both tables and views. The keyword `VIEW` is only supported for views.

INSERT

Privilege to insert or bulk load rows into the specified table or table column. Bulk loading with `ybload` also requires the `BULK LOAD` privilege on the database.

UPDATE

Privilege to update the specified table or the specified columns in a table. In most cases, UPDATE commands require `SELECT` privilege on the same table. UPDATE commands reference columns to determine which rows to update and to compute column values. For example, you would have to give `SELECT` access on the whole `team` table to `vicky` for this UPDATE command to work:

```
premdb=# grant select(teamid) on team to vicky;
GRANT
premdb=# grant update(teamid) on team to vicky;
GRANT
premdb=# \c premdb vicky
You are now connected to database "premdb" as user "vicky".
premdb=> update team set teamid=0 where teamid=3000;
ERROR:  permission denied for relation team
```

DELETE

Privilege to delete rows from the specified table or the specified columns in a table. In most cases, DELETE commands require `SELECT` privilege on the same table.

DELETE commands reference columns to determine which rows to update.

TRUNCATE

Privilege to truncate a table.

REFERENCES

Privilege to create a foreign-key constraint (required on both the referencing table and the referenced table).

Examples

Grant all privileges on the `match` table to the user `bobr`:

```
premdb=# grant all on match to bobr;
GRANT
```

Grant `SELECT` privilege on all tables in the `sys` schema to the user `vicky`:

```
premdb=# grant select on all tables in schema sys to vicky;
GRANT
```

Grant `SELECT` privilege on a view called `teamview` to the user `bobr`:

```
premdb=# grant select on view teamview to bobr;
GRANT
```

You can also use the `TABLE` keyword to refer to views:

```
premdb=# grant select on table teamview to bobr;
GRANT
```

ROLE

Platforms: All platforms

Parent topic: GRANT

Grant membership in a role to one or more other roles.

Syntax

```
GRANT role_name [, ...] TO role_name [, ...] [ WITH ADMIN OPTION ]
```

Parameters

Membership propagates the privileges granted to a role to each of its members. `WITH ADMIN OPTION` means that the role can grant and revoke membership in the role to others. Database superusers can grant or revoke membership in any role to anyone.

Roles that have `CREATEROLE` privilege (as conferred with CREATE ROLE statements) can grant or revoke membership in any role that is not a superuser.

You cannot grant membership in a role to `PUBLIC`.

See also the separate GRANT...ON ROLE command.

Examples

For example, grant membership to the `allqa` role for a `newhire` role:

```
premdb=# create role newhire;
CREATE ROLE
premdb=# grant allqa to newhire;
GRANT ROLE
premdb=# \du
                                 List of roles
  Role name  |                         Attributes                         | Member of
-------------+------------------------------------------------------------+-----------
 ...
 allqa       | Create DB, Cannot login                                    | {}
 newhire     | Cannot login                                               | {allqa}
 ...
```

Plan Hinting: Overview, Setup, and General Information

In this section:

Overview

Plan Hinting is a powerful feature that allows users to influence the query planner by enforcing specific plan shapes using various hints. These hints can guide the query execution strategy by specifying join methods, ordering joins, modifying expected row counts or changing certain configuration parameter values for the query duration.

Plan Hinting is especially useful for complex queries where the default planner may not generate the most optimal plan, giving users more control over performance.

Enabling and Disabling Plan Hinting

Two configuration parameters control whether Plan Hinting is enabled or disabled:

1. `enable_query_hint_injection`: Enables or disables the Plan Hinting feature.
2. `enable_wlm_query_hint_injection`: Allows hints to be injected via Workload Manager (WLM) rules.

By default, both configuration parameters are set to `off` and need to be set to `on` to activate the feature.

Example :

To set these properties for only the current session, one can run:

```sql
SET enable_query_hint_injection = on;
SET enable_wlm_query_hint_injection = on;
```

Note: These two parameters can not be directly injected in a query using SET hints.

Referring to Tables and Joins

When referring to tables or joins in a hint, you can use:

- **Table names** (qualified or unqualified).
- **Table aliases** (if used in the query).
- **Subquery aliases** or **CTE names** to target subqueries or CTEs.

Case Sensitivity

By default, table names are not case-sensitive. To enforce case sensitivity, place quotes around the table names.

```sql
SETTING (ROWS(orders, c # 42)) SELECT * FROM orders o JOIN customers c ON o.customer_id = c.id;
```

Here, we're telling the planner the output of join in between orders and c is estimated to contain 42 rows.

WLM Hint Injection

Plan Hinting can also be applied via **Workload Manager (WLM)** rules of type `hinting`. This allows hints to be injected into the query plan without modifying the original query text by using the syntax :

```javascript
w.addPlanHint("hint");
```

where `hint` must be one of the available hints :

- SET hint
- ROWS hint
- JOIN METHOD hint
- LEADING hint

However, if manual hints are embedded in the query, WLM-injected hints will not be applied.

Note: Hinting rules can only be created via the SQL command CREATE WLM RULE

Example of creation of a hinting WLM Rule:

```sql
DROP WLM RULE IF EXISTS "hinting_rule";
CREATE WLM RULE "hinting_rule" (
    TYPE hinting, RULE_ORDER 100, SUPERUSER true, ENABLED true,
    JAVASCRIPT
    $$
        w.addPlanHint("LEADING ( ( (A, C), B))");
        w.addPlanHint("HASHJOIN (A, C)");
        w.addPlanHint("NESTLOOP (A, C, B)");
    $$
);
ALTER WLM PROFILE "default" ACTIVATE;
SET enable_wlm_query_hint_injection = true;
```

To track the hints that were added to a query during the hinting stage, you can check `sys.query_rule_event` :

```sql
select * from sys.query_rule_event where rule_type='hinting';
```

Viewing Applied Hints

To see which hints were applied to a query, you can use the `EXPLAIN(hint)` command. To view both applied and non-applied hints, use the `verbose` option in conjunction the the `hint` option.

Example:

```sql
EXPLAIN (hint) SELECT * FROM orders JOIN customers ON orders.customer_id = customers.id;
EXPLAIN (verbose, hint) SELECT * FROM orders JOIN customers ON orders.customer_id = customers.id;
```

Understanding why a hint is not applied

Most errors related to query hints are silent to prevent query execution failures.

As mentioned above, you can usually determine which hints were applied or encountered errors by using the `EXPLAIN(hint, verbose)` statement.

In the case of an error, it will be displayed next to the hint in the output:

```sql
yellowbrick_test=# EXPLAIN (stable, hint, verbose) SETTING ("inlist_threshol" = 2) SELECT * FROM Foo WHERE i IN (1, 2, 3, 4);
                                    QUERY PLAN
-------------------------------------------------------------------------------------------
 workers   node
     all   SELECT
             (foo.i)
             distribute on (foo.i)
     all   SCAN foo
             (foo.i)
             distribute on (foo.i)
             ((foo.i = $3) OR ((foo.i = $2) OR ((foo.i = $0) OR (foo.i = $1))))

 ------------------------------------ Query Hints -------------------------------------

 Raised error hints:
        inlist_threshol = 2 -> unrecognized configuration parameter "inlist_threshol"
```

The one exception to that are typos and syntax errors:

- When query hints are manually injected, syntax errors will prevent the query from being parsed and will result in an immediate error:

```sql
yellowbrick_test=# EXPLAIN (confidence, hint) SETTING ( LEADING ( ( (A, C)  B) )  ) SELECT * from A INNER JOIN B USING (c) INNER J
ERROR:  syntax error at or near "B"
LINE 1: ... (confidence, hint) SETTING ( LEADING ( ( (A, C)  B) )  ) SE...
```

- For WLM (Workload Management) injection, the system does not terminate queries that could otherwise run. By default, WLM hint syntax errors are logged and ignored. You can change this behavior and adjust the level of messages received by modifying the configuration parameter `query_hint_report_level`.

```sql
yellowbrick_test=# set query_hint_report_level = error;
SET
yellowbrick_test=# EXPLAIN (verbose,  hint) SELECT * from A INNER JOIN B USING (c) INNER JOIN A as C using(c);
ERROR:  Skipping WLM hint "LEADING ( ( (A, C) , B )": syntax error at end of input
```

Important Considerations When Using Plan Hinting

While **Plan Hinting** provides powerful tools to control the execution of queries, it's important to be mindful of a few key considerations to ensure the best performance and maintainability.

1. Balance Manual Control with Automatic Optimizations

The query planner is built to make intelligent decisions based on your data and workload. While forcing specific strategies can offer benefits, in some cases, overriding the planner may result in suboptimal performance. We recommend applying hints only when needed and in areas where you're confident that the forced behavior is better suited to your query.

2. Consider Planning Time

For complex queries with many joins, forcing specific join methods or orders can lead to increased planning time. While this may be a necessary trade-off for improving query execution, it's important to ensure the additional planning time does not outweigh the benefits.

3. Verify Hint Application

Not all hints may be applied, depending on the query and the context. We recommend using `EXPLAIN(hint)` to verify that your hints are recognized and applied as expected. In some cases, the planner may ignore hints that cannot be followed, and `EXPLAIN(verbose, hint)` will help you identify these scenarios.

4. Monitor Changing Workloads

As your data and query patterns evolve, previously applied hints may no longer provide the best results. Periodic review of hint usage ensures that your queries remain optimal over time and minimizes the need for future troubleshooting.

Plan Hinting: JOIN METHOD Hint

Overview

The Join Method hint allows you to specify which join method (Hash Join, Nested Loop Join) should be used in prioriy for the join in between all specified tables. You can also prevent the use of certain join methods.

Supported Methods:

- **HashJoin**
- **NestLoop**
- **NoHashJoin**
- **NoNestLoop**

```sql
<Method> ( <table_name>, <table_name> [, <table_name> ...] )
```

Note: If the requested method would prevent a valid plan from being made, the planner will fall back to any other method that is available, based on its regular costing algorithm.

Examples:

Force a HASH join in between two tables

Here, the planner is instructed to use a Hash Join between `orders` and `customers`.

```sql
yellowbrick=# EXPLAIN SETTING (HASHJOIN (orders, customers))
yellowbrick-# SELECT * FROM orders JOIN customers ON orders.customer_id = customers.id;
                                          QUERY PLAN
-----------------------------------------------------------------------------------------------------------
 id   rows_planned   workers   node
 13             1       all   SELECT
  1             1       all   INNER HASH JOIN ON (customers.id = orders.customer_id)
  4             5       all   |-SCAN orders
                              |   orders.customer_id = bloom(7) AND scan_constraints: min_max(orders.customer_id)
  7             1       all   |-BUILD
 14             1       all     DISTRIBUTE REPLICATE
 10             1       all     SCAN customers
```

Forbid usage of HASH join in between orders, customers and product tables

Let's now also tell the planner to avoid using a Hash Join when joining all three tables `products`, `orders` and `customers`.

```sql
yellowbrick=# EXPLAIN SETTING (NOHASHJOIN (orders, products,  customers); HASHJOIN (orders, customers))
yellowbrick-# SELECT * FROM orders
yellowbrick-# JOIN customers ON orders.customer_id = customers.customer_id
yellowbrick-# JOIN products on orders.order_id = products.id;

                                                QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
 id   rows_planned   workers   node
 20             1       all   SELECT
  1             1       all   INNER LOOP JOIN ON (orders.order_id = products.id)
  4             1       all   |-SCAN products
 22        100000       all   |-DISTRIBUTE REPLICATE
  8        100000       all     INNER HASH JOIN ON (customers.customer_id = orders.customer_id)
 11        100000       all     |-SCAN orders
                                |   orders.customer_id = bloom(14) AND scan_constraints: min_max(orders.customer_id)
 14          1000       all     |-BUILD
 23          1000       all       DISTRIBUTE REPLICATE
 17          1000       all       SCAN customers
```

Plan Hinting: LEADING Hint

Overview

The LEADING hint allows you to specify the order in which tables are joined, and you can also control which tables are on the probe and build sides of the join. You can enforce both the order and the handedness of the join.

There are two forms of the Join Order hint:

1. **Force join order only**:

```sql
LEADING ( <table_name>, <table_name> [, <table_name>...] )
```

2. **Force both order and handedness**:

```sql
LEADING ( (<table_name>, <table_name>) [, ...] )
```

Each set of parenthesis creates a pair representing a join which handedness is set, in the form of `(probe, build)`.

Examples:

Force Join Order Only

In this example, the planner is instructed to first join `orders` with `customers`, and then join the result with `products`.

```sql
yellowbrick=# EXPLAIN SETTING (LEADING (orders, customers, products))
yellowbrick-# SELECT * FROM orders JOIN customers ON orders.customer_id = customers.id JOIN products ON products.id = orders.produ
                                        QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
 id   rows_planned   workers   node
 21             1       all   SELECT
  1             1       all   INNER HASH JOIN ON (orders.product_id = products.id)
  4            10       all   |-SCAN products
                              |   products.id = bloom(7) AND scan_constraints: min_max(products.id)
  7             1       all   |-BUILD
  8             1       all     DISTRIBUTE ON HASH(orders.product_id)
  9             1       all     INNER HASH JOIN ON (customers.id = orders.customer_id)
 12             5       all     |-SCAN orders
                                |   orders.customer_id = bloom(15) AND scan_constraints: min_max(orders.customer_id)
 15             1       all     |-BUILD
 22             1       all       DISTRIBUTE REPLICATE
 18             1       all       SCAN customers
```

But we can also instruct the planner to first join `products` with `orders`, and then join the result with `customers`.

```sql
yellowbrick=# EXPLAIN SETTING (LEADING (products, orders, customers))
yellowbrick-# SELECT * FROM orders JOIN customers ON orders.customer_id = customers.id JOIN products ON products.id = orders.produ
                                                    QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------
 id   rows_planned   workers   node
 20             1       all   SELECT
  1             1       all   INNER HASH JOIN ON (customers.id = orders.customer_id)
  2             5       all   |-INNER HASH JOIN ON (orders.product_id = products.id)
  5            10       all   | |-SCAN products
                              | |   products.id = bloom(8) AND scan_constraints: min_max(products.id)
  8             5       all   | |-BUILD
  9             5       all   |   DISTRIBUTE ON HASH(orders.product_id)
 11             5       all   |   SCAN orders
                              |     orders.customer_id = bloom(14) AND scan_constraints: min_max(orders.customer_id)
 14             1       all   |-BUILD
 21             1       all     DISTRIBUTE REPLICATE
 17             1       all     SCAN customers
```

In both of these, the planner is free to set any table it wants as the probe/build of each join, based on their estimated rowcounts.

Force Join Order and Handedness

Let's now force both order and handedness.

```sql
yellowbrick=# EXPLAIN SETTING (LEADING (((customers, orders), products)) )
yellowbrick-# SELECT * FROM orders JOIN customers ON orders.customer_id = customers.id JOIN products ON products.id = orders.produ
                                                   QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------
 id   rows_planned   workers   node
 20             1       all   SELECT
  1             1       all   INNER HASH JOIN ON (products.id = orders.product_id)
  2             1       all   |-INNER HASH JOIN ON (orders.customer_id = customers.id)
  5             1       all   | |-SCAN customers
                              | |   customers.id = bloom(8) AND scan_constraints: min_max(customers.id)
  8             5       all   | |-BUILD
  9             5       all   |   DISTRIBUTE ON HASH(orders.customer_id)
 11             5       all   |   SCAN orders
                              |     orders.product_id = bloom(14) AND scan_constraints: min_max(orders.product_id)
 14            10       all   |-BUILD
 21            10       all     DISTRIBUTE REPLICATE
 17            10       all     SCAN products
```

In the above plan, the planner had no choice and joined `customers` with `orders` first with `orders` as the build. Then, it joined the result with `products`, using product as build.

Limitations

The LEADING hint needs to specify the join ordering of the whole plan it is working on. Only specifying two tables out of three will result in the hint being ignored. This results in a few limitations.

`join_collapse_limit` and `from_collapse_limit`

The `join_collapse_limit` and `from_collapse_limit` parameters control how the planner treats the join order and subqueries when building query execution plans. In particular:

- `from_collapse_limit` affects whether the planner flattens subqueries in the FROM clause or treats them as separate subqueries.
- `join_collapse_limit` affects the join order optimization, controlling how many join combinations the planner considers when generating a query plan.

When there are more joins in the query than either of these values, one needs one LEADING hint per separated subquery the planner is working with.

If `join_collapse_limit` is set to 8 (its default value)

```sql
yellowbrick=# EXPLAIN SETTING (LEADING (((customers, orders), products)) )
yellowbrick-# SELECT * FROM orders JOIN customers ON orders.customer_id = customers.id JOIN products ON products.id = orders.produ
                                          QUERY PLAN
-----------------------------------------------------------------------------------------------------------------
 id   rows_planned   workers   node
 20             1       all   SELECT
  1             1       all   INNER HASH JOIN ON (products.id = orders.product_id)
  2             1       all   |-INNER HASH JOIN ON (orders.customer_id = customers.id)
  5             1       all   | |-SCAN customers
                              | |   customers.id = bloom(8) AND scan_constraints: min_max(customers.id)
  8             5       all   | |-BUILD
  9             5       all   |   DISTRIBUTE ON HASH(orders.customer_id)
 11             5       all   |   SCAN orders
                              |     orders.product_id = bloom(14) AND scan_constraints: min_max(orders.product_id)
 14            10       all   |-BUILD
 21            10       all     DISTRIBUTE REPLICATE
 17            10       all     SCAN products
```

The three tables are in the same planning group and the hint is correctly applied. Now, let's set `join_collapse_limit` to 2. There are now two planning groups: customer with orders and then the result with products [[customers, orders], products]. If we leave the hint as is, the hint is not used and the planner works as usual:

```sql
yellowbrick=# EXPLAIN SETTING (LEADING (((customers, orders), products)) )
yellowbrick-# SELECT * FROM orders JOIN customers ON orders.customer_id = customers.id JOIN products ON products.id = orders.produ
                                                        QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------
 id   rows_planned   mem_planned   mem_fixed   mem_actual   workers   node
 21             1      8.00Mi N      8.00Mi         0.00       all   SELECT
  1             1        0.00 N        0.00         0.00       all   INNER HASH JOIN ON (orders.product_id = products.id)
  4            10     99.00Mi H     99.00Mi         0.00       all   |-SCAN products
                                                                     |   products.id = bloom(7) AND scan_constraints: min_max(pro
  7             1     45.13Mi N     44.13Mi         0.00       all   |-BUILD
  8             1     18.00Mi H     18.00Mi         0.00       all     DISTRIBUTE ON HASH(orders.product_id)
  9             1        0.00 H        0.00         0.00       all     INNER HASH JOIN ON (customers.id = orders.customer_id)
 12             5     99.00Mi H     99.00Mi         0.00       all     |-SCAN orders
                                                                     |   orders.customer_id = bloom(15) AND scan_constraints: m
 15             1     45.13Mi H     44.13Mi         0.00       all     |-BUILD
 22             1     16.00Mi H     16.00Mi         0.00       all       DISTRIBUTE REPLICATE
 18             1     99.00Mi H     99.00Mi         0.00       all       SCAN customers
Planner configuration parameters
  join_collapse_limit                   2

 ------------------------------------ Query Hints -------------------------------------

Not used hints:
        LEADING( ((customers, orders), products))
```

To force the join in between customer and orders again, you need to make it its own hint:

```sql
yellowbrick=# EXPLAIN(HINT)  SETTING (LEADING ((customers, orders)) )
yellowbrick-# SELECT * FROM orders JOIN customers ON orders.customer_id = customers.id JOIN products ON products.id = orders.produ
                                          QUERY PLAN
-----------------------------------------------------------------------------------------------------------------
 id   rows_planned   workers   node
 21             1       all   SELECT
  1             1       all   INNER HASH JOIN ON (orders.product_id = products.id)
  4            10       all   |-SCAN products
                              |   products.id = bloom(7) AND scan_constraints: min_max(products.id)
  7             1       all   |-BUILD
```

```
 8             1       all     DISTRIBUTE ON HASH(orders.product_id)
 9             1       all     INNER HASH JOIN ON (orders.customer_id = customers.id)
12             1       all     |-SCAN customers
                               |   customers.id = bloom(15) AND scan_constraints: min_max(customers.id)
15             5       all     |-BUILD
16             5       all       DISTRIBUTE ON HASH(orders.customer_id)
18             5       all       SCAN orders
Planner configuration parameters
  join_collapse_limit                   2

------------------------------------ Query Hints -------------------------------------

Used hints:
        LEADING( (customers, orders))
```

Genetic Query Optimization

The Genetic Query Optimization (GEQO) prevents use of the LEADING hint as it will not explore all join combinations.

Plan Hinting: ROWS Hint

Overview

The ROWS hint allows you to adjust the estimated row count at the SCAN, JOIN, or SUBQUERY level. This adjustment affects the cost estimation used by the planner and can be applied with different correction methods.

Correction Methods:

- `+`: Adds to the existing estimate.
- `-`: Subtracts from the existing estimate.
- `#`: Overrides the estimate.
- `*`: Multiplies the estimate.

```sql
ROWS( <table_name> <correction_method> <row_correction> )
```

Examples:

Override row estimate at SCAN level

Assume we have a table t with 5 rows :

```sql
yellowbrick=# EXPLAIN(CONFIDENCE) SELECT * FROM t;
                              QUERY PLAN
-----------------------------------------------------------------------
 id   rows_planned   rows_max_planned   confidence   workers   node
  5             5                 5         High       all   SELECT
  2             5                 5         High       all   SCAN t
```

If one wants to let the planner think the table contains 42k rows instead:

```sql
yellowbrick=# EXPLAIN(CONFIDENCE) SETTING( ROWS( t # 42000) ) SELECT * FROM t;
                              QUERY PLAN
-----------------------------------------------------------------------
 id   rows_planned   rows_max_planned   confidence   workers   node
  5         42000             42000       Hinted       all   SELECT
  2         42000             42000       Hinted       all   SCAN t
```

The estimated row count for the table `t` was changed 42000.

Add to a JOIN row output estimate

Let's now join table `t` with another table `t2`:

```sql
yellowbrick=# EXPLAIN(CONFIDENCE) SELECT * FROM t INNER JOIN t2 USING(i);
                                                QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
 id   rows_planned   rows_max_planned   confidence   workers   node
 13             3                15          Low       all   SELECT
  1             3                15          Low       all   INNER HASH JOIN ON (t2.i = t.i)
  4             5                 5         High       all   |-SCAN t
                                                             |   t.i = bloom(7) AND scan_constraints: min_max(t.i)
  7             3                 3         High       all   |-BUILD
 10             3                 3         High       all     SCAN t2
```

We can use SETTING to add 100 rows to the join output:

```sql
yellowbrick=# EXPLAIN(CONFIDENCE) SETTING (ROWS(t, t2 + 100)) SELECT * FROM t INNER JOIN t2 USING(i);
                                                QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
 id   rows_planned   rows_max_planned   confidence   workers   node
 13           103               103       Hinted       all   SELECT
  1           103               103       Hinted       all   INNER HASH JOIN ON (t2.i = t.i)
  4             5                 5         High       all   |-SCAN t
                                                             |   t.i = bloom(7) AND scan_constraints: min_max(t.i)
  7             3                 3         High       all   |-BUILD
 10             3                 3         High       all     SCAN t2
```

Plan Hinting: SET Hint

Platforms: All platforms

Parent topic: Plan Hinting

Overview

SET hints allow users to set planner related configuration parameters to influence the query planner's decision-making process. The configuration parameter is applied for the duration of the query only.

```sql
<parameter_name> = <parameter_value>
```

Examples

Setting a configuration parameter for a SELECT statement

The following example sets a parameters in the `SETTING` clause.

```sql
premdb=# SETTING (ybd_query_tags='tagged_query')
SELECT * FROM foo ORDER BY 1;
```

Note that the tags that were set in this example are logged in `sys.query` and `sys.log_query`:

```sql
premdb=# SELECT * FROM sys.log_query WHERE tags='tagged_query';
-[ RECORD 1 ]----------------+-----------------------------------------------------------------------------------------------------------------
query_id                     | 327395
session_id                   | 24780
transaction_id               | 0
plan_id                      | 8EjFRXhVcfqyE8H1P+YBUgdcO-XUleSHmHFGw4lrkI4=
state                        | done
username                     | yellowbrick
application_name             | ybsql
database_name                | premdbtype                          | select
tags                         | tagged_query
error_code                   | 00000
error_message                | [NULL]
query_text                   | SETTING (ybd_query_tags='tagged_query') SELECT * FROM foo ORDER BY 1;
...
```

Setting a configuration parameter for an INSERT statement

The following example sets the same parameter for an `INSERT` statement:

```sql
premdb=# SETTING (ybd_query_tags='tagged_query')
INSERT INTO foo SELECT x FROM bar;
```

Setting a configuration parameter for an EXPLAIN statement

The following example sets `ybd_query_tags` for an `EXPLAIN` statement:

```sql
premdb=# EXPLAIN SETTING (ybd_query_tags='COUNT_STAR_QUERY')
SELECT COUNT(*) FROM newmatchstats WHERE seasonid>12;
                                                QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
 id   rows_planned   workers   node
  1             1    single   SELECT
  3             1    single   AGGREGATE SIMPLE
  4            10       all   DISTRIBUTE SINGLE
  5            10       all   AGGREGATE SIMPLE
  6        342000       all   EXPRESSION
                              calculate: (NULL cast)
  9        342000       all   SCAN newmatchstats
                              (newmatchstats.seasonid::INT4 > $1) AND scan_constraints: (newmatchstats.seasonid > $1)
 Database: premdb
 Version: 5.3.0-20210825161549
 Hostname: yb007-mgr1.sjc.yellowbrick.io

(13 rows)
```

In turn, you can see the value that was set for `ybd_query_tags` in the `tags` column of `sys.log_query`:

```sql
premdb=# SELECT * FROM sys.log_query WHERE tags='COUNT_STAR_QUERY';
-[ RECORD 1 ]----------+-----------------------------------------------------------------------------------------------
query_id               | 337496
session_id             | 24780
transaction_id         | 0
plan_id                | 6lgLXILZfBFbqhwrnxWnhLMRxyx-bQfznLd9BPx8wGY=
state                  | done
username               | yellowbrick
application_name       | ybsql
database_name          | premdb
type                   | EXPLAIN
tags                   | COUNT_STAR_QUERY
error_code             | 00000
error_message          | [NULL]
query_text             | EXPLAIN SETTING (ybd_query_tags='COUNT_STAR_QUERY') SELECT COUNT(*) FROM newmatchstats WHERE season
...
```

Improving performance by preventing join flipping

Let's now assume we have two tables :

- `orders` : Contains 1 million rows, mostly with `customer_id = 1` and `order_date = '1997-03-21'`
- `customers` : Contains 1000 rows, with distinct `customer_id` values.

If you want to retrieve a list of all customers along with their recent orders (if any), you might write a query like this:

```sql
SELECT c.customer_id, c.name, o.order_id, o.order_date
FROM customers c LEFT JOIN orders o ON c.customer_id = o.customer_id AND o.order_date >= '2024-01-01';
```

Usually, the planner should start by scanning customers as it is the smaller table and then join it with order and will flip the LEFT into a RIGHT join internally:

```sql
yellowbrick=# EXPLAIN(ANALYZE)
yellowbrick-# SELECT c.customer_id, c.name, o.order_id, o.order_date
yellowbrick-# FROM customers c LEFT JOIN orders o ON c.customer_id = o.customer_id AND o.order_date >= '2024-01-01';
```

```
                                        QUERY PLAN
-----------------------------------------------------------------------------------------------------
 id   rows_planned   rows_actual   workers   node
 13          34133          1000       all   SELECT
  1          34133          1000       all   RIGHT OUTER HASH JOIN ON (c.customer_id = o.customer_id)
  2          33333            10       all   |-DISTRIBUTE ON HASH(o.customer_id)
  4          33333            10       all   | SCAN orders AS o
                                             |   (o.order_date >= $0) AND scan_constraints: (orders.order_date >= $0)
  7           1000          1000       all   |-BUILD
 14           1000          1000       all     EXPRESSION
                                               calculate: (NULL cast)
 10           1000          1000       all     SCAN customers AS c
```

But here, the data is highly skewed and the filter `WHERE o.order_date >= '2024-01-01'` will actually remove almost all rows from the orders table. It is therefore better to force the planner to use a LEFT join by forbidding RIGHT joins. Let's set two parameters : forbidding right joins to improve performances and a tag, so we can easily identify the new plan in `sys.log_query`. Note that they must be separated by a semicolon:

sql

```
yellowbrick=# EXPLAIN(ANALYZE) SETTING(enable_right_joins=false, ybd_query_tags='force_left_join')
yellowbrick-# SELECT c.customer_id, c.name, o.order_id, o.order_date
yellowbrick-# FROM customers c LEFT JOIN orders o ON c.customer_id = o.customer_id AND o.order_date >= '2024-01-01';

                                        QUERY PLAN
-----------------------------------------------------------------------------------------------------
 id   rows_planned   rows_actual   workers   node
 13          33333          1000       all   SELECT
  1          33333          1000       all   LEFT OUTER HASH JOIN ON (o.customer_id = c.customer_id)
  4           1000          1000       all   |-SCAN customers AS c
  7             10            10       all   |-BUILD
 14             10            10       all     EXPRESSION
                                               calculate: (NULL cast)
  8          33333            10        all    DISTRIBUTE ON HASH(o.customer_id)
 10          33333            10        all    SCAN orders AS o
                                                 (o.order_date >= $0) AND scan_constraints: (orders.order_date >= $0)

Planner configuration parameters
   enable_right_joins                    off
```

SELECT

In this section:

Write a query that returns data from tables and views.

Syntax

```
[ SETTING (hint [; ...])]
[ WITH name as (subquery) [, ...] ]
SELECT [ ALL | DISTINCT expression [ [ AS ] output_name ] [, ...] ]
   [ * | expression [ [ AS ] output_name ] [, ...] ]
   [ FROM table_reference [, ...] ]
   [ WHERE condition ]
   [ GROUP BY { expression [, ...] ]|
     GROUPING SETS (expression [, ...]) |
     ROLLUP (expression [, ...]) |
     CUBE (expression [, ...]) } ]
   [ HAVING condition ]
   [ WINDOW window_name AS ( window_definition ) [, ...] ]
   [ { UNION | INTERSECT | { EXCEPT | MINUS } } [ ALL | DISTINCT ] select ]
   [ ORDER BY expression [ ASC | DESC ] [ NULLS { FIRST | LAST } ] [, ...] [ COLLATE collation ] ]
   [ LIMIT { count | ALL } ]
   [ OFFSET start [ ROW | ROWS ] ]
```

FROM Clause

Yellowbrick Documentation > Reference > SQL Reference > SQL Commands > SELECT > FROM Clause

Platforms: All platforms

Parent topic: SELECT

In this section:

TABLESAMPLE

Use the FROM clause to specify one or more table references, such as table names, view names, and tables derived from subqueries. Join tables in either the FROM clause or the WHERE clause. See also Subquery Examples.

```
FROM
table_reference [ * ] [ [ AS ] table_alias [ ( column_alias [, ...] ) ] ]
[ join_type table_reference ON join_condition | USING ( join_column [, ...] ) ]
[, ...]
```

Parameters

table_reference

Name a persistent or temporary table, a view, a table derived from a subquery, or a `SETOF` stored procedure.

Regular table references may consist of one, two, or three parts, separated by periods:

- Database name
- Schema name
- Table name

For example: `premdb.public.team`.

Fully qualified (three-part) table names are required for remote table references in cross-database queries. However, note that you must `SELECT` from (or `CALL`) stored procedures from the database where the procedures were created.

You cannot combine user-defined tables and system tables or system views in the same query. One exception to this rule is the sys.const table, which can be queried in conjunction with regular tables.

table_alias

Give an alias to a table. The alias serves as the table name for the duration of the query.

column_alias

Give an alias to the columns in a table. The aliases serve as the table's column names for the duration of the query.

join_type

Use one of the following join types. The INNER and OUTER keywords are optional.

- INNER JOIN or JOIN
- FULL OUTER JOIN or FULL JOIN
- LEFT OUTER JOIN or LEFT JOIN

- RIGHT OUTER JOIN or RIGHT JOIN
- CROSS JOIN

An inner join returns matching rows, based on a FROM clause or WHERE clause join condition.

An outer join must be specified in the FROM clause. An outer join returns the same set of rows that an inner join would return, but it also returns non-matching rows from the "left" table (left outer join), the "right" table (right outer join), or both tables (full outer join). In this context, the table that is listed first in the FROM clause is the left table, and the table listed second is the right table. The output columns that contain NULLs indicate the non-matching rows that the outer join returns.

A cross join returns the Cartesian product of the rows in both tables. It is an unqualified join without a join condition. The `CROSS JOIN` syntax is equivalent to writing simply `FROM table1, table2`. You can use a `WHERE` clause condition with a cross join if required, but the `ON` syntax is not supported.

ON join_condition

Express each inner or outer join in terms of a search condition, such as an equality condition:

```
hometeam join team on hometeam.htid=team.htid
```

Inequality conditions are supported for inner joins, left and right outer joins, and cross joins. Full outer joins with inequality conditions are not supported. For example, the following join is supported:

```
hometeam left outer join awayteam on hometeam.htid>awayteam.atid
```

An `ON` join retains both joining columns in its intermediate result set. Parentheses are not required for the join condition but may be used for clarity.

USING (join_column)

Express an inner or outer join in terms of a list of one or more columns that exist in both tables. Parentheses are required. For example:

```
hometeam join team using(htid)
```

The implicit join condition is an equality condition. A `USING` join retains one of the joining columns in its intermediate result set, not both.

Examples

Join two tables with `USING` syntax:

```
premdb=# select teamid, htid, atid, team.name, nickname, city, stadium, capacity
from hometeam join team using(htid)
where teamid between 20 and 29
order by teamid, htid, atid;
 teamid | htid | atid |       name        |  nickname  |     city      |      stadium       | capacity
--------+------+------+-------------------+------------+---------------+--------------------+----------
     20 |   21 |   70 | Ipswich Town      | Blues      | Ipswich       | Portman Road       |    30311
     21 |   22 |   71 | Leeds United      | Whites     | Leeds         | Elland Road        |    39460
     22 |   23 |   72 | Leicester City    | Foxes      | Leicester     | King Power Stadium |    32262
     23 |   24 |   73 | Liverpool         | Reds       | Liverpool     | Anfield            |    44742
     24 |   25 |   74 | Manchester City   | Citizens   | Manchester    | Etihad Stadium     |    55097
     25 |   26 |   75 | Manchester United | Red Devils | Manchester    | Old Trafford       |    75635
     26 |   27 |   76 | Middlesbrough     | Boro       | Middlesbrough | Riverside Stadium  |    34742
     27 |   28 |   77 | Newcastle United  | Magpies    | Newcastle     | St. James Park     |    52405
     28 |   29 |   78 | Norwich City      | Canaries   | Norwich       | Carrow Road        |    27244
     29 |   30 |   79 | Nottingham Forest | Forest     | Nottingham    | City Ground        |    30445
(10 rows)
```

Here is the same query with `JOIN ON` syntax:

```
premdb=# select teamid, team.htid, atid, team.name, nickname, city, stadium, capacity
from hometeam join team on hometeam.htid=team.htid
where teamid between 20 and 29
order by teamid, htid, atid;
```

The following query uses a full outer join to return matching and non-matching rows in the `hometeam` and `awayteam` tables:

```
premdb=# select ht.htid, at.atid, ht.name
from hometeam ht full outer join awayteam at on ht.name=at.name
order by ht.name nulls first;
 htid | atid |          name
------+------+------------------------
      |   98 |
      |   99 |
      |  100 |
   51 |      |
   50 |      |
   49 |      |
   2 |   51 | Arsenal
   3 |   52 | Aston Villa
   4 |   53 | Barnsley
   5 |   54 | Birmingham City
   6 |   55 | Blackburn Rovers
   7 |   56 | Blackpool
...
```

The equivalent left outer join query returns fewer rows:

```
premdb=# select ht.htid, at.atid, ht.name
from hometeam ht left outer join awayteam at on ht.name=at.name
order by ht.name nulls first;
 htid | atid |          name
------+------+------------------------
   51 |      |
   50 |      |
   49 |      |
   2 |   51 | Arsenal
   3 |   52 | Aston Villa
   4 |   53 | Barnsley
   5 |   54 | Birmingham City
   6 |   55 | Blackburn Rovers
   7 |   56 | Blackpool
...
```

Return the unqualified cross-join results from two tables:

```
premdb=# select * from awayteam cross join hometeam;
 atid |          name          | htid |          name
------+------------------------+------+------------------------
   51 | Arsenal                |    2 | Arsenal
   51 | Arsenal                |    3 | Aston Villa
   51 | Arsenal                |    4 | Barnsley
   51 | Arsenal                |    5 | Birmingham City
   51 | Arsenal                |    6 | Blackburn Rovers
...
```

TABLESAMPLE

> Prerequisite: while the feature is in beta, you must explicitly enable it by setting `enable_full_samplescan` to `ON`.

Use the `TABLESAMPLE` clause to retrieve a random sample of rows from a table. This clause is typically appended to a `FROM` clause and is useful for quickly analyzing a representative subset of a large data set.

```sql
FROM <table_name> [ AS ] <alias> TABLESAMPLE <sampling_method> ( <sample_percentage> ) [ REPEATABLE ( seed ) ]
```

Parameters

table_name

Name of the table from which to sample rows.

sampling_method

The method used to perform the sampling. Supported methods:

- **BERNOULLI** — The only supported sampling method. Performs sampling at the row level, evaluating each row independently and including it in the sample with the specified probability.
- **SYSTEM** — Currently this is just an alias for using BERNOULLI.

sample_percentage

The approximate percentage of the table's rows to return. Must be a numeric literal between 0 and 100. For example, `TABLESAMPLE BERNOULLI (10)` returns approximately 10% of the table.

REPEATABLE (seed)

Optional clause to make the sampling deterministic. If specified, the same seed value will always return the same sampled rows (provided that the underlying data hasn't changed). When omitted, the seed is set randomly using a random number generator. The RNG can be seeded by the `setseed` function, which is another way to achieve determinism.

Notes

- `TABLESAMPLE` returns an *approximate* number of rows. There is no guarantee that the exact percentage specified will be returned.
- The clause only supports sampling entire tables, not derived tables or views.
- The reproducibility of samples is also influenced by the `setseed()` function (which sets the random seed for the session). However, the seed provided in the `REPEATABLE` clause takes precedence over any value set with `setseed()`.

Examples

Sample approximately 1% of the rows from the `team` table using the `BERNOULLI` method with the seed set to 2:

```sql
CREATE TABLE t AS SELECT * FROM sys.rowgenerator LIMIT 1000000;
SELECT COUNT(*) FROM t TABLESAMPLE BERNOULLI (1) REPEATABLE (2);
```

```
 count
-------
 10127
(1 row)
```

GROUP BY Clause

Platforms: All platforms

Parent topic: SELECT

In this section:

Use the GROUP BY clause in conjunction with aggregate functions in the SELECT list. Name the columns that are grouped in order to calculate aggregate values such as counts, averages, and sums. You can also specify GROUP BY extensions: GROUPING SETS, ROLLUP, and CUBE.

```
[ GROUP BY { expression [, ...] |
GROUPING SETS (expression [, ...]) |
ROLLUP (expression [, ...]) |
CUBE (expression [, ...])
} ]
```

When an aggregate function is used in the SELECT list, all non-aggregate columns or expressions in the SELECT list must be specified as grouping columns in the GROUP BY clause.

Grouping columns can be specified with their original names, SELECT list aliases, or ordinal numbers. You cannot use parameters defined in prepared statements, and you cannot use the result of a subquery as a grouping column. Subqueries in the GROUP BY clause return a syntax error.

Examples

Use the COUNT function to find out how many times each team has won the league:

```
premdb=# select winners, count(winners)
from season
where winners is not null
group by winners;
     winners      | count
------------------+-------
 Manchester City  |     2
 Arsenal          |     3
 Chelsea          |     4
 Blackburn Rovers |     1
 Leicester City   |     1
 Manchester United |    13
(6 rows)
```

You can also use an ordinal number to specify a grouping column:

```
premdb=# select winners, count(winners) from season
where winners is not null
group by 1;
     winners      | count
------------------+-------
 Manchester City  |     2
 Arsenal          |     3
```

```
 Chelsea           |     4
 Blackburn Rovers  |     1
 Leicester City    |     1
 Manchester United |    13
(6 rows)
```

You can also use a SELECT list alias to specify a grouping column:

```
premdb=# select winners team_name, count(winners) timeswon
from season
where winners is not null
group by team_name;
    team_name     | timeswon
------------------+----------
 Manchester City   |        2
 Arsenal           |        3
 Chelsea           |        4
 Blackburn Rovers  |        1
 Leicester City    |        1
 Manchester United |       13
(6 rows)
```

Note that the WHERE clause *does not* accept SELECT list aliases.

GROUP BY CUBE

Platforms: All platforms

Parent topic: GROUP BY Clause

Return aggregated values for the given list of grouping sets and all possible subsets of that list.

```
GROUP BY CUBE (expression [, ...])
```

`GROUP BY CUBE` is a shorthand form of an exhaustive `GROUPING SETS` list. For example, the following clause:

```
CUBE(expr1, expr2, expr3)
```

is evaluated in the same way as:

```
GROUPING SETS(
(expr1, expr2, expr3),
(expr1, expr2),
(expr1, expr3),
(expr1),
(expr2, expr3),
(expr2),
(expr3),
()
)
```

Note that this list includes the empty set `()`.

Duplicate grouping sets are not allowed.

`GROUP BY CUBE` queries are limited to a maximum of 7 columns.

Examples

The examples in this section are based on the following set of rows in the `player` table:

```
premdb=# select * from player;
 playerid | teamid | seasonid | firstname | lastname  | position |    dob     | weekly_wages | avg_mins_per_match | matches_played
----------+--------+----------+-----------+-----------+----------+------------+--------------+--------------------+---------------
       1 |      2 |       27 | NULL      | NULL      | NULL     | NULL       |         NULL |               NULL |          NULL
       2 |     41 |       27 | Harry     | Kane      | F        | 1993-07-28 |       250000 |            84.1567 |          36.1
       3 |     41 |       27 | Harry     | Winks     | M        | 1996-02-02 |        40000 |            72.3412 |          31.2
       4 |     24 |       27 | Kevin     | De Bruyne | M        | 1991-06-28 |       418000 |            86.0981 |          35.3
       5 |     25 |       27 | Paul      | Pogba     | M        | 1993-03-15 |       378000 |            76.5341 |          29.4
       6 |     25 |       27 | Alexis    | Sanchez   | F        | 1988-12-19 |       410000 |            71.2309 |          31.5
       7 |     13 |       27 | Gonzalo   | Higuain   | F        | 1987-12-10 |       352000 |            61.9803 |          26.6
       8 |      1 |       27 | Mesut     | Ozil      | M        | 1988-10-15 |       350000 |            64.4561 |          32.7
       9 |     13 |       27 | Eden      | Hazard    | M        | 1991-01-07 |       200000 |            81.2387 |          36.8
      10 |     24 |       27 | Sergio    | Aguero    | F        | 1988-06-02 |       220000 |            80.0032 |          33.9
      11 |     25 |       27 | Romelu    | Lukaku    | M        | 1993-05-13 |       250000 |            85.1897 |            34
```

```
      12 |     25 |       27 | David     | de Gea    | G        | 1990-11-07 |       200000 |            88.5764 |           35.1
      13 |     23 |       27 | Mo        | Salah     | F        | 1992-06-15 |       200000 |            82.2765 |           36.2
      14 |     24 |       27 | Riyad     | Mahrez    | M        | 1991-02-21 |       200000 |            55.1908 |           24.3
      15 |     23 |       27 | Virgil    | Van Dijk  | D        | 1991-07-08 |       180000 |            87.6345 |           37.4
      16 |     24 |       27 | David     | Silva     | M        | 1986-01-08 |       250000 |            79.6723 |           30.5
      17 |     22 |       27 | Jamie     | Vardy     | F        | 1987-01-11 |        80000 |            77.1986 |           31.6
      18 |     41 |       27 | Dele      | Alli      | M        | 1996-04-11 |       100000 |            75.9013 |           30.7
      19 |     41 |       27 | Marcus    | Rashford  | F        | 1997-10-31 |       250000 |            79.9912 |           32.8
      20 |     47 |       27 | NULL      | NULL      | NULL     | NULL       |         NULL |               NULL |           NULL
(20 rows)
```

This example returns aggregated wages over all combinations of two columns, `position` and `cob`:

```
 position |     cob     |   sum
----------+-------------+---------
 D        | Netherlands |  180000
 D        | [NULL]      |  180000
 F        | Argentina   |  572000
 F        | Chile       |  410000
 F        | Egypt       |  200000
 F        | England     |  580000
 F        | [NULL]      | 1762000
 G        | Spain       |  200000
 G        | [NULL]      |  200000
 M        | Algeria     |  200000
 M        | Belgium     |  250000
 M        | England     |  140000
 M        | France      |  378000
 M        | Germany     |  350000
 M        | Netherlands |  618000
 M        | Spain       |  250000
 M        | [NULL]      | 2186000
 [NULL]   | Algeria     |  200000
 [NULL]   | Argentina   |  572000
 [NULL]   | Belgium     |  250000
 [NULL]   | Chile       |  410000
 [NULL]   | Egypt       |  200000
 [NULL]   | England     |  720000
 [NULL]   | France      |  378000
 [NULL]   | Germany     |  350000
 [NULL]   | Netherlands |  798000
 [NULL]   | Spain       |  450000
 [NULL]   | [NULL]      | 4328000
 [NULL]   | [NULL]      |  [NULL]
 [NULL]   | [NULL]      |  [NULL]
 [NULL]   | [NULL]      |  [NULL]
(31 rows)
```

The following example groups by the cube of three grouping sets, showing how cubed results grow exponentially when multiple expressions are used in the `GROUP BY CUBE` clause. Note the inclusion of subtotal rows (rows with two `NULL` values) and a grand total row (the last row, with three `NULL` values). There are no completely `NULL` rows in this case because of the `WHERE` clause constraint.

```
premdb=# select position, weekly_wages wages, cob country, sum(matches_played) matches
from player
where wages is not null
group by cube(position, wages, country)
order by 1, 2, 3;
position | wages  |   country   | matches
----------+--------+-------------+---------
 D        | 180000 | Netherlands |    37.4
 D        | 180000 | [NULL]      |    37.4
 D        | [NULL] | Netherlands |    37.4
 D        | [NULL] | [NULL]      |    37.4
```

```
F        |  80000 | England     |    31.6
F        |  80000 | [NULL]      |    31.6
F        | 200000 | Egypt       |    36.2
F        | 200000 | [NULL]      |    36.2
F        | 220000 | Argentina   |    33.9
F        | 220000 | [NULL]      |    33.9
F        | 250000 | England     |    68.9
F        | 250000 | [NULL]      |    68.9
F        | 352000 | Argentina   |    26.6
F        | 352000 | [NULL]      |    26.6
F        | 410000 | Chile       |    31.5
F        | 410000 | [NULL]      |    31.5
F        | [NULL] | Argentina   |    60.5
F        | [NULL] | Chile       |    31.5
F        | [NULL] | Egypt       |    36.2
F        | [NULL] | England     |   100.5
F        | [NULL] | [NULL]      |   228.7
G        | 200000 | Spain       |    35.1
G        | 200000 | [NULL]      |    35.1
G        | [NULL] | Spain       |    35.1
G        | [NULL] | [NULL]      |    35.1
M        |  40000 | England     |    31.2
M        |  40000 | [NULL]      |    31.2
M        | 100000 | England     |    30.7
M        | 100000 | [NULL]      |    30.7
M        | 200000 | Algeria     |    24.3
M        | 200000 | Netherlands |    36.8
M        | 200000 | [NULL]      |    61.1
M        | 250000 | Belgium     |      34
M        | 250000 | Spain       |    30.5
M        | 250000 | [NULL]      |    64.5
M        | 350000 | Germany     |    32.7
M        | 350000 | [NULL]      |    32.7
M        | 378000 | France      |    29.4
M        | 378000 | [NULL]      |    29.4
M        | 418000 | Netherlands |    35.3
M        | 418000 | [NULL]      |    35.3
M        | [NULL] | Algeria     |    24.3
M        | [NULL] | Belgium     |      34
M        | [NULL] | England     |    61.9
M        | [NULL] | France      |    29.4
M        | [NULL] | Germany     |    32.7
M        | [NULL] | Netherlands |    72.1
M        | [NULL] | Spain       |    30.5
M        | [NULL] | [NULL]      |   284.9
[NULL]   |  40000 | England     |    31.2
[NULL]   |  40000 | [NULL]      |    31.2
[NULL]   |  80000 | England     |    31.6
[NULL]   |  80000 | [NULL]      |    31.6
[NULL]   | 100000 | England     |    30.7
[NULL]   | 100000 | [NULL]      |    30.7
[NULL]   | 180000 | Netherlands |    37.4
[NULL]   | 180000 | [NULL]      |    37.4
[NULL]   | 200000 | Algeria     |    24.3
[NULL]   | 200000 | Egypt       |    36.2
[NULL]   | 200000 | Netherlands |    36.8
[NULL]   | 200000 | Spain       |    35.1
[NULL]   | 200000 | [NULL]      |   132.4
[NULL]   | 220000 | Argentina   |    33.9
[NULL]   | 220000 | [NULL]      |    33.9
[NULL]   | 250000 | Belgium     |      34
[NULL]   | 250000 | England     |    68.9
[NULL]   | 250000 | Spain       |    30.5
[NULL]   | 250000 | [NULL]      |   133.4
[NULL]   | 350000 | Germany     |    32.7
[NULL]   | 350000 | [NULL]      |    32.7
[NULL]   | 352000 | Argentina   |    26.6
[NULL]   | 352000 | [NULL]      |    26.6
```

```
  [NULL]   | 378000 | France      |    29.4
  [NULL]   | 378000 | [NULL]      |    29.4
  [NULL]   | 410000 | Chile       |    31.5
  [NULL]   | 410000 | [NULL]      |    31.5
  [NULL]   | 418000 | Netherlands |    35.3
  [NULL]   | 418000 | [NULL]      |    35.3
  [NULL]   | [NULL] | Algeria     |    24.3
  [NULL]   | [NULL] | Argentina   |    60.5
  [NULL]   | [NULL] | Belgium     |      34
  [NULL]   | [NULL] | Chile       |    31.5
  [NULL]   | [NULL] | Egypt       |    36.2
  [NULL]   | [NULL] | England     |   162.4
  [NULL]   | [NULL] | France      |    29.4
  [NULL]   | [NULL] | Germany     |    32.7
  [NULL]   | [NULL] | Netherlands |   109.5
  [NULL]   | [NULL] | Spain       |    65.6
  [NULL]   | [NULL] | [NULL]      |   586.1
 (89 rows)
```

GROUP BY GROUPING SETS

Platforms: All platforms

Parent topic: GROUP BY Clause

Name zero or more `GROUP BY` expressions as grouping sets. Aggregates are computed for each discrete group.

```
GROUP BY GROUPING SETS (expression [, ...])
```

Within the outer pair of parentheses, expressions may be either named in a simple comma-separated list or isolated within their own inner pairs of parentheses. For example, `grouping sets((city, state))` and `grouping sets((city), (state))` are both valid; however, these constructs return different results:

- `grouping sets((city, state))` means group by `city` *and* `state` in one set of rows.
- `grouping sets((city), (state))`, or `grouping sets(city, state)`, means group by `city` in a set of rows, then group by state in a separate set of rows.

You may want to use inner sets of parentheses consistently for clarity.

Usage Notes

- Duplicate grouping sets are not allowed.
- Queries are limited to a maximum of 128 grouping sets.
- The maximum length of each grouping set is 200 elements.
- `GROUP BY CUBE` queries are limited to a maximum of 7 columns.
- If the target table is empty, a grouping set query returns 0 rows.
- Aliases defined in the `SELECT` list may be referenced in the `GROUPING SETS` clause. For example:

```
select c1 as a, c2 as b, ...
from ...
group by grouping sets((a), (b))
...
```

Ordinal numbers that reference `GROUP BY` expressions are also valid. For example:

```
select city, state, ...
from ...
group by grouping sets((1), (2))
...
```

Examples

Assume that you have a `citypop` table that stores population numbers by city and state, and that the table contains five rows:

```
testdb=# select * from citypop order by 2,3;
   city     | state |   pop
------------+-------+----------
```

```
 San Jose    | CA    |  1799000
 Los Angeles | CA    | 12459000
 Naples      | FL    |   398000
 Miami       | FL    |  6167000
 Boise       | ID    |   455000
(5 rows)
```

Note the results of the following two queries. The first one defines `(city, state)` as a single unit of aggregation:

```
testdb=# select city, state, sum(pop) from citypop
group by grouping sets((city, state)) order by 1;
   city     | state |   sum
------------+-------+----------
 Boise      | ID    |   455000
 Los Angeles | CA    | 12459000
 Miami      | FL    |  6167000
 Naples     | FL    |   398000
 San Jose   | CA    |  1799000
(5 rows)
```

The query returns only five rows, with one sum for each different city/state group. This is the exact same result that you would get with a simple `group by city, state` clause. The use of `GROUPING SETS` in this particular case does not add any value but is a valid alternative to the basic `GROUP BY` syntax.

The second query defines `city` and `state` as distinct groups for aggregation within the same result set:

```
testdb=# select city, state, sum(pop) from citypop group by grouping sets((city), (state)) order by 1,2,3;
   city     | state |   sum
------------+-------+----------
 Boise      | NULL  |   455000
 Los Angeles | NULL  | 12459000
 Miami      | NULL  |  6167000
 Naples     | NULL  |   398000
 San Jose   | NULL  |  1799000
 NULL       | FL    |  6565000
 NULL       | CA    | 14258000
 NULL       | ID    |   455000
(8 rows)
```

This query returns eight rows: five rows for each different city, plus three rows for each different state. In this case, the use of `GROUPING SETS` provides a simple way to return results that have multiple levels of aggregation within a single query. `NULL` values are used to complete the empty columns.

To take this example one step further, you can also produce a *total* row for the query by specifying `()` as a third, but empty grouping set:

```
testdb=# select city, state, sum(pop) from citypop group by grouping sets((city), (state), ()) order by 1,3;
   city     | state |   sum
------------+-------+----------
 Boise      | NULL  |   455000
 Los Angeles | NULL  | 12459000
 Miami      | NULL  |  6167000
 Naples     | NULL  |   398000
 San Jose   | NULL  |  1799000
 NULL       | ID    |   455000
 NULL       | FL    |  6565000
 NULL       | CA    | 14258000
 NULL       | NULL  | 21278000
(9 rows)
```

The ninth row is the grand total for the preceding sets of rows. An empty grouping set means that all rows are aggregated in a single group. This total row is produced even when no input rows exist.

Throughout these examples, note that `NULL` values occupy the empty columns when there is no match for a particular grouping column. Total rows contain `NULL` values for all grouping columns.

NULL Handling with Grouping Sets

When you are running queries that group by grouping sets, you have two types of `NULL` values to think about: those that exist in the table data itself, and those produced in aggregated rows.

When `NULL` values exist in the data for grouping set columns and expressions, aggregated rows return `NULL` values in the result. For example, the 14th row in the following query result represents the group where both `position` and `cob` are `NULL`. There are two such rows in the `player` table. In both of those rows, `weekly_wages` is also `NULL`. Therefore the 14th row is a series of three `NULL` values.

```
yellowbrick=# select position, cob, sum(weekly_wages)
from player
group by grouping sets((position, cob)) order by 1,2,3;
 position |     cob     |  sum
----------+-------------+--------
 D        | Netherlands | 180000
 F        | Argentina   | 572000
 F        | Chile       | 410000
 F        | Egypt       | 200000
 F        | England     | 580000
 G        | Spain       | 200000
 M        | Algeria     | 200000
 M        | Belgium     | 250000
 M        | England     | 140000
 M        | France      | 378000
 M        | Germany     | 350000
 M        | Netherlands | 618000
 M        | Spain       | 250000
 [NULL]   | [NULL]      | [NULL]
(14 rows)
```

If you do not want this kind of row to be evaluated or displayed, filter out `NULL` values in the `WHERE` clause.

For grouping sets queries, you will see `NULL` values used in two other ways:

- To complete the blanks for subtotal and grand total rows. Total rows contain `NULL` values for all grouping columns.
- To complete the blanks when grouped aggregation occurs at different levels. `NULL` values occupy the empty columns when there is no match for a particular grouping column.

If you did not intend to return these aggregated rows, you may need to rewrite the query. If you want to return these rows only under certain conditions, you can apply a filter in the `HAVING` clause.

GROUP BY ROLLUP

Return aggregated results over the specified list of grouping sets and rolled up sets from that list.

```
GROUP BY ROLLUP (expression [, ...])
```

Like `GROUP BY CUBE`, `GROUP BY ROLLUP` is a convenient shorthand for a more detailed `GROUPING SETS` construct.

For example, the following clause:

```
ROLLUP(expr1, expr2, expr3)
```

is evaluated in the same way as:

```
GROUPING SETS(
(expr1, expr2, expr3),
(expr1, expr2),
(expr1),
()
)
```

`GROUP BY ROLLUP` is useful for analyzing data that has a hierarchical structure, such as total population by city, county, state, and country, or goals scored by match, competition, season, and club career.

Duplicate grouping sets are not allowed.

Examples

The examples in this section are based on the following set of rows in the `player` table:

```
premdb=# select * from player;
 playerid | teamid | seasonid | firstname | lastname  | position |    dob     | weekly_wages | avg_mins_per_match | matches_played
----------+--------+----------+-----------+-----------+----------+------------+--------------+--------------------+---------------
       1 |      2 |       27 | NULL      | NULL      | NULL     | NULL       |         NULL |               NULL |          NULL
       2 |     41 |       27 | Harry     | Kane      | F        | 1993-07-28 |       250000 |            84.1567 |          36.1
       3 |     41 |       27 | Harry     | Winks     | M        | 1996-02-02 |        40000 |            72.3412 |          31.2
       4 |     24 |       27 | Kevin     | De Bruyne | M        | 1991-06-28 |       418000 |            86.0981 |          35.3
       5 |     25 |       27 | Paul      | Pogba     | M        | 1993-03-15 |       378000 |            76.5341 |          29.4
       6 |     25 |       27 | Alexis    | Sanchez   | F        | 1988-12-19 |       410000 |            71.2309 |          31.5
       7 |     13 |       27 | Gonzalo   | Higuain   | F        | 1987-12-10 |       352000 |            61.9803 |          26.6
       8 |      1 |       27 | Mesut     | Ozil      | M        | 1988-10-15 |       350000 |            64.4561 |          32.7
       9 |     13 |       27 | Eden      | Hazard    | M        | 1991-01-07 |       200000 |            81.2387 |          36.8
      10 |     24 |       27 | Sergio    | Aguero    | F        | 1988-06-02 |       220000 |            80.0032 |          33.9
      11 |     25 |       27 | Romelu    | Lukaku    | M        | 1993-05-13 |       250000 |            85.1897 |            34
      12 |     25 |       27 | David     | de Gea    | G        | 1990-11-07 |       200000 |            88.5764 |          35.1
      13 |     23 |       27 | Mo        | Salah     | F        | 1992-06-15 |       200000 |            82.2765 |          36.2
      14 |     24 |       27 | Riyad     | Mahrez    | M        | 1991-02-21 |       200000 |            55.1908 |          24.3
```

```
      15 |     23 |       27 | Virgil    | Van Dijk  | D        | 1991-07-08 |       180000 |            87.6345 |           37.4
      16 |     24 |       27 | David     | Silva     | M        | 1986-01-08 |       250000 |            79.6723 |           30.5
      17 |     22 |       27 | Jamie     | Vardy     | F        | 1987-01-11 |        80000 |            77.1986 |           31.6
      18 |     41 |       27 | Dele      | Alli      | M        | 1996-04-11 |       100000 |            75.9013 |           30.7
      19 |     41 |       27 | Marcus    | Rashford  | F        | 1997-10-31 |       250000 |            79.9912 |           32.8
      20 |     47 |       27 | NULL      | NULL      | NULL     | NULL       |         NULL |               NULL |           NULL
(20 rows)
```

Note that this table contains stored `NULL` values in two of its rows.

Use `GROUP BY ROLLUP` over the `position` and `cob` columns, with a `SUM` on `weekly_wages`:

```
premdb=# select position, cob, sum(weekly_wages)
from player
group by rollup(position, cob)
order by 1, 2, 3;
 position |     cob     |   sum
----------+-------------+---------
 D        | Netherlands |  180000
 D        | NULL        |  180000
 F        | Argentina   |  572000
 F        | Chile       |  410000
 F        | Egypt       |  200000
 F        | England     |  580000
 F        | NULL        | 1762000
 G        | Spain       |  200000
 G        | NULL        |  200000
 M        | Algeria     |  200000
 M        | Belgium     |  250000
 M        | England     |  140000
 M        | France      |  378000
 M        | Germany     |  350000
 M        | Netherlands |  618000
 M        | Spain       |  250000
 M        | NULL        | 2186000
 NULL     | NULL        | 4328000
 NULL     | NULL        |    NULL
 NULL     | NULL        |    NULL
(20 rows)
```

This example rolls up over three different columns:

```
premdb=# select position, weekly_wages wages, cob country, sum(matches_played) matches
from player
group by rollup(position, wages, country)
order by 1, 2, 3;
 position | wages  |   country   | matches
----------+--------+-------------+---------
 D        | 180000 | Netherlands |    37.4
 D        | 180000 | [NULL]      |    37.4
 D        | [NULL] | [NULL]      |    37.4
 F        |  80000 | England     |    31.6
 F        |  80000 | [NULL]      |    31.6
 F        | 200000 | Egypt       |    36.2
 F        | 200000 | [NULL]      |    36.2
 F        | 220000 | Argentina   |    33.9
 F        | 220000 | [NULL]      |    33.9
 F        | 250000 | England     |    68.9
 F        | 250000 | [NULL]      |    68.9
 F        | 352000 | Argentina   |    26.6
 F        | 352000 | [NULL]      |    26.6
 F        | 410000 | Chile       |    31.5
 F        | 410000 | [NULL]      |    31.5
 F        | [NULL] | [NULL]      |   228.7
```

```
 G        | 200000 | Spain       |    35.1
 G        | 200000 | [NULL]      |    35.1
 G        | [NULL] | [NULL]      |    35.1
 M        |  40000 | England     |    31.2
 M        |  40000 | [NULL]      |    31.2
 M        | 100000 | England     |    30.7
 M        | 100000 | [NULL]      |    30.7
 M        | 200000 | Algeria     |    24.3
 M        | 200000 | Netherlands |    36.8
 M        | 200000 | [NULL]      |    61.1
 M        | 250000 | Belgium     |      34
 M        | 250000 | Spain       |    30.5
 M        | 250000 | [NULL]      |    64.5
 M        | 350000 | Germany     |    32.7
 M        | 350000 | [NULL]      |    32.7
 M        | 378000 | France      |    29.4
 M        | 378000 | [NULL]      |    29.4
 M        | 418000 | Netherlands |    35.3
 M        | 418000 | [NULL]      |    35.3
 M        | [NULL] | [NULL]      |   284.9
[NULL]    | [NULL] | [NULL]      |  [NULL]
[NULL]    | [NULL] | [NULL]      |   586.1
[NULL]    | [NULL] | [NULL]      |  [NULL]
[NULL]    | [NULL] | [NULL]      |  [NULL]
(40 rows)
```

A similar example rolls up by team, position, and country, and produces the average weekly wage (cast as an integer) for each set:

```
premdb=# select teamid team, position, cob country, avg(weekly_wages)::int matches
from player
group by rollup(team, position, country)
order by 1, 2, 3, 4;
  team  | position |   country   | matches
--------+----------+-------------+---------
     1 | M        | Germany     |  350000
     1 | M        | [NULL]      |  350000
     1 | [NULL]   | [NULL]      |  350000
     2 | [NULL]   | [NULL]      |  [NULL]
     2 | [NULL]   | [NULL]      |  [NULL]
     2 | [NULL]   | [NULL]      |  [NULL]
    13 | F        | Argentina   |  352000
    13 | F        | [NULL]      |  352000
    13 | M        | Netherlands |  200000
    13 | M        | [NULL]      |  200000
    13 | [NULL]   | [NULL]      |  276000
    22 | F        | England     |   80000
    22 | F        | [NULL]      |   80000
    22 | [NULL]   | [NULL]      |   80000
    23 | D        | Netherlands |  180000
    23 | D        | [NULL]      |  180000
    23 | F        | Egypt       |  200000
    23 | F        | [NULL]      |  200000
    23 | [NULL]   | [NULL]      |  190000
    24 | F        | Argentina   |  220000
    24 | F        | [NULL]      |  220000
    24 | M        | Algeria     |  200000
    24 | M        | Netherlands |  418000
    24 | M        | Spain       |  250000
    24 | M        | [NULL]      |  289333
    24 | [NULL]   | [NULL]      |  272000
    25 | F        | Chile       |  410000
    25 | F        | [NULL]      |  410000
    25 | G        | Spain       |  200000
    25 | G        | [NULL]      |  200000
    25 | M        | Belgium     |  250000
    25 | M        | France      |  378000
```

```
     25 | M        | [NULL]      |  314000
     25 | [NULL]   | [NULL]      |  309500
     41 | F        | England     |  250000
     41 | F        | [NULL]      |  250000
     41 | M        | England     |   70000
     41 | M        | [NULL]      |   70000
     41 | [NULL]   | [NULL]      |  160000
     47 | [NULL]   | [NULL]      |  [NULL]
     47 | [NULL]   | [NULL]      |  [NULL]
     47 | [NULL]   | [NULL]      |  [NULL]
 [NULL] | [NULL]   | [NULL]      |  240444
(43 rows)
```

Subqueries

Platforms: All platforms

Parent topic: SELECT

In this section:

In general, Yellowbrick supports subqueries in the standard way.

Subquery expressions with ALL are not supported.

Correlated Subqueries

A correlated subquery contains one or more correlation references between its own columns and the columns that the outer query produces. Yellowbrick processes correlated subqueries by rewriting them and "decorrelating" these references.

Correlated subqueries take on several different patterns. In a Yellowbrick database, the general form of a supported correlated subquery is as follows:

```
select * from table1
where table1.x =
(select sum(table2.a) from table2 where table2.b = table1.x);
```

In this pattern:

- `table1` is a table in the outer query (the main query), and `table2` is a table in the inner query (the subquery). Each row that the outer query produces is qualified (or disqualified) based on the result of the subquery. (Additional tables may appear in the outer and inner queries.)
- The equal to (`=`) comparison operator is the qualifier in this example. Other operators may be used here: <, >, >=, <=
- The `SUM` function is a non-windowed, non-distinct aggregate function that produces a single value that can be compared to each row in the outer query. `AVG` , `MIN` , and `MAX` aggregate functions may also be used here, but `COUNT` is not supported.
- The correlated subquery appears in the `WHERE` clause of the outer query, and its correlation reference is in the `WHERE` clause of the subquery:

```
where table2.b = table1.x
```

- The subquery may have other predicates or join conditions that only apply to the tables in the `FROM` clause of the subquery.

Select-List Subqueries

Correlated subqueries are supported in the select list only if the value being returned by the subquery comes directly from that subquery. For example, `team.teamid` comes directly from `team` in the subquery:

```
premdb=# select (select teamid from team where team.atid=awayteam.atid) from awayteam;
```

In this case, the returned value (`awayteam.atid`) is not from the subquery (which is using `team`):

```
premdb=# select (select awayteam.atid from team where team.name=awayteam.name) from awayteam;
ERROR:  Correlated references in SELECT lists are not supported
```

Note that correlated subqueries in the select list are allowed to contain `COUNT()` and `COUNT(DISTINCT)` aggregates, unlike those in the `WHERE` clause.

Unsupported Patterns

Given this general supported pattern, note the following restrictions. The following correlated subqueries return an error message:

Correlated subqueries connected with an OR condition:

You cannot connect two correlated subqueries with an `OR` condition. For example, the following query pattern returns an error:

```
select * from table1 where
  table1.x >
  (select max(table2.a) from table2
  where table2.b = table1.x)
  OR
  table.y <
  (select min(table2.a) from table2
  where table2.b = table1.x)
  ;
```

Correlated expressions on the right side of ALL, IN, NOT IN, EXISTS, and NOT EXISTS conditions:

The following query pattern returns an error:

```
select * from table1 where EXISTS
  (select sum(table2.a) from table2
  where table2.b = table1.x);
```

Correlated subqueries with `EXISTS` and `NOT EXISTS` conditions are not supported in general but do work in some cases.

COUNT aggregation in a correlated subquery:

The following query pattern returns an error:

```
select * from table1 where table1.x =
  (select COUNT(table2.a) from table2
  where table2.b = table1.x);
```

Correlated subqueries with GROUP BY or set operations (UNION, INTERSECT, EXCEPT) or correlations in the SELECT list, HAVING clause, ORDER BY clause, or WINDOW clause:

For example, the following query pattern returns an error:

```
select * from table1 where table1.x =
  (select sum(table2.a) from table2
  where table2.b = table1.x GROUP BY table2.c);
```

Skip-level correlation references

Correlation references that skip a query block are not supported. For example, in the following query pattern, the inner block that contains the correlation reference is separated from the main query block by another subquery. The correlation reference tries to correlate `table1` and `table3`.

```
select * from table1
where table1.x =
(select sum(table2.a) from table2
where table2.b =
(select sum(table3.a from table3 where table1.x=table3.b));
```

Examples

The following example is a simple case of a correlated subquery that returns the output of a `SUM` function and compares it with a column from a table in the outer query. The correlation reference is `ht.htid=at.atid`.

```
premdb=# select * from awayteam at
where at.atid=
(select sum(ht.htid) from hometeam ht
where ht.htid=at.atid);
 atid |  name
------+---------
   51 | Arsenal
(1 row)
```

The following similar query is not supported because it replaces the `SUM` function with `COUNT`:

```
premdb=# select * from awayteam at
where at.atid=(select count(ht.htid) from hometeam ht
where ht.htid=at.atid);
ERROR:  COUNT aggregate functions in correlated subqueries are not supported
```

Subquery Examples

Platforms: All platforms

Parent topic: Subqueries

Yellowbrick supports subqueries in the select list, FROM clause, WHERE clause, and WITH clause. See also SQL Conditions and Correlated Subqueries.

The following FROM clause subquery finds results from the `match` table where the half-time and full-time scores were the same for a given season and home team.

```
premdb=# select * from
(select seasonid, htid, atid, concat_ws(' : ',htscore,ftscore) score
   from match where htid=45 and seasonid=10) s
where substr(score,1,3)=substr(score,7,9);

 seasonid | htid | atid |   score
----------+------+------+-----------
       10 |   45 |   51 | 1-1 : 1-1
       10 |   45 |   62 | 2-0 : 2-0
       10 |   45 |   63 | 2-1 : 2-1
       10 |   45 |   71 | 0-0 : 0-0
       10 |   45 |   72 | 1-0 : 1-0
(5 rows)
```

The following query uses two subqueries in the FROM clause to compares goals scored at home and away for each team during a specific season:

```
premdb=# select a.name, homegoals, awaygoals,
homegoals+awaygoals as goals_for
from
(select t1.name, sum(substr(ftscore,1,1)::int)
from match m1, team t1, season s1
where (m1.htid=t1.htid and m1.seasonid=s1.seasonid)
and season_name='2003-2004'
group by t1.name order by 2 desc) as a(name,homegoals),
(select t2.name, sum(substr(ftscore,3,1)::int)
from match m2, team t2, season s2
where (m2.atid=t2.atid and m2.seasonid=s2.seasonid)
and season_name='2003-2004'
group by t2.name) as b(name,awaygoals)
where a.name=b.name
order by 4 desc;
         name           | homegoals | awaygoals | goals_for
------------------------+-----------+-----------+-----------
 Arsenal                |        40 |        33 |        73
 Chelsea                |        34 |        33 |        67
 Manchester United      |        37 |        27 |        64
 Manchester City        |        31 |        24 |        55
 Liverpool              |        29 |        26 |        55
 Newcastle United       |        33 |        19 |        52
 Fulham                 |        29 |        23 |        52
 Blackburn Rovers       |        25 |        26 |        51
 Charlton Athletic      |        29 |        22 |        51
 Bolton Wanderers       |        24 |        24 |        48
 Leicester City         |        19 |        29 |        48
 Aston Villa            |        24 |        24 |        48
 Tottenham Hotspur      |        33 |        14 |        47
 Portsmouth             |        35 |        12 |        47
 Everton                |        27 |        18 |        45
```

```
 Middlesbrough           |        25 |        19 |        44
 Southampton             |        24 |        20 |        44
 Birmingham City         |        26 |        17 |        43
 Leeds United            |        25 |        15 |        40
 Wolverhampton Wanderers |        23 |        15 |        38
(20 rows)
```

HAVING Clause

Use the HAVING clause to filter the results of aggregate functions. (You cannot filter these results in the WHERE clause.)

```
[ HAVING condition ]
```

A HAVING condition must either repeat the aggregate function expression that was used in the SELECT list (if the expression appears there) or it can reference the column alias defined for the expression. The HAVING condition may not refer to an expression with an ordinal number. The HAVING clause may introduce an aggregate function that is not referenced elsewhere in the query.

For example, find the cities that have total stadium capacities that are greater than 50000:

```
premdb=# select city, sum(capacity) cap
from team
group by city
having cap>50000
order by 2 desc;
   city    |  cap
-----------+--------
 London     | 296983
 Manchester | 130732
 Birmingham |  99801
 Liverpool  |  84963
 Sheffield  |  72434
 Newcastle  |  52405
(6 rows)
```

Note: You may specify column aliases in the HAVING clause. In this example, you can write `having cap>50000`. You do not have to repeat the function expression `sum(capacity)`.

The following example introduces an aggregate function in the HAVING clause. The query asks for total stadium capacities for cities where the average capacity is greater than 33000.

```
premdb=# select city, sum(capacity) cap
from team
group by city
having avg(capacity)>33000
order by 2 desc;
    city      |  cap
--------------+--------
 Manchester    | 130732
 Birmingham    |  99801
 Liverpool     |  84963
 Sheffield     |  72434
 Newcastle     |  52405
 Sunderland    |  49000
 Leeds         |  39460
 Middlesbrough |  34742
 Derby         |  33597
```

```
 Cardiff       |  33280
(10 rows)
```

You can express complex conditions in the HAVING clause just as you would in the WHERE clause. For example, you can use `AND`:

```
premdb=# select city, sum(capacity) cap
from team
group by city
having avg(capacity)>33000 and sum(capacity)>50000
order by 2 desc;
   city    |  cap
-----------+--------
 Manchester | 130732
 Birmingham |  99801
 Liverpool  |  84963
 Sheffield  |  72434
 Newcastle  |  52405
(5 rows)
```

LIMIT Clause

Platforms: All platforms

Parent topic: SELECT

Use the LIMIT clause to limit the number of rows that a query returns.

```
[ LIMIT { count | ALL } ]
```

`LIMIT ALL` means return all rows. `LIMIT 0` means return no rows. Use an ORDER BY clause if you want a query to consistently return the same subset of rows.

For example:

```
premdb=# select * from match order by matchday limit 0;
 seasonid | matchday | htid | atid | ftscore | htscore
----------+----------+------+------+---------+---------
(0 rows)
```

```
premdb=# select * from match order by matchday limit 3;
 seasonid |      matchday       | htid | atid | ftscore | htscore
----------+---------------------+------+------+---------+---------
        1 | 1992-08-01 00:00:00 |    2 |   52 | 0-1     | -
        1 | 1992-08-01 00:00:00 |    2 |   55 | 0-1     | -
        1 | 1992-08-01 00:00:00 |    2 |   63 | 2-1     | -
(3 rows)
```

OFFSET Clause

Use the OFFSET clause to define a starting point for the set of rows that a query returns. (You must use OFFSET in combination with ORDER BY to guarantee where the offset falls with respect to an ordered set of rows.)

```
[ OFFSET start [ ROW | ROWS ] ]
```

The `start` argument defines the starting number for the rows that are returned (row 10 or row 10000, for example). You cannot use a negative number. `ROW` and `ROWS` are optional keywords that have no effect.

For example:

```
premdb=# select * from match
order by matchday
limit 10
offset 5000;
 seasonid |      matchday       | htid | atid | ftscore | htscore
----------+---------------------+------+------+---------+---------
       13 | 2004-12-28 00:00:00 |   42 |   65 | 1-1     | 0-0
       13 | 2004-12-28 00:00:00 |   27 |   78 | 2-0     | 0-0
       13 | 2004-12-28 00:00:00 |    8 |   55 | 0-1     | 0-1
       13 | 2004-12-28 00:00:00 |    3 |   75 | 0-1     | 0-1
       13 | 2004-12-28 00:00:00 |   13 |   67 | 2-0     | 0-0
       13 | 2004-12-29 00:00:00 |   28 |   51 | 0-1     | 0-1
       13 | 2005-01-01 00:00:00 |   13 |   51 | 1-3     | 1-1
       13 | 2005-01-01 00:00:00 |   24 |   63 | 0-1     | 0-0
       13 | 2005-01-01 00:00:00 |    8 |   93 | 1-1     | 0-1
       13 | 2005-01-01 00:00:00 |   27 |   75 | 0-2     | 0-1
(10 rows)
```

ORDER BY Clause

Platforms: All platforms

Parent topic: SELECT

Use the ORDER BY clause to define how the final result set for a query (or an intermediate result set for a subquery) is sorted and displayed.

```
[ ORDER BY expression [ ASC | DESC ] [ NULLS { FIRST | LAST } ] [, ...] [ COLLATE collation ] ]
```

expression

One of the following:

- Column name
- Column expression
- Column alias
- Column ordinal (a number that refers to the position of the column or expression in the select list)

ASC | DESC

`ASC` (ascending) is the default order. `DESC` is often used for ranking numeric results from high to low. Each of these keywords may follow any ORDER BY expression.

NULLS FIRST | LAST

`NULL` values can be sorted first or last for any ORDER BY expression. `NULLS LAST` is the default when ascending order is in effect, and `NULLS FIRST` is the default when descending order is in effect.

COLLATE collation

The `COLLATE` clause is supported in UTF8 databases only. This clause defines the sort order based on a specific predefined collation. You cannot create collations in a Yellowbrick database. See Supported Collations (UTF8 Databases Only).

Usage Notes

- ORDER BY expressions must appear in the select list when DISTINCT is used in the select list.
- Parameters defined in prepared statements cannot be used in the ORDER BY clause.
- Whole-row references are not supported in the ORDER BY clause.
- By default, the `C` collation order is used for both `LATIN9` databases and `UTF8` databases. However, `UTF8` databases support the `COLLATE` clause. See Supported Collations (UTF8 Databases Only).

Examples

Assume that the select list looks like this:

```
select winners as club, season_name as season
```

For this select list, all of the following are valid ORDER BY clauses:

```
order by winners nulls first, season_name desc;
order by club nulls first, season desc;
order by 1 nulls first, 2 desc;
```

Here is a complete query with an ORDER BY clause:

```
premdb=# select * from season
order by winners nulls first, seasonid;
 seasonid | season_name | numteams |      winners
----------+-------------+----------+-------------------
       25 | 2016-2017   |       20 |
        6 | 1997-1998   |       20 | Arsenal
       10 | 2001-2002   |       20 | Arsenal
       12 | 2003-2004   |       20 | Arsenal
        3 | 1994-1995   |       22 | Blackburn Rovers
       13 | 2004-2005   |       20 | Chelsea
       14 | 2005-2006   |       20 | Chelsea
       18 | 2009-2010   |       20 | Chelsea
       23 | 2014-2015   |       20 | Chelsea
...
```

This example demonstrates the restriction for SELECT DISTINCT queries with ORDER BY clauses:

```
premdb=# select distinct season_name, winners from season
order by winners nulls first, seasonid;
ERROR:  for SELECT DISTINCT, ORDER BY expressions must appear in select list
LINE 1: ... winners from season order by winners nulls first, seasonid;
```

This example demonstrates the restriction on whole-row references in the ORDER BY clause:

```
premdb=# select team.* from team order by team.*;
ERROR:  Whole row references are not allowed: team.*
LINE 1: select team.* from team order by team.*;
```

The following example runs in a UTF8 database and contains a `COLLATE` clause:

```
premdb_utf8=# select * from chinese_names order by name collate "zh_CN";
         name
------------------------
 利物浦
 曼彻斯特城
 曼联队
 切尔西
 托特纳姆热刺足球俱乐部
(5 rows)
```

Supported Collations (UTF8 Databases Only)

UTF8 databases support the following collations in the `ORDER BY` ... `COLLATE` clause. The default `C` collation is also supported.

```
ar_AE
ar_BH
ar_DZ
ar_EG
ar_IQ
```

```
ar_JO
ar_KW
ar_LB
ar_LY
ar_MA
ar_OM
ar_QA
ar_SA
ar_SD
ar_SY
ar_TN
ar_YE
be_BY
bg_BG
ca_ES
cs_CZ
da_DK
de_AT
de_CH
de_DE
de_LU
el_CY
el_GR
en_AU
en_CA
en_GB
en_IE
en_IN
en_NZ
en_PH
en_SG
en_US
en_ZA
es_AR
es_BO
es_CL
es_CO
es_CR
es_DO
es_EC
es_ES
es_GT
es_HN
es_MX
es_NI
es_PA
es_PE
es_PR
es_PY
es_SV
es_US
es_UY
es_VE
et_EE
fi_FI
fr_BE
fr_CA
fr_CH
fr_FR
fr_LU
ga_IE
hi_IN
hr_HR
hu_HU
is_IS
it_CH
it_IT
iw_IL
```

```
ja_JP
ko_KR
lt_LT
lv_LV
mk_MK
ms_MY
mt_MT
nl_BE
nl_NL
pl_PL
pt_BR
pt_PT
ro_RO
ru_RU
sk_SK
sl_SI
sq_AL
sr_ME
sr_RS
sv_SE
th_TH
tr_TR
uk_UA
vi_VN
zh_CN
zh_HK
zh_SG
zh_TW
```

SELECT List

Define a list of output columns for a query.

```
SELECT [ ALL | DISTINCT expression [ [ AS ] output_name ] [, ...] ]
[ * | expression [ [ AS ] output_name ] [, ...] ]
```

ALL

Return all rows for the expression, retaining duplicates (the default select list behavior).

DISTINCT

Discard duplicate rows for the expression. When DISTINCT is used, the ORDER BY clause must only include expressions from the SELECT list. For example, the following query returns an error:

```
select distinct numteams, winners from season order by seasonid;
ERROR:  for SELECT DISTINCT, ORDER BY expressions must appear in select list
```

DISTINCT ON syntax is not supported.

* (star)

Return all rows and columns from the table. See also TABLE, which provides equivalent syntax for a `SELECT *` query from a single table.

output_name

Define an alias for each resulting column. SQL functions return default names if you do not provide aliases. Column aliases can be referenced later in the select list itself and later in the query, such as in the WHERE and HAVING clauses.

Examples

For example, find distinct combinations of `numteams` and `winners` values in the `season` table:

```
premdb=# select distinct numteams, winners from season
where winners is not null
order by 2;
 numteams |      winners
----------+-------------------
       20 | Arsenal
       22 | Blackburn Rovers
       20 | Chelsea
       20 | Leicester City
       20 | Manchester City
       20 | Manchester United
       22 | Manchester United
(7 rows)
```

Whole row references are not allowed. For example:

```
premdb=# select distinct team from team order by 1;
ERROR:  Whole row references are not allowed: team
LINE 1: select distinct team from team order by 1;
```

Column Aliases

In the select list of a query, you can define aliases for expressions and reuse them immediately in subsequent select list expressions or later in the query. For example, in this query the `cap` alias is defined in the first select list expression, then used in a calculation in the second expression:

```
premdb=# select avg_att/capacity as cap, cap*1000 from team where avg_att>0;
      cap        |     ?column?
-----------------+-----------------
 0.00099475605708 | 0.99475605708000
 0.00078742549959 | 0.78742549959000
 0.00097601186322 | 0.97601186322000
 0.00099608765571 | 0.99608765571000
 0.00093833555513 | 0.93833555513000
 0.00094786305661 | 0.94786305661000
 0.00099810923067 | 0.99810923067000
 ...
```

In the following example, the alias `c` is reused in the WHERE clause, and the alias `sa` is reused in the ORDER BY clause:

```
premdb=# select city c, sum(avg_att) sa from team
where c in('London','Manchester') group by c order by sa desc;
    c      |   sa
-----------+---------
 London     | 196.766
 Manchester | 129.327
(2 rows)
```

Multiple expressions may have the same alias in the same select list; if this kind of duplication occurs, the leftmost alias is used to resolve later references.

When a *volatile* function, such as the RANDOM() function, is the target of an alias, the function is computed each time; its initial result does not persist. For example, in this query, `r1`, `r2`, and `r3` all produce different results:

```
premdb=# select random() r1, r1 r2, r1 r3 from sys.const;
       r1        |        r2         |        r3
-----------------+-------------------+-------------------
 0.49950508915522 | 0.918498391599146 | 0.750150247759709
(1 row)
```

SETTING Clause

Yellowbrick Documentation > Reference > SQL Reference > SQL Commands > SELECT > SETTING Clause

Platforms: All platforms

Parent topic: SELECT

As a way to influence planner optimization and query execution, you can specify a `SETTING` clause at the beginning of the following SQL commands:

- `SELECT`
- `INSERT INTO SELECT`
- `UPDATE`
- `DELETE`
- `CREATE TABLE AS (CTAS)`
- `EXPLAIN` (with the commands in this list)

Note: The values you specify in the `SETTING` clause remain in effect for the duration of a single SQL statement. Values do not persist for the life of the session or transaction.

The `SETTING` clause syntax is as follows:

```
SETTING ( hint [ ; hint ... ] )
```

where `hint` must be one of the available hints :

- SET hint
- ROWS hint
- JOIN METHOD hint
- LEADING hint

If a `SELECT` statement contains a `WITH` clause (common table expression) and a `SETTING` clause, the `SETTING` clause must precede the `WITH` clause.

Note When using SETTING to inject multiple hints, they must be separated by a semicolon

See Plan Hinting for all details on available hints and how to use them.

UNION, INTERSECT, EXCEPT

Platforms: All platforms

Parent topic: SELECT

These set operators compare and merge the results of two separate query expressions. You can select compatible column information from two separate tables, finding only the common rows (`INTERSECT`), rows that are in one result set but not the other (`EXCEPT` or `MINUS`), or a complete `UNION` of all of the rows from both sides.

```
query
 { UNION | INTERSECT | { EXCEPT | MINUS } } [ ALL | DISTINCT ]
query
```

Each `query` must have the same number of columns, and the columns within each `query` must have compatible data types. You can write queries that contain multiple set operators connecting multiple query expressions.

You can use the `ALL` and `DISTINCT` keywords to return or discard any duplicate rows. The default behavior is to discard duplicates (`DISTINCT`).

To qualify for the `EXCEPT` or `MINUS` result set, rows must exist in the result of the first query expression but not in the result of the second. `EXCEPT` and `MINUS` are synonyms.

For example:

```
premdb=# select count(*) from
(select htid, atid from team except select htid, atid from match) exc;
 count
-------
   50
(1 row)
```

```
premdb=# select count(*) from
(select htid, atid from match minus select htid, atid from team) exc;
 count
-------
  1562
(1 row)
```

WHERE Clause

Use the WHERE clause for two distinct purposes: to filter rows based on some search criteria and to join tables based on one or more join conditions.

```
[ WHERE condition ]
```

Note the following rules:

- The results of aggregate functions may not be constrained in the WHERE clause. See HAVING Clause.
- Columns that do not exist in tables defined in the FROM clause may not be restricted in the WHERE clause.
- Inner joins can be expressed in the FROM clause or the WHERE clause. Outer joins must be expressed in the FROM clause.
- Equality conditions are supported for all join types. Inequality conditions are not supported for full outer joins.

For example, the following query joins four tables, and it filters rows based on additional search criteria. In this case, parentheses are used to separate the join conditions from the other constraints:

```
premdb=# SELECT season_name, matchday, h.name hname, a.name aname, ftscore
FROM season s, hometeam h, awayteam a, match m
WHERE (s.seasonid=m.seasonid AND h.htid=m.htid AND a.atid=m.atid)
AND hname='Chelsea' AND season_name='2011-2012'
ORDER BY matchday;
 season_name |      matchday       |  hname  |          aname          | ftscore
-------------+---------------------+---------+-------------------------+---------
 2011-2012   | 2011-08-20 00:00:00 | Chelsea | West Bromwich Albion    | 2-1
 2011-2012   | 2011-08-27 00:00:00 | Chelsea | Norwich City            | 3-1
 2011-2012   | 2011-09-24 00:00:00 | Chelsea | Swansea City            | 4-1
 2011-2012   | 2011-10-15 00:00:00 | Chelsea | Everton                 | 3-1
 2011-2012   | 2011-10-29 00:00:00 | Chelsea | Arsenal                 | 3-5
 2011-2012   | 2011-11-20 00:00:00 | Chelsea | Liverpool               | 1-2
 2011-2012   | 2011-11-26 00:00:00 | Chelsea | Wolverhampton Wanderers | 3-0
 2011-2012   | 2011-12-12 00:00:00 | Chelsea | Manchester City         | 2-1
 2011-2012   | 2011-12-26 00:00:00 | Chelsea | Fulham                  | 1-1
 2011-2012   | 2011-12-31 00:00:00 | Chelsea | Aston Villa             | 1-3
 2011-2012   | 2012-01-14 00:00:00 | Chelsea | Sunderland              | 1-0
 2011-2012   | 2012-02-05 00:00:00 | Chelsea | Manchester United       | 3-3
 2011-2012   | 2012-02-25 00:00:00 | Chelsea | Bolton Wanderers        | 3-0
 2011-2012   | 2012-03-10 00:00:00 | Chelsea | Stoke City              | 1-0
 2011-2012   | 2012-03-24 00:00:00 | Chelsea | Tottenham Hotspur       | 0-0
 2011-2012   | 2012-04-07 00:00:00 | Chelsea | Wigan Athletic          | 2-1
 2011-2012   | 2012-04-29 00:00:00 | Chelsea | Queens Park Rangers     | 6-1
 2011-2012   | 2012-05-02 00:00:00 | Chelsea | Newcastle United        | 0-2
 2011-2012   | 2012-05-13 00:00:00 | Chelsea | Blackburn Rovers        | 2-1
(19 rows)
```

WINDOW Clause

Platforms: All platforms

Parent topic: SELECT

Use the `WINDOW` clause as a shortcut to define the same behavior for multiple window functions in the same query.

```
[ WINDOW window_name AS ( window_definition ) [, ...] ]
```

In this context, the `window_definition` is the contents of the `OVER` clause, not including the `OVER` keyword.

For example:

```
premdb=# select name, city,
sum(capacity) over w,
max(capacity) over w
from team
window w as (partition by city order by capacity);
         name           |     city      |  sum   |  max
------------------------+---------------+--------+-------
 Barnsley               | Barnsley      |  23009 | 23009
 West Bromwich Albion   | Birmingham    |  27000 | 27000
 Birmingham City        | Birmingham    |  57016 | 30016
 Aston Villa            | Birmingham    |  99801 | 42785
 Blackburn Rovers       | Blackburn     |  31367 | 31367
 Blackpool              | Blackpool     |  17338 | 17338
 Bolton Wanderers       | Bolton        |  28723 | 28723
 Bournemouth            | Bournemouth   |  11464 | 11464
 Bradford City          | Bradford      |  25136 | 25136
 Burnley                | Burnley       |  22546 | 22546
...
```

WITH Clause

Platforms: All platforms

Parent topic: SELECT

Define common table expressions (CTEs) for reuse in queries. CTEs are defined once as subqueries in the WITH clause that precedes a SELECT query and are reusable throughout that query.

```
[ WITH name AS (subquery) [, ...] ]
```

Give each WITH subquery a unique name. The WITH clause may define one or more subqueries, using any supported SELECT statement syntax. CTEs cannot contain DML commands (INSERT, UPDATE, DELETE).

The following query is a simple example of a join between the `match` table and a subquery defined in the WITH clause (`w1`):

```
premdb=# with w1 as (select * from awayteam)
select * from match, w1
where w1.atid=match.atid;
 seasonid |      matchday       | htid | atid | ftscore | htscore | atid |          name
----------+---------------------+------+------+---------+---------+------+------------------------
        1 | 1992-08-01 00:00:00 |    2 |   52 | 0-1     | -       |   52 | Aston Villa
        1 | 1992-08-01 00:00:00 |    2 |   55 | 0-1     | -       |   55 | Blackburn Rovers
        1 | 1992-08-01 00:00:00 |    2 |   63 | 2-1     | -       |   63 | Chelsea
        1 | 1992-08-01 00:00:00 |    2 |   64 | 3-0     | -       |   64 | Coventry City
        1 | 1992-08-01 00:00:00 |    2 |   65 | 3-0     | -       |   65 | Crystal Palace
...
```

ABORT

Roll back the current transaction.

```
ABORT [ WORK | TRANSACTION ]
```

`WORK` and `TRANSACTION` are both optional keywords; they have no effect.

See also ROLLBACK.

ALTER CLUSTER

Alter the attributes or state of an existing cluster, using one or more `WITH()` options. You can also suspend, resume, or rename a cluster.

```
ALTER [ COMPUTE ] CLUSTER name
WITH (
[ NODE_COUNT number ]
[ AUTO_SUSPEND number | NULL ]
[ AUTO_RESUME TRUE | FALSE ]
[ WLM_PROFILE name ]
[ TIMEOUT timeout_seconds ]
)
[ ACTIVATE wait_seconds [ WITH CANCEL | WITHOUT CANCEL ] ]
```

```
ALTER [ COMPUTE ] CLUSTER name
[ SUSPEND | SUSPEND wait_seconds [ WITH CANCEL | WITHOUT CANCEL ] ]
[ RESUME [ TIMEOUT timeout_seconds ] ]
[ RENAME TO new_name ]
```

COMPUTE
Optional keyword; in the current release, all clusters are compute clusters. They do the work of executing queries and other operations such as bulk loads.

name
Unique name for a cluster within an instance.

NODE_COUNT
The number of compute nodes assigned to the cluster.

AUTO_SUSPEND
The number of seconds of user inactivity, after which the cluster is automatically suspended.

AUTO_RESUME
Whether to automatically resume the instance when a new user query is submitted against a suspended cluster.

WLM_PROFILE
WLM (workload management) profile associated with the cluster. A default profile is assigned when a cluster is created.

TIMEOUT timeout_seconds
The number of seconds before the operation times out.

ACTIVATE wait_seconds
The number of seconds to wait before activating the modified cluster, with or without canceling queries that are currently running.

SUSPEND
Suspend a running cluster. Optionally, specify a wait time and whether to cancel running queries.

RESUME
Resume a suspended cluster. Optionally, specify a timeout in seconds.

RENAME TO
Rename a cluster.

Examples with Modified Attributes

The following example alters a compute cluster called `large-default-cluster`, giving it two nodes and automatic suspension after 600 seconds of inactivity:

```
premdb=> ALTER CLUSTER "large-default-cluster" WITH (
NODE_COUNT 2,
AUTO_SUSPEND 600
);
ALTER COMPUTE CLUSTER
```

The following example suspends a compute cluster called `test2`:

- The cluster will be suspended in 60 seconds.
- The cluster will not cancel existing queries before suspending.

```
premdb=> ALTER CLUSTER "test2"
SUSPEND 60
WITHOUT CANCEL;
ALTER COMPUTE CLUSTER
```

Examples with Actions on a Cluster

The following example resumes a cluster:

```
premdb=> alter cluster "bobr-rc6-April12-cluster1" resume;
ALTER COMPUTE CLUSTER
```

The following example renames a cluster:

```
premdb=> alter cluster "bobr-rc6-April12-cluster1" rename to "bobr-rc6-cluster1";
ALTER COMPUTE CLUSTER
```

ALTER DATABASE

Alter attributes of an existing physical database.

```
ALTER DATABASE name [ [ WITH ] ALLOW_CONNECTIONS { true | false }  ]
ALTER DATABASE name CONNECTION LIMIT number
ALTER DATABASE name RENAME TO new_name
ALTER DATABASE name OWNER TO { new_owner | CURRENT_USER | SESSION_USER }
ALTER DATABASE name RESET MAX_SIZE
ALTER DATABASE name SET HOT_STANDBY ON | OFF
ALTER DATABASE name SET MAX_SIZE [=] 'size'
ALTER DATABASE name SET MAX_SIZE DISABLE
ALTER DATABASE name SET READONLY ON [ WITH [ IMMEDIATE | TIMEOUT n { SECOND | SECONDS | MINUTE | MINUTES } ] ] | OFF
```

ALLOW_CONNECTIONS

Specify whether the database accepts user connections: `true` or `false`.

CONNECTION LIMIT

Set a limit on the number of concurrent database connections by regular users (superuser connections do not count). Some of these connections may be used up by internal database services.

RENAME TO

Rename a database. You cannot rename the current database (the database you are connected to) or any database that is currently involved in replication.

OWNER TO

Change the owner of a database. See CURRENT_USER and SESSION_USER.

RESET MAX_SIZE

Disable the disk usage limit for the specified database. Similar to `ALTER DATABASE name SET MAX_SIZE DISABLE`.

SET HOT_STANDBY ON | OFF

`ON`: Prepare an *empty* database to serve as a target for a restore operation or replication. `OFF`: Take a database out of `HOT_STANDBY` mode because you no longer want to use it for incremental restores or replication.

SET MAX_SIZE [=] 'size'

Specify the maximum amount of space an individual database may use, where `'size'` is the number of bytes. `'size'` must be a positive integer with the possible inclusion of units: MB, GB, or TB. If units are included, the command will be accepted with and without a space between the integer and unit.

SET MAX_SIZE DISABLE

Disable the disk usage limit for the specified database. Similar to `ALTER DATABASE name RESET MAX_SIZE`.

SET READONLY ON | OFF

Put a database in `READONLY` mode and prevent new writes to the database. By default, the system waits for any running read/write transactions to complete. Optionally, you can specify `WITH IMMEDIATE` or a `TIMEOUT` of some number of seconds or minutes.

`WITH IMMEDIATE` causes any running write transactions to be rolled back. A `TIMEOUT` setting waits the specified number of seconds or minutes for read/write transactions to complete, then aborts any remaining read/write transactions. The database is set to `READONLY` immediately if no read/write transactions are running.

You can also set `READONLY` mode to `OFF`. The `WITH IMMEDIATE` and `TIMEOUT` options do not apply in this case.

The database mode is reflected in the results of `ybsql \l` command and logged to the `sys.database` view.

Note: If a restore operation is in progress for the database named in the `ALTER DATABASE` command, the command has to wait for an exclusive system catalog lock. The `ALTER DATABASE` command will proceed when the lock being held by the restore operation is released.

Examples

For example, attempt to alter the name of the current database, then switch to another database and retry.

```
shakespeare=# alter database shakespeare rename to shakes;
ERROR:  current database cannot be renamed
shakespeare=# \c premdb
You are now connected to database "premdb" as user "brumsby".
premdb=# alter database shakespeare rename to shakes;
ALTER DATABASE
```

Change the owner of the `premdb` database:

```
premdb=# alter database premdb owner to bobr;
ALTER DATABASE
```

Disable connections to the `yellowbrick` database:

```
premdb=# alter database yellowbrick allow_connections false;
ALTER DATABASE
premdb=# \c yellowbrick
FATAL:  database "yellowbrick" is not currently accepting connections
Previous connection kept
```

Put a database into `HOT_STANDBY` mode:

```
premdb=# alter database newpremdb set hot_standby on;
ALTER DATABASE
```

Put a database into `READONLY` mode after waiting for 10 minutes for read/write transactions to complete:

```
premdb=# alter database premdb_secure_all set readonly on with timeout 10 minutes;
ALTER DATABASE
```

Turn off `HOT_STANDBY` mode for a database that was in use for replication:

```
yellowbrick=# alter database premdb set hot_standby off;
WARNING:  This database will no longer accept restore and replication operations
ALTER DATABASE
```

Set a quota for the `newpremdb` database:

```
premdb=# alter database newprembd set max_size='1TB';
ALTER DATABASE
```

ALTER DATABASE ADD REPLICA

Create a replica for a source database.

```
ALTER DATABASE source_database_name
ADD REPLICA replica_name
TO remote_server
WITH (FREQUENCY seconds
   [, ALIAS 'target_database_name' ]
   [, EXCLUDE 'schema_list' ]
   [, SECURITY_MODE 'ALL' | 'NONE' ]
   [, BW_LIMIT megabytes ]
   [, REVERSE_REPLICA 'reverse_replica_name' ]
   [, USER_RESOLUTION_MODE 'PLACEHOLDER' | 'DB_OWNER' ]
)
```

Syntax

source_database_name

Specify the name of a database on the source system. Data will be replicated from this database to the named target database. You can create only 1 replica per database.

Note: You cannot create a replica for the `yellowbrick` database.

replica_name

Specify a unique name for this replica. A replica is not a database; it is a logical object that identifies a remote server, a source database, and a target database.

remote_server

Specify the name of an existing remote server, as created with the CREATE REMOTE SERVER command.

ALIAS target_database_name

Specify a database name, which will be the name of the target database for replication using this replica. If the named database exists on the target system, it must be empty and in `HOT_STANDBY` mode. If the named database does not exist, it is automatically created and placed in `HOT_STANDBY` mode.

Note: The target database does not have to be empty in one exceptional case: if you seed the replica by restoring the database from a backup before starting replication. See Seeding a Replica.

If you do not specify an alias, the command creates a database with the same name as the source database on the target system. For loopback replication, you must replicate to an aliased database.

FREQUENCY seconds

Specify the replication interval, in seconds. The value must be an integer. The minimum frequency is `5` seconds. There is no default value. For example, to use a replication interval of 1 minute, set the frequency to `60`. For one hour, set it to `3600`.

EXCLUDE 'schema_list'

Optionally, specify a list of schemas that you want to exclude when replication runs. If you do not specify this option, all objects in all schemas are replicated. *Enclose the list in single quotes and separate the schema names with commas. Do not use leading or trailing spaces in the list.*

You can also expand the list of excluded schemas by using the wildcard character * (an asterisk). No other wildcard characters are supported. For example, the following syntax excludes the `public` schema, all schemas with the prefix `premdb_`, and all schemas with the prefix `epl`:

```
EXCLUDE 'public,premdb_*,epl*'
```

Tip: You can use the `EXCLUDE` wildcard character to prevent the replication of new schemas that are added to a database, assuming that there is a convention in place for naming schemas with a standard set of prefixes.

SECURITY_MODE 'ALL' | 'NONE'

Specify this option to replicate all or none of the following security-related objects and attributes: users, roles, grants, and ACLs. The default behavior is `ALL`.

BW_LIMIT megabytes

Specify a bandwidth limit in megabytes per second, which is the maximum speed of transmission traffic for replication operations. Typically, this limit does not need to be imposed, allowing replication to run at full throttle. If you have a business requirement to control the rate of traffic over the link between the two databases, you can set this option to some number of megabytes per second.

REVERSE_REPLICA

Create a reverse replica for use during a failover procedure. See Replication Failover and Failback.

Note: Allow some time for forward replication to start before trying to create a reverse replica.

USER_RESOLUTION_MODE 'PLACEHOLDER' | 'DB_OWNER'

Specify the user resolution mode when replicated users and roles cannot be resolved on the target system, either because they never existed on the target or because they were previously replicated, then dropped. When `SECURITY_MODE` is set to `ALL`, references to missing replicated users are mapped to the database owner by default. In turn, default privileges for those users are not replicated. You can change this behavior by creating the replica as follows:

```
WITH (..., security_mode 'ALL', user_resolution_mode 'placeholder'...)
```

`PLACEHOLDER` creates a placeholder role on the target system for the role on the source system and maps all references to that role, including default privileges.
`DB_OWNER` maps all references to the database owner.

Examples

Create a replica with a target database named `premdb_replica_db` and a replication interval of `10` seconds.

```
premdb=# alter database premdb
add replica premdb_replica to ybd_repl_svr
with (alias premdb_replica_db, frequency 10);
ADD REPLICA
```

Create a replica for a different database but use the same remote server. This replica has a frequency of `60` seconds and excludes some schemas:

```
premdb=# alter database yb100db
add replica yb100db_replica to ybd_repl_svr
with (alias yb100db_replica_db, frequency 60, exclude 'public,yb100schema_*');
ADD REPLICA
```

ALTER DATABASE ALTER REPLICA

Start, pause, or resume replication for a specific replica. Also use this command to modify other attributes of the replica.

```
ALTER DATABASE local_database_name
ALTER REPLICA replica_name
START [ WITH CHAIN chain_name ] |
PAUSE |
RESUME |
FORCE |
PROMOTE [ WITH DATA LOSS ] |
WITH (FREQUENCY seconds)
```

Syntax

local_database_name

Specify the name of a database on the source system whose replica you want to start or modify.

replica_name

Specify the unique name of the replica you want to start or modify.

START

Start running replication for the specified replica and its target database, as defined with the `ALTER DATABASE ADD REPLICA` command. The replication interval will be the frequency that was set when the replica was created.

WITH CHAIN chain_name

Optionally, specify an existing backup chain name. By default, the replica will have a backup chain name that is the same as the replica name, and you do not need to specify it when you start replication. When replication has started, the specific backup chain in use cannot be dropped or used for future backup or restore operations. The replication process depends on that chain for incremental restore operations to the replica database.

If `WITH CHAIN` is specified, the remote database is seeded with a new backup chain. A new backup chain is created in the source database, and a full backup is transmitted and restored via the replication service.

You can query the `sys.replica` view to get information about the backup chains and replicas that are in use.

Note: If you seeded the replica from a backup, you must start replication with the same backup chain that the backup used.

PAUSE

Pause replication for the specified replica and its target database.

RESUME

Resume replication for the specified replica and its target database.

FORCE

Start replication now for the specified replica and its target database. Replication must be in `PAUSED` state in order for this command to run.

PROMOTE [WITH DATA LOSS]

Reverse the roles of the primary and secondary systems, using a reverse replica. If you specify the `WITH DATA LOSS` option, the `PROMOTE` command will not try to force replication from the primary to the secondary before reversing the roles. Data written to the source database on the primary system may not have been replicated and may be lost when replication starts in the reverse direction.

See also Replication Failover and Failback.

Note: A `ROLLBACK DATABASE TO SNAPSHOT` command is run *automatically* as part of an `ALTER DATABASE...ALTER REPLICA...PROMOTE` command. You cannot run multiple `PROMOTE` operations (or explicit `ROLLBACK DATABASE` operations) in parallel. Because of exclusive locks on shared dependencies (required by the rollback), the `PROMOTE` operations will block each other and fail. For example:

```
alter database repldb1 alter replica repldb1_failover promote;
alter database repldb2 alter replica repldb2_failover promote;
Error:
ERROR: Failed to promote replica
DETAIL: Unable to promote replica “repldb1_failover” Reason: ... ROLLBACK DATABASE failed to lock shared relations
...
ERROR: Failed to promote replica
DETAIL: Unable to promote replica “repldb2_failover” Reason: ... ROLLBACK DATABASE failed to lock shared relations
```

WITH (FREQUENCY seconds)

Change the frequency for a replica that is not running (paused or not started). Specify the number of seconds for the interval.

Examples

Start replication with the default backup chain:

```
premdb=# alter database premdb
alter replica premdb_replica start;
START REPLICA
```

Start replication with a designated new backup chain:

```
premdb=# alter database premdb
alter replica premdb_replica
start with chain premdb_replica_chain;
START REPLICA
```

Attempt to start replication when a backup chain is already in use:

```
premdb=# alter database premdb
alter replica premdb_replica start;
ERROR:  Backup chain for replica "premdb_replica" already exists. Did you mean to RESUME this replica or START WITH CHAIN?
premdb=# alter database premdb alter replica premdb_replica resume;
RESUME REPLICA
```

Alter the frequency of a paused replica:

```
premdb=# alter database premdb alter replica premdb_replica1 with (frequency 30);
ALTER REPLICA
```

See also the examples for ALTER DATABASE ADD REPLICA.

ALTER DATABASE DROP REPLICA

Drop a replica object from a source system.

```
ALTER DATABASE local_database_name DROP REPLICA replica_name
```

When you drop a replica, its corresponding backup chain on the source system is also dropped. Note that dropping a replica on the source system does not drop the backup chain on the target system.

For example:

```
premdb=# alter database premdb drop replica premdb_replica;
DROP REPLICA
```

In this example, the replica cannot be dropped because replication is in progress:

```
premdb=# alter database premdb drop replica premdb_replica;
ERROR:  Failed to drop replica
DETAIL:  Cannot drop replica premdb_replica from database premdb because it is currently running
```

In order to drop this replica, you have to pause replication first:

```
premdb=# alter database premdb alter replica premdb_replica pause;
PAUSE REPLICA
premdb=# alter database premdb drop replica premdb_replica;
DROP REPLICA
```

ALTER DEFAULT PRIVILEGES

Grant or revoke default access privileges on database objects that are created in the future. This command does not affect privileges defined on existing objects.

You can apply default privileges on the following objects:

- Tables and views (`ON TABLES`)
- Sequences (`ON SEQUENCES`)
- Stored procedures (`ON PROCEDURES`)

This command is very useful for databases and schemas in which new tables and other objects are created routinely as part of daily operations. By setting default privileges for users and roles, you avoid having to grant permissions on separate new objects as they are created. For example, you can make sure that users who belong to a specific role can automatically select from new temporary tables that are created in a specific schema.

You can alter privileges either for objects in the current database or for objects in the specified schema or schemas. If you specify default privileges on one or more schemas, these privileges are added to any global default privileges for the type of object in question.

```
ALTER DEFAULT PRIVILEGES
   [ FOR { ROLE | USER } creator_role [, ...] ]
   [ IN SCHEMA schema [, ...] ]
   { grant_command | revoke_command }
```

where `grant_command` is one of the following:

```
GRANT { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES }
   [, ...] | ALL [ PRIVILEGES ] }
   ON TABLES
   TO { role | PUBLIC } [, ...] [ WITH GRANT OPTION ]

GRANT { { USAGE | SELECT | UPDATE }
   [, ...] | ALL [ PRIVILEGES ] }
   ON SEQUENCES
   TO { role | PUBLIC } [, ...] [ WITH GRANT OPTION ]

GRANT { EXECUTE | ALL [ PRIVILEGES ] }
   ON PROCEDURES
   TO { role | PUBLIC } [, ...] [ WITH GRANT OPTION ]
```

and `revoke_command` is one of the following:

```
REVOKE [ GRANT OPTION FOR ]
   { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES }
   [, ...] | ALL [ PRIVILEGES ] }
   ON TABLES
   FROM { role | PUBLIC } [, ...]
   [ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
   { { USAGE | SELECT | UPDATE }
```

```
    [, ...] | ALL [ PRIVILEGES ] }
    ON SEQUENCES
    FROM { role | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
    { EXECUTE | ALL [ PRIVILEGES ] }
    ON PROCEDURES
    FROM { role | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]
```

Note: The default privileges for any object type normally grant all permissions to the object owner, and may grant some privileges to `PUBLIC` as well. However, you can change this behavior by altering global default privileges with the `ALTER DEFAULT PRIVILEGES` command.

Before you can drop a role for which default privileges have been altered, you need to reverse the changes that were made. Alternatively, you can use the DROP OWNED BY command to remove the default privileges for the role (along with the objects owned by the role). For example:

```
premdb=# drop role role1;
ERROR:  role "role1" cannot be dropped because some objects depend on it
DETAIL:  owner of default privileges on new relations belonging to role role1
premdb=# drop owned by role1;
DROP OWNED
premdb=# drop role role1;
DROP ROLE
```

Parameters

creator_role

Specify an existing role of which the current user is a member. If you do not specify `FOR ROLE`, the default target is the current role. This role specification defines an *object creator*. (It is not sufficient to be the object owner because ownership can be transferred.) Objects created by this role will be subject to the GRANT or REVOKE action that follows in the ALTER DEFAULT PRIVILEGES command. You can change default privileges only for objects that will be created by yourself or by roles that you are a member of. (`FOR USER` is synonymous with `FOR ROLE` in this context.)

schema

Specify an existing schema so that default privileges are altered for objects that are subsequently created in that schema. If you do not specify `IN SCHEMA`, default privileges are altered for all schemas in the current database.

grant_command | revoke_command

In general, see the GRANT and REVOKE command descriptions for more information about the syntax that you can use, which is an abbreviated version of those two commands. Remember that the ALTER DEFAULT PRIVILEGES command sets permissions for all objects of a given type, not for one or more named objects.

role_name

Specify the name of one or more existing roles that will be the receivers of the GRANT or REVOKE action. You can specify `PUBLIC` instead of a role name.

Examples

Grant `INSERT` privileges to `bobr` on all new tables created in the `public` schema by the current user (`ybdadmin@yellowbrickcloud.com`). You can use the `ybsql \ddp` command to display default access privileges.

```
yellowbrick=> alter default privileges in schema public grant insert on tables to bobr;
ALTER DEFAULT PRIVILEGES
yellowbrick=> \ddp
                            Default access privileges
           Owner            | Schema | Type  |          Access privileges
```

```
------------------------------+--------+-------+----------------------------------------
 ybdadmin@yellowbrickcloud.com | public | table | bobr=a/"ybdadmin@yellowbrickcloud.com"
(1 row)
```

In this case, `bobr` has `INSERT` privilege (`bobr=a`) on tables owned by `ybdadmin` . See also ybsql Slash (\) Commands.

Now log in as `bobr` and grant `SELECT` privileges to user `ybsql10` :

```
yellowbrick=> alter default privileges in schema public grant select on tables to ybsql10;
ALTER DEFAULT PRIVILEGES
yellowbrick=> \ddp
                            Default access privileges
            Owner             | Schema | Type  |           Access privileges
------------------------------+--------+-------+---------------------------------------
 bobr                         | public | table | ybsql10=r/bobr
 ybdadmin@yellowbrickcloud.com | public | table | bobr=a/"ybdadmin@yellowbrickcloud.com"
(2 rows)
```

Granting Schema-Level Privileges to Members of a Role

When you are setting up users and roles that will require access to specific schemas and tables, you may need to use a combination of `GRANT` commands and the `ALTER DEFAULT PRIVILEGES` command. This combination guarantees access to both existing objects and new objects when they are created.

In this example, `dbuser1` is the owner of the database `newdb` . The users `dbuser2` and `dbuser3` belong to a role named `dbrole` .

```
premdb=# yellowbrick=> create user dbuser1 login password 'dbuser1' role "ybdadmin@yellowbrickcloud.com";
CREATE ROLE
premdb=# create user dbuser2 login password 'dbuser2';
CREATE ROLE
premdb=# create user dbuser3 login password 'dbuser3';
CREATE ROLE
premdb=# create role dbrole nologin role dbuser2, dbuser3;
CREATE ROLE
premdb=# create database newdb with owner=dbuser1;
CREATE DATABASE
```

Members of `dbrole` are granted permission to connect to `newdb` and usage on a new schema called `dbo` :

```
newdb=# grant connect on database newdb to dbrole;
GRANT
newdb=# create schema dbo;
CREATE SCHEMA
newdb=# grant usage on schema dbo to dbrole;
GRANT
```

Note: This `GRANT USAGE` statement must be issued explicitly by the schema owner for every applicable schema in the database. If this command is not run, only the schema owner has visibility to the underlying tables, procedures, and views. However, although this privilege allows the role to see objects within the schema, it does not grant `SELECT` on the tables in the schema.

Now `dbuser1` creates a simple table in the `dbo` schema and inserts a value:

```
newdb=# create table dbo.table1(c1 int);
CREATE TABLE
newdb=# insert into dbo.table1 values(1);
INSERT 0 1
newdb=# select * from dbo.table1;
```

```
 c1
----
  1
(1 row)
```

As expected, `dbuser2` cannot select from the new table:

```
newdb=# \c newdb dbuser2
Password for user dbuser2:
You are now connected to database "newdb" as user "dbuser2".
newdb=> select * from dbo.table1;
ERROR:  permission denied for relation table1
```

A quick solution to the problem is for `dbuser1` to grant access to all tables in schema `dbo` for all members of `dbrole`:

```
newdb=> \c newdb dbuser1
Password for user dbuser1:
You are now connected to database "newdb" as user "dbuser1".
newdb=# grant all on all tables in schema dbo to dbrole;
GRANT
newdb=# \c newdb dbuser2
Password for user dbuser2:
You are now connected to database "newdb" as user "dbuser2".
newdb=> select * from dbo.table1;
 c1
----
  1
(1 row)
```

This command is useful, but it only solves the problem for *existing* tables in that database and schema. When `dbuser1` creates a new `dbo` table, `dbuser2` and `dbuser3` will not be able to access it. The solution is to run the following `ALTER DEFAULT PRIVILEGES` command, which allows all members of `dbrole` to access all future tables created by `dbuser1`:

```
newdb=# alter default privileges for user dbuser1 grant all privileges on tables to dbrole;
ALTER DEFAULT PRIVILEGES
newdb=# create table dbo.table2(c1 int);
CREATE TABLE
newdb=# insert into dbo.table2 values(1);
INSERT 0 1
newdb=# \c newdb dbuser3
Password for user dbuser3:
You are now connected to database "newdb" as user "dbuser3".
newdb=> select * from dbo.table2;
 c1
----
  1
(1 row)
newdb=# \c newdb dbuser2
Password for user dbuser2:
You are now connected to database "newdb" as user "dbuser2".
newdb=> select * from dbo.table2;
 c1
----
  1
(1 row)
```

Note that the `ALTER DEFAULT PRIVILEGES` command must be run by the object creator (user `dbuser1` in this case).

ALTER EXTERNAL AUTHENTICATION

Platforms: EE: All cloud platforms

Parent topic: SQL Commands

Enable or disable an external authentication object. Disabled ones will not be considered when matching.

```
ALTER EXTERNAL AUTHENTICATION name SET ENABLED { true | false }
```

ALTER EXTERNAL FORMAT

Alter an external format object.

```
ALTER EXTERNAL FORMAT name
TYPE [mode] file_type
WITH (option value [,option value] ...)
```

For details about `TYPE` and `WITH` options, see CREATE EXTERNAL FORMAT.

For example:

```
premdb=# alter external format premdbs3format type text with (nullmarker 'NULL');
ALTER EXTERNAL FORMAT
```

You must have the correct privileges to run this command. See ON EXTERNAL object.

ALTER EXTERNAL LOCATION

Alter an external location object.

```
ALTER EXTERNAL LOCATION name
[ PATH string ]
[ EXTERNAL STORAGE name ]
[ EXTERNAL FORMAT name ]
```

For example, alter the path and the external storage object for a location:

```
premdb=> alter external location premdbazuredata path 'premdbnew' external storage "premdbAzure";
ALTER EXTERNAL LOCATION
```

You must have the correct privileges to run this command. See ON EXTERNAL object.

You cannot alter the `USAGE` of an external location (`PRIMARY` or `EXTERNAL`).

ALTER EXTERNAL STORAGE

Alter one or more attributes of an external storage object.

```
ALTER EXTERNAL STORAGE name
TYPE S3 | AZURE | AZDL2 | MINIO
ENDPOINT 'endpoint_uri'
[ REGION 'region_string']
[ IDENTITY 'identity_string']
[ CREDENTIAL 'credential_string']
```

You must specify the storage type (`S3` , `AZURE` , `AZDL2` , or `MINIO`) and the endpoint URL.

For information about the `ENDPOINT` , `REGION` , `IDENTITY` , and `CREDENTIAL` options, see CREATE EXTERNAL STORAGE.

For example, alter the endpoint and region for an S3 storage object:

```
premdb=> alter external storage "premdbs3"
type s3 endpoint 'https://s3.us-east-1.amazonaws.com' region 'us-east-1';
ALTER EXTERNAL STORAGE
```

Alter the endpoint for an Azure storage object:

```
premdb=> alter external storage "premdbAzure"
type azure endpoint 'https://ybbobr.blob.core.windows.net/';
ALTER EXTERNAL STORAGE
```

You must have the correct privileges to run this command. See ON EXTERNAL object.

ALTER PROCEDURE

Change the name, schema, or owner of a stored procedure.

```
ALTER PROCEDURE name()
{ RENAME TO new_name() |
  OWNER TO role |
  SET SCHEMA schema }
```

For example, rename a procedure:

```
premdb=# alter procedure proc1() rename to ybsp1;
ALTER PROCEDURE
premdb=# \dfp
                          List of functions
 Schema | Name  | Result data type | Argument data types |       Type
--------+-------+------------------+---------------------+------------------
 public | ybsp1 | void             |                     | stored procedure
(1 row)
```

Now change the owner of the procedure:

```
premdb=# alter procedure ybsp1() owner to bobr;
ALTER PROCEDURE
```

Now change the schema of the procedure and list information about it with the `\dfp` command:

```
premdb=# alter procedure ybsp1() set schema prem;
ALTER PROCEDURE
premdb=# set schema 'prem';
SET
premdb=# \dfp
                          List of functions
 Schema | Name  | Result data type | Argument data types |       Type
--------+-------+------------------+---------------------+------------------
 prem   | ybsp1 | void             |                     | stored procedure
(1 row)
```

ALTER REMOTE SERVER

Make changes to an existing remote server that is used for replication.

```
ALTER REMOTE SERVER name
WITH ([HOST remote_host_name] [,] [SQL_PORT number] )
```

You can change the remote host name or the SQL port number. You cannot change the `NOHOSTNAMECHECK` option; use `CREATE OR REPLACE REMOTE SERVER` to modify that setting.

Examples

```
premdb_rep=# alter remote server ybd_repl_svr
with(sql_port 5432);
ALTER REMOTE SERVER
```

```
premdb_rep=# alter remote server ybd_repl_svr
with(host 'yb100');
ALTER REMOTE SERVER
```

You cannot specify an IP address for the `HOST` value. See also CREATE REMOTE SERVER.

ALTER ROLE

Alter the attributes and privileges of an existing user or role. `ROLE` and `USER` are synonymous keywords in this command.

In general, the ability to alter roles requires the system-level `ALTER ANY ROLE` privilege. To alter roles and set `CREATEROLE` or `NOCREATEROLE` also requires the privilege `CREATE ROLE WITH GRANT OPTION`. To alter roles and set `CREATEDB` or `NOCREATEDB` also requires the privilege `CREATE DATABASE WITH GRANT OPTION`. See ON SYSTEM.

```
ALTER ROLE role_specification [ WITH ] option [ ... ]

where option can be:

     SUPERUSER | NOSUPERUSER
   | CREATEDB | NOCREATEDB
   | CREATEROLE | NOCREATEROLE
   | INHERIT | NOINHERIT
   | LOGIN | NOLOGIN
   | CONNECTION LIMIT connlimit
   | [ ENCRYPTED | UNENCRYPTED ] PASSWORD 'password'
   | VALID UNTIL 'timestamp'

ALTER ROLE name RENAME TO new_name

ALTER ROLE { role_specification | ALL } [ IN DATABASE database_name ]
   SET configuration_parameter { TO | = } { value | DEFAULT } | FROM CURRENT
ALTER ROLE { role_specification | ALL } [ IN DATABASE database_name ]
   RESET configuration_parameter | RESET ALL

where role_specification can be:

role_name | CURRENT_USER | SESSION_USER
```

Note: For system-defined users and roles with ID values less than `16384`, you can only alter the password (no other attributes). Query the sys.user `and` sys.role views to check ID values.

role_specification

Use a valid role name or the current value of the CURRENT_USER or SESSION_USER function.

option

For details about the list of options, see CREATE ROLE.

CAUTION:

If you create and update unencrypted passwords with CREATE ROLE and ALTER ROLE commands, passwords are transmitted in plain text and may appear in log messages.

RENAME

Rename the role.

configuration_parameter

You can set or reset configuration parameters to specific default values for sessions initiated by the specified role.

- `DEFAULT` (or `RESET`) removes the role-specific setting; the role inherits the system-wide default setting in new sessions.
- `RESET ALL` removes all role-specific settings.
- `SET FROM CURRENT` saves the session's current value as the role-specific value.
- `IN DATABASE` sets or removes the parameter for the specific role and database only.

See also the SET, RESET, ALTER SYSTEM, and SHOW commands, also ALTER USER SET DEFAULT_CLUSTER and ALTER USER ALTER PASSWORD_POLICY.

Examples

```txt
premdb=# alter role henry connection limit 10;
```

```txt
premdb=# alter role all set search_path to 'premdb';
```

ALTER SCHEMA

Alter attributes of an existing schema. You can change the owner of a schema, change its name, and set the size of or disable disk quota.

```
ALTER SCHEMA name OWNER TO { new_owner | CURRENT_USER | SESSION_USER }
ALTER SCHEMA name RENAME TO new_name
ALTER SCHEMA name SET MAX_SIZE [=] 'size'
ALTER SCHEMA name SET MAX_SIZE DISABLE
```

OWNER TO

Change the owner of a schema. See CURRENT_USER and SESSION_USER.

RENAME TO

Rename a schema.

SET MAX_SIZE [=] 'size'

Specify the maximum amount of space an individual schema may use, where `'size'` is the number of bytes. `'size'` must be a positive integer with the possible inclusion of units: MB, GB, or TB. If units are included, the command will be accepted with and without a space between the integer and unit.

SET MAX_SIZE DISABLE

Disable the disk usage limit for the specified schema.

For example, rename a schema, then change its owner. The new owner must be an existing database user.

```
premdb=# alter schema premdb rename to newprem;
ALTER SCHEMA
premdb=# alter schema newprem owner to bobr;
ALTER SCHEMA
```

For example, set a quota for a schema:

```
premdb=# alter schema premdb set max_size='1GB';
ALTER SCHEMA
```

ALTER SEQUENCE

Alter attributes of an existing sequence. You can change the name of a sequence, its owner, or its schema. You can also restart a sequence.

```
ALTER SEQUENCE [ IF EXISTS ] name
RESTART [ [ WITH ] number ] |
START [ WITH ] number |
OWNER TO { new_owner | CURRENT_USER | SESSION_USER } |
RENAME TO new_name |
SET SCHEMA new_schema
```

You cannot run an `ALTER SEQUENCE` command on a database that is in `HOT_STANDBY` mode.

IF EXISTS

Alter the sequence if it exists, but do not return an error message if it does not exist.

name

Name of the sequence to alter, optionally schema-qualified.

RESTART

Reset the starting value of a sequence:

- If you do not specify a `RESTART` number, and a `START` value was previously defined for the sequence, it is restarted from that value.
- If you do not specify a `RESTART` number, and no `START` value was previously defined for the sequence, it is restarted from 1024.
- If you specify a number, the sequence restarts from that value. `WITH` is an optional keyword.

START

Set a new starting number for generated sequence values, such as `START 1000000`. When you alter a sequence, this value is a *pending* start value. The new start value takes effect only when you *restart* the sequence with the `RESTART` option. Alternatively, you can provide the starting number in the `RESTART` command itself. (Note that when you *create* a sequence, its `START` value takes effect immediately.)

OWNER TO

Change the owner of the sequence. (See also CURRENT_USER and SESSION_USER.)

RENAME TO

Change the name of the sequence. This name cannot be schema-qualified.

SET SCHEMA

Change the schema that the sequence belongs to.

Examples

Rename a sequence:

```
premdb=# alter sequence public.matchid rename to matchkey;
ALTER SEQUENCE
```

Restart the sequence:

```
premdb=# alter sequence matchkey restart;
ALTER SEQUENCE
premdb=# select nextval('matchkey');
  nextval
------------
 5000001535
(1 row)
```

Alter the `START` value for the sequence and restart it:

```
premdb=# alter sequence matchkey start 10000;
ALTER SEQUENCE
premdb=# alter sequence matchkey restart;
ALTER SEQUENCE
premdb=# select nextval('matchkey');
 nextval
---------
   11263
(1 row)
```

Change the schema for a sequence:

```
premdb=# alter sequence matchkey set schema bobr;
ALTER SEQUENCE
```

```
premdb=# alter sequence if exists matchid set schema bobr;
ALTER SEQUENCE
```

ALTER SYSTEM

Alter the system by setting or resetting the value of a specific configuration parameter.

```
ALTER SYSTEM SET configuration_parameter { TO | = } { value | 'value' | DEFAULT }
ALTER SYSTEM RESET configuration_parameter
ALTER SYSTEM RESET ALL
```

You must be a `sysadmin` user to run this command. You cannot run it inside a transaction block.

ALTER SYSTEM modifies behavior for all future sessions on the system but not current sessions. See Precedence of Settings for Configuration Parameters.

Parameter changes made with ALTER SYSTEM do not take effect until the configuration is reloaded (via PG_RELOAD_CONF() or the database is stopped and restarted. See Configuration Parameters for more details.

Syntax

configuration_parameter
: Specify a system configuration parameter that is modifiable. You can use the SHOW ALL command to return a list of parameters; however, not all of these are modifiable.

value
: Specify a new value for the parameter, using a quoted string or a number, as appropriate for the parameter in question.

DEFAULT
: Set the configuration parameter to its default value (equivalent to using `RESET`).

RESET | RESET ALL
: Reset one or all configuration parameters to their default values. Reset values are subject to the rules under Precedence of Settings for Configuration Parameters.

Examples

The following ALTER SYSTEM command takes effect when the configuration has been reloaded. After the reload, the new setting applies to all sessions for all regular users. The changes will not apply to sessions run by superusers.

```
premdb=# show idle_session_timeout;
 idle_session_timeout
----------------------
 1h
(1 row)
premdb=# alter system set idle_session_timeout to '10min';
WARNING:  Will be effective after the next server configuration reload, or after the next server restart in the case of parameters
ALTER SYSTEM
```

After a reload (or a server restart, though a restart is not needed in this specific case), user `bobr` and all other regular users will see and be subject to the new timeout setting:

```
premdb=# \c premdb bobr
Password for user bobr:
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)
You are now connected to database "premdb" as user "bobr".
premdb=> show idle_session_timeout;
 idle_session_timeout
----------------------
 10min
(1 row)
```

To reset the `idle_session_timeout` parameter to its default value (`1h`), run the following command as a `sysadmin` user:

```
premdb=# alter system reset idle_session_timeout;
WARNING:  Will be effective after the next server configuration reload, or after the next server restart in the case of parameters
ALTER SYSTEM
```

Precedence of Settings for Configuration Parameters

Note the following rules of precedence when you modify system configuration parameters:

- The ALTER SYSTEM command sets values for parameters that are applied *globally* to all future sessions (assuming that a database restart is done for those parameters that require a restart to take effect).
- You can override these global settings for specific users by using ALTER ROLE commands. An ALTER ROLE...SET command sets a parameter value for all sessions that are run by the specified user or role.
- You can override global and user-specific parameters by using SET commands. A SET command sets a parameter value for the current session only. No other sessions are affected.

See also the SET, RESET, ALTER ROLE, and SHOW commands.

ALTER SYSTEM SET CLUSTER

Set either the default cluster for users, or the system cluster for system-generated queries.

Setting the *default cluster* applies to all users who have not set their own default cluster. Any user can run this command.

Setting the *system cluster* applies to all system work, including automatic `ANALYZE` operations, flushing of rows from the row store, and "GC" operations. You must have `sysadmin` privileges to alter the system cluster.

```
ALTER SYSTEM SET { DEFAULT_CLUSTER name | SYSTEM_CLUSTER } name
```

Use double quotes for the cluster name if it contains special characters.

Examples

Set the default cluster:

```
yellowbrick=# alter system set default_cluster "bobr-rc6-cluster1";
ALTER SYSTEM SET DEFAULT_CLUSTER
```

Attempt to set the system cluster as a regular user, then try again as a user with `sysadmin` privileges:

```
premdb=> alter system set system_cluster "bobr-may3-small-cluster";
ERROR:  must be superuser or sysadmin to do SET SYSTEM_CLUSTER
premdb=> \c premdb ybsql_consumer
Password for user ybsql_consumer:
You are now connected to database "premdb" as user "ybsql_consumer".
premdb=> alter system set system_cluster "bobr-may3-small-cluster";
ALTER SYSTEM SET SYSTEM_CLUSTER
```

ALTER TABLE

Modify the attributes of a table.

```
ALTER TABLE [ IF EXISTS ] name
ADD column_name data_type [ column_constraint [ ... ] ] |
ADD table_constraint
ALTER [ COLUMN ] column_name SET DEFAULT expression |
ALTER [ COLUMN ] column_name DROP DEFAULT |
ALTER [ COLUMN ] column_name DROP NOT NULL |
ALTER [ COLUMN ] varchar_column_name VARCHAR(new_length) |
DROP CONSTRAINT [ IF EXISTS ] constraint_name [ RESTRICT | CASCADE ] |
OWNER TO { new_owner | CURRENT_USER | SESSION_USER } |
RENAME column_name TO new_column_name |
RENAME TO new_name |
SET MAX_SIZE [=] 'size' |
SET MAX_SIZE DISABLE |
SET RETENTION_SIZE [=] 'size' |
SET RETENTION_SIZE DISABLE |
SET RETENTION_AGE [=] 'duration' |
SET RETENTION_AGE DISABLE |
SET SCHEMA new_schema |
SET ROWSTORE_SIZE_LIMIT 'size' |
SET ROWSTORE_FULL_ACTION { { 'B' | 'BLOCK' } | { 'C' | 'CANCEL' } | { 'S' | 'SLOW' } }
```

IF EXISTS

Alter the table if it exists, but do not return an error message if it does not exist.

ADD column_name

Add a column to a table. You must specify the data type.

You cannot drop columns from tables. `ALTER TABLE DROP COLUMN` is not supported.

Tables have a limit of 2,000 user-defined columns.

ADD column_constraint

Specify column constraints for added columns, such as `UNIQUE`, `NULL`, and `DEFAULT`; however, you cannot add a column with a `NOT NULL` constraint. See CREATE TABLE for complete column constraint syntax.

Note: When you add a column with a `DEFAULT` constraint, the default value is applied only for new rows that are inserted after the column was added. Existing rows in the table will contain a `NULL` value for that column.

ADD table_constraint

Add a table constraint: `PRIMARY KEY`, `FOREIGN KEY`, or `UNIQUE` (see CREATE TABLE for the complete syntax).

Note: `PRIMARY KEY`, `FOREIGN KEY`, and `UNIQUE` constraints are not enforced.

SET DEFAULT expression

Define a default value for the column, which applies only to new rows that are inserted or updated; existing rows do not change.

DROP DEFAULT

Drop the default value for the column. Make sure the expression is compatible with the data type of the column. This change applies only to new rows that are inserted or updated; existing rows do not change.

DROP NOT NULL

Drop the `NOT NULL` constraint for the column. `NULL` values will be allowed in new rows. This command applies only to columns that were defined `NOT NULL`. (`SET NOT NULL` is not supported.)

varchar_column_name

Specify an existing `VARCHAR` column in the table. The column must be a `VARCHAR` column; you cannot alter column lengths for other data types. You cannot alter the length of a column that is referenced in a view.

VARCHAR(new_length)

The new length of the `VARCHAR` column you want to modify. This number must be greater than the current length of the column. For example, you can change a `VARCHAR(20)` column to a `VARCHAR(50)`.

DROP CONSTRAINT [IF EXISTS] constraint_name [RESTRICT | CASCADE]

Drop a constraint on a table, such as a foreign key constraint. The `CASCADE` option automatically drops objects that depend on the dropped constraint. The `RESTRICT` option (the default) does not drop the constraint if dependent objects exist.

OWNER TO

Change the owner of the table.

RENAME TO

Rename a column in the table or the name of the table itself.

SET MAX_SIZE [=] 'size'

Specify the maximum amount of space an individual table may use, where `'size'` is the number of bytes. `'size'` must be a positive integer with the possible inclusion of units: MB, GB, or TB. If units are included, the command will be accepted with and without a space between the integer and unit.

SET MAX_SIZE DISABLE

Disable the disk usage limit for the specified table.

SET RETENTION_SIZE [=] 'size'

Specify the maximum amount of space an individual table may use before the retention process starts removing old records, where `'size'` is the number of bytes. `'size'` must be a positive integer with the possible inclusion of units: MB, GB, or TB. If units are included, the command will be accepted with and without a space between the integer and unit.

Only some specific system tables accept a retention policy (see Retention policies for system relations for more details). Tables may also have minimum value for the retention size.

SET RETENTION_SIZE DISABLE

Disable the retention policy size limit for the specified table.

Only some specific system tables accept a retention policy (see Retention policies for system relations for more details).

SET RETENTION_AGE [=] 'duration'

Specify the age threshold for an individual table record before the retention process starts removing old records, where `'duration'` is a positive interval representing a duration.

Only some specific system tables accept a retention policy (see Retention policies for system relations for more details). Tables may also have minimum value for the retention age.

SET RETENTION_AGE DISABLE

Disable the retention policy age limit for the specified table.

Only some specific system tables accept a retention policy (see Retention policies for system relations for more details).

SET SCHEMA

Change the schema that the table belongs to.

SET ROWSTORE_SIZE_LIMIT 'size'

Specify a row store size limit for the table, using a number and a unit in single quotes. For example:

- `'1MB'`
- `'10MB'`
- `'50GB'`

A value of `-1` means infinity (equivalent to no limit other than the size of the row store itself). A value of `0` means that rows are immediately flushed for every insert (creating very small shards in the column store file system).

Specified limits are converted into files and rounded down. The default file size is 32MB. For example, a limit of `32GB` converts to a maximum of 1024 32MB files.

The current limit for each table is logged in the sys.table view.

SET ROWSTORE_FULL_ACTION

Specify the action to take when the row store limit for the table is reached. For more details, see Managing the Row Store.

- `'S'` (`'SLOW'`): Insert rows directly into the column store. This option will make sure the data continues to load but performance will not be optimal. This option is the default. When rows start being loaded into the column store instead of the row store, you will see a warning message.
- `'B'` (`'BLOCK'`): Block the insert until a flush is possible, then resume.

  Note: When the `BLOCK` action is set, inserts tracked in the sys.query view remain in the `plan` state.
- `'C'` (`'CANCEL'`): Cancel the insert and return an error.

The current action for each table is logged in the sys.table view.

Examples

Change the schema of a table:

```
premdb=# alter table team set schema private;
ALTER TABLE
```

Add a new column to a table:

```
premdb=# alter table matchstats add column matchid uuid null;
ALTER TABLE
```

Define a default value for a column:

```
premdb=# alter table match alter column matchday set default '2017-10-21';
ALTER TABLE
```

Remove a default value for a column:

```
premdb=# alter table match alter column matchday drop default;
ALTER TABLE
```

Remove a `NOT NULL` constraint from a column:

```
premdb=# alter table newmatchstats alter column seasonid drop not null;
ALTER TABLE
```

Add a `UNIQUE` constraint to a table:

```
premdb=# alter table match add constraint uniquematch unique(matchid);
ALTER TABLE
```

Add a `FOREIGN KEY` constraint to a table:

```
premdb=# alter table city add constraint fk_city_country foreign key (countrycode) references country(code);
```

Increase the length of the `VARCHAR` column `city` in the `team` table. The column was originally defined as `VARCHAR(20)`.

```
premdb=# alter table team alter column city varchar(30);
ALTER TABLE
```

Drop a foreign-key constraint from a table:

```
premdb=# alter table match drop constraint match_seasonid_fkey;
ALTER TABLE
```

Set the row store size limit and the row store full action for the `match` table:

```
premdb=# alter table match set rowstore_size_limit '10GB';
ALTER TABLE
premdb=# alter table match set rowstore_full_action 'C';
ALTER TABLE
premdb=# select * from sys.table where name='match';
-[ RECORD 1 ]--------+------------------------------
table_id             | 16515
database_id          | 16498
name                 | match
schema_id            | 2200
owner_id             | 10
distribution         | hash
sort_key             | [NULL]
distribution_key     | seasonid
cluster_keys         |
```

```
partition_keys       |
is_temp              | f
definition           | [NULL]
is_auto_analyze      | t
auto_analyze_policy  | default_policy
last_analyzed        | 2020-06-01 17:44:35.669725-07
creation_time        | 2020-06-01 11:40:25.059889-07
rowstore_bytes       | 0
rowstore_row_count   | 0
rowstore_disk_usage  | 0
rowstore_size_limit  | 10737418240
rowstore_full_action | C
```

Set a quota for the match table:

```
premdb=# alter table match set max_size='1MB';
ALTER TABLE
```

ALTER USER

Alter the attributes and privileges of an existing user or role. `ROLE` and `USER` are synonymous keywords in this command.

```
ALTER ROLE role_specification [ WITH ] option [ ... ]

where option can be:

     SUPERUSER | NOSUPERUSER
   | CREATEDB | NOCREATEDB
   | CREATEROLE | NOCREATEROLE
   | INHERIT | NOINHERIT
   | LOGIN | NOLOGIN
   | CONNECTION LIMIT connlimit
   | [ ENCRYPTED | UNENCRYPTED ] PASSWORD 'password'
   | VALID UNTIL 'timestamp'

ALTER ROLE name RENAME TO new_name

ALTER ROLE { role_specification | ALL } [ IN DATABASE database_name ]
   SET configuration_parameter { TO | = } { value | DEFAULT } | FROM CURRENT
ALTER ROLE { role_specification | ALL } [ IN DATABASE database_name ]
   RESET configuration_parameter | RESET ALL

where role_specification can be:

role_name | CURRENT_USER | SESSION_USER
```

For details, see ALTER ROLE.

See also ALTER USER SET DEFAULT_CLUSTER and ALTER USER ALTER PASSWORD_POLICY.

ALTER USER SET DEFAULT_CLUSTER

Set the default cluster for a specific user.

```
ALTER USER [ IF EXISTS ] user_name SET DEFAULT_CLUSTER cluster_name
```

This command is useful as an alternative to running a `USE CLUSTER` command. Use double quotes for the cluster name if it contains special characters.

Important: This command does not have the same behavior as a grantable privilege and does not apply the default cluster setting to members of a role. You must set the default cluster for each user separately, regardless of role membership.

Also make sure each user has been granted `USAGE ON CLUSTER` for the default cluster you are setting. (`GRANT USAGE ON CLUSTER` can be set for a role or for individual users.)

Example

Set the default cluster for user `ybsql10` :

```
yellowbrick=# alter user ybsql10 set default_cluster "bobr-rc6-cluster1";
ALTER USER
```

Load Balancing Compute Clusters

To learn about how to use the compute cluster load balancer see Load Balancer for Compute Clusters.

ALTER VIEW

Alter the attributes of an existing view.

```
ALTER VIEW [ IF EXISTS ] name OWNER TO { new_owner | CURRENT_USER | SESSION_USER }
ALTER VIEW [ IF EXISTS ] name RENAME TO new_name
ALTER VIEW [ IF EXISTS ] name SET SCHEMA new_schema
ALTER VIEW [ IF EXISTS ] name SET REPRESENTATION {DEFAULT | TEXT | BINARY}
```

console

IF EXISTS

Alter the view if it exists, but do not return an error message if it does not exist.

OWNER TO

Change the owner of the view.

RENAME TO

Rename the view. The new name cannot be schema-qualified.

SET SCHEMA

Change the schema that the view belongs to.

SET REPRESENTATION

Change the view representaion to use the default configuration, text storage or binary storage.

For example, create a view and rename it:

```
premdb=# create view v1 as select * from season;
CREATE VIEW
premdb=# alter view v1 rename to season_view;
ALTER VIEW
premdb=# \d season_view
          View "public.season_view"
   Column    |         Type          | Modifiers
-------------+-----------------------+-----------
 seasonid    | smallint              |
 season_name | character(9)          |
 numteams    | smallint              |
 winners     | character varying(30) |
```

ALTER WLM PROFILE

Make a workload management (WLM) profile active, change its default pool, or rename it.

```
ALTER WLM PROFILE name
( DEFAULT_POOL name ) |
ACTIVATE [ seconds ] [ WITH CANCEL | WITHOUT CANCEL ] |
RENAME TO name
```

Only superusers and users who have been granted `CONTROL WLM ON SYSTEM` can create, alter, and drop WLM profiles.

PROFILE name

The profile name must match the name of an existing profile in the database.

(DEFAULT_POOL name)

Change the default pool for the named profile. This option must be enclosed in parentheses. The named pool may be an existing resource pool or a resource pool that has not yet been created. However, if the named default pool does not exist, you cannot activate the profile.

ACTIVATE [seconds]

When you activate a profile, the currently active profile is deactivated. You can activate a profile immediately, or after an interval specified in seconds. If you activate immediately, any running queries are canceled. If you specify an interval, active queries have some time to complete before the new profile is activated.

Note: The target of this operation is the current compute cluster, which must be running when the `ACTIVATE` command is run. This command does not persist between cluster restarts and will fail if the target cluster is suspended. See also Creating WLM Profiles.

WITH CANCEL | WITHOUT CANCEL

`WITH CANCEL` (the default): Cancel any running queries when the activation timeout expires. `WITHOUT CANCEL`: cancel the activation itself if queries are still running when the timeout expires. For example, `activate 10 with cancel` means that the system will wait for up to 10 seconds before canceling queries, then it will activate the profile. However, `activate 10 without cancel` means that if queries are not complete after 10 seconds, the activation will fail.

RENAME TO name

Rename a profile.

NOTE

Renaming a profile is disallowed if any compute cluster is currently activated with the profile, or has set the profile in its cluster configuration.

To rename a profile, it must not be in use by any compute cluster.

See ALTER CLUSTER and `ALTER WLM PROFILE name ACTIVATE`

Examples

For example, activate a profile immediately:

```
premdb=# alter wlm profile shortquerybias activate;
ALTER WLM PROFILE
```

Now activate the `maintenance` profile after a waiting period of 60 seconds (`WITH CANCEL` by default):

```
premdb=# alter wlm profile maintenance activate 60;
ALTER WLM PROFILE
```

Note: To activate the `default` profile, you must quote the profile name:

```
premdb=# alter wlm profile "default" activate;
ALTER WLM PROFILE
```

For example, change the default pool for a profile:

```
premdb=# alter wlm profile sqb (default_pool sqbpool);
ALTER WLM PROFILE
```

For example, rename a profile:

```
premdb=# alter wlm profile shortquery rename to shortquerybias;
ALTER WLM PROFILE
```

Query the sys.wlm_active_profile view to see the current profiles in the system and which profile is active.

ALTER WLM RESOURCE POOL

Alter the name, the profile, or other options for a WLM resource pool.

```
ALTER WLM RESOURCE POOL name
(PROFILE name [ options ]) |
RENAME TO name
```

Only superusers and users who have been granted `CONTROL WLM ON SYSTEM` can create, alter, and drop WLM resource pools.

PROFILE name

You can alter the profile that is associated with this pool by naming an existing profile. The profile cannot be set to `null`. A single set of parentheses is required around the profile name and the subsequent list of options (if any).

---

options

For details about the available options you can change, see CREATE WLM RESOURCE POOL. The profile name, if specified, and subsequent options must be enclosed in parentheses.

ALTER commands only change the options you specify; other options remain as they were previously defined. To remove a value for an option that accepts `null`, use `null` as the new value in the ALTER command.

RENAME TO name

You can rename a resource pool.

Examples

For example, alter a resource pool and attach it to a profile:

```
premdb=# alter wlm resource pool bobr_pool (profile bobr_profile);
ALTER WLM RESOURCE POOL
```

For example, alter two settings for a resource pool:

```
premdb=# alter wlm resource pool bobr_pool (slots 5, queue_size 10);
ALTER WLM RESOURCE POOL
```

For example, rename a resource pool:

```
premdb=# alter wlm resource pool longpool rename to longest_pool;
ALTER WLM RESOURCE POOL
```

ALTER WLM RULE

Alter either the name or the options set for a WLM rule.

```
ALTER WLM RULE name
(PROFILE name [ options ]) |
RENAME TO name
```

Only superusers and users who have been granted `CONTROL WLM ON SYSTEM` can create, alter, and drop WLM rules.

PROFILE name

You can alter the profile that is associated with this rule by naming an existing profile. The profile cannot be set to `null`. A single set of parentheses is required around the profile name and the subsequent list of options (if any).

options

For details about the available options you can change, see CREATE WLM RULE. The profile name, if specified, and subsequent options must be enclosed in parentheses.

ALTER commands only change the options you specify; other options remain as they were previously defined. To remove a value for an option that accepts `null`, use `null` as the new value in the ALTER command.

Examples

For example, set the profile for a rule:

```
premdb=# alter wlm rule fastlane (profile bobr_profile);
ALTER WLM RULE
```

For example, rename a rule:

```
premdb=# alter wlm rule fastlane rename to freeway;
ALTER WLM RULE
```

BEGIN

Yellowbrick Documentation > Reference > SQL Reference > SQL Commands > BEGIN

Platforms: All platforms

Parent topic: SQL Commands

Start a transaction block.

```
BEGIN [ WORK | TRANSACTION ]
```

SQL statements that follow a BEGIN statement are executed in a single transaction until an explicit COMMIT or ROLLBACK command is executed. `WORK` and `TRANSACTION` are both optional keywords; they have no effect.

`START TRANSACTION` is accepted alternative syntax for this command.

Explicit transactions are useful because they offer a consistent view of the database to other users and sessions when you are making a series of related changes. The complete set of changes is visible when the transaction commits, but no intermediate states are visible.

By default, Yellowbrick automatically commits transactions one SQL statement at a time. Client tools should never turn off automatic commit; turning it off may create downstream blocking of transactions for other users in the system.

The only supported transaction isolation level is `read committed`. Any user can set the transaction isolation level to `read committed`.

For example, drop five tables in one transaction:

```
premdb=# begin;
BEGIN
premdb=# drop table season;
DROP TABLE
premdb=# drop table team;
DROP TABLE
premdb=# drop table hometeam;
DROP TABLE
premdb=# drop table awayteam;
DROP TABLE
premdb=# drop table match;
DROP TABLE
premdb=# commit;
COMMIT
```

Setting Transactions to Read-Only

You can control read-only access for users at the session level by setting the `default_transaction_read_only` command. When this parameter is set to `ON` in a session, the user is restricted to running queries and cannot create or modify objects. For example:

```
premdb=# create table t1(c1 int);
CREATE TABLE
premdb=# drop table t1;
DROP TABLE
premdb=# set default_transaction_read_only to on;
SET
```

```
premdb=# create table t1(c1 int);
ERROR:  cannot execute CREATE TABLE in a read-only transaction
```

CALL

Yellowbrick Documentation > Reference > SQL Reference > SQL Commands > CALL

Platforms: All platforms

Parent topic: SQL Commands

Execute a stored procedure.

```
CALL procedure( [ argvalue [, ...] ] )
```

For example, call a procedure that has no arguments:

```
premdb=# call proc1();
CALL
```

The empty parentheses are required.

If the procedure was created with arguments, call the procedure with appropriate values for those arguments. For example, the procedure `addteam` has one `VARCHAR` argument:

```
premdb=# call addteam('Brighton');
CALL
```

If the procedure was created with arguments that have default values, you can call the procedure with empty parentheses. The default values will be used.

See also CREATE PROCEDURE.

CANCEL

Cancel a running query.

```
CANCEL query_id
```

For example:

```
premdb=# cancel 22669;
CANCEL
```

You can select the `query_id` from the `sys.query` system view, or you can look it up in Yellowbrick Manager (go to **Query Activity**). See also Canceling Queries.

CLOSE

Yellowbrick Documentation > Reference > SQL Reference > SQL Commands > CLOSE

Platforms: All platforms

Parent topic: SQL Commands

Close a cursor.

```
CLOSE { name | ALL }
```

You can close all open cursors or a specific cursor that you name. After you have closed a cursor, no subsequent operations are allowed on it. Close a cursor when it is no longer needed.

See also DECLARE, FETCH, and MOVE cursor.

Examples

For example, close a named cursor within a script:

```
begin;
declare match cursor for select * from match order by matchday;
fetch next from match;
...
close match;
end;
```

Close all currently open cursors:

```
premdb=# close all;
CLOSE CURSOR ALL
```

COMMENT ON

Comment on a database object.

```
COMMENT ON object name IS 'comment'
COMMENT ON CONSTRAINT name ON table IS 'comment'
```

object

One of the following: `COLUMN`, `DATABASE`, `ROLE`, `SCHEMA`, `SEQUENCE`, `TABLE`, `VIEW`, `KEY`, `PROCEDURE`.

name

Database object name. Use `table.name` for column names, such as `match.ftscore`.

If a stored procedure has arguments, you must identify them in the procedure name. For example, either of the following names would work for a procedure `addteam` with one `VARCHAR(30)` argument:

```
addteam(varchar(30))
addteam(name varchar(30))
```

If a stored procedure does not have any arguments, specify its name with parentheses. For example:

```
droptable()
```

CONSTRAINT name ON table

For constraints, specify the table name as well as the constraint name.

'comment'

The text string that you want to save for the named object. Use `null` to remove an existing comment. Use a new `COMMENT ON` command to modify a comment. You can see the current comment for an object by using the appropriate backslash command with the plus sign, such as `\d+` or `\l+`.

Do not put sensitive information in comments; any user can see comments on shared objects such as databases and roles. In general, only object owners and superusers can comment on objects. You must be a superuser to comment on a superuser role. You must have `CREATEROLE` privilege to comment on non-superuser roles.

Examples

Comment on a table and a view:

```
premdb=# comment on table awayteam is 'Just a reference for atid in the match table';
COMMENT
premdb=# comment on view matchview is 'View of match table';
COMMENT
premdb=# \d+
```

```
                              List of relations
 Schema |   Name    | Type  |  Owner  |                Description
--------+-----------+-------+---------+-------------------------------------------
 public | awayteam  | table | brumsby | Just a reference for atid in the match table
 public | hometeam  | table | brumsby |
 public | match     | table | brumsby |
 public | matchview | view  | brumsby | View of match table
 public | season    | table | brumsby |
 public | team      | table | brumsby |
(6 rows)
```

Comment on a table column:

```
premdb=# comment on column match.ftscore is 'Full-time score';
COMMENT
premdb=# \d+ match
                      Table "public.match"
  Column  |            Type             | Modifiers |   Description
----------+-----------------------------+-----------+-----------------
 seasonid | smallint                    |           |
 matchday | timestamp without time zone |           |
 htid     | smallint                    |           |
 atid     | smallint                    |           |
 ftscore  | character(3)                |           | Full-time score
 htscore  | character(3)                |           |

Distribution: Hash (seasonid)
```

Comment on a stored procedure:

```
premdb=# comment on procedure addteam(varchar(30)) is 'Stored proc with varchar arg';
COMMENT
```

Comment on a primary key constraint for a table:

```
premdb=# comment on constraint match_pkey on match is 'Primary key constraint on seasonid';
COMMENT
```

COMMIT

Commit the current transaction block.

```
COMMIT [ WORK | TRANSACTION ]
```

`WORK` and `TRANSACTION` are both optional keywords; they have no effect.

See also BEGIN.

For example, commit the results of two INSERT statements as part of a single transaction:

```
premdb=# begin;
BEGIN
premdb=# insert into awayteam select atid,name from team;
INSERT 0 50
premdb=# insert into hometeam select htid,name from team;
INSERT 0 50
premdb=# commit;
COMMIT
```

COPY

Copy data *to* a table from `STDIN`, or copy data *from* a table or query to `STDOUT`. Any user can copy data with this command. See also ybsql \copy Command, which you can use to copy data to and from files.

Syntax

```
COPY table_name [ ( column_name [, ...] ) ]
   FROM STDIN [ [ WITH ] ( option [, ...] ) ]

COPY { table_name [ ( column_name [, ...] ) ] | ( query ) }
   TO STDOUT [ [ WITH ] ( option [, ...] ) ]

where option is:

   FORMAT text | csv | binary
   DELIMITER 'delimiter_character'
   NULL 'null_string'
   HEADER [ 'true' | 'false' ]
   QUOTE 'quote_character'
   ESCAPE 'escape_character'
   FORCE_QUOTE { ( column_name [, ...] ) | * }
   FORCE_NULL ( column_name [, ...] )
   FORCE_NOT_NULL ( column_name [, ...] )
   ENCODING 'encoding_name'
```

Note: The options list is enclosed in parentheses and separated by commas. The `WITH` keyword is not required.

For example:

```
premdb=# copy season to stdout;
1   1992-1993   22  Manchester United
2   1993-1994   22  Manchester United
3   1994-1995   22  Blackburn Rovers
4   1995-1996   20  Manchester United
5   1996-1997   20  Manchester United
6   1997-1998   20  Arsenal
...
```

```
premdb=# copy season to stdout with (format csv);
1,1992-1993,22,Manchester United
2,1993-1994,22,Manchester United
3,1994-1995,22,Blackburn Rovers
4,1995-1996,20,Manchester United
5,1996-1997,20,Manchester United
6,1997-1998,20,Arsenal
...
```

CREATE CLUSTER

Create a new compute cluster or replace an existing cluster.

```
CREATE [ OR REPLACE ] [ COMPUTE ] CLUSTER [ IF NOT EXISTS ] name
WITH (
[ NODE_COUNT number, ]
[ DEFAULT_CLUSTER [ TRUE | FALSE ], ]
[ HARDWARE_INSTANCE name, ]
[ WLM_PROFILE name,]
[ AUTO_SUSPEND number | NULL,]
[ AUTO_RESUME TRUE | FALSE,]
[ MAX_SPILL_PCT number | NULL,]
[ INITIALLY_SUSPENDED ]
[ TIMEOUT timeout_seconds ]
)
```

OR REPLACE

Create a new compute cluster, or replace an existing compute cluster if one exists with the same name within the same instance.

COMPUTE

Optional keyword; in the current release, all clusters are compute clusters. They do the work of executing queries and other operations such as bulk loads.

name

Unique name for a cluster within an instance. Use double quotes for the cluster name if it contains special characters.

NODE_COUNT

The number of compute nodes assigned to the cluster.

DEFAULT_CLUSTER

Whether this cluster is declared the default cluster for the instance. Only one cluster within a given instance may be the default cluster.

HARDWARE_INSTANCE

Hardware instance type: `small-v1`, `small-v2`, `large-v1`, or `large-v2`. The cluster nodes must have the same hardware instance type.

WLM_PROFILE

WLM (workload management) profile associated with the cluster. A default profile is assigned when a cluster is created.

AUTO_SUSPEND

The number of seconds of user inactivity, after which the cluster is automatically suspended.

AUTO_RESUME

Whether to automatically resume the cluster when a new user query is submitted against the cluster (and it is suspended).

MAX_SPILL_PCT

Maximum disk space allocated for spilling as a percentage of total disk space.

INITIALLY_SUSPENDED

Put the cluster in suspended state on creation. The user must resume the cluster when it is ready for use.

TIMEOUT timeout_seconds

The number of seconds before the operation times out.

Examples

The following example creates a compute cluster called `premdb-queries` with:

- One node
- The default WLM profile
- The small hardware instance type (`small-v1`)
- Automatic suspension after 300 seconds
- Automatic resumption when a new query is run against the cluster

```
CREATE CLUSTER "premdb-queries" WITH (
NODE_COUNT 1,
WLM_PROFILE 'default',
HARDWARE_INSTANCE 'small-v1',
AUTO_SUSPEND 300,
AUTO_RESUME TRUE
);
CREATE COMPUTE CLUSTER
```

The following example creates a compute cluster called `April12_LargeV1_2Nodes` with:

- 2 nodes
- The `flex` WLM profile
- A large hardware instance (`large-v1`)
- Automatic suspension after 30 seconds
- No automatic resumption when a new query is run against the cluster
- An initial suspended state (not running on creation)

```
CREATE CLUSTER "April12_LargeV1_2Nodes" WITH (
  NODE_COUNT 2
, WLM_PROFILE 'flex'
, HARDWARE_INSTANCE 'large-v1'
, AUTO_SUSPEND 30
, AUTO_RESUME FALSE
, INITIALLY_SUSPENDED
);
CREATE COMPUTE CLUSTER
```

CREATE DATABASE

Create a new physical database with create-time encoding and locale plus additional connection properties.

```
CREATE DATABASE name
   [ [ WITH ] [ OWNER [=] user_name ]
          [ ENCODING [=] encoding ]
          [ LC_COLLATE [=] lc_collate ]
          [ LC_CTYPE [=] lc_ctype ]
          [ ALLOW_CONNECTIONS [=] { true | false } ] ]
          [ CONNECTION LIMIT number ]
          [ HOT_STANDBY ON ]
```

name

Database name. Names longer than 128 characters are allowed but truncated. See also SQL Identifiers. Space characters are not allowed in database names.

Note: You can also use Yellowbrick Manager to create databases. When you create a database in Yellowbrick Manager, the database name is implicitly quoted and the case of the name is preserved. In `ybsql`, you need to use an explicit quoted identifier to create a database with a case-sensitive name.

OWNER

Define the user who owns the database.

ENCODING

Define the database encoding. Only the `LATIN9` and `UTF8` encodings are supported. The default encoding is `LATIN9`. For multi-byte character sets, use the `UTF8` encoding. If your data is all single-byte, use the `LATIN9` encoding, which provides better query performance.

Note: To load UTF-16 data with `ybload`, create a `UTF8` database and set the `ybload --encoding` option to `UTF16`. See ybload Options.

LC_COLLATE

Collation order to use in the new database. This affects the sort order applied to strings. For example, in queries with `ORDER BY`, the default is `C`. For UTF8 databases you typically want a value that matches your `LC_CTYPE`. See Usage Notes below for additional details.

LC_CTYPE

Character classification to use in the new database. This affects the categorization of characters. For example, lower, upper and digit. The default is `C`. In UTF8 databases, if you need case sensitive comparison of multi-byte characters, you want to explicitly set a locale. The default `C` encoding means that functions that rely on character classification will respect the ANSI C character classification. See Usage Notes below for additional details.

ALLOW_CONNECTIONS

Specify whether the database accepts user connections: `true` or `false`.

CONNECTION LIMIT

Set a limit on the number of concurrent database connections by regular users (superuser connections do not count). Some of these connections may be used up by internal database services.

HOT_STANDBY ON

Create a database that is ready for use as a target for restore operations or replication.

Note: You can run queries that reference tables in different databases. See Cross-Database Queries.

Usage Notes

To create a database, you must be a superuser or have the special `CREATEDB` privilege. See CREATE USER.

You can set the `LC_COLLATE` and `LC_CTYPE` database attributes only at the time of creation and only via SQL.

- Once created, the encoding of a database cannot be changed.
- The SMC does not support setting the locale during database creation.
- If you have created a database with the default `C` encoding but need to change it, contact Yellowbrick Technical Support.

You can create a maximum of 1,000 databases per appliance. (The database `yellowbrick` is included in this number.). A very large number of databases is not recommended. Each database creates additional overhead and and is often an indicator that databases are being used in place of schemas which is not a best practice.

Because Yellowbrick supports cross database statements, care needs to be taken when mixing `LATIN9` and `UTF8` databases in a single statement to avoid "`invalid UTF8 character`" errors. In a cross database statement, the database from which the statement is executed determines the encoding that is used. Yellowbrick does not do automatic transcoding (For example, converting of high-bit ASCII to UTF8 characters and vice versa).

If you change the `LC_COLLATE` or `LC_CTYPE` for any database used in cross-database statements, you want to be consistent across all databases involved in those queries. Using different `LC_COLLATE` or `LC_CTYPE` locales can result in inconsistent behavior between databases and unexpected behavior from cross database statements.

Use locales only if you actually need them. Using locales other than the default of `C` for `LC_COLLATE` or `LC_CTYPE` does have a performance impact on character handling scenarios with ordering or case sensitive functions. The drawback of not using them is that case conversion and ordering of multi-byte characters may not occur. For example, when using the default of locale of `C` with Danish UTF8 characters, `LOWER(Å)` returns `Å`, not `å`.

The valid locales for `LC_COLLATE` and `LC_CTYPE` are the supported locales on the Yellowbrick manager node. You can list them using the Linux command `locale -a`. Several behaviours follow the locale settings. In particular, the locale affects:

- Case-insensitive matching and the classification of characters by character-class regular expressions. For example, pattern matching operators including `LIKE`, `SIMILAR TO`, and POSIX-style regular expressions.
- Sort order in queries containing an `ORDER BY` clause or the standard comparison operators on textual data.
- The `UPPER`, `LOWER`, and `INITCAP` functions.
- The `TO_CHAR` family of functions.

All other comparisons including joining, grouping, and aggregates use a simple binary comparison.

Examples

Create a database with the default `LATIN9` encoding specifying the owner.

```
yellowbrick=# create database mydb with owner=yb_dbo;
```

Create a `UTF8` database with an explicit collation and ctype.

```
yellowbrick=# create database mydb with encoding=utf8 lc_ctype='en_US.UTF-8' lc_collate='en_US.UTF-8' ;
```

Create a database that does not allow connections to it.

```
yellowbrick=# create database mydb allow_connections=false;
```

Create a database with a maximum limit of 100 concurrent connections.

```
yellowbrick=# create database mydb connection limit 100;
```

Create a database in hot_standby mode.

```
yellowbrick=# create database premdb_hot with hot_standby on;
```

You cannot create a database with an encoding other than `LATIN9` or `UTF8`.

```
yellowbrick=# create database shakespeare encoding=LATIN1;
ERROR: Only LATIN9 and UTF8 database encodings are supported
```

CREATE EXTERNAL AUTHENTICATION

Configures a Yellowbrick Instance to authenticate with an external IDP. The information supplied in this command tells the instance how to validate a supplied JWT token, and how to map the named claim to a database username.

Authentication properties are stored in the system view `sys.external_authentication`.

```
CREATE EXTERNAL AUTHENTICATION name
ISSUER 'string'
USER_MAPPING_CLAIM 'string'
GRANT (role_name [, role_name] …)
GRANT_CLAIM 'grant_claim'
[ AUDIENCE ('string', [, 'string'] ...) ]
[ AZP 'string' ]
[ ALLOWED_ROLES (role_name [, role_name] …) ]
[ DISALLOWED_ROLES (role_name [, role_name] …) ]
[ KEY 'public_key' ]
[ AUTO_CREATE true | false ]
[ ENABLED true | false ]
```

ISSUER

The issuer identifier established in the IDP integration, created in Yellowbrick Manager or through Kubernetes CRDs. Typically this is the URL of the authorization server that issues the JWT. Required.

USER_MAPPING_CLAIM

A property-name in the payload of the JWT whose value is to map to a Yellowbrick database user. For example, `email` or `preferred_username`. Required.

GRANT

The names of Yellowbrick database roles that this database user should be a member of when auto-creating a role. Optional if there is a `GRANT_CLAIM`. The database roles need not be present when issuing this command.

GRANT_CLAIM

The name of a claim in the JWT that holds a list of database role names that the auto-created user is made a member of. Optional as long there is a `GRANT`.

AUDIENCE

A list of strings, any of which that must match the `aud` claim of the JWT. Optional. During JWT validation, one of the strings supplied in here must match. Note that if the `aud` claim in the JWT is a list of strings, then any of those strings must match the ones supplied here.

AZP

Name of the authorized party that must be present in the `azp` claim of the JWT. Optional.

ALLOWED_ROLES

List of primary login roles that will pass authentication. Optional. This check is skipped if the list is empty.

DISALLOWED_ROLES

List of primary login roles that will fail authentication. Optional. Can be used to selectively disallow named login roles.

AUTO_CREATE

If true, then automatically create the database user if it does not exist. Optional. If not specified, defaults to false.

ENABLED

If true, then this authentication is active. Optional. If not specified, defaults to false.

PUBLIC_KEY

The public key of the issuer for verification of JWTs. Optional. Normally the public key is fetched from the issuer on demand when validating JWTs; specifying a key here disables this behavior.

CREATE EXTERNAL LOCATION

Platforms: EE: All cloud platforms, CE

Parent topic: SQL Commands

Create an external location that you can either use for primary database storage or that you can reference when you load a table from external storage. This object defines either the target storage container for table data or the location of source files that you intend to load.

```
CREATE EXTERNAL LOCATION [ IF NOT EXISTS ] name
PATH 'path'
EXTERNAL STORAGE storage_name
[ EXTERNAL FORMAT format_name ]
[ USAGE PRIMARY DEFAULT | EXTERNAL ]
```

IF NOT EXISTS

Create the object if it does not already exist. If it does exist, do not create it and do not return an error.

PATH

Name of the S3 bucket or Azure container that contains (or will contain) source files for the load. For example:

```
path 'yb_premdb'
```

Do not specify any sub-folders or slash characters, just the bucket or container name. (If folders exist inside the bucket, you can specify them as part of the prefix string in the `LOAD TABLE` command.)

EXTERNAL STORAGE

Name of an external storage object. See CREATE EXTERNAL STORAGE. This parameter is required.

EXTERNAL FORMAT

Name of an external format object. See CREATE EXTERNAL FORMAT. The format you specify will be used as the default format for parsing files for this external location object. This parameter is optional; when the table is loaded, the format can be defined with a set of load-specific options that override the default format.

USAGE PRIMARY DEFAULT | EXTERNAL

Whether the external location (and associated object storage) is intended for storing persistent table data or loading and unloading (data movement). A `USAGE PRIMARY DEFAULT` location will be used to store all the table data in the databases that belong to an instance. A `USAGE EXTERNAL` location will be used for loading and unloading data. If you intend to create a `USAGE PRIMARY DEFAULT` location, you must first deselect the **Create initial external storage** option in Yellowbrick Manager when you create the instance. See Configuring Primary Storage in AWS.

You can use the same primary storage location for multiple data warehouse instances.

You must have the correct privileges to run this command. See ON EXTERNAL object.

Examples

Create an external location object that references an external storage object:

```
premdb=> create external location premdbazuredata
path 'premdb' external storage "premdbAzure";
CREATE EXTERNAL LOCATION
```

Drop an external location object and re-create it with the optional external format now specified:

```
premdb=> drop external location if exists premdbs3data;
DROP EXTERNAL LOCATION

premdb=> create external location premdbs3data
path 'ybpremdb'
external storage premdbs3
external format premfiles;
CREATE EXTERNAL LOCATION
```

Create an external location object that is used for primary database storage:

```
yellowbrick=# create external location azurevpc
path 'ybownbucket'
external storage azurevpc_storage
usage primary default;
CREATE EXTERNAL LOCATION
```

Configuring Primary Storage in AWS

Primary storage is per-instance object storage that is used for storing the data in Yellowbrick tables and other database objects. If you want to use your own primary storage, as opposed to allowing Yellowbrick to create that storage for you when you create an instance, follow these steps:

1. In the AWS console, using the same account you used to install the VPC, create a new bucket. In the **Permissions** tab, add the following policy, where `ybownbucket` is the bucket name:

```
{
    "Version": "2012-10-17",
    "Id": "EnableBucketEncryption",
    "Statement": [
        {
            "Sid": "Deny non-TLS",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:*",
            "Resource": [
                "arn:aws:s3:::ybownbucket",
                "arn:aws:s3:::ybownbucket/*"
            ],
            "Condition": {
                "Bool": {
                    "aws:SecureTransport": "false"
                }
            }
        }
    ]
}
```

2. Create a new data warehouse instance with the **Create initial external storage** option *deselected*. (The default is selected.)
3. Before creating any clusters, databases, or tables in this instance, create an external storage object and an external location object that references the storage object. Connect to the instance you just created (you do not need a cluster). For example:

```
create external storage inst4ownbucket
type S3
endpoint 'https://s3.us-east-1.amazonaws.com'
region 'us-east-1'
identity '******************'
credential '************************************';

create external location inst4awsvpc
path 'ybownbucket'
external storage inst4ownbucket
usage primary default;
```

4. Start creating clusters, databases, and tables in the new instance. The bucket named in the `PATH` for the `CREATE EXTERNAL LOCATION` object will be used to store Yellowbrick data.

Configuring Primary Storage in Microsoft Azure

CREATE EXTERNAL MOUNT

Create an NFS mount for use with external tables.

```
CREATE EXTERNAL MOUNT [ IF NOT EXISTS ] 'reference_name'
AS 'fully_qualified_nfs_path'
[ WITH ( option value [, ...] ) ]
READONLY | WRITEONLY
```

You must be a superuser to run this command.

The following options are supported in the `WITH` clause:

- `GID <gid>` : Group ID to use when mounting and interacting with the NFS volume
- `UID <uid>` : User ID to use when mounting and interacting with the NFS volume

By default, the command runs a verification check that the NFS volume allows both read and write access. If you want to limit this check to verifying read-only access or write-only access, specify `READONLY` or `WRITEONLY` at the end of the command (*not* as a `WITH` option).

Read-only access to a mount will work for `SELECT FROM EXTERNAL` (`ybload`) operations; external table files can be selected from in queries. Write-only access to a mount will work for `CREATE EXTERNAL TABLE` (`ybunload`) operations; external table files can be written and listed.

The command will either complete successfully or return an `"Insufficient permissions"` error.

Note: Make sure that you have an external mount available that allows external table log files to be written to it. If you are using a read-only mount for running external table queries, you still need to be able to write the `ybload` logs for those queries to a writable mount directory. You can use the `logdir` option in the `CREATE EXTERNAL TABLE` command or the `SELECT FROM EXTERNAL` command to specify a writable mount directory.

Examples

Create a new external mount and verify that it is available for reads and writes:

```
premdb=# create external mount '/qumulo/ext_db/' as
'nfs://qumulo:/data/bobrum/ext_db/'
with (gid 1234, uid 4567);
```

Drop this mount and re-create it, but this time verify only that the mount is available for writes:

```
premdb=# drop external mount '/qumulo/ext_db/';
DROP EXTERNAL MOUNT
premdb=# create external mount '/qumulo/ext_db/' as
'nfs://qumulo:/data/bobrum/ext_db/'
with (gid 1234, uid 4567)
writeonly;
```

CREATE EXTERNAL STORAGE

Platforms: EE: All cloud platforms, CE

Parent topic: SQL Commands

Create an external storage object, which can be referenced in the definition of specific external storage *locations*. An external storage object contains metadata for connecting to a specific object store, including the type, endpoint, and access credentials.

```
CREATE EXTERNAL STORAGE [ IF NOT EXISTS ] <name>
TYPE <storage_type>
[ ENDPOINT 'endpoint_uri']
[ REGION 'region_string']
[ IDENTITY 'identity_string']
[ CREDENTIAL 'credential_string']
[ WITH ADVANCED (<name> <value> [,<name> <value>]...) ]
```

IF NOT EXISTS

Create the object if it does not already exist. If it does exist, do not create it and do not return an error.

TYPE

Storage type: `S3`, `AZURE`, `AZDL2`, or `MINIO`

ENDPOINT

S3 endpoint URIs must be of the following form, using `https://` or `http://`:

```
https://s3.us-east-1.amazonaws.com
```

or:

```
http://s3.us-west-1.amazonaws.com
```

Note that these endpoints are standard region-based S3 endpoints that begin with `https://s3.` or `http://s3.`

These endpoint URIs are not prefaced with a bucket name. See the AWS Documentation for a complete list of regional endpoints.

Azure and AZDL2 endpoint URIs must be of the following form:

```
https://<storage_account_name>.blob.core.windows.net/
```

REGION

For example, in S3: `us-east-2` or `us-west-1`

For Azure object storage, do not specify a region.

IDENTITY

Access key ID for S3 endpoints, or a storage account name for Azure.

If you are going to load from a public bucket or container, you can connect anonymously and do not need to specify this option.

CREDENTIAL

Secret access key for S3 or Azure endpoints. If you are going to load from a public bucket or container, you can connect anonymously and do not need to specify this option.

WITH ADVANCED

Advanced configuration options that vary by object store provider. For AWS S3, you can specify the following Boolean options:

- Default to `false`:

```
dualStackEnabled
accelerateModeEnabled
pathStyleAccessEnabled
```

- Default to `true`:

```
checksumValidationEnabled
chunkedEncodingEnabled
```

For example:

```
with advanced(dualStackEnabled true, chunkedEncodingEnabled true)
```

You must have the correct privileges to run this command. See ON EXTERNAL object.

Note: If the type of external storage is `AZURE` or `AZDL2`, then the SAS token should be generated at the storage account level, not at the container level.

Examples

Create an Azure external storage object with login credentials:

```
create external storage "premdbAzure" type azure
endpoint 'https://trebor.blob.core.windows.net/'
identity 'trebor'
credential 'MPuQsUdSiyxBSrJxV354dGSUukNgyYwTwrOiXYZ6nSiUWsJMQNMS+HB/LK44dpqI+mgv7vRbAimWuajKIuHI6T==';
```

Create an S3 external storage object with login credentials:

```
create external storage premdbs3
type S3
endpoint 'https://s3.us-east-1.amazonaws.com'
region 'us-east-1'
identity 'ABIA4CDEFG6CUDXUY7NM'
credential '2jr0tnuNRo5Xi0Kh8WxapWDB2S+ojzVc4Jd6AMe4';
```

Create a similar external storage object but for anonymous use:

```
create external storage my_public_s3_bucket
type S3
```

```
endpoint 'https://s3.us-west-2.amazonaws.com'
region 'us-west-2';
```

CREATE KEY

Create a new encryption key in a database.

```
CREATE KEY [ IF NOT EXISTS ] key [ WITH ] SECRET 'hex_string';
```

IF NOT EXISTS

Create the key if it does not already exist. If it does exist, do not create it and do not return an error.

key

A `key` name with a maximum length of 128 bytes. Names longer than 128 bytes are automatically truncated. See SQL Identifiers. Encryption keys are referenced in `ENCRYPT_KS` and `DECRYPT_KS` functions and in `ENCRYPTED WITH` column constraints in `CREATE TABLE`. See also Encrypting Sensitive Data.

hex_string

A valid hexadecimal string with a minimum of 1 character and maximum of 32768 characters. Hexadecimal strings can only contain the following characters: `[0-9]` and `[a-f]`. The `0x` prefix is not supported.

A newly created key is only accessible by the user creating the key. To grant other users permission to use a key, see GRANT.

The `CREATE KEY` operation is non-transactional and will error out when run inside a transaction block.

Note that keys are not backed up during backup operations because they are not actually stored in the database. The remote keystore administrator must make sure keys are backed up and stored safely.

You cannot create keys in one database and reference them in a table that you create in another database.

Examples

For example, create a key named `yb100key` with secret `a1b2c3d4`:

```
premdb=# create key yb100key secret 'a1b2c3d4';
CREATE KEY
```

Create the key `playerkey` if it does not already exist:

```
premdb=# create key if not exists playerkey secret 'a1b2c3d4e5';
CREATE KEY
```

Using an invalid hexadecimal string as a secret will result in the following error:

```
premdb=# create key yb100key secret 'g1h2i3j4';
ERROR:  invalid secret. HINT: secret must be a valid hexadecimal string (i.e. non-empty and without a '0x' prefix) smaller than 32
```

CREATE PROCEDURE

Create a stored procedure, using the `plpgsql` language. For details about `plpgsql`, see the PostgreSQL 9.5 documentation.

```
CREATE [ OR REPLACE ] PROCEDURE name
( [ [ argname ] argtype [ { DEFAULT | = } default_expr ] [, ...] ] )
 [ RETURNS rettype
AS $$
[ DECLARE alias [ aliastype ] [ ALIAS FOR $n ]; [ ... ] ]
BEGIN
  [ body ];
END;
$$ LANGUAGE plpgsql;
```

Note: You cannot create procedures that reference tables in remote databases; see Cross-Database Queries.

PROCEDURE name

You can create multiple procedures with the same name if they have different signatures (arguments and data types). Use empty parentheses after the name if the procedure has no arguments.

You cannot create a stored procedure in a remote database, and you must `SELECT` from (or `CALL`) a stored procedure from the database where it was created. However, remote table references are allowed in the body of the procedure.

Note: Although superusers are allowed to create stored procedures in the `sys` schema, this practice is not recommended. All users should create stored procedures in `public` or a user-defined schema.

[argname] argtype]

The name (or alias) for an argument is optional. Its data type is required. Stored procedures may have arguments that use any of the supported Yellowbrick data types.

DEFAULT | =

Define a value for the argument that will be supplied automatically when the procedure is called without a value for that argument. If an argument has a default value, all of the arguments that follow must have default values.

RETURNS rettype

Return a value or a set of values from the procedure:

- `data_type`: specify the return data type for a return value
- `SETOF table`: return a set of values instead of a single value

Note: A procedure that returns a set must be called with a `SELECT` statement, and the query must not contain table references. For example:

```
select * from proc_set() order by 1;
```

DECLARE

You can declare aliases explicitly here, or you can specify them in the argument list after the procedure name. You can use the `ALIAS FOR $n` syntax if you do not specify a data type for the alias. For example:

```
declare day alias for $1;
```

BEGIN

For detailed information about the syntax that you can use in the body of a `plpgsql` stored procedure, see the PostgreSQL 9.5 documentation. To implement transaction commit and rollback support, use the `BEGIN...EXCEPTION` construct inside the body. The entire outer `BEGIN` block must end with a semicolon.

END

The `END` keyword must be followed by a semicolon.

LANGUAGE plpgsql

Yellowbrick stored procedures support the `plpgsql` language only.

Transaction Control Within PL/pgSQL Procedures

Within procedures executed with the `CALL` command, you can end transactions by using the `COMMIT` and `ROLLBACK` commands. A new transaction is started automatically after a transaction is ended with these commands; there is no separate `START TRANSACTION` command. (Note that `BEGIN` and `END` have different meanings in PL/pgSQL.)

For example:

```
CREATE PROCEDURE transaction_test1()
AS $$
BEGIN
   FOR i IN 0..9 LOOP
       INSERT INTO test1 (a) VALUES (i);
       IF i % 2 = 0 THEN
           COMMIT;
       ELSE
           ROLLBACK;
       END IF;
   END LOOP;
END
$$ LANGUAGE plpgsql;
CREATE PROCEDURE
premdb=# create table test1(a int);
CREATE TABLE
premdb=# CALL transaction_test1();
CALL
```

Examples

For example, create a procedure that runs an `INSERT`:

```
premdb=# create procedure addteam(name varchar(30)) as
$$ begin
insert into team(name) values(name);
end;
$$ language plpgsql;
CREATE PROCEDURE
```

Create (or replace) a similar `INSERT` procedure that takes two parameters and declares aliases:

```
premdb=# create or replace procedure insert_hometeam(integer,varchar(30))
as $$
declare
   htid alias for $1;
   name alias for $2;
begin
   insert into hometeam(htid, name)
   values ($1, $2);
end;
$$ language plpgsql;
CREATE PROCEDURE
```

Create a similar procedure but provide default values for both arguments:

```
premdb=# create or replace procedure
insert_hometeam(integer=1,varchar(30)='Team 1')
...
```

Create a procedure that runs an `UPDATE`:

```
premdb=# create or replace procedure update_match
(timestamp, char(3))
as $$
declare
   day alias for $1;
   score alias for $2;
begin
   update match
   set ftscore=$2
   where matchday>$1;
end;
$$ language plpgsql;
CREATE PROCEDURE
```

Here is an example `CALL` command that runs the `update_match` procedure:

```
premdb=# call update_match('2015-01-01','0-0');
CALL
```

Create and run a simple procedure that returns its own input value:

```
premdb=# create procedure sp2(val int4) returns int as $$
begin
return val;
end$$ language plpgsql;
CREATE PROCEDURE
premdb=# call sp2(20);
CALL
premdb=# select sp2(20);
 sp2
-----
  20
(1 row)
```

Create and run a procedure that selects all rows from the `team` table:

```
premdb=# create procedure sp_team_table() returns setof team as $$
declare
rec record;
begin
for rec in EXECUTE 'select * from team order by 1,2,3,4,5' loop
return next rec;
end loop;
end
$$ language plpgsql;
CREATE PROCEDURE

premdb=# select name, stadium, capacity from sp_team_table()
where capacity <25000 and name is not null
order by 3;
       name         |     stadium      | capacity
--------------------+------------------+----------
 Bournemouth        | Vitality Stadium |    11464
 Oldham Athletic    | Boundary Park    |    13309
 Swindon Town       | County Ground    |    15728
 Blackpool          | Bloomfield Road  |    17338
 Queens Park Rangers | Loftus Road     |    18439
 Swansea City       | Liberty Stadium  |    20520
 Portsmouth         | Fratton Park     |    21100
 Watford            | Vicarage Road    |    21577
 Burnley            | Turf Moor        |    22546
 Barnsley           | Oakwell Stadium  |    23009
 Reading            | Madejski Stadium |    24161
(11 rows)
```

Use the `\dfp` or `\dfp+` command to return information about the stored procedures in the database. For example:

```
premdb=# \dfp
                        List of functions and stored procedures
 Schema |       Name       | Result data type |  Argument data types   |       Type
--------+------------------+------------------+------------------------+------------------
 public | addteam          | void             | name character varying | stored procedure
 public | sp2              | integer          | val integer            | stored procedure
 public | sp_team_table    | SETOF team       |                        | stored procedure
 ...
```

Usage Notes

- When a stored procedure with a return value is executed via a `CALL` statement, the return value is not displayed. If the same procedure is invoked via a `SELECT` statement, it returns the value.
- When a procedure with no return value is executed via a `SELECT` statement, it returns `NULL`.
- A procedure that returns a set must be called with a simple `SELECT` statement, and the query must not contain a `FROM` clause with table references.
- Yellowbrick does not support trigger procedures.
- Yellowbrick does not support collation of `plpgsql` variables.
- Integer data types are not implicitly downcast when they are passed as parameters to stored procedures. For example, a constant value of `2000` is interpreted as an `INTEGER` data type, not as a `SMALLINT`. Therefore, the following attempt to call a stored procedure that was created with a `SMALLINT` argument fails with an error. When the constant is explicitly cast to `SMALLINT`, the `CALL` command succeeds.

```
premdb=# call insert_hometeam(2000);
ERROR:  stored procedure insert_hometeam(integer) does not exist
HINT:  No stored procedure matches the given name and argument types.
premdb=# call insert_hometeam(2000::smallint);
CALL
```

Data types are implicitly upcast where necessary; this behavior only applies to values that require downcasting.

Invocation of Stored Procedures via CALL and SELECT

The following table shows the behavior for different kinds of stored procedures, depending on their structure, return type, and how they are invoked: via `CALL` or `SELECT`. In most cases, you cannot `CALL` a nested procedure (a stored procedure within another stored procedure), but you can use a simple `SELECT` to call a nested procedure.

Procedure	Returns	Invoked with CALL	Invoked with SELECT
Not nested	Nothing (`VOID`)	Allowed	Allowed
	Scalar value	Allowed but result discarded (not displayed)	Allowed and result returned
	A set (`SETOF`)	Disallowed, returns an error	Allowed
Nested with no arguments or constant arguments	Nothing (`VOID`)	Allowed	Allowed
	Scalar value	Disallowed, returns an error	Allowed
	A set (`SETOF`)	Disallowed, returns an error	Allowed
Nested with variable arguments	Nothing (`VOID`)	Disallowed, returns an error	Allowed
	Scalar value	Disallowed, returns an error	Allowed
	A set (`SETOF`)	Disallowed, returns an error	Allowed

For example, the procedure below (`public.raise_v_p`) fails to execute when it is nested and causes its outer procedure to fail. Attempts to `SELECT` or `CALL` `public.echo_n_raise_i_p` both fail because you cannot `CALL` a nested procedure that takes a variable as an argument.

```
CREATE OR REPLACE PROCEDURE public.raise_v_p( _arg_v varchar )
AS
$$
DECLARE
   tmp integer := 0;
BEGIN
   RAISE INFO '%', _arg_v;
END;
$$
LANGUAGE plpgsql ;

CREATE OR REPLACE PROCEDURE public.echo_n_raise_i_p( _arg_i int )
RETURNS text AS
$$
BEGIN
  CALL public.raise_v_p( _arg_i );
  RETURN _arg_i::text;
END;
$$
LANGUAGE plpgsql ;

premdb=# SELECT * FROM public.echo_n_raise_i_p(1);
ERROR:  column "_arg_i" does not exist
CONTEXT:  SQL statement "CALL public.raise_v_p( _arg_i )"
PL/pgSQL function echo_n_raise_i_p(integer) line 3 at SQL statement

premdb=# CALL public.echo_n_raise_i_p(1);
ERROR:  column "_arg_i" does not exist
CONTEXT:  SQL statement "CALL public.raise_v_p( _arg_i )"
PL/pgSQL function echo_n_raise_i_p(integer) line 3 at SQL statement
```

However, the nested procedure will run if you replace the variable with a constant:

```
...
CALL public.raise_v_p( '10' );
...
```

CREATE REMOTE SERVER

Create a remote server on the source system for replication purposes. The remote server is a reference to the target system.

```
CREATE [ OR REPLACE ] REMOTE SERVER name
WITH (HOST remote_host_name [, NOHOSTNAMECHECK true | false ] [, SQL_PORT number])
```

OR REPLACE

If the remote server exists and is replaced, an implicit `DROP REMOTE SERVER` operation occurs. You cannot drop a remote server that is referenced by an existing replica (regardless of whether replication is running).

HOST

The host name of the remote system is required by default. Typically, the host name is the fully qualified domain name (FQDN) for the host, as specified by the common name (CN) value in the SSL certificate that is used to trust the target system. You can use the IP address for the `HOST` value instead of the host name only if the `NOHOSTNAMECHECK` option is set to `true`.

NOHOSTNAMECHECK

Set to `true`, this option disables strict host name checking against installed SSL certificates. In turn, validation will not fail for a wildcard certificate, and an IP address may be specified for the host instead of the host name. For example, if the host name in the installed certificate is `*.paly.ybdatabase.com`, but the host name in the `CREATE REMOTE SERVER` statement is `yb100.paly.ybdatabase.com`, the remote server will still be created successfully and replication operations will operate normally.

If you do not specify this option, or specify `false`, strict host name checking occurs and the FQDN must be used for the `HOST` option.

SQL_PORT

The default SQL port is `5432`. The port number you specify must be open and configured for use on the remote system. See Opening Network Ports for Clients.

Remote servers are global system objects, not database objects. One remote server can support replication for multiple databases and replicas.

Examples

Specify the host name only:

```
yellowbrick=# create remote server test_repl_svr
with (host 'yb100.paly.ybdatabase.com');
CREATE REMOTE SERVER
```

Specify the host name and the SQL port number:

```
premdb=# create remote server ybd_repl_svr
with (host 'my.target.host.io', sql_port 5432);
CREATE REMOTE SERVER
```

Use an IP address for the host and disable strict checking for the host name:

```
premdb=# create or replace remote server replsvr with (host '10.30.22.203', nohostnamecheck true);
CREATE REMOTE SERVER
```

Attempt to use an IP address for the host when strict checking for the host name is enabled. The error is expected in this case.

```
premdb=# create or replace remote server yblocal with (host '10.30.22.203');
ERROR:  Failed to add remote server
DETAIL:  Error validating remote server. The hostname:port 10.30.22.203:5432 specified is configured with a certificate that does
```

CREATE ROLE

Create a database role (or user) and define attributes for the role.

Users and roles are virtually the same thing. You can use a CREATE ROLE statement to create a new role that takes on the privileges granted to an existing role. In that sense, users may become a member of a role and accrue privileges from that role.

CREATE USER syntax is accepted; however, CREATE USER and CREATE ROLE are synonymous, except for one important difference in behavior. When you use CREATE USER, the `LOGIN` attribute is assumed by default. However, `NOLOGIN` is the default when you use CREATE ROLE.

Only users and roles that have `CREATE ROLE` privilege can create other roles and users. This privilege can be inherited or granted:

- Inherited when the `CREATEROLE` option is used in the CREATE ROLE statement
- Granted explicitly with a GRANT CREATE ROLE ON SYSTEM statement

Note the difference between *attributes* assigned to roles when they are created and *grantable privileges* that may be subsequently given to roles or accrued via role membership. With the exception of `CREATEDB/NOCREATEDB` and `CREATEROLE/NOCREATEROLE`, the options defined in the CREATE ROLE statement are attributes, not privileges.

```
CREATE ROLE name [ [ WITH ] option [ ... ] ]

where option can be:

      SUPERUSER | NOSUPERUSER
    | CREATEDB | NOCREATEDB
    | CREATEROLE | NOCREATEROLE
    | INHERIT | NOINHERIT
    | LOGIN | NOLOGIN
    | CONNECTION LIMIT connlimit
    | PASSWORD 'password'
    | VALID UNTIL 'timestamp'
    | IN ROLE role_name [, ...]
    | ROLE role_name [, ...]
    | ADMIN role_name [, ...]
    | USER role_name [, ...]
```

name

Specify the role or user name, using a quoted or unquoted identifier. See SQL Identifiers.

SUPERUSER | NOSUPERUSER

Whether the role can override all access restrictions for the database. Only a superuser can create a new superuser. `NOSUPERUSER` is the default. Assign this attribute only when it is really needed. You cannot grant `SUPERUSER` privilege.

CREATEDB | NOCREATEDB

Whether the role can create databases. `NOCREATEDB` is the default. Ability to create databases is inheritable when users belong to a role that has that privilege.

CREATEROLE | NOCREATEROLE

Whether the role can create new roles (execute CREATE ROLE). A role with this attribute can also alter and drop roles. `NOCREATEROLE` is the default. Ability to create roles is inheritable when users belong to a role that has that privilege.

INHERIT | NOINHERIT

Whether a role inherits the privileges of roles that it is a member of. `NOINHERIT` means that membership in another role only grants the ability to `SET ROLE` to that other role. The privileges of the other role become available only after doing this.

`INHERIT` is the default behavior. `INHERIT` refers to inheritance of *grantable privileges* (system-level privileges, access to database objects, and memberships in roles, `CREATEDB`, `CREATEROLE`). A role with the `INHERIT` attribute can automatically use whatever privileges have been granted to all roles it is directly or indirectly a member of.

LOGIN

Whether a role is allowed to log in (that is, used as the initial session authorization name for a client connection). Think of a role that has `LOGIN` attribute as a *user*. Roles that cannot log in are useful for managing database privileges. `NOLOGIN` is the default, except when the alternative `CREATE USER` syntax is used. `CREATE USER` creates a user (role) with default `LOGIN` capability.

CONNECTION LIMIT limit

For a role with the `LOGIN` attribute, the concurrent connection limit for the role. The default is `-1` (no limit).

PASSWORD 'password'

For a role with the `LOGIN` attribute, define a password for password authentication. If no password is specified, the password is set to null and authentication always fails for this user. `PASSWORD NULL` also sets the password to null.

VALID UNTIL 'timestamp'

Expiration date and time for the password. If you do not set a date and time, the password remains valid for all time.

IN ROLE role_name

One or more existing roles that this role will become a member of. You cannot add the new role as an administrator; use the GRANT command.

ROLE role_name

One or more existing roles that will become members of this role. (The new role effectively becomes a "group.") `USER role_name` is alternative syntax for the `ROLE` clause.

ADMIN role_name

Like the ROLE clause, except that the named roles become members of the new role `WITH ADMIN OPTION`, which means they can grant membership in this role to others.

Examples

Create a role with password authentication and the ability to create other roles and create databases.

```
premdb=# create role ybuser1 createdb createrole login
password '!@#$%900' valid until '2016-12-31 23:59:59';
CREATE ROLE
```

Create a set of `LOGIN` users with the `INHERIT` option.

```
premdb=# create role alex login inherit;
CREATE ROLE
```

```
premdb=# create role vicky login inherit;
CREATE ROLE
premdb=# create role henry login inherit;
CREATE ROLE
premdb=# create role niklas login inherit;
CREATE ROLE
premdb=# \du
                                 List of roles
      Role name        |            Attributes             |       Member of
-----------------------+-----------------------------------+-----------------------
 alex                  |                                   | {}
 henry                 |                                   | {}
 niklas                |                                   | {}
 vicky                 |                                   | {}
...
```

Create a role (or group) to which these four users belong:

```
premdb=# create role allqa nologin createdb role alex, vicky, henry, niklas;
CREATE ROLE
premdb=# \du
                                 List of roles
  Role name  |                        Attributes                         | Member of
-------------+-------------------------------------------------------------+-----------
 alex        |                                                            | {allqa}
 allqa       | Create DB, Cannot login                                    | {}
 henry       |                                                            | {allqa}
 niklas      |                                                            | {allqa}
 vicky       |                                                            | {allqa}
...
```

Note that the `CREATEDB` attribute is inherited, so the `allqa` members can run as themselves and create databases. (They do not need to use `SET ROLE` to run as `allqa`.)

```
premdb=# premdb=# \c premdb alex
You are now connected to database "premdb" as user "alex".
premdb=> create database alexdb;
CREATE DATABASE
```

CREATE SCHEMA

Create a new schema in the current database.

```
CREATE SCHEMA name [ AUTHORIZATION user ] [ schema_element [ ... ] ]
CREATE SCHEMA AUTHORIZATION user | CURRENT_USER | SESSION_USER [ schema_element [ ... ] ]
CREATE SCHEMA IF NOT EXISTS name [ AUTHORIZATION user ]
CREATE SCHEMA IF NOT EXISTS AUTHORIZATION user
```

Only `sysadmin` users, the database owner, and users who have been granted the CREATE privilege can create a schema. For granting CREATE privileges, see GRANT.

AUTHORIZATION user

The owner of the schema. If you do not enter a schema `name`, the user `name` becomes the name of the schema.

CURRENT_USER

See CURRENT_USER.

SESSION_USER

See SESSION_USER.

schema_element

Use a complete CREATE TABLE or CREATE VIEW statement to create a table or view that belongs to the new schema.

IF NOT EXISTS

Create the schema if it does not already exist. If it does exist, do not create it and do not return an error.

For example:

```
premdb=# create schema league;
CREATE SCHEMA
premdb=# create schema if not exists league;
CREATE SCHEMA
```

Create a schema named and owned by user `bobr`:

```
premdb=# create schema authorization bobr;
CREATE SCHEMA
```

Within a single CREATE SCHEMA statement, create a schema, a table, and a view:

```
premdb=# create schema league
create table test(c1 int)
create view testv as select * from test;
CREATE SCHEMA
premdb=# set schema 'league';
SET
premdb=# \d
       List of relations
 Schema | Name  | Type  |  Owner
```

```
--------+-------+-------+---------
 league | test  | table | brumsby
 league | testv | view  | brumsby
(2 rows)
```

You can query tables across schemas in the same database. For example, the `awayteam` table in this query exists in both the `bobr` schema and the `public` schema:

```
premdb=# TABLE premdb.bobr.awayteam UNION TABLE premdb.public.awayteam ORDER BY 1;
 atid |          name
------+------------------------
   51 | Arsenal
   52 | Aston Villa
   53 | Barnsley
   54 | Birmingham City
   55 | Blackburn Rovers
...
```

CREATE SEQUENCE

Create a sequence number generator.

```
CREATE [ TEMPORARY | TEMP ] SEQUENCE [ IF NOT EXISTS ] name [ START [ WITH ] number ]
```

TEMP | TEMPORARY

Create a temporary sequence that is dropped at the end of the session.

IF NOT EXISTS

Do not return an error if a sequence (or another object) with the same name already exists.

name

Unique sequence name within a schema. Sequence names may be reused across schemas. A sequence may not have the same name as a table if both objects are in the same schema. The sequence name may be database- and schema-qualified. For example: `premdb.public.matchkey`

START WITH number

Starting number for the generated sequence values. Must be a positive number in the range (1 to 9223372036854774784). The maximum value a sequence can return is 9223372036854775807 (263 -1).

Usage Notes

Sequence values increment by 1024 (or greater) on each compute node. See Generating Values with Sequences.

After creating sequences, you can use the `\ds` command in `ybsql` to list them. Sequences are also included alongside tables and views in the `\d` output.

```
premdb=# \ds
          List of relations
 Schema |  Name   |   Type   |  Owner
--------+---------+----------+---------
 public | matchid | sequence | brumsby
(1 row)
```

Examples

Create a sequence and select its next value:

```
premdb=> create sequence matchid;
CREATE SEQUENCE
premdb=> select nextval('matchid') from sys.const;
 nextval
---------
    1024
(1 row)
```

Create a table called `matchkey` and generate values for the `matchid` column:

```
premdb=# create table matchkey as
select seasonid,matchday,htid,atid,ftscore,htscore,
nextval('matchid') as matchid
from match;
SELECT 8606
premdb=# select * from matchkey limit 10;
 seasonid |      matchday       | htid | atid | ftscore | htscore | matchid
----------+---------------------+------+------+---------+---------+---------
       15 | 2006-10-28 00:00:00 |    8 |   75 | 0-4     | 0-2     | 1573888
       15 | 2006-09-16 00:00:00 |    8 |   76 | 0-0     | 0-0     | 1574912
       15 | 2006-12-26 00:00:00 |    8 |   77 | 2-1     | 1-1     | 1575936
       15 | 2006-12-30 00:00:00 |    8 |   81 | 3-2     | 2-1     | 1576960
       15 | 2007-04-21 00:00:00 |    8 |   83 | 1-3     | 0-0     | 1577984
       15 | 2007-03-31 00:00:00 |    8 |   84 | 1-0     | 0-0     | 1579008
       15 | 2006-08-19 00:00:00 |    8 |   91 | 2-0     | 2-0     | 1580032
       15 | 2006-09-09 00:00:00 |    8 |   92 | 1-0     | 0-0     | 1581056
       15 | 2006-12-09 00:00:00 |    8 |   94 | 4-0     | 1-0     | 1582080
       15 | 2006-11-04 00:00:00 |    8 |   95 | 0-1     | 0-0     | 1583104
(10 rows)
```

Create a sequence with a start value and select its next value:

```
premdb=# drop sequence if exists matchid;
DROP SEQUENCE
premdb=# create sequence matchid start 5000000000;
CREATE SEQUENCE
premdb=# select nextval('matchid') from sys.const;
  nextval
------------
 5000000512
(1 row)
```

See also Loading Generated Key Values.

CREATE TABLE AS

Create a new table that stores the results of a query.

```
CREATE [ { TEMPORARY | TEMP } ] TABLE [ IF NOT EXISTS ] table_name
   [ (column_name [, ...]) ]
   [ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
   AS query
   [ WITH ]
   [ DATA | NO DATA ]
   [ DISTRIBUTE { ON (column) | [ ON ] REPLICATE | [ ON ] RANDOM } ]
   { [ SORT ON (column) ] |
     [ CLUSTER ON (column [, ... ] }
   [ PARTITION BY ( { range_partition_spec | hash_partition_spec } [ , ... ] )
   ]
```

TEMPORARY | TEMP

Create a temporary table that is automatically dropped at the end of the session (and is visible only to that session). You cannot create "global" temporary tables that are visible across sessions. Temporary tables belong to a temporary schema. You cannot qualify a temporary table name with a user-defined schema name.

IF NOT EXISTS

Create the table if it does not already exist. If it does exist, do not create it and do not return an error.

table_name

Give the table a name that is distinct from the name of any other table or view in the same schema. Optionally, qualify the name of a persistent table with the schema name. You can also qualify table names with the name of the current database, but you cannot create a table in a different database; the table must belong to the current database.

The maximum length of a table name is 128 bytes; longer names are automatically truncated. See SQL Identifiers.

column_name

Name each column uniquely. If you do not specify a name for a column, the behavior is undefined. For example, if you use `CAST` or `CONVERT` expressions without aliases, the column name may default to the name of the data type used in the expression. In turn, you may try to create a table with duplicate column names, which results in the following error:

```
premdb=# create table casts as select cast(1 as varchar), cast(2 as varchar) from sys.const;
ERROR:  column "varchar" specified more than once in "casts"
premdb=# create table casts as select cast(1 as varchar) cast1, cast(2 as varchar) cast2 from sys.const;
SELECT 1
```

Tables have a limit of 2,000 user-defined columns.

ON COMMIT

Set the behavior for temporary tables when a transaction commits.

- `PRESERVE ROWS`: do not delete rows from temporary tables when transactions commit.

- `DELETE ROWS` : delete rows from temporary tables when transactions commit.
- `DROP` : drop temporary tables when transactions commit.

AS query

Supply any query (SELECT statement) that Yellowbrick supports. The results of this query populate the table. You can also use a TABLE statement or an EXECUTE statement.

WITH

The options that follow the query definition may be preceded by the `WITH` keyword.

NO DATA | DATA

`NO DATA` creates an empty table, regardless of the rows that the query returns. If `NO DATA` is not specified or `DATA` is specified, the table will contain the results of the query.

DISTRIBUTE

Define data distribution for the table; see CREATE TABLE, which supports the same options. See also Data Distribution for CREATE TABLE AS (CTAS) Results.

Note: You cannot distribute a table on a floating-point column (`FLOAT` , `FLOAT4` , `FLOAT8`).

SORT ON

Define a sort column for the table (how the data is sorted when it is loaded). The `SORT ON` and `CLUSTER ON` options are mutually exclusive. See Sorted and Clustered Tables.

CLUSTER ON

Define cluster columns for the table; see CREATE TABLE, which supports the same syntax.

PARTITION BY

See Partitioning Options. CREATE TABLE supports the same syntax.

Note: If used, the following clauses at the end of the `CREATE TABLE AS` statement must appear in the order shown in the syntax diagram:

1. `DISTRIBUTE`
2. `SORT ON` or `CLUSTER ON`
3. `PARTITION BY`

Data Distribution for CREATE TABLE AS (CTAS) Results

If you do not specify a distribution type for a CTAS table, the resulting data distribution depends on the nature of the query that creates the table.

- A table created from columns in one or more replicated tables is also replicated.
- A table created from a single hash-distributed table is typically hash-distributed on the same column when the distribution column is included in the select list. If the distribution column is not included in the select list, random distribution is used.
- Tables resulting from equality joins over distribution columns typically preserve the hash distribution.
- In general, hash distribution is preserved where possible, but tables created from complex joins or that contain complex select list expressions may produce randomly distributed result sets.

Constraints

A CTAS statement preserves `NOT NULL` constraints on columns from the source table. Other constraints are not preserved.

Examples

Create a table that filters and sorts the rows from another table:

```
premdb=# create table season20 as
select * from match where seasonid=20 order by seasonid,matchday;
SELECT 380
premdb=# \d season20
            Table "public.season20"
  Column  |            Type             | Modifiers
----------+-----------------------------+-----------
 seasonid | smallint                    |
 matchday | timestamp without time zone |
 htid     | smallint                    |
 atid     | smallint                    |
 ftscore  | character(3)                |
 htscore  | character(3)                |

Distribution: Hash (seasonid)
```

Create a table that uses a TABLE statement to select all of the rows from another table:

```
premdb=# create table newseason as
table season;
SELECT 25
```

Create a table that holds the result of an executed prepared statement:

```
premdb=# prepare x(varchar(30)) as
select * from hometeam where name=$1;
PREPARE
premdb=# create table homectas as execute x('Leicester City');
SELECT 1
premdb=# select * from homectas;
 htid |      name
------+----------------
   23 | Leicester City
(1 row)
```

Create a table that contains no data, just the table structure with column definitions:

```
premdb=# create table ctas_season as
select * from season with no data;
SELECT 0
premdb=# \d ctas_season
          Table "public.ctas_season"
   Column    |         Type          | Modifiers
-------------+-----------------------+-----------
 seasonid    | smallint              |
 season_name | character(9)          |
 numteams    | smallint              |
 winners     | character varying(30) |

Distribution: Replicated

premdb=# select count(*) from ctas_season;
 count
-------
     0
(1 row)
```

Create a version of the `match` table that is sorted on the `htid` column:

```
premdb=# create table matchsort as
select * from match with sort on(htid);
SELECT 8606
premdb=# \d matchsort
            Table "public.matchsort"
  Column  |            Type             | Modifiers
----------+-----------------------------+-----------
 seasonid | smallint                    |
 matchday | timestamp without time zone |
 htid     | smallint                    |
 atid     | smallint                    |
 ftscore  | character(3)                |
 htscore  | character(3)                |

Distribution: Hash (seasonid)
Sort Column: (htid)
```

Create a table that is sorted and partitioned:

```
premdb=# create table stats_sorted_partitioned as
select * from newmatchstats where seasonid>4
distribute on(seasonid)
sort on(htid)
partition by(hash(matchday with 100 partitions, is null));
SELECT 39398400
```

CREATE USER

Platforms: All platforms

Parent topic: SQL Commands

Create a database user (or role) and define access privileges for the user.

CREATE USER and CREATE ROLE are synonymous, except for one difference in behavior. When you use CREATE USER, LOGIN is assumed by default. NOLOGIN is assumed when you use CREATE ROLE.

```
CREATE USER name [ [ WITH ] option [ ... ] ]

where option can be:

      SUPERUSER | NOSUPERUSER
    | CREATEDB | NOCREATEDB
    | CREATEROLE | NOCREATEROLE
    | INHERIT | NOINHERIT
    | LOGIN | NOLOGIN
    | CONNECTION LIMIT connlimit
    | [ ENCRYPTED | UNENCRYPTED ] PASSWORD 'password'
    | VALID UNTIL 'timestamp'
    | IN ROLE role_name [, ...]
    | ROLE role_name [, ...]
    | ADMIN role_name [, ...]
    | USER role_name [, ...]
```

For details about the syntax and examples, see CREATE ROLE.

CREATE VIEW

Yellowbrick Documentation > Reference > SQL Reference > SQL Commands > CREATE VIEW

Platforms: All platforms

Parent topic: SQL Commands

Create a new view or replace an existing view, based on the results of a specific query.

```
CREATE [ OR REPLACE ] [ TEMP | TEMPORARY ] VIEW name [ ( column_name [, ...] ) ]
   AS query
```

OR REPLACE

Create a new view, or replace an existing view if a view with the same name exists in the specified (or current) schema and database.

Note: When you intend to *replace* a view, you cannot drop columns, change the order of columns, change the names of columns, or change the data types of columns. However, you can add new columns to the view.

TEMP | TEMPORARY

Create a view in a temporary schema and remove it when the current session ends.

VIEW name

Specify a view name, optionally qualified with a schema name. You cannot create views in remote databases.

column_name

Provide an optional list of column names for the view definition. By default, the view assumes the column names that derive from the query. Make sure the query for the view defines a set of uniquely named columns.

AS query

Define the view in terms of a valid query (a SELECT statement). The query cannot change the names or data types of columns in the source tables.

Yellowbrick views are persistent objects. They are not dependent on the tables that they reference; therefore, you can drop or rename a table and its associated views persist in the database. For example, you can create table `team`, then create view `teamview` by selecting from `team`. If you drop `team` with `CASCADE`, the view remains in the database. If table `team` is re-created, you can still select from `teamview` provided that the columns that originally existed in the table also exist in the new instance of the table and have the same data types.

See also User-Defined Views.

Examples

For example, create a view in the `premdb` schema, based on a query against the `match` table in the `public` schema:

```
premdb=# create or replace view premdb.matchview as select * from public.match where seasonid between 5 and 15;
CREATE VIEW
premdb=# \d premdb.matchview
            View "premdb.matchview"
  Column  |           Type             | Modifiers
----------+----------------------------+-----------
 seasonid | smallint                   |
```

```
 matchday | timestamp without time zone |
 htid     | smallint                    |
 atid     | smallint                    |
 ftscore  | character(3)                |
 htscore  | character(3)                |
```

The following `CREATE OR REPLACE VIEW` statement fails because it attempts to change the column names of an existing view:

```
premdb=# create or replace view matchview(mv1,mv2,mv3,mv4,mv5,mv6) as
select * from public.match where seasonid between 5 and 15;
ERROR:  cannot change name of view column "seasonid" to "mv1"
```

The following `CREATE OR REPLACE VIEW` statement fails because it attempts to change the data type of a column in an existing view:

```
premdb=# create or replace view matchv as select seasonid::int,matchday,htid,atid,ftscore,htscore from match;
ERROR:  cannot change data type of view column "seasonid" from smallint to integer
```

The following example demonstrates the case where you use `CREATE OR REPLACE VIEW` to add columns to an existing view. The following initial version of the view contains five columns. The last column is created by concatenating two columns from the `match` table, but those columns are not included in their original form.

```
premdb=# create view matchv as select seasonid, matchday, htid, atid,
concat(ftscore, ' (HT:', htscore,')') score from match;
CREATE VIEW
premdb=# \d matchv
              View "public.matchv"
  Column  |            Type             | Modifiers
----------+-----------------------------+-----------
 seasonid | smallint                    |
 matchday | timestamp without time zone |
 htid     | smallint                    |
 atid     | smallint                    |
 score    | character varying(12)       |
```

If you later decide that you want to add columns to the view, such as `ftscore` and `htscore` in this case, you can use `CREATE OR REPLACE VIEW` and put them at the end of the original column list:

```
premdb=# create or replace view matchv as select seasonid, matchday, htid, atid,
concat(ftscore, ' (HT:', htscore,')') score, ftscore, htscore from match;
CREATE VIEW
premdb=# \d matchv
              View "public.matchv"
  Column  |            Type             | Modifiers
----------+-----------------------------+-----------
 seasonid | smallint                    |
 matchday | timestamp without time zone |
 htid     | smallint                    |
 atid     | smallint                    |
 score    | character varying(12)       |
 ftscore  | character(3)                |
 htscore  | character(3)                |
```

However, note that you cannot rearrange the columns in the view:

```
premdb=# create or replace view matchv as select seasonid, matchday, htid, atid, ftscore, htscore,
concat(ftscore, ' (HT:', htscore,')') score from match;
ERROR:  cannot change name of view column "score" to "ftscore"
```

To change column names, column order, or data types for columns, you have to drop the view, then create it again.

CREATE WLM PROFILE

Create a workload management (WLM) profile.

```
CREATE WLM PROFILE name [ ( DEFAULT_POOL name ) ]
```

Only superusers and users who have been granted `CONTROL WLM ON SYSTEM` can create, alter, and drop WLM profiles. You can use Yellowbrick Manager to create and manage WLM profiles, resource pools, and rules.

Use any valid SQL identifier as the profile name. Names longer than 128 characters are allowed but truncated. See also SQL Identifiers.

For example:

```
premdb=# create wlm profile shortquerybias;
CREATE WLM PROFILE
```

This example creates a profile with a default pool. The default pool does not have to exist when this command is run.

```
premdb=# create wlm profile sqb (default_pool sqbpool);
CREATE WLM PROFILE
```

Use the ALTER WLM PROFILE command to make a profile the active profile or change its default pool. Query the sys.wlm_active_profile view to see the current profiles in the system and which profile is active. Alternatively, use the `ybsql \dwp` command to return a list of profiles.

CREATE WLM RESOURCE POOL

Create a workload management (WLM) resource pool.

```
CREATE WLM RESOURCE POOL name (PROFILE name [ options ])
```

Only superusers and users who have been granted `CONTROL WLM ON SYSTEM` can create, alter, and drop WLM resource pools.

POOL name

Use any valid SQL identifier as the pool name. Names longer than 63 characters are allowed but truncated. See also SQL Identifiers.

PROFILE name

A single set of parentheses is required around the profile name and the subsequent list of options (if any). The resource pool must belong to an existing profile. For example: `(profile myprofile)`

You cannot attach a user-defined resource pool to the system profiles (`default` and `maintenance`).

---

options

Express each option as a key-value pair and separate the options with commas. You can list options in any order. If you do not specify any options, the result is a resource pool with default values. The profile name and subsequent options must be enclosed in parentheses.

max_concurrency

Maximum number of concurrent queries that can run in this pool. The default is `1` . `max_concurrency` must be less than or equal to 100. If `max_concurrency` is greater than `min_concurrency` , the pool is a *flex pool*. See Examples of Flex Pools.

min_concurrency

Minimum number of concurrent queries that can run in this pool. The default is `1` . `min_concurrency` must be less than or equal to `max_concurrency` .

slots (deprecated)

For backward compatibility, the number of concurrent queries that can run in this pool. The default is `1` . This option cannot be `null` and must be set to a number between 1 and 100.

queue_size

Number of queries that can wait in the queue for this pool. Set this option to a number greater than `0` . The default is `10` . This option cannot be `null` .

maximum_wait_limit

Maximum number of seconds that a query can wait in this pool. This option can be set to a number greater than `0` or `null` .

maximum_row_limit

Maximum number of rows that a query can return when executed in this pool. This option can be set to a number greater than or equal to `0` or `null`.

maximum_exec_time

Maximum duration (number of seconds) for a query that executes in this pool. This option can be set to a number greater than or equal to `0` or `null`.

max_spill_pct

Maximum temporary space available for spilling. Enter a percentage from 1 to 100. Other valid values are `null` and `0`. A value of `0` disables spilling for this resource pool. A value of `null`, which is the default, is equivalent to **Any (remainder)** in Yellowbrick Manager, which means split any unallocated temporary space with other queries running under pools set to **Any**. (Requested memory per slot must be at least 1GB for pools with spilling enabled and 512MB for pools with spilling disabled.)

requested_memory

Memory restricted for this pool, expressed either in MB or as a percentage, using single quotes. For example, `'1000MB'` and `'10%'` are valid values (stored as `1000` and `10%`, respectively). You can also specify `null` (no restriction on memory).

The requested memory for the pool is divided equally among the concurrent query slots. Requested memory per slot must be at least 1GB for pools with spilling enabled and 512MB for pools with spilling disabled.

If more memory is requested than is available per compute node, the memory will be scaled down. However, the system pools will always get the memory allocated to them. See Allocating Query Memory.

next_memory_queue

Name of the next resource pool to use, if memory is exceeded for this pool. A `null` value is allowed (no next pool).

next_exec_time_limit_queue

Name of the next resource pool to use, if the execution time limit is exceeded for this pool. A `null` value is allowed (no next pool).

Query the sys.wlm_pending_pool table to see the resource pools in the system. Alternatively, use the `ybsql \dwrp` command to return a list of resource pools.

Examples

Create a resource pool with a named profile but otherwise default settings:

```
premdb=# create wlm profile testing;
CREATE WLM PROFILE
premdb=# create wlm resource pool test_pool1 (profile testing);
CREATE WLM RESOURCE POOL
premdb=# select * from sys.wlm_pending_pool where profile_name='testing';
-[ RECORD 1 ]--------------+-------------
name                       | test_pool1
profile_name               | testing
memory_requested           | remainder
memory_per_query_bytes     | 114460459008
temp_space_requested       | remainder
temp_space_per_query_bytes | 1466935803904
max_concurrency            | 1
min_concurrency            | 1
queue_size                 | 100
-[ RECORD 2 ]--------------+-------------
name                       | system
profile_name               | testing
memory_requested           | 11264MB
memory_per_query_bytes     | 3936354304
temp_space_requested       | 163840MB
temp_space_per_query_bytes | 57265881088
```

```
max_concurrency           | 3
min_concurrency           | 2
queue_size                | 100
```

Create another resource pool in the same profile but specify maximum/minimum concurrency as 12/3:

```
premdb=# create wlm resource pool test_pool_12_and_3
(max_concurrency 12, min_concurrency 3, profile testing);
CREATE WLM RESOURCE POOL
```

Create a resource pool with spilling disabled, a queue size of 2, maximum/minimum concurrency of 2/1, and a requested memory size of 1024MB:

```
premdb=# create wlm resource pool mb512_nospill_2_and_1
(max_concurrency 2, min_concurrency 1, queue_size 2, requested_memory '1024MB', max_spill_pct 0, profile testing);
CREATE WLM RESOURCE POOL
```

Attempt to create a resource pool with spilling enabled (by default), 2 slots (queue size), and a requested memory size of 1024MB. The system returns an expected error:

```
premdb=# create wlm resource pool mb512_spill_2_and_1
(max_concurrency 2, min_concurrency 1, queue_size 2, requested_memory '1024MB', profile testing);
ERROR:  requested_memory must be >= 1024MB per query
```

To avoid this error, increase the requested memory to at least 2048MB (1024MB per slot):

```
premdb=# create wlm resource pool mb512_spill_2_and_1
(max_concurrency 2, min_concurrency 1, queue_size 2, requested_memory '2048MB', profile testing);
CREATE WLM RESOURCE POOL
```

CREATE WLM RULE

Create a workload management (WLM) rule.

```
CREATE WLM RULE name [ ( PROFILE name [ options ] ) ]
```

Only superusers and users who have been granted `CONTROL WLM ON SYSTEM` can create, alter, and drop WLM rules.

The immediate result of a `CREATE WLM RULE` command is a pending rule. See Profile Activation for details about the transition of pending rules to active rules. You can query the sys.wlm_pending_rule and sys.wlm_active_rule views to see the rules in the system. Alternatively, use the `ybsql \dwr` and `\dwr+` commands to return similar information. The output of the `\dwr+` command matches the detailed output of the system views. The `\dwr` command returns a list of rules but does not include their JavaScript definitions.

RULE name

Use any valid SQL identifier as the rule name. Names longer than 128 bytes are allowed but truncated. See also SQL Identifiers.

PROFILE name

A single set of parentheses is required around the profile name and the subsequent list of options (if any). The rule may belong to an existing profile that you name in the command, or a profile that you have not yet created. For example: `(profile myprofile)`

Alternatively, if you specify `null` or do not specify a profile, the rule is global and belongs to all profiles: `(profile null)`

You cannot attach a user-defined rule to the system profiles (`default` and `maintenance`).

Options

Express each option as a key-value pair and separate the options with commas. You can list options in any order. If you do not specify any options, the result is a WLM rule with default values. However, you must specify the `javascript` for every rule you create.

type

Rule type: `submit` , `hinting` , `assemble` , `compile` , `restart_for_user` , `restart_for_error` , `runtime` , or `completion` . See Rule Types and Creating WLM Rules.

rule_order

Rule order, expressed as a number greater than or equal to `1` ; cannot be `null` . The default is `100` . This value defines the order in which rules are applied, lowest first, highest last. Rules marked as `1` are applied before rules marked as `10` or `100` , for example.

enabled

If `true` , the rule is enabled on creation. The default value is `true` . This option cannot be `null` .

superuser

If `true`, the rule applies to superuser accounts only. The default value is `false`. This option cannot be `null`.

javascript

Rule conditions and actions, expressed in JavaScript. This option is required. Use the following format:

```
javascript $$ <rule_in_javascript> $$
```

For example:

```
javascript $$ if (w.database === 'premdb') {w.resourcePool = 'shortquerypool';} $$
```

By using the `$$` notation, you will avoid potential problems with special characters that might otherwise need to be escaped. The default value is `""` (no rule condition or action).

version

Version of this rule (can be `null`).

Examples

Create a rule with `profile`, `superuser`, and `javascript` options defined:

```
premdb=# create wlm rule slowlane
(profile shortquery,
superuser true,
javascript $$ if (w.referencedTables.contains('match')) {w.maximumRowLimit = 1000000;}$$) ;
CREATE WLM RULE
```

Because no type is specified, this rule will default to the `compile` type.

The following rule is a `runtime` rule:

```
create wlm rule runtime_matchstats_critical_priority
(profile shortquerybias,
type runtime,
javascript $$
if (w.referencedTables.contains('matchstats') &&
   w.priority === 'low' ||
   w.priority === 'normal' ||
   w.priority === 'high') {
    w.priority = 'critical';
} $$);
CREATE WLM RULE
```

DEALLOCATE

Remove prepared statements that were created during the current session.

```
DEALLOCATE [ IF EXISTS ] { name | ALL }
```

You can remove a specific prepared statement by name (without arguments), or you can remove all prepared statements. If you use the `IF EXISTS` option, no error message is returned if the named statement does not exist.

For example:

```
premdb=# prepare x(varchar(30)) as select * from team where name=$1;
PREPARE
premdb=# execute x('Leicester City');
 teamid | htid | atid |      name      | nickname |   city    |      stadium       | capacity | avg_att
--------+------+------+----------------+----------+-----------+--------------------+----------+---------
    22 |   23 |   72 | Leicester City | Foxes    | Leicester | King Power Stadium |    32262 |  32.201
(1 row)
```

```
premdb=# deallocate all;
DEALLOCATE ALL
premdb=# execute x('Leicester City');
ERROR:  prepared statement "x" does not exist
```

Note that the DEALLOCATE syntax does not require you to provide the arguments to the prepared statement, just its name:

```
premdb=# prepare avgsales(int) as select ...
PREPARE
...
premdb=# deallocate avgsales;
DEALLOCATE
```

DECLARE

Yellowbrick Documentation > Reference > SQL Reference > SQL Commands > DECLARE

Platforms: All platforms

Parent topic: SQL Commands

Create a cursor within a transaction block.

```
DECLARE name [ INSENSITIVE ] [ NO SCROLL | SCROLL ] CURSOR  [ { WITH | WITHOUT } HOLD ] FOR query
```

See also CLOSE, FETCH, and MOVE (related cursor commands).

INSENSITIVE

By default, cursors are *insensitive* to any changes in the data that are made after the cursor is declared. Data is fetched based on the results of the query *when the cursor was declared*. The `INSENSITIVE` keyword need not be specified and has no effect. Sensitive cursors are not supported.

NO SCROLL | SCROLL

`NO SCROLL` is the default and need not be specified. Scrollable (`SCROLL`) cursors are intended for use with backward moves and fetches, which are not supported. However, a cursor may be created with `SCROLL` but used for forward moves and fetches only.

WITH HOLD | WITHOUT HOLD

`WITHOUT HOLD` , the default, specifies that the cursor cannot be used outside of the transaction that created it. `WITH HOLD` specifies that a cursor can continue to be used after the transaction in which it was created commits. This distinction means that you can declare a cursor `WITH HOLD` without enclosing the `DECLARE` command in an explicit begin/end transaction block. The default type of cursor (`WITHOUT HOLD`) can only be used within the current transaction and must be preceded by an explicit `BEGIN` statement.

FOR query

Specify any valid Yellowbrick `SELECT` query.

Note: If the query does not have `FROM` clause references to Yellowbrick tables, the cursor may be declared according to the rules defined in the PostgreSQL documentation. For example, backward moves and fetches will be allowed. (However, note that in some cases the query planner may add a `FROM sys.const` clause to the query that you submit, in which case the PostgreSQL rules will not apply.)

Interaction with Savepoints

Savepoints keep track of open cursors. If a cursor is invalidated by an error that occurs within a transaction, rolling back to a savepoint in that transaction makes the cursor valid again (except when the error results from a `DECLARE` , `FETCH` , or `MOVE` of the cursor itself). However, the cursor is not moved back to its position when the `SAVEPOINT` command was issued; the cursor maintains the last position it held.

Examples

The `DECLARE` command must be used within an explicit transaction block for default `WITHOUT HOLD` cursors. For example:

```
premdb=# begin;
BEGIN
```

```
premdb=# declare match cursor for select * from match order by matchday;
DECLARE CURSOR
...
premdb=# end;
```

You cannot access such a cursor after the current transaction has ended:

```
premdb=# begin;
BEGIN
premdb=# declare season cursor for select * from season order by 1;
DECLARE CURSOR
premdb=# end;
COMMIT
premdb=# fetch next from season;
ERROR:  cursor "season" does not exist
```

A `WITH HOLD` cursor does not require an explicit `BEGIN` statement when it is declared and is usable in subsequent transactions:

```
premdb=# declare season cursor with hold for select * from season order by 1;
DECLARE CURSOR
premdb=# fetch next from season;
 seasonid | season_name | numteams |      winners
----------+-------------+----------+-------------------
        1 | 1992-1993   |       22 | Manchester United
(1 row)

premdb=# fetch next from season;
 seasonid | season_name | numteams |      winners
----------+-------------+----------+-------------------
        2 | 1993-1994   |       22 | Manchester United
(1 row)

premdb=# begin;
BEGIN
premdb=# fetch next from season;
 seasonid | season_name | numteams |     winners
----------+-------------+----------+-----------------
        3 | 1994-1995   |       22 | Blackburn Rovers
(1 row)

premdb=# end;
COMMIT
premdb=# fetch next from season;
 seasonid | season_name | numteams |      winners
----------+-------------+----------+-------------------
        4 | 1995-1996   |       20 | Manchester United
(1 row)
...
```

DELETE

Yellowbrick Documentation > Reference > SQL Reference > SQL Commands > DELETE

Platforms: All platforms

Parent topic: SQL Commands

Delete rows from a table based on the results of a condition.

Note: The Yellowbrick column store is designed to optimally store, compress, and organize data for high-volume read-oriented analytics. As a result, when data is deleted it continues to exist in the row store for a period of time until it can be efficiently removed ("garbage-collected" or "vacuumed") and data can be reorganized. These optimizations are fully automatic and require no administration or manual interaction.

```
[ WITH name AS (subquery) [, ...] ]
DELETE FROM table_name [ * ] [ [ AS ] alias ]
   [ USING list ]
   [ WHERE condition ]
```

WITH

See WITH Clause for syntax details. You can reference a WITH subquery in the USING clause of the DELETE statement.

table_name

Specify the name of a table in the current database.

Note: You cannot use `DELETE` commands on tables in remote databases; see Cross-Database Queries.

Warning: Running `EXPLAIN ANALYZE` on an `INSERT`, `UPDATE`, or `DELETE` statement modifies the target table, without warning. To avoid this problem, you can run the `EXPLAIN ANALYZE` statement inside a transaction, then roll it back if needed.

USING

Name one or more tables or table references, which in turn can be referenced in WHERE clause conditions for the DELETE operation. Do not list the target table for the DELETE (except for self-join purposes).

WHERE

Use this WHERE clause just as you would use the WHERE clause in a SELECT statement. Define a condition on either the target table or a table that is listed in the USING clause. For example, you can join the target table to another listed table to qualify rows for the DELETE operation.

`DELETE FROM table` without a `WHERE` clause deletes all of the rows from the table.

For example, delete rows from the `match` table that fall within a date range for the `matchday` column:

```
premdb=# delete from match
where extract(year from matchday) between 1992 and 1999;
DELETE 3101
```

For example, delete rows from the `match` table based on a join to the `season` table:

```
premdb=# delete from match
using season
```

```
where match.seasonid=season.seasonid;
DELETE 8606
```

This example uses a WITH clause that contains a ROW_NUMBER function. This statement removes duplicates from the `season` table based on values in the `seasonid` column. The system column ROWUNIQUE is used to find matching rows.

```
premdb=# insert into season(seasonid) values(10);
INSERT 0 1
premdb=# select * from season where seasonid=10;
 seasonid | season_name | numteams | winners
----------+-------------+----------+---------
       10 | 2001-2002   |       20 | Arsenal
       10 | [NULL]      |   [NULL] | [NULL]
(2 rows)

premdb=# with rows(rid, rn) as (select rowunique, row_number() over(partition by seasonid order by seasonid) from season)
delete from season using rows where rows.rn>1 and season.rowunique=rows.rid;
DELETE 1
premdb=# select * from season where seasonid=10;
seasonid | season_name | numteams | winners
----------+-------------+----------+---------
       10 | 2001-2002   |       20 | Arsenal
(1 row)
```

DESCRIBE

Describe the definition of a schema object. Optionally, return the DDL that created the object or the secret associated with the key.

```
DESCRIBE { table | view | sequence | PROCEDURE procedure }
[ { WITH DDL | ONLY DDL } ]
DESCRIBE KEY key [ WITH SECRET ]
DESCRIBE EXTERNAL AUTHENTICATION name [ WITH DDL ]
```

For information about generating DDL for a specific database, see Generating DDL.

Parameters

table | view | sequence | PROCEDURE procedure

Name of a persistent table, temporary table, view, sequence, or stored procedure. You can specify these objects with or without the schema name.

For stored procedures, specify the keyword `PROCEDURE` before the procedure name. Also specify the data type arguments in the procedure name. For example:

```
describe procedure addteam(varchar)
```

KEY key

Name of an encryption key that was created with a `CREATE KEY` statement.

WITH DDL

Return the complete DDL that created the object as well as the table definition. The table definition is equivalent to the output of the `ybsql \d` command.

ONLY DDL

Return the complete DDL that created the object but not the table definition.

WITH SECRET

Return the secret that was created with the key. Only the owner or superusers can view the secret of a key.

Examples

For example, describe the `match` table, including the DDL:

```
premdb=# describe match with ddl;

--------------------------------------------------------------------------------------------
                     Table "public.match"
 ----------+----------------------------+-----------+-------------
   Column  |            Type            | Modifiers | Description
 ----------+----------------------------+-----------+-------------
  seasonid | smallint                   | not null  |
  matchday | timestamp without time zone |           |
```

```
 htid     | smallint                 |           |
 atid     | smallint                 |           |
 ftscore  | character(3)             |           |
 htscore  | character(3)             |           |

Distribution: Hash (seasonid)
Sort Column: (seasonid)
Columns:
   seasonid PRIMARY KEY
Foreign-key constraints:
Primary-key constraints:
   "match_pkey" PRIMARY KEY (seasonid)
   "match_seasonid_fkey" FOREIGN KEY (seasonid) REFERENCES season(seasonid)

-- SHOW DDL
-- Name: match
-- Schema: public
--------------------------------------------------------------------------------
CREATE TABLE match (
   seasonid smallint NOT NULL,
   matchday timestamp without time zone,
   htid smallint,
   atid smallint,
   ftscore character(3),
   htscore character(3),
   CONSTRAINT "match_pkey" PRIMARY KEY (seasonid),
   CONSTRAINT "match_seasonid_fkey" FOREIGN KEY (seasonid) REFERENCES season(seasonid)
)
DISTRIBUTE ON (seasonid)
SORT ON (seasonid);
(39 rows)
```

Return DDL only for the `season` table:

```
premdb=# describe season only ddl;

-------------------------------------
 -- SHOW DDL
 -- Name: season
 -- Schema: public
 -- --------------------------------
 CREATE TABLE season (
    seasonid smallint,
    season_name character(9),
    numteams smallint,
    winners character varying(30)
 )
 DISTRIBUTE REPLICATE
 ;
(12 rows)
```

Return DDL only for a sequence. Note that the start number is guaranteed to be equal to or greater than the originally specified start number. (See also Generating Values with Sequences.)

```
premdb=# create sequence sequence1 start 100000;
CREATE SEQUENCE
premdb=# describe sequence1 only ddl;

-----------------------------------------------
 -- SHOW DDL
 -- Name: sequence1
 -- Schema: public
 -- -------------------------------------------
```

```
 CREATE SEQUENCE sequence1 START WITH 100352;
(5 rows)
```

Describe the `ybd100key` encryption key:

```
premdb=# describe key ybd100key;

-------------------------
   Key "premdb.public"
 ----------------------
 CREATE KEY ybd100key ;
(3 rows)
```

Describe `ybd100key` and return its secret:

```
premdb=# describe key ybd100key with secret;

-----------------------------------------------
            Key "premdb.public"
 ---------------------------------------------
 CREATE KEY ybd100key WITH SECRET 'a1b2c3d4';
(3 rows)
```

Attempting to view the secret of a key without proper privileges results in the following error:

```
premdb=# describe key yb100key with secret;
ERROR:  No privilege to check the secret of key "yb100key"
```

Describe the `addteam(varchar)` stored procedure, including the DDL:

```
premdb=# describe procedure addteam(varchar) with ddl;

-------------------------------------------------------------------------------
   Procedure "public.addteam"
 ----------+-------------------
  Argument |       Type
 ----------+-------------------
  name     | character varying

 Returns: void

 -- SHOW DDL
 -- Procedure Name: addteam
 -- Schema: public
 ---------------------------------------------------------------------------
 CREATE PROCEDURE public.addteam(name character varying) RETURNS void
    LANGUAGE plpgsql
    AS $$ begin
 insert into team(name) values(name);
 end;
 $$;
(21 rows)
```

Describe only the DDL for the `insert_hometeam` procedure:

```
premdb=# describe procedure insert_hometeam(integer,varchar(30)) only ddl;
-------------------------------------------------------------------------------
 -- SHOW DDL
```

```
-- Procedure Name: insert_hometeam
-- Schema: public
--------------------------------------------------------------------------------
CREATE PROCEDURE public.insert_hometeam(integer, character varying) RETURNS void
   LANGUAGE plpgsql
   AS $_$
declare
  htid alias for $1;
  name alias for $2;
begin
  insert into hometeam(htid, name)
  values ($1, $2);
end;
$_$;
(15 rows)
```

DROP BACKUP CHAIN

Drop a backup chain. Optionally, also drop its backup snapshots.

```
DROP BACKUP CHAIN 'chain_name' [ CASCADE ]
```

Use the `CASCADE` option to drop the snapshots that belong to the specified backup chain. Typically, you should run the command with this option.

Run this command when you plan on no longer using a specific backup chain to take backups. When you drop a backup chain, the system reclaims space previously taken up by delete information that was pending the next backup. When a backup chain has been dropped, it can no longer be used for incremental or cumulative backups. Backups of the same database using other chains are not affected and can continue.

Run the command from the database that owns the chain. Chains are database-specific.

Although the `ybbackupctl` client tool provides a command to delete physical backups, this command does not connect to the database and does not drop the referenced backup chain.

For example, the `August2020` backup chain for database `premdb` has the following snapshots:

```
premdb=# select * from sys.backup_snapshot;
 database_id | backup_txid |  backup_txsnap   |                snapshot_name                 |         creation_time
-------------+-------------+------------------+----------------------------------------------+-------------------------------
       16395 |      175440 | 0:175440:175440: | Aug21Full                                    | 2020-08-21 13:20:37.795147-07
       16395 |      177767 | 0:177767:177767: | Aug21Incr                                    | 2020-08-21 13:22:09.254674-07
       16395 |      177989 | 0:177989:177989: | Aug21IncrNew                                 | 2020-08-21 13:24:38.32832-07
       16395 |      175439 | 0:3:3:           | ybd_cf861c56-0cf7-19a8-f459-3382036212c8_epoch | 2020-08-21 13:20:37.781983-07
(4 rows)
```

The following command drops the chain and all the associated snapshots:

```
premdb=# drop backup chain 'August2020' cascade;
DROP BACKUP CHAIN
premdb=# select * from sys.backup_chain;
 database_id | chain_name | policy | oldest_backup_point_id | oldest_rollback_point_id | inprogress_backup_point_id | creation_tim
-------------+------------+--------+------------------------+--------------------------+----------------------------+-------------
(0 rows)
premdb=# select * from sys.backup_snapshot;
 database_id | backup_txid | backup_txsnap | snapshot_name | creation_time
-------------+-------------+---------------+---------------+---------------
(0 rows)
```

DROP CLUSTER

Drop a virtual compute cluster.

```
DROP [ COMPUTE ] CLUSTER [ IF EXISTS ] name
```

Use double quotes for the cluster name if it contains special characters.

If you use the `IF EXISTS` option, no error message is returned if the named cluster does not exist.

Example

```
drop cluster "bobr-rc6-cluster1";
```

DROP DATABASE

Drop an existing physical database.

```
DROP DATABASE [ IF EXISTS ] name
```

For example:

```
premdb=# drop database shakespeare;
DROP DATABASE
```

If you use the `IF EXISTS` option, no error message is returned if the named database does not exist. For example:

```
premdb=# drop database shakespeare;
ERROR:  database "shakespeare" does not exist
premdb=# drop database if exists shakespeare;
DROP DATABASE
```

You cannot drop a database that is in `HOT_STANDBY` mode.

DROP EXTERNAL AUTHENTICATION

Platforms: EE: All cloud platforms

Parent topic: SQL Commands

Drop an external authentication object.

```
DROP EXTERNAL AUTHENTICATION name
```

DROP EXTERNAL FORMAT

Platforms: EE: All cloud platforms, CE

Parent topic: SQL Commands

Drop an external format object.

```
DROP EXTERNAL FORMAT [ IF EXISTS ] name [ CASCADE ]
```

IF EXISTS

Drop the object, if found, but do not return an error message if it is not found.

CASCADE

Drop dependent objects as well, if any exist.

For example:

```
premdb=> drop external format premdbs3format;
ERROR:  cannot drop external format premdbs3format because other objects depend on it
DETAIL:  external location premdbs3data depends on external format premdbs3format
HINT:  Use DROP ... CASCADE to drop the dependent objects too.
premdb=> drop external format premdbs3format cascade;
DROP EXTERNAL FORMAT
```

The `IF EXISTS` option drops the object, if found, but does not return an error message if it is not found.

You must have the correct privileges to run this command. See ON EXTERNAL object.

DROP EXTERNAL LOCATION

Platforms: EE: All cloud platforms, CE

Parent topic: SQL Commands

Drop an external location object.

```
DROP EXTERNAL LOCATION [ IF EXISTS ] name [ CASCADE ]
```

IF EXISTS

Drop the object, if found, but do not return an error message if it is not found.

CASCADE

Drop dependent objects as well, if any exist.

You must have the correct privileges to run this command. See ON EXTERNAL object.

Example

```
premdb=> drop external location premdbs3data;
DROP EXTERNAL LOCATION
```

DROP EXTERNAL MOUNT

Drop an NFS mount on an appliance that was used for external tables.

```
DROP EXTERNAL MOUNT [ IF EXISTS ] 'reference_name'
```

where `reference_name` is the name that was given to the mount when it was created. You must be a superuser to run this command.

Examples

```
premdb=# select * from sys.mounts;
       ref_name        |            resource            | options
-----------------------+--------------------------------+---------
 /local_external_tables | /home/brumsby/external_tables/ | {}
(1 row)
premdb=# drop external mount '/local_external_tables';
DROP EXTERNAL MOUNT
```

The following example demonstrates the use of the `IF EXISTS` syntax. The command does not return an error if the named mount does not exist. If it does exist, the mount is dropped.

```
premdb=# drop external mount if exists 'local';
DROP EXTERNAL MOUNT
```

DROP EXTERNAL STORAGE

Platforms: EE: All cloud platforms, CE

Parent topic: SQL Commands

Drop an external storage object.

```
DROP EXTERNAL STORAGE [ IF EXISTS ] name [ CASCADE ]
```

IF EXISTS

Drop the object, if found, but do not return an error message if it is not found.

CASCADE

Drop dependent objects as well, if any exist. You must use `CASCADE` to drop an external storage object that has a dependent external location.

You must have the correct privileges to run this command. See ON EXTERNAL object.

Example

```
premdb=> drop external storage premdbs3;
ERROR:  cannot drop external storage premdbs3 because other objects depend on it
DETAIL:  external location premdbs3data depends on external storage premdbs3
HINT:  Use DROP ... CASCADE to drop the dependent objects too.
premdb=> drop external storage premdbs3 cascade;
DROP EXTERNAL STORAGE
```

DROP KEY

Drop a current key from a database.

```
DROP KEY [ IF EXISTS ] key CASCADE
```

IF EXISTS

Do not return a warning if the key does not exist.

key

A current `key` name created with the CREATE KEY command.

Note: You must be the owner of a key to drop it.

CASCADE

This parameter is required but currently has no effect.

Note that the `DROP KEY` operation is non-transactional and will error out when run inside a transaction block.

See also Encrypting Sensitive Data.

For example:

```
premdb=# drop key ybd100key cascade;
DROP KEY
```

If a user who is not the owner of a key attempts to drop it, the command fails with the following error message:

```
premdb=# drop key yb100key cascade;
ERROR:  must be owner of key yb100key
```

DROP OWNED BY

Drop all the objects in the current database that are owned by one more specified roles.

```
DROP OWNED BY { role | CURRENT_USER | SESSION_USER } [, ...] [ CASCADE | RESTRICT ]
```

This command also revokes any privileges that were granted to the specified roles on objects in the current database.

For example:

```
premdb=# premdb=# drop owned by bobr;
DROP OWNED
```

In this example, user `yb100` cannot be dropped until objects owned by `yb100` are dropped.

```
premdb=# drop user yb100;
ERROR:  role "yb100" cannot be dropped because some objects depend on it
DETAIL:  owner of table bobr_match
owner of table bobr_team
owner of table bobr_season
premdb=# drop owned by yb100;
DROP OWNED
premdb=# drop user yb100;
DROP ROLE
```

You can use the `CASCADE` option to automatically drop objects that depend on the affected objects. `RESTRICT` is the default.

Note: Only the objects within a database can be dropped; role privileges, database-level privileges, and system-level privileges are ignored by this command.

DROP PROCEDURE

Drop a stored procedure from the database.

```
DROP PROCEDURE [IF EXISTS] procedure([ [ argname ] argtype [, ...] ])
```

For example:

```
premdb=# drop procedure if exists proc1();
DROP PROCEDURE
```

If you use the `IF EXISTS` option, `ybsql` does not return a warning if the view does not exist.

If the procedure has arguments, you must identify them by data type or by name and type. For example, either of the following commands would work for a procedure named `addteam` with one `VARCHAR` argument:

```
premdb=# drop procedure addteam(varchar(30));
DROP PROCEDURE
```

```
premdb=# drop procedure addteam(name varchar(30));
DROP PROCEDURE
```

You must drop a procedure from the database where it was created.

DROP REMOTE SERVER

Drop an existing remote server that is used for replication.

```
DROP REMOTE SERVER [IF EXISTS] name
```

If you specify the `IF EXISTS` option, the system does not return a warning if the remote server does not exist.

You cannot drop a remote server that is referenced by an existing replica (regardless of whether replication is running).

Examples

```
yellowbrick=# drop remote server ybd_repl_svr;
DROP REMOTE SERVER
```

```
premdb=# drop remote server if exists ybdrepl3;
DROP REMOTE SERVER
```

DROP ROLE

Drop one or more roles or users from the database.

```
DROP ROLE [ IF EXISTS ] name [, ...]
```

`ROLE` and `USER` are synonymous keywords in this command.

The ability to run DROP ROLE commands requires the system-level `DROP ANY ROLE` privilege. See ON SYSTEM.

Examples

For example:

```
premdb=# drop role alex, vicky, henry, niklas;
DROP ROLE
```

You cannot drop roles that have dependencies in any of your databases. Before dropping a role, drop all the objects it owns (or reassign their ownership) and revoke any privileges the role has been granted on other objects. See the DROP OWNED command. For example:

```
premdb=# drop role allqa;
ERROR:  role "allqa" cannot be dropped because some objects depend on it
DETAIL:  owner of database alexdb
```

When you use the `IF EXISTS` option, the command does not return a warning if the role does not exist.

```
premdb=# create role bobr;
CREATE ROLE
premdb=# drop role bobr;
DROP ROLE
premdb=# drop role if exists bobr;
DROP ROLE
```

DROP SCHEMA

Drop a schema from the database.

```
DROP SCHEMA [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

IF EXISTS

Do not return a warning if the schema does not exist.

name

Name the schema to drop. You can drop multiple schemas with one statement. You must drop schemas from the database where they were created.

CASCADE

If other objects depend on the schema, such as tables and views, drop them as well.

RESTRICT

If other objects depend on the schema, do not drop the schema. This is the default.

For example:

```
premdb=# drop schema premdb;
DROP SCHEMA
```

For example, attempt to drop a schema that has a dependent table:

```
premdb=# drop schema bobr;
ERROR:  cannot drop schema bobr because other objects depend on it
DETAIL:  table bobr.new depends on schema bobr
HINT:  Use DROP ... CASCADE to drop the dependent objects too.
premdb=# drop schema bobr cascade;
DROP SCHEMA
```

DROP SEQUENCE

Drop a sequence number generator.

```
DROP SEQUENCE [ IF EXISTS ] name [, ...]
```

IF EXISTS

When you use the `IF EXISTS` option, the command does not return a warning if the sequence does not exist.

name

One or more sequences to drop (optionally schema-qualified).

For example:

```
premdb=# drop sequence seq999, public.seq1000;
DROP SEQUENCE
```

DROP TABLE

Drop a table from the database.

```
DROP TABLE [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

IF EXISTS

Do not return a warning if the table does not exist.

name

Name the table to drop, optionally qualified by its schema name. You can drop multiple tables with one statement. You must drop tables from the database where they were created.

CASCADE

If other objects depend on the table, drop them as well.

Views are not treated as dependent on tables; therefore, they persist in the database when their tables are dropped with the `CASCADE` option. For example, you can create table `team`, then create view `teamview` by selecting from `team`. If you drop team with `CASCADE`, the view remains in the database. If table `team` is re-created, you can select from `teamview` based on the data in `team`.

RESTRICT

If other objects depend on the table, do not drop the table. This is the default; however, views are not dependent objects.

For example, drop three tables with one statement:

```
premdb=# drop table awayteam, hometeam, team;
DROP TABLE
```

Create a view, then drop the table that the view selects from:

```
premdb=# create view htv as select * from hometeam;
CREATE VIEW
premdb=# drop table hometeam restrict;
DROP TABLE
```

Note that the `RESTRICT` option *does not* prevent the table from being dropped. The view is considered independent and persists until it is dropped explicitly with a `DROP VIEW` statement.

Attempt to drop a table from a remote database:

```
yellowbrick=# drop table premdb.public.season;
ERROR:  cross-database references are not allowed: "premdb.public.season"
```

DROP USER

Drop one or more roles or users from the database.

```
DROP ROLE [ IF EXISTS ] name [, ...]
```

`ROLE` and `USER` are synonymous keywords in this command. See also DROP ROLE.

DROP VIEW

Drop a view from a database.

```
DROP VIEW [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

IF EXISTS

Do not return a warning if the view does not exist.

name

Name the view to drop, optionally qualified by its schema name. You can drop multiple views with one statement. You must drop views from the database where they were created.

CASCADE

If other objects depend on the view, such as other views, drop them as well.

RESTRICT

If other objects depend on the view, do not drop the view. This is the default.

For example, drop a view called `matchview`:

```
premdb=# drop view matchview;
DROP VIEW
```

For example, restrict a view from being dropped if it has dependencies:

```
premdb=# create view depview as select * from valuesview where id1=0;
CREATE VIEW
premdb=# drop view valuesview restrict;
ERROR:  cannot drop view valuesview because other objects depend on it
DETAIL:  view depview depends on view valuesview
HINT:  Use DROP ... CASCADE to drop the dependent objects too.
```

For example, note the use of the IF EXISTS option in the second statement (no error message is returned):

```
premdb=# drop view myview;
ERROR:  view "myview" does not exist
premdb=# drop view if exists myview;
DROP VIEW
```

Attempt to drop a view from a remote database:

```
yellowbrick=# drop view premdb.public.vteam;
ERROR:  cross-database references are not allowed: "premdb.public.vteam"
```

DROP WLM PROFILE

Drop a workload management (WLM) profile.

```
DROP WLM PROFILE [ IF EXISTS ] name
```

This command also drops resource pools and rules associated with the profile. Only superusers and users who have been granted `CONTROL WLM ON SYSTEM` can create, alter, and drop WLM profiles.

IF EXISTS

Do not return a warning if the profile does not exist.

name

The profile name must match the name of an existing profile in the database.

For example:

```
premdb=# drop wlm profile myprofile;
DROP WLM PROFILE
```

Query the sys.wlm_active_profile view to see the current profiles in the system and which profile is active.

Note: The `default`, `flex`, and `maintenance` profiles cannot be dropped.

DROP WLM RESOURCE POOL

Drop a workload management (WLM) resource pool.

```
DROP WLM RESOURCE POOL [ IF EXISTS ] name
```

Only superusers and users who have been granted `CONTROL WLM ON SYSTEM` can create, alter, and drop WLM resource pools. When you drop a WLM profile, its associated resource pools and rules are dropped.

IF EXISTS

Do not return a warning if the resource pool does not exist.

name

The pool name must match the name of an existing resource pool in the database.

For example:

```
premdb=# drop wlm resource pool longest_pool;
DROP WLM RESOURCE POOL
```

Query the sys.wlm_active_pool view to see the current pools in the system.

Note: The default and admin resource pools cannot be dropped.

DROP WLM RULE

Drop a workload management (WLM) rule.

```
DROP WLM RULE [ IF EXISTS ] name
```

Only superusers and users who have been granted `CONTROL WLM ON SYSTEM` can create, alter, and drop WLM rules. When you drop a WLM profile, its associated resource pools and rules are dropped.

IF EXISTS

Do not return a warning if the rule does not exist.

name

The rule name must match the name of an existing rule in the database.

For example:

```
premdb=# drop wlm rule freeway;
DROP WLM RULE
```

Query the sys.wlm_pending_rule view to see the current rules in the system.

Note: System rules, which begin with the `sys.` prefix, cannot be dropped.

END

Commit the current transaction block.

```
END [ WORK | TRANSACTION ]
```

`WORK` and `TRANSACTION` are both optional keywords; they have no effect. See also COMMIT.

EXECUTE

Execute a prepared statement.

```
EXECUTE name [ ( parameter [, ...] ) ]
```

To execute a prepared statement, supply its name and a list of parameter values. See PREPARE.

For example:

```
premdb=# prepare insert_team(int,varchar(30),varchar(20),varchar(20),varchar(50),int)
as insert into hometeam values($1, $2, $3, $4, $5, $6);
PREPARE
premdb=# execute insert_team(20,'Arsenal','The Gunners','London','Emirates Stadium',60260);
INSERT 0 1
premdb=# select * from hometeam;
 htid |       name        |  nickname   |    city    |      stadium       | capacity
------+-------------------+-------------+------------+--------------------+----------
   20 | Arsenal           | The Gunners | London     | Emirates Stadium   |    60260
    1 | Manchester United | Red Devils  | Manchester | Old Trafford       |    75635
  200 | Tottenham Hotspur | Spurs       | London     | White Hart Lane    |    36284
    2 | Leicester City    | Foxes       | Leicester  | King Power Stadium |    32262
(4 rows)
```

EXPLAIN

Return the text version of the query plan for a `SELECT` query statement, a SQL statement that runs a query, or a running query identified by its query ID.

Syntax

```
EXPLAIN ( [ ANALYZE ] [ , VERBOSE ] [ , STABLE ] [ , TYPE ] [ , NODE_OUTPUT ] [ , DISTRIBUTION ] [ , PARTITIONS ]) query | query_i
```

ANALYZE

Run the query and return both the query plan and execution details. Runtime statistics are reported, including rows returned, execution time (in milliseconds), memory use, and cache hits. The read efficiency statistic is a percentage that describes how effective the execution engine was in not reading rows from the storage engine early in the plan. A value of 0% means that no rows were discarded in this way. A high value means that the system was very efficient and incurred less I/O. Sorting a table on a column that is frequently constrained may influence read efficiency.

If you do not specify this option, EXPLAIN returns the query plan without running the query.

Warning: Running `EXPLAIN ANALYZE` on an `INSERT`, `UPDATE`, or `DELETE` statement modifies the target table, without warning. To avoid this problem, you can run the `EXPLAIN ANALYZE` statement inside a transaction, then roll it back if needed.

VERBOSE

Return a more detailed plan output. You can use this option with or without `ANALYZE` and/or `STABLE`.

STABLE

Strip out details that might vary in the plan. Only the basic structure of the plan is rendered, regardless of actual statistics. You can use this option to isolate real differences between plans when the same query is run multiple times under different conditions. You can use this option in conjunction with `VERBOSE` and/or `ANALYZE`.

Note: `LIMIT` queries that do not have an `ORDER BY` clause are non-deterministic; therefore, using the `STABLE` option for these queries is not recommended.

TYPE

Return the data types for the columns in the output.

NODE_OUTPUT

Return the list of columns that are projected from each plan node.

DISTRIBUTION

Return distribution information for each plan node.

PARTITIONS

Return information about partitioned table columns when the plan uses partitioned execution.

query

Any valid Yellowbrick query. You can run the EXPLAIN command on all of the following SQL statements:

- SELECT
- SELECT INTO
- CREATE TABLE AS (CTAS)
- INSERT
- UPDATE
- DELETE

You cannot run the EXPLAIN command on a TRUNCATE statement.

query_id

Return EXPLAIN output for a query that is currently running. Specify the query by using the `query_id` value from the `sys.query` view.

Return the plan for a SELECT * query

```
premdb=# explain select * from awayteam;
                 QUERY PLAN
-----------------------------------------------
 id   rows_planned   workers   node
  1            50    single   SELECT
  3            50    single   SCAN awayteam
 Database: premdb
 Version: 5.1.0-20201008122214
 Hostname: yb007-mgr0.yellowbrick.io

(7 rows)
```

Run the same query and return both the plan and statistics

```
premdb=# explain (analyze) select * from awayteam;
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------
 Planning time: 0.140 ms
 id   rows_planned   rows_actual   skew   workers   node
  1            50            50   0.00    single   SELECT
  3            50            50   0.00    single   SCAN awayteam
                                                     read_efficiency=0.0%, rows_from_column_store=50, rows_from_row_store=0

 50 rows returned
 Read: 40.00kiB, Distributed: 893.00B
 Database: premdb
 Version: 5.1.0-20201008122214
 Hostname: yb007-mgr0.yellowbrick.io
 query_id: 2859092
 Lookup cache hit: false
 Code cache hit: false
 Execution time: 665.396 ms, End time: 2020-11-02 12:10:48
(14 rows)
```

Run the same query and add the NODE_OUTPUT option

```
premdb=# explain (analyze, node_output) select * from awayteam;
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------
```

```
 Planning time: 0.185 ms
 id   rows_planned   rows_actual   skew   workers   node
  1             50            50   0.00    single   SELECT
                                                     (awayteam.atid, awayteam.name)
  3             50            50   0.00    single   SCAN awayteam
                                                     (awayteam.atid, awayteam.name)
                                                     read_efficiency=0.0%, rows_from_column_store=50, rows_from_row_store=0
 50 rows returned
 Read: 40.00kiB, Distributed: 893.00B
 Database: premdb
 Version: 5.1.0-20201008122214
 Hostname: yb007-mgr0.yellowbrick.io
 query_id: 43353441
 Lookup cache hit: false
 Code cache hit: true
 Execution time: 27.893 ms, End time: 2020-12-14 12:47:16
(16 rows)
```

Run the same query and add the VERBOSE option

```
premdb=# explain (analyze, node_output, verbose) select * from awayteam;
                                                                      QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------
 Planning time: 0.178 ms
 id   rows_planned   rows_actual   skew   mem_planned   mem_actual   workers   node
  1             50            50   0.00      8.00Mi H     256.14ki    single   SELECT
                                                                                (awayteam.atid, awayteam.name)
                                                                                distribute single
  3             50            50   0.00     57.00Mi H       1.00Mi    single   SCAN awayteam
                                                                                (awayteam.atid, awayteam.name)
                                                                                distribute single
                                                                                read_efficiency=0.0%, rows_from_column_store=50, r
 50 rows returned
 Memory planned (fixed): 71.00MiB, Memory planned (variable): 128.00MiB, High confidence, Memory actual: 16.00MiB
 Read: 40.00kiB, Distributed: 893.00B
 Database: premdb
 Version: 5.1.0-20201008122214
 Hostname: yb007-mgr0.yellowbrick.io
 query_id: 43353867
 Lookup cache hit: false
 Code cache hit: true
 Execution time: 30.014 ms, End time: 2020-12-14 12:48:04
(19 rows)
```

Return the plan for a query that filters rows returned from a sorted table

Note the read efficiency statistic in this example. (The table `newmatchstats` is sorted on `seasonid`.)

```
premdb=# premdb=# explain (analyze) select * from newmatchstats where seasonid=10;
                                                                 QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------
 Planning time: 0.244 ms
 id   rows_planned   rows_actual   skew   workers   node
  1          33820         33820   1.00       all   SELECT
  3          33820         33820   1.00       all   SCAN newmatchstats
                                                     (newmatchstats.seasonid::INT4 = $0) AND scan_constraints: (newmatchstats.seas
                                                     read_efficiency=99.6%, rows_from_column_store=33820, rows_from_row_store=0
 33820 rows returned
 Read: 296.00kiB, Distributed: 726.60kiB
 Database: premdb
 Version: 5.1.0-20201008122214
```

```
 Hostname: yb007-mgr0.yellowbrick.io
 query_id: 2864664
 Lookup cache hit: false
 Code cache hit: false
 Execution time: 735.490 ms, End time: 2020-11-02 12:14:02
(15 rows)
```

Use the STABLE option on the previous query

```
explain (analyze, stable) select * from newmatchstats where seasonid=10;
                                                QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
 rows_actual   workers   node
      33820       all   SELECT
                          distribute on (newmatchstats.seasonid)
      33820       all   SCAN newmatchstats
                          distribute on (newmatchstats.seasonid)
                          (newmatchstats.seasonid::INT4 = $0) AND scan_constraints: (newmatchstats.seasonid = $0)
 33820 rows returned
(7 rows)
```

Add the TYPE option to the previous query

```
premdb=# explain (analyze, stable, type) select * from newmatchstats where seasonid=10;
                                                                                QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------------------
 rows_actual   workers   node
      33820       all   SELECT
                          (newmatchstats.seasonid INT2, newmatchstats.matchday DATE, newmatchstats.htid INT2, newmatchstats.atid I
                          distribute on (newmatchstats.seasonid)
      33820       all   SCAN newmatchstats
                          (newmatchstats.seasonid INT2, newmatchstats.matchday DATE, newmatchstats.htid INT2, newmatchstats.atid I
                          distribute on (newmatchstats.seasonid)
                          (newmatchstats.seasonid::INT4 = $0) AND scan_constraints: (newmatchstats.seasonid = $0)
 33820 rows returned
(9 rows)
```

Use the DISTRIBUTION and PARTITIONS options for a query on a partitioned table

Note that the `newmatchstats` table in this example is a partitioned table:

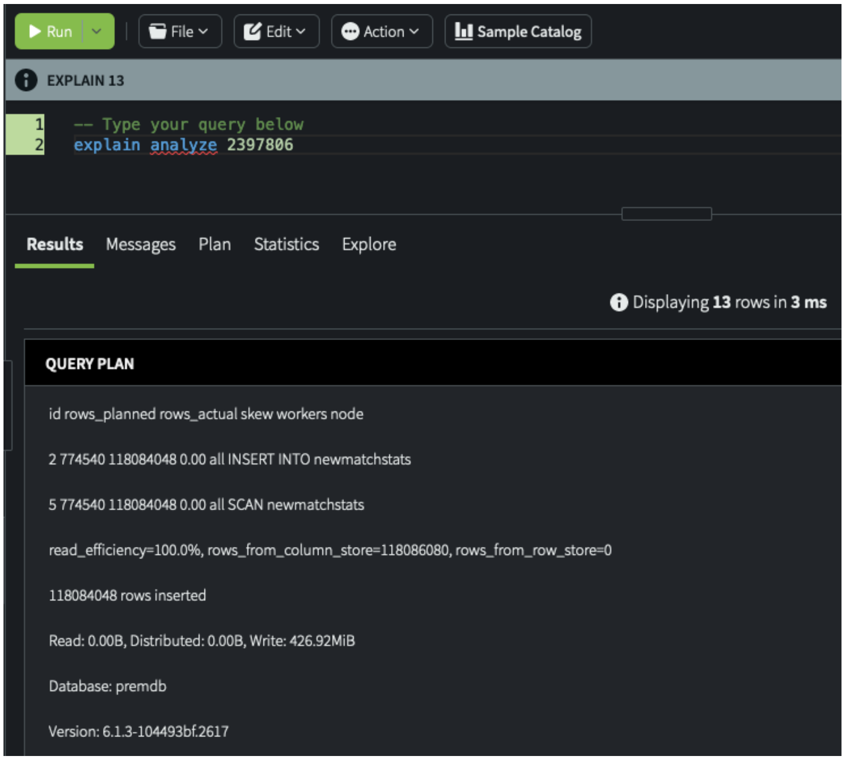
```
premdb=# describe newmatchstats;

-----------------------------------------------------------------------------------------------------------
              Table "public.newmatchstats"
 ----------+---------------------+-----------+-------------
   Column  |         Type        | Modifiers | Description
 ----------+---------------------+-----------+-------------
  seasonid | smallint            | not null  |
  matchday | date                |           |
  htid     | smallint            |           |
  atid     | smallint            |           |
  moment   | character varying(5) |           |

 Distribution: Hash (seasonid)
 Partition Columns:
    "matchday"   RANGE (BETWEEN date '1992-08-01' AND date '2017-05-31' EACH interval '1 year', IS NULL)
```

```
    "htid"       RANGE (BETWEEN 2 AND 51 EACH 1, IS NULL, OUTSIDE RANGE)

(15 rows)
```

```
premdb=# explain (distribution, partitions) select matchday, count(*) from newmatchstats
where matchday like '1996-%' group by matchday order by 1;
                                 QUERY PLAN
-----------------------------------------------------------------------------------
 id   rows_planned   workers   node
  1             11    single   SELECT
                                 distribute single
  3             11       all   DISTRIBUTE SORT
                                 distribute single
  4             11       all   SORT ON (newmatchstats.matchday ASC NULLS LAST)
                                 distribute on (newmatchstats.matchday)
  5             11       all   PARTITION BY (newmatchstats.matchday)
                                 distribute on (newmatchstats.matchday)
                                 partition by (newmatchstats.matchday)
  7              1       all   GROUP BY (newmatchstats.matchday)
                                 distribute on (newmatchstats.matchday)
  8              1       all   DISTRIBUTE ON HASH(newmatchstats.matchday)
                                 distribute on (newmatchstats.matchday)
  9              1       all   GROUP BY PARTIAL (newmatchstats.matchday)
                                 distribute on (newmatchstats.seasonid)
 11            206       all   SCAN newmatchstats
                                 distribute on (newmatchstats.seasonid)
                                 partition by (newmatchstats.matchday)
                                 newmatchstats.matchday::VARCHAR(64000)  LIKE '$1'
 Database: premdb
 Version: 5.1.0-20201008122214
 Hostname: yb007-mgr0.yellowbrick.io

(24 rows)
```

Return the query plan for a running INSERT, using its query ID

This example returns query plan information for an INSERT query that is currently running. The query ID (`2397806`) was previously selected from the `sys.query` view, then used in the EXPLAIN ANALYZE command. This example was run in the Yellowbrick Manager Query Editor.



EXPLAIN ANALYZE for a DELETE

This example shows the behavior to expect when you use the `ANALYZE` option on a statement that writes data. In this case, three rows are deleted from the season table. If you run this type of `EXPLAIN` command but do not want to delete, insert, or update the data permanently, wrap the `EXPLAIN ANALYZE` statement inside a transaction so that the changes can be rolled back.

```
premdb=# select count(*) from season;
 count
-------
   25
(1 row)

premdb=# explain analyze delete from season where winners='Arsenal';
                                                        QUERY PLAN
--------------------------------------------------------------------------------------------------------------------------
 Planning time: 0.265 ms
 id   rows_planned   rows_actual   skew   workers   node
  3              3             3   0.54    single   DISTRIBUTE RANDOM
  4              3             3   1.00    single   DELETE FROM season
  5              3             3   1.00       all   DISTRIBUTE WORKER
  7              3             3   0.00       all   SCAN season
                                                     season.winners = $0 AND scan_constraints: (season.winners = $0)
                                                     read_efficiency=0.0%, rows_from_column_store=3, rows_from_row_store=0
 3 rows deleted
 Read: 1.52MiB, Distributed: 72.00B, Write: 240.00kiB
 Database: premdb
 Version: 5.3.4-20211123225935
 Hostname: yb04-mgr0.yellowbrick.io
 query_id: 5249610
 Lookup cache hit: false
```

```
 Code cache hit: true
 Execution time: 128.120 ms, End time: 2021-11-29 14:58:35
(17 rows)

premdb=# select count(*) from season;
 count
-------
    22
(1 row)
```

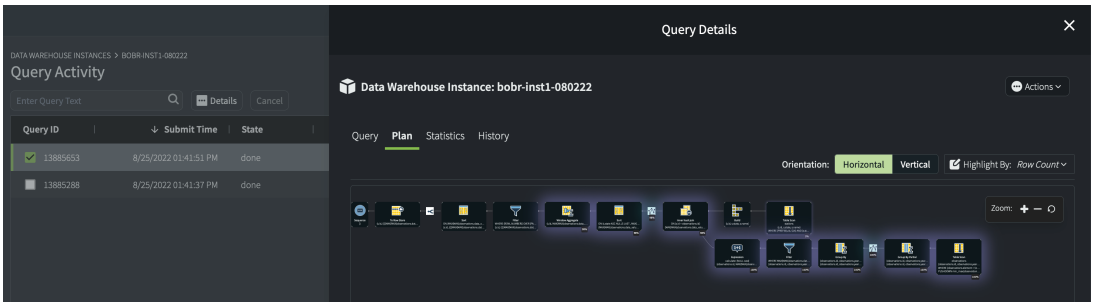
Explain I/O Metrics

Below the query plan, there is a line containing aggregated metrics for the query. These metrics may include:

- Read/Write: The amount of data read from or written to storage by the query, excluding spill I/O.
- Distributed: The number of data bytes that are distributed across workers.
- Network: The total number of bytes transmitted over the network, which may exceed the Distributed bytes field due to InfiniBand (IB) protocol overhead.
- Spill Read/Write: The amount of temporary data read or written by the query during execution.
- Spill Space: The peak amount of temporary storage space used for spilling.

Visual EXPLAIN

Tip: You may also want to use the visual EXPLAIN feature. In Yellowbrick Manager, go to **Query Activity** > **Details** > **Plan**. For example:



FETCH

Yellowbrick Documentation > Reference > SQL Reference > SQL Commands > FETCH

Platforms: All platforms

Parent topic: SQL Commands

Fetch rows from a cursor.

```
FETCH [ direction [ FROM | IN ] ] name
```

where `direction` is any one of the following options (and `name` is an open cursor):

- `NEXT | FIRST | LAST`
- `FORWARD [ count | ALL ]`
- `RELATIVE count`
- `count`
- `ALL`

If no `direction` is specified, the `FETCH` command fetches the next row.

See also MOVE cursor, which changes the position of a cursor without returning any rows. In general, the `direction` options work the same for both commands.

`FETCH` commands rely on the current position associated with the cursor. When you declare a cursor, it is positioned before the first row. After some rows are fetched, the cursor is positioned on the last retrieved row, and the next fetch proceeds from that position. If `FETCH` reaches the end of the available rows, the cursor is positioned after the last row.

NEXT, FIRST, LAST

`NEXT`, `FIRST`, and `LAST` fetch a single row after moving the cursor appropriately. If there is no such row, `FETCH` returns an empty result, and the cursor is positioned before the first row or after the last row (as appropriate).

FORWARD

Forward fetching and movement is the default behavior. `FETCH` and `MOVE` commands without a named direction default to `FORWARD` moves and fetches. Backward movement is in general not supported.

`FORWARD` returns the requested number of rows, moving forward, leaving the cursor positioned on the last-returned row (or after all rows if `count` exceeds the number of rows available). `FORWARD` without a `count` is equivalent to `NEXT`.

`MOVE FORWARD 0` and `FETCH FORWARD 0` behave as follows:

- `MOVE FORWARD 0`: Returns `MOVE 0` when specified after the cursor is declared and before any other moves or fetches. Returns `MOVE 1` when specified after any other moves or fetches.
- `FETCH FORWARD 0`: Returns no rows when specified after the cursor is declared and before any other moves or fetches. Returns an error (`Cursor can only scan forward`) when specified after other moves or fetches.

RELATIVE count

`RELATIVE count`, where `count` is positive, behaves like `FETCH count`. `RELATIVE count`, where `count` is negative, returns an error.

`RELATIVE 0` and `FORWARD 0` fetch the current row without moving the cursor. That is, they fetch the most recently fetched row again. No row is returned if the cursor is positioned before the first row or after the last row.

`MOVE RELATIVE 0` and `FETCH RELATIVE 0` behave as follows:

- `MOVE RELATIVE 0`: Returns `MOVE 0` when specified after the cursor is declared and before any other moves or fetches. Returns `MOVE 1` when specified after any other moves or fetches.
- `FETCH RELATIVE 0`: Returns no rows when specified after the cursor is declared and before any other moves or fetches. Returns an error (`Cursor can only scan forward`) when specified after other moves or fetches.

---

count

If `count` is specified by itself, it must be a positive number or `0`. See also RELATIVE count.

ALL

If `ALL` is specified by itself, it fetches all remaining rows from the current position of the cursor. `FORWARD ALL` and `ALL` are equivalent.

Minimal Support for ABSOLUTE and BACKWARD

In general, `ABSOLUTE` is not supported for moves and fetches. However, `ABSOLUTE 1` and `ABSOLUTE -1` are allowed when they are used immediately after a cursor is declared. In this case, `ABSOLUTE 1` is equivalent to `FIRST`, and `ABSOLUTE -1` is equivalent to `LAST`.

In general, `BACKWARD` and `PRIOR` are not supported for moves and fetches. Only forward fetches and movement are allowed. However, `BACKWARD count`, where `count` is a negative value, is allowed and behaves like `FETCH count`.

Examples

For example, within the following transaction, the `goals` cursor is declared, then rows are fetched forward 10 at a time until no rows are left to return:

```
premdb=# begin;
BEGIN
premdb=# declare goals cursor for
select t1.season_name, t1.winners, homegoals+awaygoals as total
from
(select season_name, winners, sum(substr(ftscore,1,1)::int) homegoals
from season, match
where season.seasonid=match.seasonid
group by season_name, winners) t1,
(select season_name, winners, sum(substr(ftscore,3,1)::int) awaygoals
from season, match
where season.seasonid=match.seasonid
group by season_name, winners) t2
where t1.season_name=t2.season_name
order by 1,2;
DECLARE CURSOR
premdb=# fetch forward 10 from goals;
 season_name |      winners      | total
-------------+-------------------+-------
 1992-1993   | Manchester United |  1222
 1993-1994   | Manchester United |  1195
 1994-1995   | Blackburn Rovers  |  1195
 1995-1996   | Manchester United |   988
 1996-1997   | Manchester United |   970
```

```
 1997-1998   | Arsenal           |  1019
 1998-1999   | Manchester United |   959
 1999-2000   | Manchester United |  1060
 2000-2001   | Manchester United |   992
 2001-2002   | Arsenal           |  1001
(10 rows)

premdb=# fetch forward 10 from goals;
 season_name |      winners      | total
-------------+-------------------+-------
 2002-2003   | Manchester United |  1000
 2003-2004   | Arsenal           |  1012
 2004-2005   | Chelsea           |   975
 2005-2006   | Chelsea           |   944
 2006-2007   | Manchester United |   931
 2007-2008   | Manchester United |  1002
 2008-2009   | Manchester United |   942
 2009-2010   | Chelsea           |  1053
 2010-2011   | Manchester United |  1063
 2011-2012   | Manchester City   |  1066
(10 rows)

premdb=# fetch forward 10 from goals;
 season_name |      winners      | total
-------------+-------------------+-------
 2012-2013   | Manchester United |  1063
 2013-2014   | Manchester City   |  1052
(2 rows)

premdb=# fetch forward 10 from goals;
 season_name | winners | total
-------------+---------+-------
(0 rows)

premdb=# end;
COMMIT
```

In this example, `FETCH LAST` returns an error because the cursor has already been moved beyond the last row. Backward fetches are not supported, so going back to return the last row in the table is not possible.

```
premdb=# declare allmatches cursor for select * from match order by 1,2;
DECLARE CURSOR
premdb=# fetch first in allmatches;
 seasonid |      matchday       | htid | atid | ftscore | htscore
----------+---------------------+------+------+---------+---------
        1 | 1992-08-01 00:00:00 |   14 |   52 | 0-1     | -
(1 row)

premdb=# move forward 8600 in allmatches;
MOVE 8600
premdb=# fetch last in allmatches;
 seasonid |      matchday       | htid | atid | ftscore | htscore
----------+---------------------+------+------+---------+---------
       22 | 2014-05-11 00:00:00 |   39 |   89 | 1-3     | 0-2
(1 row)

premdb=# fetch last in allmatches;
ERROR:  Cursor can only scan forward
```

The following `FETCH` commands demonstrate several different ways to fetch the next row:

```
premdb=# begin;
BEGIN
```

```
premdb=# declare match cursor for select * from match order by 1,2,3,4;
DECLARE CURSOR
premdb=# fetch from match;
 seasonid |      matchday       | htid | atid | ftscore | htscore
----------+---------------------+------+------+---------+---------
        1 | 1992-08-01 00:00:00 |    2 |   52 | 0-1     | -
(1 row)

premdb=# fetch in match;
 seasonid |      matchday       | htid | atid | ftscore | htscore
----------+---------------------+------+------+---------+---------
        1 | 1992-08-01 00:00:00 |    2 |   55 | 0-1     | -
(1 row)

premdb=# fetch match;
 seasonid |      matchday       | htid | atid | ftscore | htscore
----------+---------------------+------+------+---------+---------
        1 | 1992-08-01 00:00:00 |    2 |   63 | 2-1     | -
(1 row)

premdb=# fetch 1 match;
 seasonid |      matchday       | htid | atid | ftscore | htscore
----------+---------------------+------+------+---------+---------
        1 | 1992-08-01 00:00:00 |    2 |   64 | 3-0     | -
(1 row)
```

In this example, `FETCH FORWARD 0` returns no rows when run immediately after the cursor is declared, but returns an expected error when rows have been fetched or the cursor position has moved forward:

```
premdb=# declare match cursor for select * from match order by 1,2;
DECLARE CURSOR
premdb=# fetch forward 0 match;
 seasonid | matchday | htid | atid | ftscore | htscore
----------+----------+------+------+---------+---------
(0 rows)

premdb=# fetch forward 0 match;
 seasonid | matchday | htid | atid | ftscore | htscore
----------+----------+------+------+---------+---------
(0 rows)

premdb=# fetch 1 match;
 seasonid |      matchday       | htid | atid | ftscore | htscore
----------+---------------------+------+------+---------+---------
        1 | 1992-08-01 00:00:00 |   14 |   52 | 0-1     | -
(1 row)

premdb=# fetch forward 0 match;
ERROR:  Cursor can only scan forward
```

HELP

Return help text for a SQL command.

```
HELP [ command ]
```

For example:

```
yellowbrick=# help cancel;
 COMMAND |       DESCRIPTION       |     SYNTAX
---------+-------------------------+-----------------
 CANCEL  | Cancel a running query. | CANCEL query_id
(1 row)
yellowbrick=# help drop view;
  COMMAND  |         DESCRIPTION          |                          SYNTAX
-----------+------------------------------+-------------------------------------------------------------
 DROP VIEW | Drop a view from a database. | DROP VIEW [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
(1 row)
```

If you do not specify a command, `HELP` returns a list of all SQL commands. The `HELP` command returns the same output as the `ybsql \h` command.

IMPORT SSL TRUST

Import the remote SSL system configuration into the source system.

```
IMPORT SSL TRUST FROM 'certificate'
```

Non-superusers must have `sysadmin` (or `consumeradmin`) membership in order to run SSL trust commands.

For example:

```
yellowbrick=# IMPORT SSL TRUST FROM '-----BEGIN CERTIFICATE-----
yellowbrick'# MIIDjDCCAnSgAwIBAgIJAJ5N7lPpO9qOMA0GCSqGSIb3DQEBCwUAMG8xCzAJBgNV
yellowbrick'# BAYTAlVTMQswCQYDVQQIDAJDQTEfMB0GA1UECgwWWWVsbG93YnJpY2sgRGF0YSwg
yellowbrick'# SW5jLjENMAsGA1UEAwwEZW56bzEjMCEGCSqGSIb3DQEJARYUaW5mb0B5ZWxsb3di
yellowbrick'# cmljay5jb20wHhcNMTkwOTA5MTM0MDIyWhcNMjkwOTA4MTM0MDIyWjBvMQswCQYD
yellowbrick'# VQQGEwJVUzELMAkGA1UECAwCQ0ExHzAdBgNVBAoMFlllbGxvd2JyaWNrIERhdGEs
yellowbrick'# IEluYy4xDTALBgNVBAMMBGVuem8xIzAhBgkqhkiG9w0BCQEWFGluZm9AeWVsbG93
yellowbrick'# YnJpY2suY29tMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAzLTxNcTi
yellowbrick'# h3FPeALtlPPBMpa4/btvDrX78vBtoQDDrxsNY/rRohDiimXlc/GSiUUNc1c8lSN1
yellowbrick'# O3BG9JJx5S65x3xqpukv6PD1auDmqEtK6PSE7ycvqQ82w/U6QoHKunwjrSDqfFYj
yellowbrick'# klRcXo/U2F6uep2limixFvwPJ8IdA/FawPWV8vdp7fTW5PB6HAELG0l1RkaFFGnP
yellowbrick'# nnelB4tLPl8+ItiKnrivXHltY36h13Q5iRWHFHQRF+A5SRdeKWUpEOqtqGp0w6gS
yellowbrick'# eBGH3t7wFuXxBznzX5XMrW72M0s3lo0P//47WxFS+TC8ajVxoD6N1agdwS9Nxi/v
yellowbrick'# SC2rswQESmYUhQIDAQABoyswKTAJBgNVHRMEAjAAMAsGA1UdDwQEAwIF4DAPBgNV
yellowbrick'# HREECDAGggRlbnpvMA0GCSqGSIb3DQEBCwUAA4IBAQA0tjJgfK2rtwOnzgDajzXP
yellowbrick'# ra0w/PPVz8hg4qIDXKcECtdGzLZH57YOXztMwNRkj6B1vtvPyA48DCur5oEHuXD5
yellowbrick'# E6QOCylTOxMJPQzsklRfdspPdi4/YtdbXGoEkVjp/I9jj7mINhcyjGNBwUmBWTO5
yellowbrick'# 2OQ5hxN5pAyMTv7nCHfy34EecB4SuMl7AENV8Lj95MNK5N8Hzu4xIFKaLco2Urb/
yellowbrick'# cBndVlOOE1s/Nqa7yKuqRgnfR3VRlHUgDrF+E8QGeUwEyftOdpb2TrhBDiEVv1av
yellowbrick'# NemH8HbTg5/G9k61hf1e6X+dFBPtrTgnwe5FClWHqnNaKxTMwj/ffeLS7T7bPyQ9
yellowbrick'# -----END CERTIFICATE-----';
IMPORT SSL TRUST
```

INSERT

Insert rows or column values into a table either by selecting them from other tables or by specifying a list of values to insert.

```
[ WITH name AS (subquery) [, ...] ]
INSERT INTO table_name [ AS alias ] [ ( column_name [, ...] ) ]
   { DEFAULT VALUES | VALUES ( { expression } [, ...] ) [, ...] | query }
```

WITH

See WITH Clause for syntax details.

table_name

Specify the name of a table in the current database.

Note: You cannot use `INSERT` commands on tables in remote databases; see Cross-Database Queries.

Warning: Running `EXPLAIN ANALYZE` on an `INSERT`, `UPDATE`, or `DELETE` statement modifies the target table, without warning. To avoid this problem, you can run the `EXPLAIN ANALYZE` statement inside a transaction, then roll it back if needed.

DEFAULT VALUES

Insert all columns with default values if they are defined; otherwise, insert `NULL` values.

VALUES(expression [,...])

Insert a list of specific values into the table. You do not have to specify values for all columns.

To insert multiple rows, use a comma delimiter between each list of expressions. Do not repeat the `VALUES` keyword. All `VALUES` lists for a multiple-row INSERT statement must contain the same number of values.

Note: You cannot insert multiple rows at a time if the table contains encrypted columns.

VALUES (DEFAULT [,...])

Insert the specified columns with default values if they are defined; otherwise, insert `NULL` values.

Note: You can define WLM rules for `INSERT INTO...SELECT` statements but not for `INSERT INTO...VALUES` statements. `INSERT INTO...VALUES` operations are frontend queries that do not require a cluster and are not executed by the backend database.

query

Insert the results of any valid query into the named table. Use a SELECT statement.

For example, insert a set of values into the `season` table:

```
premdb=# insert into season
values(26, '2017-2018', 20, null);
INSERT 0 1
```

Insert multiple rows into the `season` table in one statement:

```
premdb=# insert into season
values(26, '2017-2018', 20, null),(27,'2018-2019',21,null);
INSERT 0 2
```

For example, load the `hometeam` table by selecting from the `team` table:

```
premdb=# insert into hometeam
select htid,name from team;
INSERT 0 50
```

Insert a row of `NULL` values into the `season` table, which has no `DEFAULT` values assigned to its columns:

```
premdb=# insert into season default values;
INSERT 0 1
```

Query Processing for INSERT Commands

`INSERT INTO...VALUES` and `INSERT INTO...SELECT` statements are processed differently. You can see the difference by using the `EXPLAIN` command. For example:

```
premdb=# explain insert into team values(200,100);
                        QUERY PLAN
------------------------------------------------------------
 Insert on team (on manager)  (cost=0.00..0.00 rows=1 width=0)
   ->  Result (on manager)  (cost=0.00..0.00 rows=1 width=0)
(2 rows)

premdb=# explain insert into team select 200,htid from hometeam;
                   QUERY PLAN
-------------------------------------------------
 id   rows_planned   workers   node
  1             50    single   INSERT INTO team
  3             50       all   SCAN hometeam
 Database: premdb
 Version: 4.1.0-23999
 Hostname: yb100

(7 rows)
```

The first `INSERT` updates the row store ("`on manager`") and the rows will later be flushed to the column store. The second `INSERT` updates the column store directly ("`workers`"). See also Managing the Row Store.

LIST BUCKETS

Platforms: EE: All cloud platforms, CE

Parent topic: SQL Commands

Return a complete or partial list of the sub-folders within a bucket.

The `LIST BUCKETS` command returns the name of each object, its size (in bytes), when it was created, and when it was last modified. The size and creation time may not be available for all objects. This command is an alternative to LIST OBJECTS, which returns a more extensive list of folders and files in object storage.

```
LIST BUCKETS
EXTERNAL LOCATION name
[ PREFIX (prefix) ]
[ LIMIT limit ]
```

EXTERNAL LOCATION

Name of an external location object. See CREATE EXTERNAL LOCATION.

PREFIX

Optional prefix string that filters the list of objects. You can specify a folder name or a common prefix that is used for a set of files. For example:

```
PREFIX ('healthcare')
PREFIX ('healthcare/allergies')
PREFIX ('healthcare/allergies_10')
```

LIMIT

An integer up to `1000` that limits the number of objects returned, given the filters applied.

LIST OBJECTS

Platforms: EE: All cloud platforms, CE

Parent topic: SQL Commands

Return a complete or partial list of the objects that are stored in a specific external location.

The `LIST OBJECTS` command returns the name of each object, its size (in bytes), when it was created, and when it was last modified. The size and creation time may not be available for all objects.

```
LIST OBJECTS
EXTERNAL LOCATION name
[ PREFIX (prefix) ]
[ SUFFIX (suffix) ]
[ LIMIT limit ]
[ START AFTER 'start_key' ]
```

EXTERNAL LOCATION

Name of an external location object. See CREATE EXTERNAL LOCATION.

PREFIX

Optional prefix string that filters the list of objects. You can specify a folder name or a common prefix that is used for a set of files. For example:

```
PREFIX ('healthcare')
PREFIX ('healthcare/allergies')
PREFIX ('healthcare/allergies_10')
```

SUFFIX

Optional suffix string that filters the list of objects. For example, you can specify a file extension or a character such as `\` (the default suffix in the **Load Data Assistant**):

```
SUFFIX('.csv')
SUFFIX('.gz')
SUFFIX('\')
```

LIMIT

An integer up to `1000` that limits the number of objects returned, given the filters applied.

START AFTER

A string that refers to an object name (or key name), which is used as a starting point for the list. For example:

```
START AFTER 'premdb/newteam.csv'
```

You can use this option to request the next set of objects when you reach the limit of 1000.

Note that the string specified here is not interpreted relative to the `PREFIX` string.

Examples

For example, you can list objects that have the `premdb` prefix within the `premdbs3data` location. The list consists of six objects, including the empty `premdb` folder. The following examples show how the `PREFIX`, `SUFFIX`, `LIMIT`, and `START AFTER` options work; they can be used in various combinations. The output for these examples is copied from the Clipboard in CSV format.

Return objects that begin with `premdb` and end with `csv`:

```
list objects
external location premdbs3data
prefix('premdb')
suffix('csv');

"object_name","size_bytes","creation_time","modify_time"
"premdb/match.csv",313142,,"2021-04-27T19:17:46-07:00"
"premdb/newmatchstats25mil.csv",629210880,,"2021-05-04T16:36:27-07:00"
"premdb/newteam.csv",3026,,"2021-04-27T19:17:43-07:00"
"premdb/player.csv",1286,,"2021-04-27T19:17:42-07:00"
"premdb/season.csv",746,,"2021-04-27T19:17:43-07:00"
```

Return the first 3 objects that start with `premdb` and end with `csv`:

```
list objects
external location premdbs3data
prefix('premdb')
suffix('csv')
limit 3;

"object_name","size_bytes","creation_time","modify_time"
"premdb/match.csv",313142,,"2021-04-27T19:17:46-07:00"
"premdb/newmatchstats25mil.csv",629210880,,"2021-05-04T16:36:27-07:00"
"premdb/newteam.csv",3026,,"2021-04-27T19:17:43-07:00"
```

Starting after `premdb/newteam.csv`, return objects that begin with `premdb` and end with `csv`:

```
list objects external location premdbs3data
prefix('premdb/')
suffix('csv')
start after 'premdb/newteam.csv';

"object_name","size_bytes","creation_time","modify_time"
"premdb/player.csv",1286,,"2021-04-27T19:17:42-07:00"
"premdb/season.csv",746,,"2021-04-27T19:17:43-07:00"
```

See also Loading a Table via the Load Assistant.

LOAD TABLE

Bulk load a table from external storage by using a SQL command (instead of the `ybload` client).

```
LOAD TABLE table_name
[ SOURCE FIELDS (col1 type [, col2 type ] [, ...]) ]
[ FROM ('string1' [,'string2'] [, ...]) ]
EXTERNAL LOCATION location_name
[ EXTERNAL FORMAT { format_name |
  (TYPE file_type WITH (option value [, option value ] [, ...])) } ]
[ WITH (option value [, option value ] [, ...]) ]
```

table_name

Name of a *regular user table* that is the target for the load. The source files will be read from external storage and written to the table. To run `LOAD TABLE` commands, you must have `BULK LOAD` privilege on the database and `INSERT` privilege on the target table.

SOURCE FIELDS

Specify all of the fields found in the source files and their data types. These fields map to the columns in the target table that you want to load. *You must specify all of the fields that appear in the source files, in order*, but you can load a table that only contains a subset of those fields. For example, you can load a table with four columns from a source file that has five or more fields. You do not have to specify the same order of columns in the `CREATE TABLE` statement. The load operation can detect which columns correspond to which source fields by name and data type.

In this example, the source file contains five fields:

```
...
source fields(seasonid smallint,
  matchday timestamp,
  htid smallint,
  atid smallint,
  moment varchar(5))
...
```

You can load a table that has one or more columns that correspond to these fields. See also LOAD TABLE Examples.

Conversely, you can also load a table that has columns that do not have corresponding fields in the source file if `DEFAULT` column constraints are specified in the `CREATE TABLE` statement. For example, you can load a two-column table from a source file that has only one field. In this case, `SOURCE FIELDS` identifies the one field that exists in the source file; the second column in the table is loaded with the specified default value. For example:

```
create table t1 (c1 int, c2 int not null default 1);
...
load table t1
source fields (c1 int)
...
```

Note: The `SOURCE FIELDS` option is not available in the Load Data Assistant.

FROM('string1','string2', ...)

Specify one or more character strings that represent a list of the S3 source files or a "prefix" that identifies those files. These files will be loaded from the specified external location, which is a bucket name.

If you do not specify a `FROM` list, the load operation will scan the entire bucket and load any files that have a schema that matches the columns in the target table. Therefore, it is recommended that you specify a `FROM` list or prefix.

If a string exactly matches the key of a specific file stored in the bucket, only that S3 file will be loaded. Otherwise, the string is checked for matches against the beginning part of the key of any objects in the bucket. All of the files that match will be loaded.

For example, the following commands all produce the same results, loading the same two files:

```
premdb=# load table match from ('/premdb/match.csv','/premdb/matchcopy.csv') external location premdbs3data;
INSERT 0 17212

premdb=# load table match from ('/premdb/match') external location premdbs3data;
INSERT 0 17212
```

Note: Wildcard patterns are not supported for S3 source files. For example, you cannot use the following `FROM` clause to load files that have a prefix of `match`:

```
from ('/premdb/mat*.csv')
```

Instead you could use:

```
from ('/premdb/mat')
```

EXTERNAL LOCATION

Name of an external location object. See CREATE EXTERNAL LOCATION.

EXTERNAL FORMAT

Name of an external format object. See CREATE EXTERNAL FORMAT. Alternatively, you can define the format in the `TYPE` clause. You can also omit the `EXTERNAL FORMAT` clause completely and use the default format settings for the table.

TYPE...WITH

Specify the source file type: `CSV`, `TEXT`, `PARQUET`, or `BCP`. You must enclose the `TYPE` clause inside parentheses, and the following `WITH` clause inside its own set of parentheses:

```
(TYPE format_type WITH(...))
```

The `WITH` clause that follows `TYPE` defines *formatting load options*, such as the line separator, field delimiter, and field definitions for different data types. Each name-value pair must consist of a valid option (in this context) and a valid setting for that option. For example:

```
(type csv with (skip_blank_lines true, delimiter '|'))
```

See External Format Load Options.

WITH

The `WITH` clause at the end of the `LOAD TABLE` statement defines *general load-processing options*. Each name-value pair must consist of a valid option (in this context) and a valid setting for that option. These are separate and distinct from the formatting load options defined by the external format object or in the `TYPE WITH()` list.

For example:

```
with (source_compression 'gz')
```

See LOAD TABLE Options.

LOAD TABLE Examples

Load a table from a single CSV file:

```
load table match
from ('/premdb/match.csv')
external location premdbs3data;
INSERT 0 8606
```

Load a table from files that match the prefix `/premdb/newmat`:

```
load table newmatchstats
from ('/premdb/newmat')
external location premdbs3data;
INSERT 0 24785280
```

Load a table from a single GZ-compressed file:

```
load table newmatchstats
from ('/premdb/newmatchstats25mil.gz')
external location premdbs3data
with(source_compression 'gz');
INSERT 0 24785280
```

Attempt to load some data that is not in the specified file format (given the specified prefix):

```
load table newmatchstats
from ('/premdb/newmatchstats')
external location premdbs3data
with(source_compression 'gz');
ERROR:  [bulksvc-load] Unable to open channel for s3://ybpremdb/premdb/newmatchstats25mil.csv CAUSED BY: Input is not in the .gz f
```

Load a table and reference an existing external format object:

```
load table newmatchstats
from ('/premdb/newmatchstats25mil.csv')
external location premdbs3data
external format premdbs3format;
```

Load a table and define the external format within the statement, instead of referencing an existing external format object:

```
load table newmatchstats
from ('/premdb/newmatchstats25mil.csv')
```

```
external location premdbs3data
external format (type csv with(skip_blank_lines true, delimiter ','));
```

Create a new table by selecting four columns from an existing five-column table. Then delete all of its rows. Reload the table by using the same source file that loaded the original five-column table. Specify all five source fields in the `LOAD TABLE` statement.

```
create table matchday as
select seasonid, matchday, htid, atid from newmatchstats;
SELECT  24785280

delete from matchday;
DELETE  24785280

load table matchday
source fields(seasonid smallint,
  matchday timestamp,
  htid smallint,
  atid smallint,
  moment varchar)
from('/premdb/newmatch')
external location premdbs3data;
INSERT  24785280
```

Create a table with two columns, `c1` and `c2`:

```
create table t1(c1 int, c2 int);
```

Load the table from a source file that contains three fields that are in a different order from the corresponding columns in the `CREATE TABLE` statement:

```
load table t1
source fields(c3 int, c2 int, c1 int)
...
```

LOAD TABLE Options

The `WITH` clause in the `LOAD TABLE` statement supports some load-processing options and source options. These options are listed alphabetically.

log_level INFO | OFF | WARN | ERROR | DEBUG | TRACE

Specify the logging level for the load output. The default level is `INFO`. Other options provide more verbose output.

max_bad_rows NUMBER

Set the maximum number of rejected rows that a load will tolerate before aborting and *starting* to roll back the transaction. The default is `0` (no bad rows); make sure that the source data you are about to load is clean and consistent if you use the default.

Note: `max_bad_rows 100` means 100 bad rows are allowed; the load will fail on the 101st bad row.

read_sources_concurrently ALWAYS | NEVER | ALLOW | <number>

Define the behavior for reading source files in parallel: `ALWAYS`, `NEVER`, `ALLOW` (the default), or a specific number of source files. See "ybload Advanced Processing Options" in the main Yellowbrick documentation.

send_compression ALWAYS | NEVER | AUTO

Define the compression policy for data buffers before they are sent from the client to the compute nodes. This option is equivalent to the `--compression-policy` option in `ybload`. See "ybload Advanced Processing Options" in the main Yellowbrick documentation.

source_compression GZ | BZIP2 | XZ | PACK200 | LZ4

This option explicitly defines the type of compression used by source data. This option applies to loads from all supported source types and overrides other compression detection methods.

Order of Precedence for Format Specifications

There are three different ways to specify the format for a `LOAD TABLE` operation. The following order of precedence is enforced:

1. In-line format options, as specified with `EXTERNAL FORMAT (TYPE...WITH(…))`
2. An in-line format name, as specified with `EXTERNAL FORMAT format_name`
3. A default format, if one exists, for the external location object specified in the `LOAD TABLE` command

If no format options are specified at all, the behavior is similar to that of `ybload` operations, which means that some values are automatically detected and the load is attempted with those settings. (The load may fail in this case.)

LOCK SEQUENCE

Lock a sequence within a transaction so that other users and sessions cannot modify the sequence until the locking transaction ends or is rolled back.

```
LOCK SEQUENCE name
```

This command must be run within a transaction block. For example:

```
premdb_session1=# begin;
BEGIN
premdb_session1=# lock sequence matchkey;
LOCK
premdb_session1=# ...
```

Another session's attempt to set a new start value for this sequence has to wait until this transaction commits or rolls back:

```
premdb_session2=# alter sequence matchid start 5000000;
...
<locking transaction ends>
ALTER SEQUENCE
```

MOVE cursor

Yellowbrick Documentation > Reference > SQL Reference > SQL Commands > MOVE cursor

Platforms: All platforms

Parent topic: SQL Commands

Change the position of a cursor without returning any rows.

```
MOVE [ direction [ FROM | IN ] ] name
```

where `direction` is one of the options defined in the description of FETCH, and `name` is an open cursor.

For example, declare a cursor, move 1, then move 10. `FETCH NEXT` returns the 12th row in the table:

```
premdb=# begin;
BEGIN
premdb=# declare team cursor for select * from team order by teamid;
DECLARE CURSOR
premdb=# move team;
MOVE 1
premdb=# move forward 10 in team;
MOVE 10
premdb=# fetch next from team;
 teamid | htid | atid |       name        | nickname |  city  |  stadium   | capacity
--------+------+------+-------------------+----------+--------+------------+----------
     12 |   13 |   62 | Charlton Athletic | Addicks  | London | The Valley |    27111
(1 row)
```

In this example, the second `MOVE 10` command returns `MOVE 6` because the end of the rows in the cursor has been reached:

```
premdb=# select count(*) from match;
 count
-------
  8606
(1 row)
premdb=# begin;
BEGIN
premdb=# declare match cursor for select * from match order by 1,2,3,4;
DECLARE CURSOR
premdb=# move 8000 match;
MOVE 8000
premdb=# fetch 600 match;
 seasonid |      matchday       | htid | atid | ftscore | htscore
----------+---------------------+------+------+---------+---------
       21 | 2012-12-08 00:00:00 |   46 |   82 | 2-2     | 1-1
       21 | 2012-12-09 00:00:00 |   18 |   91 | 2-1     | 0-0
       21 | 2012-12-09 00:00:00 |   25 |   75 | 2-3     | 0-2
       21 | 2012-12-09 00:00:00 |   45 |   73 | 2-3     | 2-1
       21 | 2012-12-10 00:00:00 |   19 |   77 | 2-1     | 1-0
       21 | 2012-12-11 00:00:00 |   39 |   83 | 3-0     | 2-0
...
premdb=# move 10 match;
MOVE 6
premdb=# fetch next match;
 seasonid | matchday | htid | atid | ftscore | htscore
```

```
----------+----------+------+------+---------+---------
(0 rows)
```

See also FETCH.

MOVE query

Move a query to another WLM resource pool.

```
MOVE query TO WLM RESOURCE POOL pool [ WITH ( resource value [,...] ) ]
```

query

The ID of a running query (`query_id` column in sys.query).

pool

Name of a resource pool in the active WLM profile. A query can only be moved to a pool, or restarted in a pool, that is part of the currently active WLM configuration. For more details, see Moving Queries.

resource value

Optional settings for resources: `memory` , `spill` , and `priority` .

Valid settings for `memory` and `spill` are numbers (explicitly or implicitly in MB) or percentages. For example, `memory '5000'` , `memory '1000MB'` , and `spill '10%'` are all valid.

Valid settings for `priority` are `low` , `normal` , `high` , and `critical` .

Examples

For example:

```
premdb# move 17831 to wlm resource pool flexpool10;
MOVE
```

The following examples request specific resources to use after the query is moved:

```
premdb=# move 5547012 to wlm resource pool long with(memory '100MB',priority low,spill '100MB');
MOVE
premdb=# move 13450004 to wlm resource pool long with(priority critical);
MOVE
```

PREPARE

Prepare a SQL statement for execution with the EXECUTE command and define parameters that you supply on execution.

```
PREPARE name [ ( data_type [, ...] ) ] AS statement
```

Prepared statements exist only for the duration of a session. (To remove prepared statements within a session, use the DEALLOCATE command.)

data_type

Define the data type for each parameter. See SQL Data Types for information about supported types. Refer to the parameters in the prepared statement by using variables of the form `$1`, `$2`, and so on.

statement

Use any supported SELECT, INSERT, UPDATE, or DELETE statement. Parameters are not supported in GROUP BY and ORDER BY clauses.

For example, this prepared query refers to a parameter for the `season_name` column twice (once in each subquery):

```
prepare goals(varchar(9)) as
select a.name, homegoals, awaygoals,
homegoals+awaygoals as goals_for
from
(select t1.name, sum(substr(ftscore,1,1)::int)
from match m1, team t1, season s1
where (m1.htid=t1.htid and m1.seasonid=s1.seasonid)
and season_name=$1
group by t1.name) as a(name,homegoals),
(select t2.name, sum(substr(ftscore,3,1)::int)
from match m2, team t2, season s2
where (m2.atid=t2.atid and m2.seasonid=s2.seasonid)
and season_name=$1
group by t2.name) as b(name,awaygoals)
where a.name=b.name
order by 4 desc;
PREPARE
```

```
premdb=# execute goals('1999-2000');
       name         | homegoals | awaygoals | goals_for
--------------------+-----------+-----------+-----------
 Manchester United  |        59 |        38 |        97
 Arsenal            |        42 |        31 |        73
 Newcastle United   |        42 |        21 |        63
 Everton            |        36 |        23 |        59
 Leeds United       |        29 |        29 |        58
 Tottenham Hotspur  |        40 |        17 |        57
 Sunderland         |        28 |        29 |        57
 Leicester City     |        31 |        24 |        55
 Chelsea            |        35 |        18 |        53
 West Ham United    |        32 |        20 |        52
 Liverpool          |        28 |        23 |        51
 Coventry City      |        38 |         9 |        47
```

```
 Middlesbrough       |        23 |        23 |        46
 Aston Villa         |        23 |        23 |        46
 Wimbledon           |        30 |        16 |        46
 Southampton         |        26 |        19 |        45
 Derby County        |        22 |        22 |        44
 Sheffield Wednesday |        21 |        17 |        38
 Bradford City       |        26 |        12 |        38
 Watford             |        24 |        11 |        35
(20 rows)
```

```
premdb=# deallocate goals;
DEALLOCATE
premdb=# execute goals('2003-2004');
ERROR:  prepared statement "goals" does not exist
premdb=#
```

REASSIGN OWNED BY

Change the ownership of database objects that belong to a database role.

```
REASSIGN OWNED BY { old_role | CURRENT_USER | SESSION_USER } [, ...]
TO { new_role | CURRENT_USER | SESSION_USER }
```

This command is typically run before a specific user is dropped. This command only affects objects within the current database, so you may have to repeat it in each database that contains objects owned by the user in question. Even then you may find that *privileges* on other objects need to be revoked before the user can be dropped. The `REASSIGN OWNED BY` command does not affect any privileges granted to roles on objects not owned by them.

The `REASSIGN OWNED BY` command must be run by a user with privileges on both the old role (or roles) and the new role.

The DROP OWNED BY command is an alternative command that drops all the database objects owned by the specified role or roles (assuming there are no dependencies).

Example

For example, reassign ownership of objects from user `bobr` to user `yb100`. Other objects in the database may be owned by `bobr`, not just the tables listed in this example. The `REASSIGN OWNED BY` command transfers ownership of all objects in the premdb database that are currently owned by `bobr`.

```
premdb=> \d
          List of relations
 Schema |    Name     | Type  |  Owner
--------+-------------+-------+---------
 public | awayteam    | table | brumsby
 public | bobr_match  | table | bobr
 public | bobr_season | table | bobr
 public | bobr_team   | table | bobr
 public | hometeam    | table | brumsby
 public | match       | table | brumsby
 public | season      | table | brumsby
 public | team        | table | brumsby
(8 rows)

premdb=> \c premdb yellowbrick
Password for user yellowbrick:
You are now connected to database "premdb" as user "yellowbrick".

premdb=# create user yb100;
CREATE ROLE
premdb=# reassign owned by bobr to yb100;
REASSIGN OWNED
premdb=# \d
          List of relations
 Schema |    Name     | Type  |  Owner
--------+-------------+-------+---------
 public | awayteam    | table | brumsby
 public | bobr_match  | table | yb100
 public | bobr_season | table | yb100
 public | bobr_team   | table | yb100
```

```
 public | hometeam    | table | brumsby
 public | match       | table | brumsby
 public | season      | table | brumsby
 public | team        | table | brumsby
(8 rows)
```

RELEASE SAVEPOINT

Destroy a savepoint within the current transaction.

```
RELEASE [ SAVEPOINT ] name
```

When a savepoint is destroyed, the transaction cannot roll back to that point. This command also destroys all savepoints that were defined after the named savepoint was defined.

For example, if you have created savepoints `one`, `two`, and `three` but you destroy savepoint `two`, savepoint `three` is also destroyed.

In the following example, when savepoint `insert1` is released, `insert2` is also destroyed, and the entire transaction is rolled back. No rows are inserted.

```
premdb=# begin;
BEGIN
premdb=# insert into season(seasonid) values(100);
INSERT 0 1
premdb=# savepoint insert1;
SAVEPOINT
premdb=# insert into season(seasonid) values(200);
INSERT 0 1
premdb=# savepoint insert2;
SAVEPOINT
premdb=# release savepoint insert1;
RELEASE
premdb=# rollback to insert2;
ERROR:  no such savepoint
premdb=# rollback to insert1;
ERROR:  no such savepoint
premdb=# commit;
ROLLBACK
premdb=# select * from season where seasonid in(100,200);
 seasonid | season_name | numteams | winners
----------+-------------+----------+---------
(0 rows)
```

RESET

Reset the value of a configuration parameter to its default value.

```
RESET parameter
```

In this example, the reset value is the built-in default for the `search_path` parameter.

```
premdb=# show search_path;
   search_path
-----------------
 "$user", public
(1 row)
premdb=# set search_path = 'public';
SET
premdb=# reset search_path;
RESET
premdb=# show search_path;
   search_path
-----------------
 "$user", public
(1 row)
```

Reset values are subject to the rules under Precedence of Settings for Configuration Parameters. If the parameter has not been changed, it resets to its built-in default; however, this default may have been overridden globally with ALTER SYSTEM, and that value may also have been changed to a user-specific default with ALTER ROLE.

See also the SET, ALTER ROLE, ALTER SYSTEM, and SHOW commands.

RESTART query

Restart a query in a resource pool with specified resources.

```
RESTART query TO WLM RESOURCE POOL name [ WITH ( resource=value [,...] ) ]
```

where `query` refers to the ID of a running query and `resource` is one of the following:

- `priority`
- `memory`
- `spool`
- `spill`

Queries are restartable when they are in the `assemble`, `compile`, `acquire resources`, and `run` states. See How Queries Are Executed.

Examples

For example, restart a query in the `large` pool:

```
premdb=# restart 344321 to wlm resource pool large;
RESTART
```

Restart a query in the `large` pool with specific requests for priority, memory, and spill space:

```
premdb=# restart 347010 to wlm resource pool large
with ( priority high, memory '500MB', memory '40%', spill '10%' );
RESTART
```

To get a list of currently running queries, select from the sys.query view. For example:

```
premdb=# select query_id from sys.query where type='insert';
 query_id
----------
  2465924
(1 row)

premdb=# restart 2465924 to wlm resource pool long;
RESTART
```

REVOKE

Revoke privileges from specific roles on database objects. Also revoke certain privileges at the system level and revoke membership from roles.

```
REVOKE [ GRANT OPTION FOR ]
   { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES }
   [, ...] | ALL [ PRIVILEGES ] }
   ON { [ TABLE ] table_name [, ...]
        | ALL TABLES IN SCHEMA schema_name [, ...] }
   FROM { role_specification | PUBLIC } [, ...]
   [ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
   { { SELECT | INSERT | UPDATE | REFERENCES } ( column_name [, ...] )
   [, ...] | ALL [ PRIVILEGES ] ( column_name [, ...] ) }
   ON [ TABLE | VIEW ] table_name | view_name [, ...]
   FROM { role_specification | PUBLIC } [, ...]
   [ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
   { { CREATE | CONNECT | TEMPORARY | TEMP |
   EXPLAIN QUERY | TRACE QUERY | VIEW QUERY TEXT |
   ALTER ANY SCHEMA | DROP ANY SCHEMA |
   BACKUP | RESTORE | BULK LOAD | CONTROL }
   [, ...] | ALL [ PRIVILEGES ] }
   ON DATABASE database_name [, ...]
   FROM { role_specification | PUBLIC } [, ...]
   [ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
   { EXECUTE | ALL [ PRIVILEGES ] }
   ON { PROCEDURE procedure_name ( [ [ argmode ] [ arg_name ] arg_type [, ...] ] ) [, ...]
        | ALL PROCEDURES IN SCHEMA schema_name [, ...] }
   FROM { [ GROUP ] role_specification | PUBLIC } [, ...]
   [ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
   { { CREATE | USAGE } [, ...] | ALL [ PRIVILEGES ] }
   ON SCHEMA schema_name [, ...]
   FROM { role_specification | PUBLIC } [, ...]
   [ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
   { { USAGE | SELECT | UPDATE } [, ...] | ALL [ PRIVILEGES ] }
   ON { SEQUENCE sequence_name [, ...]
        | ALL SEQUENCES IN SCHEMA schema_name [, ...] }
   FROM { role_specification | PUBLIC } [, ...]
   [ CASCADE | RESTRICT ]

REVOKE [ ADMIN OPTION FOR ]
   role_specification [, ...]
   FROM role_specification [, ...]
   [ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
   { ENCRYPT | DECRYPT | HMAC | ALL [ PRIVILEGES ] }
```

```
    ON KEY key_name
    FROM { role_specification } [, ...]

REVOKE [ GRANT OPTION FOR ]
    { CREATE ROLE | ALTER ANY ROLE | DROP ANY ROLE | VIEW ROLE |
      CREATE DATABASE | ALTER ANY DATABASE | DROP ANY DATABASE | BACKUP ANY DATABASE | RESTORE ANY DATABASE |
      TRACE QUERY | VIEW QUERY TEXT | EXPLAIN QUERY |
      CONTROL ANY SESSION | CONTROL LDAP }
      [, ...] | ALL [ PRIVILEGES ]
    ON SYSTEM
    FROM { role_specification } [, ...]
```

Note: See the GRANT command for details about most of the `REVOKE` options.

TABLE

When you revoke privileges on a table, any corresponding column privileges are automatically revoked. However, if a role has been granted privileges on a table, revoking the same privileges from individual columns has no effect.

GRANT OPTION FOR, ADMIN OPTION FOR

This syntax means that the grant option for the privilege is revoked, not the privilege itself. If you do not specify `GRANT OPTION FOR`, both the privilege and the grant option are revoked. If a user holds a privilege with grant option and has granted it to other users, the privileges held by those other users are *dependent privileges*.

When you revoke membership in a role, `GRANT OPTION FOR` is called `ADMIN OPTION FOR`, but the behavior is similar.

CASCADE

This syntax revokes *dependent privileges*. If the privilege or the grant option held by a user is being revoked and dependent privileges exist, those dependent privileges are also revoked if `CASCADE` is specified; if it is not specified, the revoke action fails. Affected users might keep the privilege if it was also granted through users other than the subject of this `REVOKE` command.

Examples

Revoke `USAGE` on a sequence:

```
premdb=# revoke usage on sequence matchid from bobr;
REVOKE
```

Revoke `EXECUTE` privilege on a stored procedure from the `ybd` user:

```
premdb=# revoke execute on procedure proc1() from ybd;
REVOKE
```

Revoke `ENCRYPT` privilege on key `yb100key` from user `yb100`:

```
premdb=# revoke encrypt on key yb100key from yb100;
REVOKE
```

Revoke `ENCRYPT` and `DECRYPT` privileges on key `yb100key` from user `yb100`:

```
premdb=# revoke all privileges on key yb100key from yb100;
REVOKE
```

The following example uses two `REVOKE` statements to revoke table and database privileges from user `bobr` so the user can be dropped:

```
premdb=# drop user bobr;
ERROR:  role "bobr" cannot be dropped because some objects depend on it
DETAIL:  privileges for table hometeam
privileges for table awayteam
privileges for table match
privileges for table team
privileges for table season
privileges for database premdb
premdb=# revoke all privileges on database premdb from bobr;
REVOKE
premdb=# drop user bobr;
ERROR:  role "bobr" cannot be dropped because some objects depend on it
DETAIL:  privileges for table hometeam
privileges for table awayteam
privileges for table match
privileges for table team
privileges for table season
premdb=# revoke all privileges on all tables in schema public from bobr;
REVOKE
premdb=# drop user bobr;
DROP ROLE
```

REVOKE SSL TRUST

Revoke the SSL trust configuration on the system.

```
REVOKE { 'hash' | ALL } FROM SSL TRUST
```

Non-superusers must have `sysadmin` (or `consumeradmin`) membership in order to run SSL trust commands.

'hash'

Revoke an SSL certificate from the system truststore. Use the SHOW SSL TRUST command to display certificates and determine the hash value.

ALL

Revoke all SSL certificates from the system truststore.

For example:

```
premdb=# revoke all from ssl trust;
REVOKE SSL TRUST
```

```
premdb=# revoke '2de2f78e.0' from ssl trust;
REVOKE SSL TRUST
```

ROLLBACK

Roll back the current transaction, discarding any changes that the transaction would have committed.

```
ROLLBACK [ WORK | TRANSACTION ]
```

For example, the `season` table is not dropped because the ROLLBACK command ends the transaction:

```
premdb=# begin;
BEGIN
premdb=# drop table season;
DROP TABLE
premdb=# rollback;
ROLLBACK
premdb=# \d season
            Table "public.season"
   Column    |         Type          | Modifiers
-------------+-----------------------+-----------
 seasonid    | smallint              |
 season_name | character(9)          |
 numteams    | smallint              |
 winners     | character varying(30) |

Distribution: Replicated
```

If no transaction is in progress, the ROLLBACK command returns a warning and has no other effect:

```
premdb=# rollback;
WARNING:  there is no transaction in progress
ROLLBACK
```

ROLLBACK DATABASE TO SNAPSHOT

Roll back a database to a backup snapshot.

```
ROLLBACK DATABASE TO SNAPSHOT 'rollback_point' HOT_STANDBY
```

DATABASE

This command always rolls back *the current database*. Before running this command, you must connect to the database that you want to roll back. You cannot roll back the `yellowbrick` database.

SNAPSHOT 'rollback_point'

The snapshot you specify should be the result of the sys.oldest_rollback_point_id_in_replica function.

HOT_STANDBY

The database that you roll back is placed in `HOT_STANDBY` mode. If necessary, you can use an `ALTER DATABASE` command to take the database out of `HOT_STANDBY` mode.

Use Cases

There are two main use cases for a rollback operation:

- The most common use case for rolling back a database is as part of a failover procedure when database replication is in use. A rollback of this kind occurs *automatically* as part of an `ALTER DATABASE...ALTER REPLICA...PROMOTE` command. You do not need to run the `ROLLBACK DATABASE` command manually in this case. See Replication Failover and Failback.
- You can roll back a target database that is being used for replication manually if you have taken a target database out of `HOT_STANDBY` mode for some reason, such as to run some tests that require writes. After you have run these tests, the target database and source database will be out of sync, and you will not be able to resume replication. Resuming replication in this case requires you to roll back the database to a known good snapshot (a backup snapshot created by the replication process). Rolling back a database also puts the database back into `HOT_STANDBY` mode, which is required before replication can resume. See also Rolling Back a Replicated Database.

Note: You cannot run multiple `ALTER DATABASE...ALTER REPLICA...PROMOTE` operations or explicit `ROLLBACK DATABASE` operations in parallel. Because of exclusive locks required on shared dependencies (required by the rollback), the `PROMOTE` operations will block each other and fail. For example:

```
alter database repldb1 alter replica repldb1_failover promote;
alter database repldb2 alter replica repldb2_failover promote;
Error:
ERROR: Failed to promote replica
DETAIL: Unable to promote replica “repldb1_failover” Reason: ... ROLLBACK DATABASE failed to lock shared relations
...
ERROR: Failed to promote replica
DETAIL: Unable to promote replica “repldb2_failover” Reason: ... ROLLBACK DATABASE failed to lock shared relations
```

Example

The following command rolls back a target database for replication:

```
premdb_replicated=# rollback database to snapshot 'premdb_replica_20_02_10_17_56_08' hot_standby;
ROLLBACK DATABASE TO SNAPSHOT
```

ROLLBACK TO SAVEPOINT

Roll back the commands in the current transaction that were executed after the named savepoint.

```
ROLLBACK [ WORK | TRANSACTION ] TO [ SAVEPOINT ] savepoint_name
```

The savepoint remains valid and can be rolled back to again later, if needed. The ROLLBACK TO SAVEPOINT command destroys all savepoints that were defined after the named savepoint.

For example, the following transaction is rolled back to savepoint `insert1` when a subsequent `INSERT` command within the transaction returns an error. Therefore, the row inserted into the `season` table is committed when the transaction ends. Without the savepoint that insert would have been rolled back as well.

```
premdb=# begin;
BEGIN
premdb=# insert into season(seasonid) values(100);
INSERT 0 1
premdb=# savepoint insert1;
SAVEPOINT
premdb=# insert into newseason(seasonid) values(100);
ERROR:  relation "newseason" does not exist
LINE 1: insert into newseason(seasonid) values(100);
                    ^
premdb=# rollback to savepoint insert1;
ROLLBACK
premdb=# commit;
COMMIT
premdb=# select * from season where seasonid=100;
 seasonid | season_name | numteams | winners
----------+-------------+----------+---------
     100 | [NULL]      |   [NULL] | [NULL]
(1 row)
```

SAVEPOINT

Create a savepoint within the current transaction.

```
SAVEPOINT name
```

Within a transaction, commands that are executed after a savepoint may be rolled back. The transaction is restored to its state before the savepoint was established.

You can define a maximum of 1023 savepoints per transaction.

For example, the following transaction is rolled back to savepoint `insert1` when a subsequent `INSERT` command within the transaction returns an error. Therefore, the row inserted into the `season` table is committed when the transaction ends. Without the savepoint that insert would have been rolled back as well.

```
premdb=# begin;
BEGIN
premdb=# insert into season(seasonid) values(100);
INSERT 0 1
premdb=# savepoint insert1;
SAVEPOINT
premdb=# insert into newseason(seasonid) values(100);
ERROR:  relation "newseason" does not exist
LINE 1: insert into newseason(seasonid) values(100);
                    ^
premdb=# rollback to savepoint insert1;
ROLLBACK
premdb=# commit;
COMMIT
premdb=# select * from season where seasonid=100;
 seasonid | season_name | numteams | winners
----------+-------------+----------+---------
      100 | [NULL]      |   [NULL] | [NULL]
(1 row)
```

SELECT INTO

Write a query and insert its results into a new table.

For details about the syntax, see SELECT.

Syntax

```
[ WITH name AS (subquery) [, ...] ]
SELECT [ ALL | DISTINCT expression [ [ AS ] output_name ] [, ...] ]
   [ * | expression [ [ AS ] output_name ] [, ...] ]
   INTO [ TEMPORARY | TEMP ] [ TABLE ] new_table
   [ FROM from_item [, ...] ]
   [ WHERE condition ]
   [ GROUP BY { expression [, ...] ]|
     GROUPING SETS (expression [, ...]) |
     ROLLUP (expression [, ...]) |
     CUBE (expression [, ...]) } ]
   [ HAVING condition [, ...] ]
   [ WINDOW window_name AS ( window_definition ) [, ...] ]
   [ { UNION | INTERSECT | { EXCEPT | MINUS } [ ALL | DISTINCT ] select ]
   [ ORDER BY expression [ ASC | DESC ] [ NULLS { FIRST | LAST } ] [, ...] [ COLLATE collation ] ]
   [ LIMIT { count | ALL } ]
   [ OFFSET start [ ROW | ROWS ] ]
```

See SELECT for details about the clauses in the SELECT INTO statement. The table that you select into must be a new temporary or persistent table.

For example:

```
premdb=# select * into season10 from match where seasonid=10;
SELECT 380
premdb=# \d season10
            Table "public.season10"
  Column  |           Type             | Modifiers
----------+----------------------------+-----------
 seasonid | smallint                   |
 matchday | timestamp without time zone |
 htid     | smallint                   |
 atid     | smallint                   |
 ftscore  | character(3)               |
 htscore  | character(3)               |

Distribution: Hash (seasonid)
```

In the following example, note the use of aliases for the `name` column. If these columns do not have distinct names, the SELECT INTO statement returns an error.

```
premdb=# SELECT season_name, matchday, h.name htname, a.name atname, ftscore
into results
FROM season s JOIN match m ON(m.seasonid=s.seasonid)
JOIN hometeam h ON(m.htid=h.htid) JOIN awayteam a ON(m.atid=a.atid)
```

```
WHERE h.name='Chelsea' AND season_name='2011-2012' ORDER BY matchday;
SELECT 19
```

SET

Set the value of a configuration parameter.

```
SET [ SESSION | LOCAL ] parameter { TO | = } { value | 'value' | DEFAULT }
```

SESSION

Set the value for the duration of the current session.

LOCAL

Set the value for the duration of the current transaction.

Note: `SET SCHEMA 'value'` is equivalent to `SET search_path TO value`, but you can only specify one schema with the `SET SCHEMA` syntax.

Some parameters cannot be changed at the session or local level. See Configuration Parameters.

See also the RESET, ALTER SYSTEM, ALTER ROLE, and SHOW commands.

Examples

Set the `search_path`:

```
premdb=# create schema premier;
CREATE SCHEMA
premdb=# create schema league;
CREATE SCHEMA
premdb=# set search_path to premier, league;
SET
premdb=# show search_path;
   search_path
-----------------
 premier, league
(1 row)
```

Set the `application_name` parameter. For example:

```
premdb=# set application_name to ybsql_yb100;
SET
```

Setting the application name is useful for workload management configuration. For example, you can create WLM rules that refer to the application name in a condition.

SET ROLE

Set the current user for the current session.

```
SET [ SESSION | LOCAL ] ROLE role_name
SET [ SESSION | LOCAL ] ROLE NONE
RESET ROLE
```

After this command is run, permissions checking for subsequent SQL commands is applied based on the privileges of the named role. The `role_name` must be a role that the current session user is a member of. If the session user is a superuser, any role can be specified.

`LOCAL` can only be specified within a transaction.

`NONE` and `RESET` reset the current user to the current session user. Any user can run these forms of the command.

For example:

```
premdb=# select session_user, current_user;
 session_user | current_user
--------------+--------------
 brumsby      | brumsby
(1 row)

premdb=# set role bobr;
SET
premdb=> select session_user, current_user;
 session_user | current_user
--------------+--------------
 brumsby      | bobr
(1 row)
```

SET SESSION AUTHORIZATION

Set the session user and the current user for the current SQL session.

```
SET [ LOCAL ] SESSION AUTHORIZATION user_name
SET [ LOCAL ] SESSION AUTHORIZATION DEFAULT
RESET SESSION AUTHORIZATION
```

`LOCAL` can only be used when this command is run within a transaction block and specifies that the command takes effect only for that transaction.

You can use this command, for example, to temporarily become an unprivileged user and later switch back to being a `sysadmin` user. The session user is initially set to the authenticated user name provided by the client. The current user is the user whose permissions are checked when commands are issued. The current user is normally the same as the session user, but may be changed with the SET ROLE command.

You can change the session user only if the initial session user (the authenticated user) had superuser privileges. Otherwise, the command is accepted only if it specifies the authenticated user name.

The `DEFAULT` and `RESET` options reset the session and current users to the originally authenticated user name. Any user can run these commands.

For example:

```
premdb=# select current_user, session_user;
 current_user | session_user
--------------+--------------
 yellowbrick  | yellowbrick
(1 row)

premdb=# set session authorization bobr;
SET
premdb=> select current_user, session_user;
 current_user | session_user
--------------+--------------
 bobr         | bobr
(1 row)

premdb=> drop table match;
ERROR:  must be owner of relation match
```

SHOW

Show the current value of a configuration parameter or show the current values for all parameters.

```
SHOW parameter | ALL
```

For example:

```
premdb=> show search_path;
   search_path
------------------
 public, allpremdb
(1 row)
```

```
premdb=# show yb_server_version;
 yb_server_version
-------------------
 5.4.0
(1 row)
```

`SHOW ALL` includes a description of each parameter.

```
yellowbrick=# show all;
              name                 |            setting             |                                 description
-----------------------------------+--------------------------------+-------------------------------------------------------------
 application_name                  | ybsql                          | Sets the application name to be reported in statistics a
 block_size                        | 32768                          | Shows the size of a disk block.
 bytea_output                      | hex                            | Sets the output format for bytea.
 client_encoding                   | UTF8                           | Sets the client's character set encoding.
 client_min_messages               | warning                        | Sets the message levels that are sent to the client.
 DateStyle                         | ISO, MDY                       | Sets the display format for date and time values.
 enable_alternative_round          | on                             | Translates round to round_var which allows for a non con
 enable_silent_coerce              | off                            | Enables silent and lossy coerces.
 extra_float_digits                | 0                              | Sets the number of digits displayed for floating-point v
 idle_in_transaction_session_timeout | 0                            | Sets the maximum allowed duration of any idling transact
 idle_session_timeout              | 0                              | Sets the maximum allowed session duration after any stat
 lc_collate                        | C                              | Shows the collation order locale.
 lc_ctype                          | C                              | Shows the character classification and case conversion l
 lc_time                           | C                              | Sets the locale for formatting date and time values.
 listen_addresses                  | *                              | Sets the host name or IP address(es) to listen to.
 log_timezone                      | UTC                            | Sets the time zone to use in log messages.
 max_connections                   | 2300                           | Sets the maximum number of concurrent connections.
 max_index_keys                    | 32                             | Shows the maximum number of index keys.
 max_user_connections              | 2000                           | Sets the maximum number of concurrent user connections.
 max_wal_size                      | 16GB                           | Sets the WAL size that triggers a checkpoint.
 search_path                       | "$user", public                | Sets the schema search order for names that are not sche
 server_encoding                   | LATIN9                         | Sets the server (database) character set encoding.
 ssl                               | on                             | Enables SSL connections.
 ssl_ca_dir                        | /mnt/ybdata/certs/trust-database | Location of the SSL certificate authority directory.
 ssl_cert_file                     | /mnt/ybdata/certs/ybd.crt      | Location of the SSL server certificate file.
```

```
 ssl_ciphers                    | ECDHE:!3DES:!DES:!NULL:!RC4     | Sets the list of allowed SSL ciphers.
 ssl_key_file                   | /mnt/ybdata/certs/ybd.key       | Location of the SSL server private key file.
 ssl_prefer_server_ciphers      | on                              | Give priority to server ciphersuite order.
 statement_timeout              | 0                               | Sets the maximum allowed duration in milliseconds of any
 tcp_keepalives_count           | 10                              | Maximum number of TCP keepalive retransmits.
 tcp_keepalives_idle            | 10                              | Time between issuing TCP keepalives.
 tcp_keepalives_interval        | 1                               | Time between TCP keepalive retransmits.
 TimeZone                       | UTC                             | Sets the time zone for displaying and interpreting time
 transaction_isolation          | read committed                  | Sets the current transaction's isolation level.
 wal_block_size                 | 8192                            | Shows the block size in the write ahead log.
 wal_level                      | minimal                         | Set the level of information written to the WAL.
 yb_last_execid                 |                                 | Shows the execution-id of the last backend query run in
 yb_server_version              | 5.4.0                           | Shows the Yellowbrick Database server version.
 yb_server_version_num          | 50400                           | Shows the Yellowbrick Database server version as an inte
 ybd_query_tags                 |                                 | Sets the optional query tags to be stored in query histo
(40 rows)
```

For more details about available configuration parameters, see Configuration Parameters.

SHOW SSL CA

Export the SSL CA configuration for replication purposes.

```
SHOW SSL CA
```

See Configuring SSL Trust for details about when to use this command. For example:

```
premdb=# show ssl ca;
                        CA CERTIFICATE
-----------------------------------------------------------------
 -----BEGIN CERTIFICATE-----
 MIIF3jCCA8agAwIBAgIJAJNf54QU/KfsMA0GCSqGSIb3DQEBCwUAMH8xCzAJBgNV
 BAYTAlVTMQswCQYDVQQIDAJDQTEfMB0GA1UECgwWWVsbG93YnJpY2sgRGF0YSwg
 SW5jLjEtMCsGA1UECwwkYTk5M2FkNGUtNDQ2ZC00ZTY2LTkwYWMtY2ExYmQzNjll
 NGUyMRMwEQYDVQQDDApzeXNfeWJkX2NhMB4XDTE5MTIxMzE4MjUzN1oXDTI5MTIx
 MjE4MjUzN1owfzELMAkGA1UEBhMCVVMxCzAJBgNVBAgMAkNBMR8wHQYDVQQKDBZZ
 ZWxsb3dicmljayBEYXRhLCBJbmMuMS0wKwYDVQQLDCRhOTkzYWQ0ZS00NDZkLTRl
 NjYtOTBhYy1jYTFiZDM2OWU0ZTIxEzARBgNVBAMMCnN5c195YmRfY2EwggIiMA0G
 CSqGSIb3DQEBAQUAA4ICDwAwggIKAoICAQC+rCGJSoJ+nhV4EJqXyMzW8EWIPR+N
 1Y0Heru27NH1WSPpQWu7qjKVXkegQBeA1R4pZwNVFFEvrZ6fSmNxN/F80POJMA9F
 eVUBHJAri7cJqENyaJ5qt8rYsdL4OFBEQ4uy8pxW8wcNuFjiX9E2l4oroe7ZhTBC
 Tm6n344xjiUvBjvNAiE7TGJIX5jsiWybQykL2Ah+1+PkgOyHrupANe99T11z5ian
 jrn9BImuwwD5LYTwMbvCUqTcOcGgotsEGVcFQvKJ5dh1ASCkXfw7RkaX1+mZQO3b
 gj+reHvZAdlY9jbCLVpjCxN+mbEvYNE0FHHos9pobQktD6FXbRlTkSlgEhEzK8qb
 OFgUN/N6V5UeqSZTWkSvhRh8Bw+MlYAVzkYvwnKqCLShMZRNXjQrfiRWf25XV1t+
 Cq4lfusvJJ89B1jlTwbn5u6mBxkoxYPe0s3iXtCkK6HRMm3uH9kwpBaHTExc2D8V
 BLUrnRBWYJaYh7Zec2je24l8KjJNH/d7ZOwEieiU1TT/N+5Hp/6pi8Gl171chfSj
 eIS0+eTfu5HFH7VR4pJvkzCFilnfpZhejtFOqMQxkOauXas+M9Q3wbc16BDje31D
 dodK3K9JThzn1AOekrvFy1Oc3uVofKkclfb9kyO+lulK3uIZpC3SHjW+xjYHPQbn
 EAiSNXVopdPrWwIDAQABo10wWzAMBgNVHRMEBTADAQH/MAsGA1UdDwQEAwIB5jAd
 BgNVHQ4EFgQUckz3dAaFdjZ9MBK/Z8FWGoBgo0QwHwYDVR0jBBgwFoAUckz3dAaF
 djZ9MBK/Z8FWGoBgo0QwDQYJKoZIhvcNAQELBQADggIBAJ+1b22CfNldfZZQCKl5
 6Yt7rr5YZ7klwjITG6OX1TOJCjfnOe8zeMnOcOjtzWiiRGFluiDyLJPBRxpOwr42
 PtBOREpGIwp7djbKxIcco03wgA2bm/t63vMQi1YJ3uBLKuqSgm1+yV9eMYABy4fG
 mTcWkKx8D+RWhIboIHnADq3A3u6Ws1s2qq/vT/Mln8AplB3si93JRUqJXVLLi+RO
 iD1tI8p4+Dthq6Dym9KFhBWdZwSVNBJdV9PGq1AZwMtk3w2aByzvc67B0LqveHKY
 uIZpO0PPVd6er6uzxO2U5hx4IJe3fLFk8R4j0Ckbds27Jv5ZPSDGgA05QLKdEXzJ
 /kAr01612BLQYdAv0Jvom99AFV7MdNppn2AexRenGYF5aQ3Ec4nxsREO29J4ttEp
 P5Kh4eN707QIBn27ZtngUEcwHCWr527aRCdTLpqn06OVZ2Plynf0yLmrhO9glyVV
 7P97Jpu08Zx5Osn9z2ZQ/jUEdMGklt6UDeACQfKo7XLiO1w53oQjrnTfWtRZIYdg
 wcp+KQqinONXYWtVBpDnHJ4R7+tMne9kJqHTMmiyrCceFNBoE6AiKtXaKZPyynFi
 kqU7fQQDu8pHwRFXfnxJQ5ATzjnXDaNnAByU+IPPaHjJPmDYEpue6efrqEolSsdV
 3rhJxLSKZeIKIbS5m3zRbcoH
 -----END CERTIFICATE-----
(34 rows)
```

SHOW SSL IDENTITY

Platforms: All platforms

Parent topic: SQL Commands

Display the login identity certificate, which is only used at runtime for replication login.

```
SHOW SSL IDENTITY
```

For example:

```
premdb=# show ssl identity;
                    IDENTITY CERTIFICATE
-------------------------------------------------------------------
 -----BEGIN CERTIFICATE-----
 MIIFuzCCA6OgAwIBAgIJAPbhAtmlePVuMA0GCSqGSIb3DQEBCwUAMH8xCzAJBgNV
 BAYTAlVTMQswCQYDVQQIDAJDQTEfMB0GA1UECgwWWVsbG93YnJpY2sgRGF0YSwg
 SW5jLjEtMCsGA1UECwwkODYyNzcwMDMtMDNlNi00OTdjLWIxNWEtOTkyMGQ5Zjgx
 MDZjMRMwEQYDVQQDDApzeXNfeWJkX2NhMB4XDTIwMDExNzIwMDIyM1oXDTMwMDEx
 NjIwMDIyM1owgYMxCzAJBgNVBAYTAlVTMQswCQYDVQQIDAJDQTEfMB0GA1UECgwW
 WWVsbG93YnJpY2sgRGF0YSwgSW5jLjEtMCsGA1UECwwkODYyNzcwMDMtMDNlNi00
 OTdjLWIxNWEtOTkyMGQ5ZjgxMDZjMRcwFQYDVQQDDA5zeXNfeWJkX3N5c3RlbTCC
 AiIwDQYJKoZIhvcNAQEBBQADggIPADCCAgoCggIBAL00gVwx9W0K7JupLNHjKYl2
 CRJdnHbtFv6e03wZLNhlmNGUXezs6Tura8hoeHhxAZKlJ/jlN+iRaEkTWdpG6xGK
 firsoZicZjtFL33NF5i6Olo64FPJFkESrLz/PCVpJpZm+oKIzMGW4QWHOM8Kh3ny
 KPZORj/2UzGirgzcUNslfz0E3ioD5QNhxFTGyBeCqRxPdwzyBbpk5Er4c2vJtIya
 8X8ul43pKTblwJYpelVRyiZ0R2e+UI1QsuD4vjsORJovYCI7c91vUfP3v47eJtR/
 cX6WjijvS1nWpRFaxpmJUJXDCqp2SyYfv219Ev4gcjaE9BPKZB1N/9hizrGO3OB8
 GSbZEJODdhERyZK9sYwKOMRalaCsURYRfhN1uoGIykNwocBUen2vrq7IOc40Zs0Q
 2pPn/SNrxY32LseTQPkZJUSrTgJOJ/PrLOjIcMDgHcsFDihxqHjoa5/2EMuqvbgN
 upCkawXS2xVyNDaHZnEp1yHFw0L2bviGIqFXUsRfObKWWm07VQVTTfMyQLaAUpYh
 kt/GaqtUmLhisiwufgnEmxX/L0cyU3iHbfCusKsdwUJq2nvuNaJhVTkGrksl1dTt
 YMaun28ZCtGdCmkRTC3jFUSie2MHoB0woVTPYa7O9MVIRXqCeDGyYsmXaXQ1Vz60
 K8zN4GH0Cn3YCTtP2uPTAgMBAAGjNTAzMAkGA1UdEwQCMAAwCwYDVR0PBAQDAgXg
 MBkGA1UdEQQSMBCCDnN5c195YmRfc3lzdGVtMA0GCSqGSIb3DQEBCwUAA4ICAQB0
 hLyG381TBnGxbAi2DKaLkZh/ANHY7VnMUM5NwbniVy13sL7NPKg84d9zK4pJkN9O
 VHyT1DnIaFQbn6fR98x3LYQztrSveGZq9jej/7X40iHsUnpGZNnzCdJdV5y62QdY
 P73+93NrC44hIEocJXLPXXlfSHh6uqMbIuMkGFgwDg70O+pGxjQcwBI/Q3z/7wgM
 CUItl4ofX3icPX3u2CKU2KJKGXKr2LC0VX4keTgvSNTsJzlVv7l1EPQgKUl1n6Du
 1pCs0cYsTS0Lks0DJTQeeUm80+EtT586+Dc4V3aYcbihI/UKbMmhOvMLspb3H7S5
 Nlc88IUG/t/duYGcLOcCzI4Guzt/ah2B5MSAFku8bbMJnkQWSg0Hj7jTyLiPVl9y
 aqJJECW6w/imAFf2+SXB2l+RxkvzJv8hhLZMdBeJwbZ0pVyUjIjMxQuHr929TD7W
 ftw1ecwehzJm9TLYX1uFImHj7oi5vuTLQvF0NK9h59Nk3Oyi++sSYXhGKAl1fbwW
 LiTrqvjvlWUnNmfNQClBjqDrLhRv9TtCQzpFbfzw7lNvvFwiesKrxqphdrKDi2MJ
 Coss6HlYUksyA8nDGcgkGv9roQyEFiBopShJk7CrRCX3WIabZ3eBv9hp7eAebu7x
 SUAZZa680O5Po9nbR/8tzeR+IEMMZYQlT6yKZSDYHw==
 -----END CERTIFICATE-----
(33 rows)
```

SHOW SSL SYSTEM

Export the system certificate for SSL trust on a remote system that is used for replication.

```
SHOW SSL SYSTEM
```

See Configuring SSL Trust for details about when to use this command.

For example:

```
premdb=# show ssl system;
                     SYSTEM CERTIFICATE
----------------------------------------------------------------------
 -----BEGIN CERTIFICATE-----
 MIIFyDCCA7CgAwIBAgIJAKbfKdX9Np6DMA0GCSqGSIb3DQEBCwUAMIGCMQswCQYD
 VQQGEwJVUzELMAkGA1UECAwCQ0ExHzAdBgNVBAoMFlllbGxvd2JyaWNrIERhdGEs
 IEluYy4xLTArBgNVBAsMJDQ1M0M4M0JBLTBCMEMtNDAyMS1BMjcxLTE5NzdDRkJC
 MTlGQjEWMBQGA1UEAwwNeWJpbml0X2Rldl9jYTAeFw0xOTEyMTMxNzEzMzFaFw0y
 OTEyMTIxNzEzMzFaMHwxCzAJBgNVBAYTAlVTMQswCQYDVQQIDAJDQTEfMB0GA1UE
 CgwWWWVsbG93YnJpY2sgRGF0YSwgSW5jLjEtMCsGA1UECwwkOTU0OGZlZWYtMWNj
 MC00YTg1LTliN2EtYjVmMjk4YjI1YjVlMRAwDgYDVQQDDAdicnVtc2J5MIICIjAN
 BgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEAm+QpCcsdkfHjoLkfmuLL6ltKAmvD
 P3d6MyA/PANarYQTP8stSxE/aa1OZ8ZxXaY4aLc468qYxIjtOiE51odI6pG3jOab
 zOYZ+Fvx5643wJ5nTXcqf+nHp4cTrpJX37DVsI5rbGjJzUn1y246zSfBoJqMmDu5
 aVjrO8Gz9D2IrMHxOrlk4U2xu/qSZTThvn1zVmo+qMX2NY181Iql3PvT7Iuw1/NR
 MQ4FmDOpFyB0w67/msT2G6DFvZlCPluoPH+56oTVhaB8UFj/kvSdBQX6ylOasNzB
 wEu/XpRnL/WIBLWyN/Q0umuVKGuHQa+6iY8oJOtOV+5cXpOj1+Y+lV9f0JTrPwJG
 g5Xxvk2zyllOfuiMJHPEyLB4BrU2BroQPabGYPlMD5Z0Y30DZWhvyj+l4gWX2L+d
 tzV+V832iSw3/1rGMLRWjiwhUkcwSBV99HcwMEOzr0cejQGGLXgRcL8rTDCH1j4o
 hcAJ8ZxKaUvh1Z+nBXnw6MR1B/8rXqyenLVKukKxea0couKxJnp7oi5RuekmdowO
 PbJ6t3iCgGqZdIsgsGJyA0UDBAO2IopHdXQhi11gsBJMU1BcIvkRgxGoPE2E3QKt
 AhSAZ6Qf6vEHovMSoyrGXzPvfDNHBCLQ6ZZcU2F0NdmVhZxXZZKlNuQRUoahNAVl
 rWp+Av07BtiA98UCAwEAAaNGMEQwCQYDVR0TBAIwADALBgNVHQ8EBAMCBeAwKgYD
 VR0RBCMwIYIHYnJ1bXNieYIWYnJ1bXNieS55ZWxsb3dicmljay5pbzANBgkqhkiG
 9w0BAQsFAAOCAgEAf6+7CTBQJZLdjc9pk9RQP1DL5kvxuQayTgkpF3uYnIyAiD15
 GLtwuRXnbuzN0UCMNl1747aNmCV6LLoq6MZWZpSAyug6lYyNZP7uA9bdv0whIYVA
 PJMJDtz2AgqxK3mEnRfTvLmQ+6mQRwP+kYPpe4DR191KhmUpsO25vugciV6tufNr
 VHZox04f/B5aasFU5A9JJ2x11D/7ZjG71meDB76b21FyhQURQKx5N+/ReMAcqvaF
 pf0e1qrWEd5SA/onX6EGbXmxBBDb1A+5XSpjor6pr6Uap8MfCLtqrwzBz74/sC0t
 vLDADPC5DXXZupEOOXvPGV8oshArdYhNzX5Y1B2RR78Wqz2Xrstb41eD/TBUUO7N
 LpTVF/aGATjubgK0vCjVVdwdzC+J3UfQmoLElNOrL6+6bzsu8EzWwu18wwKrUQmb
 Llzd7CSrXPR9F3qoQ7PtTH2plFskjdY6Tqgog8OStkMHG+plnw1AAaEVRPOv4ZWX
 yJeRWi4g5FlgV+Fkc66biBicZW9+vxw5uoRw/zW49vtjYOZllupVKHV0dg+3r+B0
 jgaJSCDXTsmRJmEoefeC/zEgO4AGrwk3BmCE2i6Zwt+lqCfKZOJCQmt7yQwHp0s0
 VCN9CLDM20JZ2kvBk9K8BDUdMeN7jOFQrdILcIq2ZKFdFdu0yzlW1FTOvtQ=
 -----END CERTIFICATE-----
(33 rows)
```

SHOW SSL TRUST

Validate the results of the IMPORT SSL TRUST command.

```
SHOW SSL TRUST
```

Non-superusers must have `sysadmin` (or `consumeradmin`) membership in order to run SSL trust commands.

For example:

```
premdb=# show ssl trust;
-[ RECORD 1 ]-----------------------------------------------------------------------------------------------------
hash        | 2c5c660b.0
details     | notBefore=Dec 13 18:25:37 2019 GMT                                                                +
            | notAfter=Dec 12 18:25:37 2029 GMT                                                                 +
            | issuer= /C=US/ST=CA/O=Yellowbrick Data, Inc./OU=a993ad4e-446d-4e66-90ac-ca1bd369e4e2/CN=sys_ybd_ca +
            | subject= /C=US/ST=CA/O=Yellowbrick Data, Inc./OU=a993ad4e-446d-4e66-90ac-ca1bd369e4e2/CN=sys_ybd_ca+
            | serial=935FE78414FCA7EC                                                                           +
            |
certificate | -----BEGIN CERTIFICATE-----                                                                        +
            | MIIF3jCCA8agAwIBAgIJAJNf54QU/KfsMA0GCSqGSIb3DQEBCwUAMH8xCzAJBgNV                                  +
            | BAYTAlVTMQswCQYDVQQIDAJDQTEfMB0GA1UECgwWWWVsbG93YnJpY2sgRGF0YSwg                                  +
            | SW5jLjEtMCsGA1UECwwkYTk5M2FkNGUtNDQ2ZC00ZTY2LTkwYWMtY2ExYmQzNjll                                  +
            | NGUyMRMwEQYDVQQDDApzeXNfeWJkX2NhMB4XDTE5MTIxMzE4MjUzN1oXDTI5MTIx                                  +
            | MjE4MjUzN1owfzELMAkGA1UEBhMCVVMxCzAJBgNVBAgMAkNBMR8wHQYDVQQKDBZZ                                  +
            | ZWxsb3dicmljayBEYXRhLCBJbmMuMS0wKwYDVQQLDCRhOTkzYWQ0ZS00NDZkLTRl                                  +
            | NjYtOTBhYy1jYTFiZDM2OWU0ZTIxEzARBgNVBAMMCnN5c195YmRfY2EwggIiMA0G                                  +
            | CSqGSIb3DQEBAQUAA4ICDwAwggIKAoICAQC+rCGJSoJ+nhV4EJqXyMzW8EWIPR+N                                  +
            | 1Y0Heru27NH1WSPpQWu7qjKVXkegQBeA1R4pZwNVFFEvrZ6fSmNxN/F80POJMA9F                                  +
            | eVUBHJAri7cJqENyaJ5qt8rYsdL4OFBEQ4uy8pxW8wcNuFjiX9E2l4oroe7ZhTBC                                  +
            | Tm6n344xjiUvBjvNAiE7TGJIX5jsiWybQykL2Ah+1+PkgOyHrupANe99T11z5ian                                  +
            | jrn9BImuwwD5LYTwMbvCUqTcOcGgotsEGVcFQvKJ5dh1ASCkXfw7RkaX1+mZQO3b                                  +
            | gj+reHvZAdlY9jbCLVpjCxN+mbEvYNE0FHHos9pobQktD6FXbRlTkSlgEhEzK8qb                                  +
            | OFgUN/N6V5UeqSZTWkSvhRh8Bw+MlYAVzkYvwnKqCLShMZRNXjQrfiRWf25XV1t+                                  +
            | Cq4lfusvJJ89B1jlTwbn5u6mBxkoxYPe0s3iXtCkK6HRMm3uH9kwpBaHTExc2D8V                                  +
            | BLUrnRBWYJaYh7Zec2je24l8KjJNH/d7ZOwEieiU1TT/N+5Hp/6pi8Gl171chfSj                                  +
            | eIS0+eTfu5HFH7VR4pJvkzCFilnfpZhejtFOqMQxkOauXas+M9Q3wbc16BDje31D                                  +
            | dodK3K9JThzn1AOekrvFy1Oc3uVofKkclfb9kyO+lulK3uIZpC3SHjW+xjYHPQbn                                  +
            | EAiSNXVopdPrWwIDAQABo10wWzAMBgNVHRMEBTADAQH/MAsGA1UdDwQEAwIB5jAd                                  +
            | BgNVHQ4EFgQUckz3dAaFdjZ9MBK/Z8FWGoBgo0QwHwYDVR0jBBgwFoAUckz3dAaF                                  +
            | djZ9MBK/Z8FWGoBgo0QwDQYJKoZIhvcNAQELBQADggIBAJ+1b22CfNldfZZQCKl5                                  +
            | 6Yt7rr5YZ7klwjITG6OX1TOJCjfnOe8zeMnOcOjtzWiiRGFluiDyLJPBRxpOwr42                                  +
            | PtBOREpGIwp7djbKxIcco03wgA2bm/t63vMQi1YJ3uBLKuqSgm1+yV9eMYABy4fG                                  +
            | mTcWkKx8D+RWhIboIHnADq3A3u6Ws1s2qq/vT/Mln8AplB3si93JRUqJXVLLi+RO                                  +
            | iD1tI8p4+Dthq6Dym9KFhBWdZwSVNBJdV9PGq1AZwMtk3w2aByzvc67B0LqveHKY                                  +
            | uIZpO0PPVd6er6uzxO2U5hx4IJe3fLFk8R4j0Ckbds27Jv5ZPSDGgA05QLKdEXzJ                                  +
            | /kAr01612BLQYdAv0Jvom99AFV7MdNppn2AexRenGYF5aQ3Ec4nxsREO29J4ttEp                                  +
            | P5Kh4eN707QIBn27ZtngUEcwHCWr527aRCdTLpqn06OVZ2Plynf0yLmrhO9glyVV                                  +
            | 7P97Jpu08Zx5Osn9z2ZQ/jUEdMGklt6UDeACQfKo7XLiO1w53oQjrnTfWtRZIYdg                                  +
            | wcp+KQqinONXYWtVBpDnHJ4R7+tMne9kJqHTMmiyrCceFNBoE6AiKtXaKZPyynFi                                  +
            | kqU7fQQDu8pHwRFXfnxJQ5ATzjnXDaNnAByU+IPPaHjJPmDYEpue6efrqEolSsdV                                  +
            | 3rhJxLSKZeIKIbS5m3zRbcoH                                                                          +
            | -----END CERTIFICATE-----                                                                         +
            |
```

SQL Identifiers

Identifiers are names of tables, columns, schemas, databases, and other database objects. Identifiers are frequently used in the commands described in this section. Yellowbrick supports standard SQL identifiers and quoted identifiers.

SQL Identifiers

Note the following restrictions on *unquoted* SQL identifiers:

- SQL identifiers that are not quoted must begin with a letter (a-z) or an underscore (_). The `pg_` prefix is also disallowed and reserved for system objects.
- Subsequent characters in an unquoted identifier can be letters, digits (0-9), or underscores. Unquoted SQL identifiers are case-insensitive.
- Special characters such as `#`, `$`, and so on, are not allowed anywhere in an unquoted identifier.
- Unquoted SQL identifiers are case-insensitive.

Quoted SQL Identifiers

Quoted identifiers (or delimited identifiers) are names enclosed in double quotes ("). Quoted identifiers are case-sensitive. By using quoted identifiers, you can create object names that contain explicit uppercase and lowercase letters, as well as special characters. However, you cannot use double quotes within object names.

Note: Space characters are not allowed in database names.

For example:

```
premdb=# create table "COG$"(cost dec(5,3));
CREATE TABLE
premdb=# \d "COG$"
       Table "public.COG$"
 Column |     Type     | Modifiers
--------+--------------+-----------
 cost   | numeric(5,3) |

premdb=# create table "Pound#"(c1 int);
CREATE TABLE
premdb=# \d "Pound#"
   Table "public.Pound#"
 Column |  Type   | Modifiers
--------+---------+-----------
 c1     | integer |

Distribution: Hash (c1)
```

You can use reserved words as column aliases if you specify either the `AS` keyword or double quotes. For example, the following `SELECT` statements both work:

```
premdb=> select '123' "advanced" from sys.const;
 advanced
----------
 123
```

```
(1 row)

premdb=> select '123' AS advanced from sys.const;
 advanced
----------
 123
(1 row)
```

The following statement returns an error:

```
premdb=> select '123' advanced from sys.const;
ERROR:  syntax error at or near "advanced"
LINE 1: select '123' advanced from sys.const;
```

Maximum Length of Identifiers

In practice, the maximum length of an identifier is 128 bytes. Longer names may be used in queries, and in commands such as CREATE TABLE and CREATE VIEW, but these names are automatically truncated to 128 bytes.

When you write queries and command statements that use the full name of a long identifier, `ybsql` returns the truncated version of the name. You can see information messages when this truncation occurs by setting `client_min_messages` to `NOTICE`:

```
premdb=# set client_min_messages to NOTICE;
SET
premdb=# select * from t1... ;
NOTICE:  identifier "t1..." will be truncated to "t1..."
...
```

START TRANSACTION

Platforms: All platforms

Parent topic: SQL Commands

Start a transaction block.

```
START TRANSACTION
```

This command is identical to BEGIN. See the description of `BEGIN` for details.

SUSPEND INSTANCE

Platforms: EE: All cloud platforms

Parent topic: SQL Commands

Suspend the instance that you are currently connected to, and suspend its associated clusters.

```
SUSPEND INSTANCE
```

This command does not take any parameters.

TABLE

Return the same results as a `select *` query against a table.

```
TABLE table_name [ * ]
```

You can use WITH, UNION, INTERSECT, EXCEPT, ORDER BY, LIMIT, and OFFSET in conjunction with the TABLE command, just as you would with a SELECT query.

For example:

```
premdb=# TABLE hometeam;
 htid |          name
------+------------------------
    2 | Arsenal
    3 | Aston Villa
    4 | Barnsley
    5 | Birmingham City
    6 | Blackburn Rovers
...
```

```
premdb=# TABLE hometeam UNION TABLE awayteam ORDER BY 1 DESC;
 htid |          name
------+------------------------
  100 |
   99 |
   98 |
   97 | Wolverhampton Wanderers
   96 | Wimbledon
...
```

TRUNCATE

Delete all of the rows from one or more tables.

```
TRUNCATE [ TABLE ] name [, ... ]
   [ CASCADE | RESTRICT ]
```

This is a faster operation than an unqualified DELETE operation. When data is deleted or truncated it continues to exist in the system for a period of time until it can be efficiently removed and data can be reorganized. These optimizations are fully automatic and require no administration or manual interaction.

Use the CASCADE option to automatically truncate tables that have foreign-key references to the named tables.

Note: You cannot truncate tables in hot-standby databases.

Rollback is supported for TRUNCATE operations.

For example, truncate three tables:

```
premdb=# truncate table hometeam, awayteam, team;
TRUNCATE TABLE
```

UNLOAD TABLE

> Prerequisite: This feature is currently in beta. To enable it, set `enable_sql_unload` to `ON`.

Bulk unload table data to external storage using a SQL command. This feature is a SQL-native alternative to the `ybunload` client.

```sql
UNLOAD TABLE [ table_name | SELECT query ]
INTO 'file_path'
EXTERNAL LOCATION location_name
[ EXTERNAL FORMAT { format_name |
  (TYPE file_type WITH (option value [, option value ] [, ...])) } ]
```

table_name or SQL view

Name of the table or view to unload. Data will be read from the table and written to external storage. To execute `UNLOAD TABLE`, you must have `SELECT` privilege on the specified table and / or view.

query

A valid `SELECT` statement for unload. You cannot unload the results of other SQL statements, such as `INSERT` or `CREATE TABLE AS` (`CTAS`). `SELECT` privilege is required on all referenced tables and views in the query.

file_path

Path within the specified external location where output files will be written.

EXTERNAL LOCATION

Name of an external location object. See CREATE EXTERNAL LOCATION.

Note: Only S3-compatible external storage locations are currently supported.

EXTERNAL FORMAT

Name of an external format object. See CREATE EXTERNAL FORMAT. Alternatively, you can define the format inline using the `TYPE` clause. You may also omit the `EXTERNAL FORMAT` clause to use the default format configured for the external location.

Note: The External Format specified must have `unload` mode.

TYPE...WITH

Specify the target file type: `CSV` or `TEXT`. You must enclose the `TYPE` clause inside parentheses, and the following `WITH` clause inside its own set of parentheses:

```sql
(TYPE format_type WITH(...))
```

The `WITH` clause that follows `TYPE` defines *formatting unload options*, such as the line separator, field delimiter, and field definitions for different data types. Each name-value pair must consist of a valid option (in this context) and a valid setting for that option. For example:

```sql
(TYPE csv WITH (delimiter ',', linesep '\n'))
```

Options

The `TYPE...WITH` clause in the `UNLOAD TABLE` statement supports options listed in External Format Unload Options

---

Order of Precedence for Format Specifications

There are three different ways to specify the format for an `UNLOAD TABLE` operation. The following order of precedence is enforced:

1. In-line format options, as specified with `EXTERNAL FORMAT (TYPE...WITH(…))`
2. An in-line format name, as specified with `EXTERNAL FORMAT format_name`
3. A default format, if defined, for the external location object specified in the `UNLOAD TABLE` command

---

Examples

An example of the `premdbs3data` external location can be found here.

Unload a table to CSV files:

```sql
unload table match
into '/premdb/match/'
external location premdbs3data
external format (type csv);
```

Unload a table to GZ-compressed files:

```sql
unload table newmatchstats
into '/premdb/newmatchstats/'
external location premdbs3data
external format (type csv with (compress 'gzip'));
```

Unload a table and reference an existing external format object:

```sql
unload table newmatchstats
into '/premdb/newmatchstats25mil/'
external location premdbs3data
external format premdbs3format;
```

Unload a table and reference an existing external location that has external format implicitly defined:

```sql
unload table newmatchstats
into '/premdb/newmatchstats25mil/'
external location premdbs3data;
```

Unload data using a select query from an existing table.

```sql
create table matchday as
select seasonid, matchday, htid, atid from newmatchstats;
--SELECT  24785280

unload table (select seasonid, matchday from matchday)
into '/premdb/newmatch'
external location premdbs3data;
```

UPDATE

Update rows or columns in a table by setting values based on a query or a condition.

```
[ WITH name AS (subquery) [, ...] ]
UPDATE table_name [ * ] [ [ AS ] alias ]
   SET { column_name = { expression | DEFAULT } |
         ( column_name [, ...] ) = ( { expression | DEFAULT } [, ...] ) |
         ( column_name ) = ( subquery )
       } [, ...]
   [ FROM list ]
   [ WHERE condition ]
```

WITH

See WITH Clause for syntax details. You can reference a WITH subquery in the FROM clause of the `UPDATE` statement.

table_name

The target table for the `UPDATE` statement. You can specify the table as: `database.schema.table`, `schema.table`, or `table`.

Note: You cannot use `UPDATE` commands on tables in remote databases; see Cross-Database Queries.

Warning: Running `EXPLAIN ANALYZE` on an `INSERT`, `UPDATE`, or `DELETE` statement modifies the target table, without warning. To avoid this problem, you can run the `EXPLAIN ANALYZE` statement inside a transaction, then roll it back if needed.

AS alias

An optional alias for the table.

If you specify an alias, the alias corresponds to the exact specification of the table. For example, this syntax is valid, where `sn` corresponds to `public.season`:

```
premdb=# update public.season sn set sn.seasonid=100;
UPDATE 25
```

However, the following syntax returns an error because `public.sn` in the `SET` clause does not correspond to `public.season`:

```
premdb=# update public.season sn set public.sn.seasonid=100;
ERROR:  Table "public"."sn" is not the UPDATE relation("public"."season") or doesn't exist
LINE 1: update public.season sn set public.sn.seasonid=100;
```

SET column_name

Set one or more columns in the target table equal to specific values, or to `DEFAULT` values (as defined in the `CREATE TABLE` statement). The column must exist in the target table and must be unambiguous.

A column name can be specified in any of the following ways:

- `column`

  For example: `seasonid`

- `table.column`

  For example: `season.seasonid`

- `schema.table.column`

  For example: `public.season.seasonid`

- `database.schema.table.column`

  For example: `premdb.public.season.seasonid`

- `alias.column`

  For example: `sn.seasonid`, where `sn` is defined as an alias for the target table

subquery

A single column may be set to equal the results of a subquery. For example:

```
premdb=# update season set seasonid=(select max(seasonid) from match);
UPDATE 25
```

The following syntax is not supported:

```
update t1 set (col1, col2) = (select col3, col4 from t2 ...);
```

However, you can use the following syntax to set multiple column values at once:

```
update t1 set (col1, col2) = (col3, col4) from t2 ...;
```

where `col3` and `col4` are columns in table `t2`. The `SET` values can be expressions or constants. For example:

```
premdb=# create table t1(col1 int, col2 int);
CREATE TABLE
premdb=# create table t2(col3 int, col4 int);
CREATE TABLE
premdb=# insert into t1 values(1,10);
INSERT 0 1
premdb=# insert into t2 values(2,20);
INSERT 0 1
premdb=# update t1 set (col1, col2)=(col3, col4) from t2;
UPDATE 1
premdb=# select * from t1;
 col1 | col2
------+------
    2 |   20
(1 row)
```

FROM

Name one or more tables or table references, other than the target table. These other tables may then be joined and/or referenced in `WHERE` clause conditions for the `UPDATE` operation. This syntax is similar to the `FROM` clause in a `SELECT` statement. You can use a comma-delimited list of tables and join them in the `WHERE` clause, or you can use the `JOIN...ON` syntax in the `FROM` clause.

When an `UPDATE` is executed, the target table is joined to the other tables in the `FROM` clause. You can explicitly join the target table to `FROM` clause tables by providing a join condition in the `WHERE` clause. For example:

```
UPDATE t1...FROM t2 WHERE t1.x=t2.x
```

You cannot write:

```
UPDATE t1...JOIN t2 on t1.x=t2.x
```

Note: *Do not list the target table for the UPDATE in the FROM clause unless you intend to specify a self-join.* For self-joins, the target table must have an alias that specifies the self-joining instance of the table.

For example, both of these statements are valid self-joins:

```
UPDATE t1 SET c1=10 FROM t1 AS t1a WHERE t1a.c2=t1.c2;

UPDATE t1 AS t1a SET c1=10 FROM t1 WHERE t1a.c2=t1.c2;
```

In both cases, `t1a` is an alias for `t1`, declared either after the initial `UPDATE` clause or in the `FROM` clause. The `AS` keyword is optional.

For more details, see Usage Notes on Aliases and Joins.

WHERE

Use this `WHERE` clause just as you would use the `WHERE` clause in a `SELECT` statement. Define a condition on either the target table or a table that is listed in the `FROM` clause. For example, you can join the target table to another listed table to qualify rows for the `UPDATE` operation.

Note: Table aliases are allowed in the `WHERE` clause. If the target table of the `UPDATE` is aliased, you must use the alias in the `WHERE` clause. For example:

```
premdb=# update match as m1 set ftscore='0-0' where match.htid=0;
ERROR:  invalid reference to FROM-clause entry for table "match"
LINE 1: update match as m1 set ftscore='0-0' where match.htid=0;
                                                    ^
HINT:  Perhaps you meant to reference the table alias "m1".
```

Usage Notes on Aliases and Joins

- You can use a table alias for the target table name.
- You can reference a target table alias in the `SET` clause, as a prefix for the column name.
- If the target table is aliased, you must use the alias (and not the base table name) in `WHERE` clause references to the target table.
- You can also alias any tables that are listed in the `FROM` clause or used in a subquery (including other instances of the target table used for self-joins).
- The `FROM` clause is only required when the target table is joined to other tables or to itself (a self-join). *Do not list the target table for the UPDATE in the FROM clause unless you intend to specify a self-join.* For self-joins, the target table must have an alias that specifies the self-joining instance of the table.
- You can specify join conditions in the `WHERE` clause or the `FROM` clause. Use the `WHERE` clause to join the target table to other tables. For example, this statement explicitly joins target table `t1` to table `t2`:

```
UPDATE t1
SET c1 = t2.c1
FROM t2
WHERE t2.c2 = t1.c2;
```

- You can specify join conditions in the `FROM` clause to explicitly join additional tables to each other. You can also use the `FROM` clause to join an aliased instance of the target table to other tables, but only if a self-join is intended. In the `FROM` clause, you can join tables with the `JOIN...ON` syntax. The first table to be joined is named directly after the `FROM` keyword. For example:

```
UPDATE t1
SET c1 = t2.c1
FROM t2 JOIN t3 on t2.c2 = t3.c2
WHERE t1.c2=t2.c2;
```

Note that `t1` is joined to `t2` in the `WHERE` clause. If you prefer, you can join all three tables explicitly in the `WHERE` clause:

```
UPDATE t1
SET c1=t2.c1
FROM t2, t3
WHERE t1.c2=t2.c2
AND t2.c2 = t3.c2;
```

- Pay special attention to the complete syntax of the `UPDATE` statement when you are joining tables. If you do not specify a full set of join conditions between all of the named tables, the query plan may generate a very costly cross-join. For example, consider this statement, in which `t1` has no join condition to the other table (`t2`) but `t1alias` does:

```
UPDATE t1
SET c1 = t2.c1
FROM t1 t1alias
JOIN t2 ON t1alias.c2 = t2.c2;
```

This is a common mistake that results in a cross-join between `t1` and the result of the other join between `t1alias` and `t2` . This mistake may derive simply from faulty join syntax, or it may derive from a misconception that the target table needs to be named (and aliased) in the `FROM` clause. *It does not need to be named unless a self-join is intended.*

- Each output row of a join of the target table to another table results in an update on the target table. Make sure that the join produces only one output row (or none) for each row to be modified. A target table row should not join to multiple rows from the other tables. If it does, the following error is returned:

```
ERROR:  Attempt to update a target row multiple times
```

Examples

For example, update the `team` table, aliased as `t` , and set four columns to the same value based on a condition:

```
premdb=# update team t
set (t.name, t.nickname, t.city, t.stadium) = ('TBD','TBD','TBD','TBD')
where t.name is null;
UPDATE 3
```

The following example uses a join to qualify rows for the `UPDATE` statement. A `stadium` column is added to the `match` table. The `UPDATE` statement joins the `match` and `team` tables to update the `match.stadium` column with the same values found in the `team.stadium` column. An alias `m` is used for the target table and referenced in

the `WHERE` clause.

```
premdb=# alter table match add column stadium varchar(50);
ALTER TABLE
premdb=# update match m set m.stadium=t.stadium from team t where t.htid=m.htid;
UPDATE 8606
```

The following query checks the results of the last `UPDATE` (for one `htid` and `atid` combination):

```
premdb=# select * from match where htid=42 and atid=89 order by seasonid;
 seasonid |      matchday       | htid | atid | ftscore | htscore |     stadium
----------+---------------------+------+------+---------+---------+-----------------
       20 | 2012-04-01 00:00:00 |   42 |   89 | 3-1     | 1-0     | White Hart Lane
       21 | 2012-12-16 00:00:00 |   42 |   89 | 1-0     | 0-0     | White Hart Lane
       22 | 2013-08-25 00:00:00 |   42 |   89 | 1-0     | 0-0     | White Hart Lane
(3 rows)
```

The following example joins the target table `match` to the `season` table. The `SET` clause uses an expression based on the `matchday` column. Rows are qualified for the update via the join condition and an additional predicate:

```
premdb=# update match m set m.matchday=m.matchday+interval '15 hours'
from season s
where m.seasonid=s.seasonid and s.seasonid=1;
UPDATE 462
```

The following example joins the following tables:

- `match` (the target table)
- `m`, an alias for `match`, used in a self-join
- `match_update` (aliased as `updt`)

All three tables are explicitly joined, using the `FROM` and `WHERE` clauses for the join conditions. The target table match is joined in the `WHERE` clause, and the other tables are joined in the `FROM` clause.

```
UPDATE match
SET ftscore = updt.ftscore, htscore = updt.htscore
FROM match_update updt
  JOIN match m ON m.matchday = updt.matchday
  AND m.htid = updt.htid
  AND m.atid = updt.atid
WHERE match.matchday = updt.matchday
  AND match.htid = updt.htid
  AND match.atid = updt.atid
  AND m.ftscore is null;
```

The following example is similar to the previous example but uses a subquery:

```
UPDATE match
SET
   ftscore  = updt.ftscore
 , htscore  = updt.htscore
FROM
(
 SELECT
    m.matchday
  , m.htid
```

```
  , m.atid
  , u.ftscore
  , u.htscore
 FROM
        match        m
    JOIN match_update u ON
         m.matchday = updt.matchday
      AND m.htid     = updt.htid
      AND m.atid     = updt.atid
 WHERE
    m.ftscore is null
) updt
WHERE
  match.matchday = updt.matchday
  AND match.htid = updt.htid
  AND match.atid = updt.atid
;
```

The following example is similar to the previous example but uses a `WITH` clause:

```
WITH updt AS
(  SELECT
    m.matchday
  , m.htid
  , m.atid
  , u.ftscore
  , u.htscore
 FROM
        match        m
    JOIN match_update u ON
         m.matchday = updt.matchday
      AND m.htid     = updt.htid
      AND m.atid     = updt.atid
 WHERE
    m.ftscore is null
)

UPDATE
  match
SET
   ftscore  = updt.ftscore
 , htscore  = updt.htscore
FROM
   updt
WHERE
  match.matchday = updt.matchday
  AND match.htid = updt.htid
  AND match.atid = updt.atid
;
```

USE CLUSTER

Set the cluster to use for queries and other database operations.

```
USE CLUSTER name
```

If the cluster name includes special characters, use double quotes around the name.

See also ALTER USER SET DEFAULT_CLUSTER.

For example, return the list of available clusters, then set the cluster:

```
yellowbrick=# select * from sys.cluster;
-[ RECORD 1 ]-------------+-----------------------------------
cluster_id                | 9068c6c5-7d09-4091-b8a9-8978f62cc970
cluster_name              | uat-rc5-cluster-01
nodes                     | 1
default_wlm_profile_name  | default
active_wlm_profile_name   | default
hardware_instance_type_id | b7da0afd-2c98-4f57-888d-98a51975a412
hardware_instance_name    | large-v1
is_default_cluster        | t
auto_suspend              | 300
auto_resume               | t
max_spill_pct             | 30
max_cache_pct             | 70
type                      | COMPUTE
state                     | RUNNING
-[ RECORD 2 ]-------------+-----------------------------------
cluster_id                | c1fab7bc-c24e-4aa2-9217-c21f414dd020
cluster_name              | bobr-rc5-april4-cluster
nodes                     | 2
default_wlm_profile_name  | default
active_wlm_profile_name   | default
hardware_instance_type_id | f43d64aa-828e-42cc-b402-60e008c544a3
hardware_instance_name    | small-v1
is_default_cluster        | f
auto_suspend              | 300
auto_resume               | t
max_spill_pct             | 30
max_cache_pct             | 70
type                      | COMPUTE
state                     | SUSPENDED

yellowbrick=# use cluster "bobr-rc5-april4-cluster";
SET
```

SQL Data Types

Yellowbrick Documentation > Reference > SQL Reference > SQL Data Types

Platforms: All platforms

Parent topic: SQL Reference

In this section:

Yellowbrick supports the following data types. The aliases listed in the table are valid alternative keywords for each data type. For more details about each type, go to the linked topic.

Data Type	Aliases	Description
BOOLEAN	`BOOL`	1-byte Boolean value
SMALLINT	`INT2`	16-bit signed integer
INTEGER	`INT`, `INT4`	32-bit signed integer
BIGINT	`INT8`	64-bit signed integer
DECIMAL	`DEC`, `NUMERIC`	Numeric values with a maximum precision of 38
REAL	`FLOAT4`	32-bit floating-point value
DOUBLE PRECISION	`FLOAT`, `FLOAT8`	64-bit floating-point value

Data Type	Aliases	Description
UUID		128-bit generated value (a sequence of lowercase or uppercase hexadecimal digits)
VARCHAR	`CHARACTER VARYING`	Variable-length character string
BYTEA	`VARBINARY`, `VARBYTE`	A sequence of octets (or bytes)
CHAR	`CHARACTER`, `BPCHAR`	Fixed-length character string
DATE		Date
TIME		Time of day value
TIMESTAMP		Timestamp without time zone
TIMESTAMP WITH TIME ZONE	`TIMESTAMPTZ`	Timestamp with time zone
IPV4		IP address (4-byte)
IPV6		IP address (16-byte)
MACADDR		MAC address (6-byte)
MACADDR8		MAC address (8-byte, EUI-64 format)
JSON		Stores the textual "raw" format of the `JSON`. It preserves the original text, whitespace and formatting. However, unlike PostgreSQL, no validation is performed
JSONB		Stores a binary representation of the `JSON` data. The binary form is optimized for fast lookups and retrieval, and it supports a set of functions for acting on the data. However, the original formatting is not preserved
GEOGRAPHY		Stores geospatial data representing points, lines, and polygons on the Earth's surface using geodetic coordinates

Data Types Currently Not Supported

- `INTERVAL` is not supported as a declared column type for a table (`INTERVAL` data cannot be stored). However, functions and expressions can operate on `INTERVAL` literals and produce `INTERVAL` output values (such as in views).
- `TEXT`: You can cast `TEXT` values to character strings or other types, but you cannot create a table with a `TEXT` column. Any non-constant `TEXT` value that the system processes is converted to a `VARCHAR(64000)` value internally. This behavior is inefficient so Yellowbrick recommends keeping use of `TEXT` values to a minimum.
- `NAME` is not supported as a column type.
- `TIMETZ` is not supported.
- `SERIAL` is not supported.

Data Type Casting

In this section:

You can explicitly cast an expression, such as a column or the result of a function, to a specific data type by using `CAST` (or `::`). See Explicit Casting.

Yellowbrick supports implicit casting in many cases. For example, if a query compares an `INTEGER` column to a `VARCHAR` column, an implicit cast occurs automatically. For details, see Implicit Casting.

Data Type Compatibility

To some extent, the ability to do explicit or implicit casting depends on data type compatibility. Yellowbrick data types fall into the following categories.

- Date/time types: `DATE`, `TIME`, `TIMESTAMP`, `TIMESTAMPTZ`, `INTERVAL`

Note that `INTERVAL` is not supported as a stored column type.

- Numeric types: `SMALLINT`, `INTEGER`, `BIGINT`, `DECIMAL` (`NUMERIC`), `REAL` (`FLOAT4`), `DOUBLE PRECISION` (`FLOAT8`)
- Boolean type: `BOOLEAN`
- UUID type: `UUID`
- IP address type: `IPV4`, `IPV6`
- MAC address types: `MACADDR`, `MACADDR8`
- String or character types: `VARCHAR`, `CHAR`

Within these categories, implicit casting is more likely to work and explicit casting is less likely to be required. However, some types can cross these boundaries; for example, character types can be cast implicitly to all other types. See the tables under Implicit Casting and Explicit Casting for details. If a specific combination of "from" and "to" types is not shown in either of these tables, no such cast is possible, explicitly or implicitly.

See SQL Data Types for general information about supported types.

Explicit Casting

This section identifies data type conversions that require explicit casts and describes functions that you can use for explicit casting.

Required Explicit Casts

The following table lists data type combinations *that must be cast explicitly*; implicit casts are not supported for these combinations. (Any combination that supports implicit casting also supports explicit casting.)

From Data Type	To Data Type(s)
`SMALLINT`	`BOOLEAN`
`INTEGER`	`BOOLEAN`, `DATE`, `TIMESTAMP`, `TIMESTAMPTZ`
`BIGINT`	`BOOLEAN`, `DATE`, `TIMESTAMP`, `TIMESTAMPTZ`
`DECIMAL`	`BOOLEAN`, `DECIMAL`
`REAL`	`BOOLEAN`, `DECIMAL`
`DOUBLE PRECISION`	`BOOLEAN`, `DECIMAL`
`BOOLEAN`	`SMALLINT`, `INTEGER`, `BIGINT`, `DECIMAL`, `REAL`, `DOUBLE PRECISION` Explicit casts yield the values `0` and `1`.
`JSONB`	`SMALLINT`, `INTEGER`, `BIGINT`, `DECIMAL`, `REAL`, `DOUBLE PRECISION`, `VARCHAR`, `BOOLEAN`, `BYTEA` See JSONB Casting

Note: Explicit casts to `DECIMAL` require a declaration of the precision and scale: `(p,s)`

CAST Function

You can explicitly cast an expression, such as a column or the result of a function, to a specific data type.

```
CAST ( expression AS type )
expression::type
```

See SQL Data Types for information about supported types.

For example, the following queries are equivalent:

```
premdb=# select avg(capacity)::int from team;
  avg
-------
 29982
(1 row)
```

```
premdb=# select cast(avg(capacity) as int) from team;
  avg
-------
 29982
(1 row)
```

For example, to compare a character string (`name`) with an integer (`teamid`):

```
premdb=# select name from team where name=teamid::varchar(30);
 name
------
(0 rows)
```

INT2, INT4, and INT8 Functions

You can use the following functions to coerce numeric values to integer data types:

- `INT2(expression)` : cast the expression to a `SMALLINT` type (16-bit)
- `INT4(expression)` : cast the expression to an `INTEGER` type (32-bit)
- `INT8(expression)` : cast the expression to a `BIGINT` type (64-bit)

For example:

```
premdb=# select int2(seasonid), int4(htid), int8(atid) from match limit 3;
 int2 | int4 | int8
------+------+------
    1 |    2 |   52
    1 |    2 |   55
    1 |    2 |   63
(3 rows)
```

FLOAT4 and FLOAT8 Functions

You can use the following functions to coerce numeric values to floating-point data types:

- `FLOAT4(expression)` : cast the expression to a `REAL` type (32-bit)
- `FLOAT8(expression)` : cast the expression to a `DOUBLE PRECISION` type (64-bit)

For example:

```
premdb=# select float4(avg_att), float8(capacity) from team limit 3;
 float4 | float8
--------+--------
 59.944 |  60260
  33.69 |  42785
      0 |  23009
(3 rows)
```

Implicit Casting

This section describes the Yellowbrick data types in terms of their ability to be *coerced*. Expressions in queries often result in *implicit casts*, where a given data type is automatically coerced to another compatible type. One-to-one matching of data values and data types is not necessary; if data types are compatible, implicit conversion occurs.

To cover certain cases where implicit conversion is not supported, you can convert data types in expressions explicitly, as described in Explicit Casting.

Supported Implicit Casts

The following table lists data types that are cast implicitly.

From Data Type	To Data Type(s)
`VARCHAR`	All types
`CHAR`	All types
`BYTEA`	`VARCHAR`, `CHAR`
`SMALLINT`	`INTEGER`, `BIGINT`, `DECIMAL`, `REAL`, `DOUBLE PRECISION`, `VARCHAR`, `CHAR`
`INTEGER`	`SMALLINT`, `BIGINT`, `DECIMAL`, `REAL`, `DOUBLE PRECISION`, `VARCHAR`, `CHAR`
`BIGINT`	`SMALLINT`, `INTEGER`, `DECIMAL`, `REAL`, `DOUBLE PRECISION`, `VARCHAR`, `CHAR`
`DECIMAL`	`SMALLINT`, `INTEGER`, `BIGINT`, `REAL`, `DOUBLE PRECISION`, `VARCHAR`, `CHAR`
`REAL`	`SMALLINT`, `INTEGER`, `BIGINT`, `DOUBLE PRECISION`, `VARCHAR`, `CHAR`
`DOUBLE PRECISION`	`SMALLINT`, `INTEGER`, `BIGINT`, `REAL`, `VARCHAR`, `CHAR`
`DATE`	`TIMESTAMP`, `TIMESTAMPTZ`, `VARCHAR`, `CHAR`
`INTERVAL`	`TIME`, `VARCHAR`, `CHAR`
`TIME`	`INTERVAL`, `VARCHAR`, `CHAR`
`TIMESTAMP`	`DATE`, `TIMESTAMPTZ`, `TIME`, `VARCHAR`, `CHAR`
`TIMESTAMPTZ`	`DATE`, `TIMESTAMP`, `TIME`, `VARCHAR`, `CHAR`
`BOOLEAN`	`VARCHAR`, `CHAR`
`UUID`	`VARCHAR`, `CHAR`
`IPV4`	`IPV6`, `VARCHAR`, `CHAR`
`IPV6`	`VARCHAR`, `CHAR`
`MACADDR`	`MACADDR8`, `VARCHAR`, `CHAR`

From Data Type	To Data Type(s)
`MACADDR8`	`VARCHAR`, `CHAR`
`JSONB`	`JSON`
`JSON`	`JSONB`, `VARCHAR`
`UNKNOWN` (special Postgres type)	All types

Implicit casting of Boolean values to integers or integers to Boolean values is not supported. However, the values of `0` and `1` are comparable with `false` and `true` for Boolean columns.

In most cases when implicit conversions are not possible, you will see the following `HINT` as part of the error message:

```
HINT:  No operator matches the given name and argument type(s). You might need to add explicit type casts.
```

Down-casting Character Strings

By default, character strings are not cast down to fit into smaller `CHAR` or `VARCHAR` types. For example, attempting to insert a 3-character string into a 2-character column returns an error. Down-casting and truncation of strings can be allowed by setting the `enable_silent_coerce` configuration parameter.

Data Type Hierarchy

When data types are cast implicitly, they are cast "up" or "down" according to the following hierarchy.

For example, `TIMESTAMPTZ` has precedence over the other datetime types. In turn, a comparison of a `TIMESTAMP` value and a `TIMESTAMPTZ` value produces a `TIMESTAMPTZ` result.

For character strings, `VARCHAR` has precedence over `CHAR`, and `CHAR` has precedence over `UNKNOWN`.

- `TIMESTAMPTZ`
- `TIMESTAMP`
- `DATE`
- `INTERVAL`
- `TIME`
- `DOUBLE PRECISION` (`FLOAT8`)
- `REAL` (`FLOAT4`)
- `DECIMAL` (`NUMERIC`)
- `BIGINT`
- `INT`
- `SMALLINT`
- `BOOLEAN`
- `UUID`
- `IPV6`
- `IPV4`
- `MACADDR8`
- `MACADDR`

- `JSONB`
- `JSON`
- `VARCHAR`
- `CHAR`
- `BYTEA`
- `UNKNOWN`

Note: `UNKNOWN` is a special PostgreSQL type that is used for single-quoted strings and `NULL` values.

Data Types for Expression Results

When different types are compared or participate in expressions, including comparisons in join conditions, filters, sorts, and `GROUP BY` expressions, the following general typing rules apply:

- All data types are implicitly cast to the type that is highest in the hierarchy for the arguments involved.
- If no implicit cast from a lower type to a higher type is available, an error occurs.
- If multiple, equally valid, casts are available and the first rule does not apply, an error occurs. For example, `SUM(NULL)` returns an error because the data type of `NULL` cannot be inferred, and `TO_CHAR(string, string)` returns an error because there are multiple, equally valid, casts for the first argument.

The following table provides more details for certain operators, functions, and expressions:

Expression	Rule Details
`AND` / `OR`	Cast all arguments to `boolean`.
`NOT <expr>`	Attempt to coerce `<expr>` to `boolean`.
`CASE` expressions	The resulting output is the highest type of each branch (including the `ELSE` part of the expression). All other branches are implicitly cast to that type. Given this form: `CASE <expr> WHEN <expr0> then <expr0'> ... WHEN <exprN> then <exprN'>` all `<expr0>...<exprN>` and `<expr>` parts are cast to the highest type.
`DECODE`	Apply the same rule as for `CASE`.
`COALESCE(<expr0>, <expr1>, ... <exprN>)`	Coerce all expressions to the highest of `<expr0>...<exprN>`. The result is of the same type as the highest.
`GREATEST`, `LEAST`	Apply the same rule as for `COALESCE`.
`NVL(<bool>, <expr0>)`	Return the type of `<expr0>`.
`NVL2(<bool>, <expr0>, <expr1>)`	Return the highest type of `<expr0>` and `<expr1>`.

Expression	Rule Details
`SELECT <expr0> AS x` `UNION / EXCEPT / INTERSECT` `...` `SELECT <exprN>`	Coerce `x` to the highest type for `<expr0>...<exprN>`.All other types are implicitly cast to that type. If an implicit cast is unavailable, the query returns an error.
+, -, /, *, %	When one side of an operator is a string type and the other is numeric, implicitly cast the string to the numeric type. (Even when a string is added to an integer, en error can still result if the string representation of the integer overflows the target integer type.)When one side of an operator is a datetime type and the other is not: - If the non-datetime type can be cast to the same type as the datetime type, that type is used. - Otherwise, the highest available cast in the datetime hierarchy is used.

See also Implicit Casting Examples.

Casting Strings with UNKNOWN Literals

When a common type must be found between two strings, and one or both of the strings is `UNKNOWN`, note the following behavior:

- `UNKNOWN` + `UNKNOWN` results in a `VARCHAR`. For example:

```
'abc  ' || 'zzz  ' -> 'abc  zzz  '
```

- `UNKNOWN` + `CHAR` results in a `CHAR`. For example:

```
'abc  '::char(10) || 'zzz   '  -> 'abczzz'
```

- `UNKNOWN` + `VARCHAR` results in a `VARCHAR`. For example:

```
'abc  '::varchar(10) || 'zzz   ' -> 'abc   zzz'
```

- `VARCHAR` + `UNKNOWN` + `CHAR` results in a `VARCHAR`. For example:

```
'abc  '::varchar(10) || 'zzz   ' || '+++   ':char(10)  -> 'abc   zzz   +++'
```

These examples show the || operator, but the same behavior applies to other operators, functions, and expressions where a common type must be found, such as set operators, `CASE` expressions, and simple `>` or `<` less than comparisons.

Assignment of Inserted and Updated Values

Assignment operations occur during the execution of statements such as `INSERT`, `UPDATE`, and `FETCH`, and when values are assigned to variables in stored procedures. These operations are treated differently from expression evaluation.

In these cases, both implicit and explicit casts should be emitted automatically. Given that the left side of the assignment has a known type, the other side must be cast, and there is only one candidate for the cast that can be chosen. For example, consider the following table:

```
premdb=# create table assign (a numeric(10,2), b float);
CREATE TABLE
premdb=# \d assign
        Table "public.assign"
 Column |       Type       | Modifiers
--------+------------------+-----------
 a      | numeric(10,2)    |
 b      | double precision |

Distribution: Hash (a)
```

```
premdb=# insert into assign values (1.2::float, 1.2::numeric(10,2));
INSERT 0 1
premdb=# update assign set a = b;
UPDATE 1
```

Note how the `INSERT` and `UPDATE` statements are executed. The inserted values are ultimately cast to the types defined for the columns, not the types defined in the `INSERT` statement.

```
premdb=# explain verbose insert into assign values (1.2::float, 1.2::numeric(10,2));
                                    QUERY PLAN
-----------------------------------------------------------------------------------------
 Insert on public.assign (on manager)  (output dist=None)  (cost=0.00..0.00 rows=1 width=0)
   ->  Result (on manager)  (output dist=None)  (cost=0.00..0.00 rows=1 width=0)
         Output: 1.20::numeric(10,2), '1.2'::double precision
(3 rows)
```

In the `UPDATE` statement, column `a` is set by casting column `b` to `NUMERIC(10,2)`, which is the stored data type of column `a`.

```
premdb=# explain verbose update assign set a = b;
                                                                QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------
 id   rows_planned   mem_planned   mem_actual   workers   node
  2           100      3.00Gi H         0.00       all   INSERT INTO assign
                                                          (assign.b::NUMERIC(10, 2), assign.b, assign.rowunique, assign.rowid)
                                                          distribute on (assign.b::NUMERIC(10, 2))

...
```

Implicit Casting Examples

This section shows some examples of queries in which an implicit cast converts a value or expression of a given data type to another data type.

Casts for Dates and Timestamps

The following example implicitly casts a `TIMESTAMP` column (`matchday`) to a character string in order to return the result of the `SUBSTR` function.

```
premdb=# select matchday, substr(matchday,1,4) yr from match where htid=3 and atid=56;
     matchday       |  yr
--------------------+------
 2010-11-10 00:00:00 | 2010
(1 row)
```

The following example inserts a `DATE` value into a `TIMESTAMP` column:

```
premdb=# insert into match(matchday) select current_date;
INSERT 0 1
premdb=# select * from match where matchday=(select current_date);
 seasonid |      matchday       |  htid  |  atid  | ftscore | htscore
----------+---------------------+--------+--------+---------+---------
   [NULL] | 2020-09-25 00:00:00 | [NULL] | [NULL] | [NULL]  | [NULL]
(1 row)
```

This example uses a table called `chartime` , which contains four `VARCHAR` columns. However, these columns contain `VARCHAR` strings that represent `TIME` , `TIMESTAMP` , `TIMESTAMPTZ` , and `DATE` values.

```
premdb=# \d chartime
         Table "public.chartime"
 Column |         Type          | Modifiers
--------+-----------------------+-----------
 c1     | character varying(12) |
 c2     | character varying(24) |
 c3     | character varying(24) |
 c4     | character varying(10) |

Distribution: Hash (c1)

premdb=# select * from chartime;
   c1    |         c2          |           c3           |     c4
---------+---------------------+------------------------+------------
 00:00:00 | 2012-12-22 00:00:00 | 2012-12-22 00:00:00-07 | 2012-12-22
 12:55:00 | 2020-09-25 12:55:00 | 2020-09-25 12:55:00-07 | 2020-09-25
(2 rows)
```

Implicit casts make it possible to query and compare these columns directly with datetime functions such as `EXTRACT` . For example:

```
premdb=# select extract(day from c2) from chartime;
 date_part
-----------
        25
        22
(2 rows)
```

You can use an `EXPLAIN` command with the `VERBOSE` option to see the implicit cast on the `c2` column in this example:

```
premdb=# explain (verbose) select extract(day from c2) from chartime;
                                        QUERY PLAN
-------------------------------------------------------------------------------------------------------------
 id   rows_planned   mem_planned   mem_actual   workers   node
  1           100      8.00Mi H         0.00       all   SELECT
                                                          (EXTRACT(day FROM chartime.c2::TIMESTAMPTZ))
                                                          distribute on (chartime.c1)
  3           100     57.00Mi H         0.00       all   SCAN chartime
                                                          (chartime.c2)
                                                          distribute on (chartime.c1)
...
```

A join condition between one a `VARCHAR` column in the `chartime` table and the `match.matchday` `TIMESTAMP` column also benefits from an implicit cast:

```
premdb=# select * from match, chartime where match.matchday=chartime.c2;
 seasonid |      matchday       | htid | atid | ftscore | htscore |    c1    |         c2          |          c3           |
----------+---------------------+------+------+---------+---------+----------+---------------------+-----------------------+-----
       21 | 2012-12-22 00:00:00 |   44 |   78 | 2-1     | 1-1     | 00:00:00 | 2012-12-22 00:00:00 | 2012-12-22 00:00:00-07 | 2012-
       21 | 2012-12-22 00:00:00 |   45 |   67 | 1-2     | 1-0     | 00:00:00 | 2012-12-22 00:00:00 | 2012-12-22 00:00:00-07 | 2012-
       21 | 2012-12-22 00:00:00 |   46 |   51 | 0-1     | 0-0     | 00:00:00 | 2012-12-22 00:00:00 | 2012-12-22 00:00:00-07 | 2012-
...
```

Again you can use the `EXPLAIN` command to see the implicit cast:

```
premdb=# explain select * from match, chartime where match.matchday=chartime.c2;
                                          QUERY PLAN
-----------------------------------------------------------------------------------------------------------
 id   rows_planned   workers   node
  1         396783       all   SELECT
 10         396783       all   INNER JOIN ON (chartime.c2::TIMESTAMP = match.matchday)
 13        8812544       all   |-SCAN match
                               |   match.matchday = bloom(2) AND scan_constraints: min_max(match.matchday)
  2           100       all   |-BUILD
  3           100       all     DISTRIBUTE REPLICATE
  5           100       all     SCAN chartime
...
```

In this example, a `VARCHAR` column contains a character string for a date. When the `DATEADD` function is applied to this string, the highest type in the datetime hierarchy (`TIMESTAMPTZ`) is used for the implicit cast:

```
premdb=# \d vardate
          Table "public.vardate"
 Column |         Type          | Modifiers
--------+-----------------------+-----------
 c1     | character varying(10) |

Distribution: Hash (c1)

premdb=# insert into vardate values('2020-12-31');
```

```
INSERT 0 1
premdb=# select dateadd(month, 1, c1) from vardate;
       dateadd
-----------------------
 2021-01-31 00:00:00-08
(1 row)

premdb=# explain verbose select dateadd(month, 1, c1) from vardate;
                                          QUERY PLAN
-----------------------------------------------------------------------------------------------------
 id   rows_planned   mem_planned   mem_actual   workers   node
  1           100     32.00Mi H         0.00       all   SELECT
                                                          (DATEADD(month, $0, vardate.c1::TIMESTAMPTZ)
                                                          distribute on (vardate.c1)
  3           100     57.00Mi H         0.00       all   SCAN vardate
                                                          (vardate.c1)
                                                          distribute on (vardate.c1)
 ...
```

Numbers as Character Strings

When the `CONCAT` function concatenates data with different types, the values are implicitly cast to `VARCHAR`. In this case, `teamid` and `capacity` are integer columns:

```
premdb=# select concat(teamid,' ',name,' ',capacity) from team limit 3;
       concat
-----------------------
 37 Stoke City 27902
 38 Sunderland 49000
 39 Swansea City 20520
(3 rows)

premdb=# explain verbose select concat(teamid,' ',name,' ',capacity) from team limit 3;
                                                                QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------
 id   rows_planned   mem_planned   mem_actual   workers   node
  1             3     32.98Mi H         0.00    single   SELECT
                                                          (team.teamid::VARCHAR(6) || $0 || team.name || $1 || team.capacity::VAR
                                                          distribute single
  2             3      4.00Mi H         0.00    single   LIMIT $4
                                                          (team.teamid::VARCHAR(6) || $0 || team.name || $1 || team.capacity::VAR
                                                          distribute single
  4           100     57.00Mi H         0.00    single   SCAN team
                                                          (team.teamid, team.name, team.capacity)
                                                          distribute single
...
```

You can implicitly cast an integer column to a character string in order to apply a `LIKE` condition:

```
premdb=# select * from team where capacity like '75%';
 teamid | htid | atid |       name        |  nickname  |    city    |   stadium    | capacity
--------+------+------+-------------------+------------+------------+--------------+----------
    25 |   26 |   75 | Manchester United | Red Devils | Manchester | Old Trafford |    75635
(1 row)

premdb=# explain select * from team where capacity like '75%';
                              QUERY PLAN
-----------------------------------------------------------------------
 id   rows_planned   workers   node
  1             1    single   SELECT
  3             1    single   SCAN team
                               team.capacity::VARCHAR(64000)  LIKE '$0'
```

The following query tries to subtract one number from another number, using `SUBSTR` to extract the numbers from the `ftscore` and `htscore`. However, these columns are stored as character strings:

```
premdb=# \d match
              Table "public.match"
  Column  |            Type             | Modifiers
----------+-----------------------------+-----------
 seasonid | smallint                    |
 matchday | timestamp without time zone |
 htid     | smallint                    |
 atid     | smallint                    |
 ftscore  | character(3)                |
 htscore  | character(3)                |
```

By default, an implicit cast to a `TIMESTAMPTZ` is chosen and attempted for the `SUBSTR` expression, but the query fails.

```
premdb=# select seasonid,htid,atid,ftscore,htscore,substr(ftscore,1,1)-substr(htscore,1,1) goaldiff
from match
where seasonid=22 and htid=2 and atid between 50 and 75
order by 1,2,3;
ERROR:  invalid datetime format
DETAIL:   [reason => invalid input syntax for type timestamp with time zone: "1"]
```

If one of the arguments in the expression is explicitly cast to an integer, the query succeeds:

```
premdb=# select seasonid,htid,atid,ftscore,htscore,
substr(ftscore,1,1)::smallint-substr(htscore,1,1) goaldiff
from match
where seasonid=22 and htid=2 and atid between 50 and 75
order by 1,2,3
 seasonid | htid | atid | ftscore | htscore | goaldiff
----------+------+------+---------+---------+----------
       22 |    2 |   52 | 1-3     | 1-1     |        0
       22 |    2 |   61 | 2-0     | 0-0     |        2
       22 |    2 |   63 | 0-0     | 0-0     |        0
       22 |    2 |   65 | 2-0     | 0-0     |        2
       22 |    2 |   67 | 1-1     | 0-0     |        1
       22 |    2 |   68 | 2-0     | 0-0     |        2
       22 |    2 |   69 | 2-0     | 1-0     |        1
       22 |    2 |   73 | 2-0     | 1-0     |        1
       22 |    2 |   74 | 1-1     | 0-1     |        1
       22 |    2 |   75 | 0-0     | 0-0     |        0
(10 rows)
```

Adding Two Character Strings

This example shows the expected behavior when both sides of an addition operator are character strings. The query returns an error because the expected calculation is ambiguous in this context:

```
premdb=# create table varnum(c1 char(5), c2 varchar(5));
CREATE TABLE
premdb=# insert into varnum values(1,1);
INSERT 0 1
premdb=# insert into varnum values(10,20);
INSERT 0 1
premdb=# select c1+c2 from varnum;
ERROR:  operator is not unique: character + character varying
LINE 1: select c1+c2 from varnum;
```

```
               ^
HINT:  Could not choose a best candidate operator. You might need to add explicit type casts.
```

Note that when you cast one side of the addition operator explicitly, the other side is implicitly cast to match that type. In the first example, column `c2` is implicitly cast to an integer; in the second example, `c2` is implicitly cast to an interval:

```
premdb=# select c1::int+c2 from varnum;
 ?column?
----------
        2
       30
(2 rows)

premdb=# select c1::interval+c2 from varnum;
 ?column?
----------
 00:00:02
 00:00:30
(2 rows)
```

MACADDR Cast to MACADDR8

Seven rows are inserted into the following table, and the `MACADDR` values for column `c1` are cast implicitly to `MACADDR8` for insertion into column `c2`.

```
premdb=# \d mac
     Table "public.mac"
 Column |   Type    | Modifiers
--------+-----------+-----------
 c1     | macaddr   |
 c2     | macaddr8 |

Distribution: Hash (c1)

premdb=# insert into mac
premdb-# values('08:00:2b:01:02:03', '08:00:2b:01:02:03'),
premdb-#       ('08-00-2b-01-02-03', '08-00-2b-01-02-03'),
premdb-#       ('08002b:010203', '08002b:010203'),
premdb-#       ('08002b-010203', '08002b-010203'),
premdb-#       ('0800.2b01.0203', '0800.2b01.0203'),
premdb-#       ('0800-2b01-0203', '0800-2b01-0203'),
premdb-#       ('08002b010203', '08002b010203');
INSERT 0 7
premdb=# select * from mac;
       c1         |           c2
------------------+-------------------------
 08:00:2b:01:02:03 | 08:00:2b:ff:fe:01:02:03
 08:00:2b:01:02:03 | 08:00:2b:ff:fe:01:02:03
 08:00:2b:01:02:03 | 08:00:2b:ff:fe:01:02:03
 08:00:2b:01:02:03 | 08:00:2b:ff:fe:01:02:03
 08:00:2b:01:02:03 | 08:00:2b:ff:fe:01:02:03
 08:00:2b:01:02:03 | 08:00:2b:ff:fe:01:02:03
 08:00:2b:01:02:03 | 08:00:2b:ff:fe:01:02:03
(7 rows)
```

IPV4 Cast to IPV6

The following example shows an `IPV4` value cast to an `IPV6` value for storage in an `IPV6` column:

```
premdb=# select * from ipv4_ipv6;
     c1       |                  c2
--------------+---------------------------------------
 19.120.51.251 | 684d:1111:0222:3333:4444:5555:0006:0077
(1 row)

premdb=# insert into ipv4_ipv6(c2) select c1 from ipv4_ipv6;
INSERT 0 1
premdb=# select * from ipv4_ipv6;
     c1       |                  c2
--------------+---------------------------------------
 [NULL]        | 0000:0000:0000:0000:0000:ffff:1378:33fb
 19.120.51.251 | 684d:1111:0222:3333:4444:5555:0006:0077
(2 rows)

premdb=# explain (verbose) insert into ipv4_ipv6(c2) select c1 from ipv4_ipv6;
                                    QUERY PLAN
-------------------------------------------------------------------------------------------
 id   rows_planned   mem_planned   mem_actual   workers   node
  2             1      3.00Gi H         0.00       all   INSERT INTO ipv4_ipv6
                                                          (NULL, ipv4_ipv6.c1::IPV6)
                                                          distribute on (NULL)
  3             1    272.00Mi H         0.00       all   DISTRIBUTE ON HASH(NULL)
                                                          (NULL, ipv4_ipv6.c1::IPV6)
                                                          distribute on (NULL)
...
```

DECIMAL

In this section:

The DECIMAL data type stores signed integer values with a user-defined precision and scale. NUMERIC and DEC are valid synonyms for DECIMAL. The sign character (+ or -) is optional. Leading zeros and whitespace characters are allowed. Trailing whitespace characters are also allowed.

Precision and Scale

The *precision* of a DECIMAL column defines its maximum total number of digits, including decimal places (its *scale*). The maximum precision for a DECIMAL column is 38.

The scale of a DECIMAL column defines the maximum number of digits to the right of the decimal point. For example, the number `5999.95` has a precision of 6 and a scale of 2. A DECIMAL column that is created without a scale value has a default scale of `0`. A DECIMAL column that is created without precision and scale values defaults to `DECIMAL(18,0)`.

You cannot load a value into a DECIMAL column if the number of digits to the left of the decimal point exceeds the precision of the column minus its scale. The specified precision and scale determine the allowed range of values for the column. For example:

Column Specification	Allowed Range
DECIMAL(5,2)	`-999.99` to `+999.99` Values must round to an absolute value less than 103.
DECIMAL(19,0)	`-9999999999999999999` to `+9999999999999999999` (19 nines)Values must round to an absolute value less than 1019.
DECIMAL(38,0)	`-99999999999999999999999999999999999999` to `+99999999999999999999999999999999999999` (38 nines)Values must round to an absolute value less than 1038.
DECIMAL(38,19)	`-9999999999999999999.9999999999999999999` to `+9999999999999999999.9999999999999999999` (19 nines on either side of the decimal point)Values must round to an absolute value less than 1019.

If the number of digits to the right of the decimal point exceeds the specified scale, the input value is rounded rather than rejected.

Examples with DECIMAL Columns

The following table contains two DECIMAL columns:

```
premdb=# create table dectest(c1 dec(5,2),c2 dec(13,9));
CREATE TABLE
premdb=# \d dectest
      Table "public.dectest"
 Column |     Type      | Modifiers
```

```
--------+---------------+-----------
 c1     | numeric(5,2)  |
 c2     | numeric(13,9) |

Distribution: Hash (c1)
```

The following values are in range and are inserted into the table:

```
premdb=# insert into dectest values(178.56,2100.000000001);
INSERT 0 1
premdb=# select * from dectest;
   c1   |       c2
--------+----------------
 178.56 | 2100.000000001
(1 row)
```

The following values have to be rounded to fit the specified scale. Note that the value `178.56666` is rounded up to `178.57`.

```
premdb=# insert into dectest values(178.56666,2100.000000001111);
INSERT 0 1
premdb=# select * from dectest;
   c1   |       c2
--------+----------------
 178.56 | 2100.000000001
 178.57 | 2100.000000001
(2 rows)
```

The following value is out of range for column `c1` so the INSERT operation fails with an expected overflow error:

```
premdb=# insert into dectest(c1) values(1789.56666);
ERROR:  numeric field overflow
DETAIL:  A field with precision 5, scale 2 must round to an absolute value less than 10^3.
```

Calculations with DECIMAL Values

This section explains the expected behavior when SQL functions and operators are applied to DECIMAL values.

Resulting Precision and Scale

The following table shows how precision and scale are computed for mathematical operations that return DECIMAL results.

- `p1` and `s1`: precision and scale of the first operand in a calculation
- `p2` and `s2`: precision and scale of the second operand in a calculation

For the results of all calculations, the maximum precision is 38 and the maximum scale is 38.

Operation	Resulting Precision (p)	Resulting Scale (s)
Addition (+) and Subtraction (-)	`MAX(p1 - s1, p2 - s2) + MAX(s1, s2) + 1`	`MAX(s1, s2)`
Multiplication (*)	`p1 + p2`	`s1 + s2`
Division (/)	`p1 - s1 + s2 + MAX(6, s1 + p2 + 1)`	`MAX(6, s1 + p2 + 1)`
Modulo (%)	`MAX(p1 - s1, p2 - s2) + MAX(s1, s2)`	`MAX(s1, s2)`
UNION, INTERSECT, EXCEPT	`MAX(s1, s2) + MAX(p1 - s1, p2 - s2)`	`MAX(s1, s2)`
CASE expression	`MAX(p1 - s1, p2 - s2) + MAX(s1, s2)`	`MAX(s1, s2)`
SUM function	38	Scale of input value
AVG function	Precision of input value	Scale of input value
TRUNC, ROUND functions	Precision of input value	MIN(scale of input value, scale of second argument to function)
POWER function	38	See the following table.

Resulting Scale for POWER

The following table shows how the scale is computed in a POWER function result. The precision is always 18, but the scale may be 15, 8, or 6. A minimum scale of 6 is always preserved. `p1` and `s1` are the precision and scale of the first argument to the function, and `p2` and `s2` are the precision and scale of the second argument.

MAX(p1 - s1, s1)	p2 - s2	Result type
<= 2	<= 1	(38,15)
3	<= 1	(38, 8)
Any number	Any number	(38, 6)

For example, the following result is a `DECIMAL(38,8)` because `MAX(p1 - s1, s1) = 3` and `p2-s2 = <=1`:

```
premdb=# select pow(10.999,0.9) from sys.const;
   pow
------------
 8.65401975
(1 row)
```

The following result is a `DECIMAL(38,15)` because `MAX(p1 - s1, s1) = 2` and `p2-s2 = <=1`:

```
premdb=# select pow(10.99,0.9) from sys.const;
       pow
-------------------
 8.647646401207339
(1 row)
```

Overflow Conditions

Overflow conditions with DECIMAL calculations are managed (and where possible, prevented) as follows. See the previous table for related information about these calculations.

Addition (+) and Subtraction (-)

An addition or subtraction on two `DECIMAL` operands does not reduce the scale of the result in order to preserve the overall precision. The resulting scale is always `MAX(s1, s2)`. This behavior means that calculations are more likely to return overflow errors but will not automatically and silently round down results. In many cases, you can safeguard overflow conditions by adding explicit casts.

When precision >38, return an overflow error if the integral part of the result still has more than `p - s` number of digits.

Multiplication (*)

- When the scale of the result is `<=6`, retain that scale.

- When the scale is `>6`, reduce it to tighten overall precision and try to prevent overflow.
- Compute `(p1 + p2, s1 + s2)`:
- Return an overflow error if the integral part of the result still has more than `p - s` number of digits.
- If no overflow occurs, round the result to scale `(s1 + s2)`.

Division (/)

- When the scale of the result is `<=6`, set the scale to 6.

- When the scale is `>6`, reduce it to tighten overall precision and try to prevent overflow.
- Compute `(p1 - s1 + s2 + max(6, s1 + p2 + 1), max(6, s1 + p2 + 1))`:
- Return an overflow error if the integral part of the result still has more than `p - s` digits.
- If no overflow occurs but the scale goes over the value of `(MAX(6, s1 + p2 + 1)`, truncate (do not round) the scale.

Modulo (%)

Return an overflow error if the value of the integral part of either operand does not fit into the integral part of the result.

UNION, INTERSECT, EXCEPT operators

The behavior for DECIMAL values propagated by set operations is similar to the behavior for addition and subtraction:

- Reduce scale to tighten overall precision and try to prevent overflow.

- Return an overflow error if the integral part of the result has more than `p - s` digits.
- If no overflow occurs, round the result to scale `MAX(s1, s2)`.

CASE expressions

The behavior for DECIMAL values propagated by CASE expressions is similar to the behavior for addition and subtraction.

- Reduce scale to tighten overall precision and try to prevent overflow.
- Round the result to scale `MAX(s1, s2)` before propagation.
- Return an overflow error if the integral part of the result has more than `p - s` digits.

POWER function

If the integral part of the result is greater than 38 minus the scale of the first argument, the function may return an overflow error.

Examples of DECIMAL Calculations

This section contains some examples that demonstrate how Yellowbrick handles calculations that involve `DECIMAL` columns and constants.

Multiplication Examples

This example shows the resulting precision and scale when two `DECIMAL` values are multiplied:

```
premdb=# select avg_att, avg_att*1.1 att11 from team where avg_att>50;
 avg_att |  att11
---------+---------
  59.944 | 65.9384
  54.041 | 59.4451
  75.286 | 82.8146
(3 rows)
```

The `avg_att` column is a `5,3` column. `1.1` is a `2,1` constant value. The resulting precision and scale is `6,4`.

Assume that a table named `dec385` contains a `DECIMAL(38,5)` column. The column contains two values:

```
premdb=# select * from dec385;
                 c1
-----------------------------------------
 955555555555555555555555555555555.55553
 999999999999888777666555444333222.11111
(2 rows)
```

An attempt to multiply these values with a DECIMAL value greater than 1 returns an overflow error. The resulting precision and scale for the multiplication is `(40,6)`, computed from `(38,5)+(2,1)`. Yellowbrick DECIMAL values have a maximum precision of `38`, of which 6 digits are retained for the scale (for multiplication). The calculated values in this example do not fit into a `(38,6)` column, so the query returns an overflow error.

```
premdb=# select c1*1.1 from dec385;
ERROR:  numeric value out of range
```

Addition Examples

This example adds `(38,5)` column values to a `(38,9)` constant value. The scale of the result is not reduced, and the query returns an overflow error.

```
premdb=# select c1+12345678912345678912345678912.123456789 from dec385;
ERROR:  numeric value out of range
```

This query will run if you manually cast the result:

```
premdb=# select c1+12345678912345678912345678912.123456789::decimal(38,5) from dec385;
               ?column?
----------------------------------------
  955567901234467901234467901234467.67899
 1000012345678801123345467790012134.23457
(2 rows)
```

Assume that the `dectest` table has two DECIMAL(7,2) columns, `dec1` and `dec2`. The following functions yield results that are DECIMAL(38,2):

```
sum(dec1) = decimal(38,2)
sum(dec2) = decimal(38,2)
```

The addition of these two columns also yields a `DECIMAL(38,2)` column:

```
sum(decimal(38,2)) + sum(decimal(38,2)) = decimal(38,2)
```

JSON

> Prerequisites:
>
> - To use the JSON type and all the functions associated with it, you must be connected to a UTF-8 database.
> - While the feature is in beta, you must explicitly enable the feature by setting `enable_full_json` to `ON`.

The JSON data type stores variable-length character strings up to 64,000 bytes.

Note that the JSON data type retains an exact copy of the input text. Consequently, processing functions must reparse the JSON on each execution. Additionally, JSON data type does not validate JSON syntax. For syntax validation and better performance, use JSONB instead.

```sql
CREATE TABLE t (c JSON);

INSERT INTO t VALUES ('{"c":3, "b":2, "a":1}');

INSERT INTO t VALUES ('{"a":');

SELECT * FROM t;
```

```sql
           c
-----------------------
 {"c":3, "b":2, "a":1}
 {"a":
(2 rows)
```

Comparisons

Comparisons on JSON expressions behave like VARCHAR comparisons. Two JSONs are different if they contain the same keys with the same values in different order. For ordering, the C collation is used.

```sql
SELECT '{"a":1,"b":2}'::JSON = '{"b":2,"a":1}'::JSON;
 -- f

SELECT '{"a":1,"b":2}'::JSON < '{"b":2,"a":1}'::JSON;
-- t
```

CHECK_JSON

Validates a JSON string, which is useful for checking the format before inserting it into a JSONB column. The function returns `NULL` if the string is a well-formed JSON or if it is NULL. If the JSON string is malformed, it returns an error message with a location of the error.

```sql
CHECK_JSON(json_string)
```

json_string

A JSON expression to validate.

Examples

```sql
select check_json('{"ticker": ');
-- Invalid value. Pos: 11

select check_json('{"ticker": "INTC"}');
-- null

select check_json('{ "key1" : 1, "key2" : 2, "key1" : 3 }');
-- the key 'key1' is not unique

select check_json('{ "key1" : 1, "key2 : 2 }');
-- Missing a closing quotation mark in string. Pos: 25

select check_json('"key1""key2"');
-- The document root must not be followed by other values. Pos: 3

select check_json('{:100}');
-- Missing a name for object member. Pos: 1

select check_json('{"key1" 100}');
-- Missing a colon after a name of object member. Pos: 5

select check_json('{"key1":100"key2":200,"key3" 300}');
-- Missing a comma or '}' after an object member. Pos: 8

select check_json('"\ufffg"');
-- Incorrect hex digit after \u escape in string. Pos: 1

select check_json('{"emoji": "\uDE0A\uD83D"}');
-- The surrogate pair in string is invalid. Pos: 11
```

JSONB

In this section:

> To use the JSONB type and all the functions associated with it, you must be connected to a UTF-8 database.
>
> While the feature is in beta, you must also explicitly enable the feature by setting the configuration parameter `enable_full_json` to `ON`.

JSONB data type stores JSON documents in a binary format.

Storing data in JSONB columns eliminates the need for JSON string parsing during each function execution. The binary format includes metadata which speeds up operations such as index or key lookups. During insertion, JSON documents are validated and canonicalized: whitespace is removed, and name-value pairs are reordered. Duplicate keys are not permitted.

```sql
CREATE TABLE t (c JSONB);

INSERT INTO t VALUES ('{"c":3, "b":2, "a":1}');

INSERT INTO t VALUES ('{"a":');
-- ERROR:  invalid input syntax for type jsonb: {"a":
-- LINE 1: INSERT INTO t VALUES ('{"a":');
                               ^
SELECT * FROM t;
-- {"a":1,"b":2,"c":3}
```

Comparisons

Comparisons on JSONB expressions operate on the JSON representation of the JSONB. Since JSONB reorders the keys into a canonical form, two JSONB objects with the same key/value pairs in different order are equal. The C collation is used for ordering.

```sql
SELECT '{"a":1,"b":2}'::JSONB = '{"b":2,"a":1}'::JSONB;
--  t

SELECT '{"a":1,"b":2}'::JSONB < '{"b":2,"a":1}'::JSONB;
--  f
```

Limits

JSONB values cannot exceed 64 KB in their binary form. This limit can be hit even when the original JSON document is under 64 KB, because the binary format contains additional metadata that is used to speed up lookups.

```sql
SELECT (REPEAT('9', 63993))::JSONB;
--  ERROR:  Jsonb cast function output is longer than the maximum of 64000 bytes
```

JSONB values also have a maximum depth limit of 250. The depth of primitives is 0, and the depth of arrays and objects is 1 plus the maximum depth of any of their values.

```sql
SELECT '
{
  "level1": {
    "level2": {
      "level3": {
        ...
        "level249": {
          "level250": "deep value"
        }
      }
    }
  }
}
'::JSONB;
--  OK

SELECT '
{
  "level1": {
    "level2": {
      "level3": {
        ...
        "level250": {
          "level251": "deep value"
        }
      }
    }
  }
}
'::JSONB;
--  ERROR:  jsonb exceeded maximum nested depth 250
```

FLATTEN

`FLATTEN` is a Set Returning Function.

It returns a JSONB object with the fields "index", "key" and "value" for each element contained in the input JSONB.

```sql
FLATTEN(jsonb_expression)
```

jsonb_expression

The JSONB value to flatten.

FLATTEN behavior

Objects

If the input JSONB is an object, `FLATTEN` returns a JSONB for each key-value pair in the object. Each JSONB has

- index: `null`
- key: the key of the key-value pair
- value: the value of the key-value pair

```sql
SELECT * FROM FLATTEN('{"name":"Barnie","type":"Dog","big":true}');
```

```sql
                  flatten
--------------------------------------------
 {"index":null,"key":"big","value":true}
 {"index":null,"key":"name","value":"Barnie"}
 {"index":null,"key":"type","value":"Dog"}
(3 rows)
```

Arrays

If the input is an array, `FLATTEN` returns a JSONB for each element of the array. Each JSONB has

- index: the index of the element
- key: `null`
- value: the value of the element

```sql
SELECT * FROM FLATTEN('[7,[],1]');
```

```sql
             flatten
---------------------------------
```

```
 {"index":0,"key":null,"value":7}
 {"index":1,"key":null,"value":[]}
 {"index":2,"key":null,"value":1}
(3 rows)
```

Primitives

If the input is a primitive type (string, number, boolean, or null), `FLATTEN` returns a single JSONB that has

- index: `null`
- key: `null`
- value: the primitive itself

```sql
SELECT * FROM FLATTEN('3');
--  {"index":null,"key":null,"value":3}
```

Syntactic Sugar

There is an alternate syntax for `FLATTEN` that accepts additional parameters in the form `<jsonpath> AS <column name>`.

This applies `<jsonpath>` to each row that would have been returned by the single argument version of `FLATTEN`, and returns the value obtained after applying the jsonpath as `<column name>`.

```sql
SELECT * FROM FLATTEN('[7,[],1]', $.index AS i, $.value AS v) AS f;
```

```sql
 i | v
---+----
 0 | 7
 1 | []
 2 | 1
(3 rows)
```

The `AS <alias>` part is mandatory when using the multi argument syntax for `FLATTEN`.

JSON_OBJECT

`JSON_OBJECT` is an alias for `FLATTEN`.

Handling empty JSONB errors

Breaking behavior changes were made between 7.0 and 7.1. Among these changes is that the JSONB created from an empty string is no longer valid

```sql
SELECT ''::JSONB;
-- ERROR:  invalid input syntax for type jsonb:
-- LINE 1: SELECT ''::JSONB;
```

This is a potential problem for upgrades from versions that support this behavior to ones that do not.

There is a preupgrade check for this when upgrading from 7.0. You can hit an error regarding empty jsonbs during the preupgrade check. You can also hit the error after the upgrade tp 7.1 if the preupgrade check was not run before the upgrade.

In the sections below, we describe how to handle each of these scenarios. We demonstrate this on a table created on a 7.0 version of Yellowbrick like this

```sql
CREATE TABLE t AS (SELECT 0::INT AS i, ''::JSONB AS jb);
```

Empty JSONB error during pre-upgrade

If the pre-upgrade check for empty JSONB values fails, you will see a message like this in the preupgrader logs:

```sql
WARNING: yellowbrick_test_utf8: public.t contains 1 rows with empty JSONBs
REMEDY: To view the empty JSONBs, run this statement in the database 'yellowbrick_test_utf8':
 SELECT rowunique, jb FROM public.t WHERE jb IS NOT NULL AND json_typeof(jb) IS NULL;

 See https://docs.yellowbrick.com/latest/ybd_sqlref/jsonb_handle_empty.html for more information
```

As shown in the remedy message, you can use the condition `jb IS NOT NULL AND json_typeof(jb) IS NULL` to detect empty JSONB values, and inspect affected rows or to update the values to something else, for example by running

```sql
BEGIN;

UPDATE t SET jb = NULL WHERE jb IS NOT NULL AND json_typeof(jb) IS NULL;

COMMIT;
```

Empty JSONB in stats error during pre-upgrade

Another check that can fail is the check for empty JSONB values in pg_statistic:

```
WARNING: yellowbrick_test_utf8: The following tables contain empty JSONB values in their stats
    public.t
 See https://docs.yellowbrick.com/latest/ybd_sqlref/jsonb_handle_empty.html for more information
```

This means that pg_statistic, a catalog table responsible for storing statistical data about table columns, such as the most common values for a particular column, contains rows with empty JSONB values.

To solve this problem, you first need to follow the steps from the previous section to ensure that you have removed all empty JSONB values from the affected tables.

When those values are deleted, the pg_statistic table may still contain empty JSONB values if no analyze operations took place. To ensure these values are gone, you should explicitly run ANALYZE on the tables that are listed in the warning message.

```sql
ANALYZE t;
```

Empty JSONB error after upgrade

If pre-upgrade was not run before upgrading, you can get into a situation where you have empty JSONB values in your tables that are no longer considered valid. Many operations on these values become invalid after an upgrade, for example:

```sql
SELECT * FROM t;
-- ERROR:  empty jsonb is no longer supported, see https://docs.yellowbrick.com/latest/ybd_sqlref/jsonb_handle_empty.html
```

The condition to check for empty jsonb values is `yb_is_empty_jsonb(jb)`. Other columns in the affected rows of the table can be viewed by running:

```sql
SELECT i FROM t WHERE yb_is_empty_jsonb(jb);
--  0
```

To replace the empty JSONB values with some other value that is valid, use an update statement:

```sql
BEGIN;

UPDATE t SET jb = NULL WHERE yb_is_empty_jsonb(jb);

COMMIT;
```

JSON Accessor

The : operator can be used to path into a JSONB expression and extract values out of it. The syntax is `<jsonb_expr> : <jsonpath_expr> <error handling mode>`. The error handling mode is optional. The return type is JSONB.

JSONPath expressions

JSONPath expressions can only appear after the `:` operator.

Root Identifier

A JSONPath must always start with `$`, which refers to the root node of the input JSONB.

```sql
SELECT '{"name":"Slovakia","currency":"€"}' : $;
--  {"currency":"€","name":"Slovakia"}
```

Name Selector

A name selector selects a value from an object with the matching key. There are three ways to use a name selector.

1. `<jsonpath_expr>.name` - dot notation with an identifier
2. `<jsonpath_expr>.'name'` - dot notation with a string constant
3. `<jsonpath_expr>['name']` - bracket notation with a string constant

```sql
WITH cte AS (SELECT '{"k":"v"}'::JSONB AS jsonb)
SELECT jsonb : $.k, jsonb : $.'k', jsonb : $['k']
FROM cte;
--  "v" | "v" | "v"
```

The first way is the most concise but also the least powerful, because some names cannot be used in the grammar in that context.

```sql
SELECT '{"1":"one"}' : $.1;
-- ERROR:  syntax error at or near ".1"
-- LINE 1: SELECT '{"1":"one"}' : $.1;
                                ^
SELECT '{"1":"one"}' : $.'1';
--  "one"
```

When using dot notation with an identifier, the identifier is case sensitive:

```sql
SELECT '{"Age":25}' : $.Age;
--  25
```

Index Selector

An index selector selects a value from an array at the given index. The first element has index 0.

```sql
SELECT '[3,1,4]' : $[2];
--  4
```

Error Handling Modes

Two different behaviors are available for when a JSONPath is not able to find a value. The two behaviors are

1. Return NULL
2. Error

The default behavior can be modified by setting the json_missing_element_default configuration parameter:

```sql
SET json_missing_element_default TO 'null';

SELECT '{}' : $.k;
-- <NULL>

SET json_missing_element_default TO 'error';

SELECT '{}' : $.k;
-- ERROR:  No key found at path: 'strict $."k"' for '{}'
```

The behavior can also be explicitly set after the JSONPath expression, which has priority over the value set by the configuration parameter

```sql
SELECT '{}' : $.k NULL ON ERROR;
-- <NULL>

SELECT '{}' : $.k ERROR ON ERROR;
-- ERROR:  No key found at path: 'strict $."k"' for '{}'
```

JSON_TYPEOF

Returns the type of the root JSON element as a VARCHAR string. Possible types are `object`, `array`, `string`, `number`, `boolean`, and `null`.

```sql
JSON_TYPEOF(jsonb_expression)
```

jsonb_expression

The JSONB value to inspect.

Examples

```sql
SELECT json_typeof('{}');
-- object

SELECT json_typeof('3.14');
 json_typeof
-- number

SELECT json_typeof('"Alice"');
 json_typeof
--  string

SELECT json_typeof('null');
-- null

SELECT json_typeof(NULL);
-- <NULL>
```

JSONB Casting

JSONB can be cast to many different types. These casts can be separated into groups that share some similarities.

JSON Cast

Casting a JSONB expression to JSON gives back the JSON representation for that JSONB. The output is always valid JSON, or NULL if the JSON expression given was NULL.

```sql
SELECT '{"b":2,"a":1}'::JSONB::JSON;
-- {"a":1,"b":2}

SELECT '"string"'::JSONB::JSON;
--  "string"

SELECT 'null'::JSONB::JSON;
--  null

SELECT NULL::JSONB::JSON;
-- <NULL>
```

Extraction Casts

Extraction casts are intended for getting values out from leaf JSON values such as strings, numbers and booleans. The casts in this group are the `SMALLINT`, `INTEGER`, `BIGINT`, `DECIMAL`, `REAL`, `DOUBLE PRECISION`, `VARCHAR` and `BOOLEAN` casts. These casts:

- Return NULL if the input JSONB is a SQL NULL, or if it is a leaf JSON null value.
- Return the JSON representation of the JSONB, except:
  - Strip the outer double quotes if the JSONB expression is a string.
  - Unescape any characters that previously had to be escaped if the JSONB expression is a string.

```sql
SELECT '{"b":2,"a":1}'::JSONB::VARCHAR;
--  {"a":1,"b":2}

SELECT '"string"'::JSONB::VARCHAR;
-- string

SELECT 'null'::JSONB::BOOLEAN;
-- <NULL>

SELECT NULL::JSONB::REAL;
-- <NULL>

SELECT '"84"'::JSONB::DECIMAL(4, 2);
--  84.00
```

BYTEA Cast

Casting to BYTEA returns the internal binary representation of the JSONB, or NULL if the expression is a SQL NULL value.

```sql
SELECT '"84"'::JSONB::BYTEA;
-- \x0100000025000000038340000
```

There is no cast from `BYTEA` back to `JSONB`.

SQL String Constants

Yellowbrick Documentation > Reference > SQL Reference > SQL Data Types > SQL String Constants

Platforms: All platforms

Parent topic: SQL Data Types

In this section:

- C-style Escape Sequences in String Constants
- Escape Sequences for Non-Printable Characters
- Unicode Escape Sequences in String Constants

This section describes string constants in SQL, also known as character literals. This section also shows how to escape and include both printable and non-printable characters within string constants. Yellowbrick supports C-style and Unicode escape sequences for literal text within SQL statements.

A SQL string constant is any sequence of characters enclosed by single quotes ('). For example:

```
'We are such stuff as dreams are made on'
'Man Utd'
'06172019'
```

To identify a single-quote character inside a string constant, use two adjacent single quotes. For example:

```
'A Midsummer Night''s Dream'
'St. Mary''s Stadium'
```

Two adjacent single quotes have a distinct meaning from the double-quote character (`"`).

Although it is typically only used to enclose the text of procedures that contain embedded quotes, dollar quoting is also allowed in literal strings. For example:

```
premdb=# select $$St. Mary's Stadium$$ as dollar_quoted_string from sys.const;
 dollar_quoted_string
----------------------
 St. Mary's Stadium
(1 row)
```

C-style Escape Sequences in String Constants

A string constant that has an `E` or `e` directly before the opening single quotation mark may contain escape sequences. For example, you can type:

```
e'any string'
E'any string'
```

Within a string that starts with `E` or `e`, type a backslash character (`\`) to begin a C-style escape sequence. What follows can be one of a limited set of alphanumeric characters, an octal hex, or a unicode sequence.

The following example uses several backslash-character sequences. The result is a string constant that contains multiple tab characters and new lines:

```
premdb=# select E'\t'||E'\t'||E'\t'||'We are such stuff'||E'\n'||'As dreams are made on; and our little life'||E'\n'||'Is rounded
as verse from sys.const;
                 verse
-------------------------------------------
                        We are such stuff +
 As dreams are made on; and our little life+
 Is rounded with a sleep.
(1 row)
```

Note: The `+` characters in this example are just artifacts of how `ybsql` denotes newlines. They are not literal text inserted into the generated SQL string.

The specific escape sequences you can use are shown in the following table.

Backslash Escape Sequence	Interpretation
`\b`	Backspace
`\f`	Form feed
`\n`	Newline
`\r`	Carriage return
`\t`	Tab
`\o` , `\oo` , `\ooo`	Octal byte value (`o` = 0 to 7)
`\xh` , `\xhh`	Hexadecimal byte value (`h` = 0 to 9, A to F)
`\uxxx` , `\Uxxxxxxxx`	16-bit or 32-bit hexadecimal Unicode character value (`x` = 0 to 9, A to F)

Note: This list of escape sequences overlaps with but is not the same as the escape sequences used with `ybtools`. For example, `\us` may be used as the field delimiter in `ybload`, but it is an invalid character sequence in a SQL string.

Any other character following a backslash is taken literally. For example, you can embed a single quotation mark in a string by using either double single quotes or a backslash. An embedded backslash must also be preceded with a backslash. To create the literal string `\1's`, you can use either of the following sequences:

```
premdb=# select
 with_two_single_quotes | with_backslash_single_quote
------------------------+------------------------------
 \1's                   | \1's
(1 row)
```

Escape Sequences for Non-Printable Characters

Platforms: All platforms

Parent topic: SQL String Constants

The following table shows valid escape sequences for commonly used non-printable characters.

Dec	Backslash Char	Backslash Octal	Hex	Backslash SQL Unicode (4) **	Backslash SQL Unicode (6) **	ybload Unicode (2,4,6,8)***
0	\0 *	\000	\x00	\u0000	\U000000	U+0000
1		\001	\x01	\u0001	\U000001	U+0001
2		\002	\x02	\u0002	\U000002	U+0002
3		\003	\x03	\u0003	\U000003	U+0003
4		\004	\x04	\u0004	\U000004	U+0004
5		\005	\x05	\u0005	\U000005	U+0005
6		\006	\x06	\u0006	\U000006	U+0006
7		\007	\x07	\u0007	\U000007	U+0007
8	\b	\010	\x08	\u0008	\U000008	U+0008
9	\t	\011	\x09	\u0009	\U000009	U+0009
10	\n	\012	\x0A	\u000A	\U00000A	U+000A
11		\013	\x0B	\u000B	\U00000B	U+000B
12	\f	\014	\x0C	\u000C	\U00000C	U+000C
13	\r & \m	\015	\x0D	\u000D	\U00000D	U+000D
14		\016	\x0E	\u000E	\U00000E	U+000E
15		\017	\x0F	\u000F	\U00000F	U+000F
16		\020	\x10	\u0010	\U000010	U+0010
17		\021	\x11	\u0011	\U000011	U+0011
18		\022	\x12	\u0012	\U000012	U+0012

Dec	Backslash Char	Backslash Octal	Hex	Backslash SQL Unicode (4) **	Backslash SQL Unicode (6) **	ybload Unicode (2,4,6,8)***
19		\023	\x13	\u0013	\U000013	U+0013
20		\024	\x14	\u0014	\U000014	U+0014
21		\025	\x15	\u0015	\U000015	U+0015
22		\026	\x16	\u0016	\U000016	U+0016
23		\027	\x17	\u0017	\U000017	U+0017
24		\030	\x18	\u0018	\U000018	U+0018
25		\031	\x19	\u0019	\U000019	U+0019
26		\032	\x1A	\u001A	\U00001A	U+001A
27		\033	\x1B	\u001B	\U00001B	U+001B
28		\034	\x1C	\u001C	\U00001C	U+001C
29		\035	\x1D	\u001D	\U00001D	U+001D
30	\rs*	\036	\x1E	\u001E	\U00001E	U+001E
31	\us*	\037	\x1F	\u001F	\U00001F	U+001F
127		\177	\x7F	\u007F	\U00007F	U+007F

Notes:

- * These sequences are for `ybload` and `ybunload` only.
- ** These sequences are separate from the `U&'\nnnn` and `U&\+nnnnnn` escape sequences.
- *** These sequences are for `ybload` and `ybunload` only. They can be 2, 4, 6, or 8 hex digits wide.

Unicode Escape Sequences in String Constants

A Unicode escape string constant contains `U&` or `u&` directly before the opening quote, without any spaces. For example:

```
u&'any string'
U&'any string'
```

Within the quotes, you can specify Unicode characters in escaped form by typing a backslash followed by either the four-digit hexadecimal code-point number or a plus sign followed by a six-digit hexadecimal code-point number. For example:

```
premdb=# select U&'a\0062c\+000064' from sys.const;
 ?column?
----------
 abcd
(1 row)
```

To use a different escape character instead of a backslash, specify it in a `UESCAPE` clause after the string. For example:

```
premdb=# select U&'a$0062c$+000064' UESCAPE '$' from sys.const;
 ?column?
----------
 abcd
(1 row)
```

The escape character can be any single character except for a hexadecimal digit, the plus sign, a single quotation mark, a double quotation mark, or a whitespace character.

To include the escape character in the string literally, type it twice.

BIGINT

The BIGINT data type stores 8-byte integer values.

- The accepted range of values is from -9223372036854775808 to +9223372036854775807.
- The INT8 keyword is a synonym for BIGINT.
- Leading zeros and whitespace characters are allowed. Trailing whitespace characters are also allowed.

BINARY

A binary string is a sequence of octets (or bytes). The binary strings specifically allow storing octets of value zero and other “non-printable” octets (usually, octets outside the decimal range 32 to 126). The operations on binary strings process the actual bytes. The binary data type is used to support the storage and manipulation of binary data. This forms the foundation for the `JSONB` data type.

The binary data type has several aliases:

- `BYTEA` : `BYTEA` stands for “binary array”. The `BYTEA` type does not have a length indicator but is currently limited to the same length as the maximum `VARCHAR` .

- `VARBINARY` : `VARBINARY` or `VARBINARY(n)` takes a length parameter in bytes.

- `VARBYTE` : `VARBYTE` or `VARBYTE(n)` takes a length parameter in bytes.

Note: These type aliases all map to the same underlying `BYTEA` type.

Prerequisite

To use a binary data type, enable the `enable_full_bytea` configuration parameter.

Input BYTEA values

This section explains how to input `BYTEA` values using the following formats supported by Yellowbrick:

- hex
- escape
- converting the characters into their byte representation depending on the encoding

bytea “hex” Format

The “hex” format encodes binary data as two hexadecimal digits per byte, most significant nibble first. The entire string is preceded by the sequence `\x` (to distinguish it from the escape format). For input, the hexadecimal digits can be either uppercase or lowercase, and whitespace is permitted between digit pairs (but not within a digit pair nor in the starting `\x` sequence).

```sql
CREATE TABLE bytea_table (
    data BYTEA
);

-- Insert a row with a bytea value in hexadecimal format
INSERT INTO bytea_table (data) VALUES ('\x48656C6C6F2C20776F726C6421');

-- Select and display the inserted bytea value
SELECT * FROM bytea_table;
-- \x48656c6c6f2c20776f726c6421
```

bytea “escape” Format

The “escape” format represents a binary string using ASCII characters and converts non-ASCII bytes into special escape sequences. When entering bytea values in escape format, octets of certain values must be escaped, while all octet values can be escaped. In general, to escape an octet, convert it into its three-digit octal value and precede it by a backslash. Backslash itself (octet decimal value 92) can alternatively be represented by double backslashes.

```sql
CREATE TABLE bytea_table (
    data BYTEA
);

-- Insert a row with a bytea value in escape format
INSERT INTO bytea_table (data) VALUES (E'Hell\157 wo\162ld!');

SET bytea_output TO escape;

-- Select and display the inserted bytea value
SELECT * FROM bytea_table;
-- Hello world!
```

Converting the characters into their byte representation depending on the encoding

```sql
CREATE TABLE bytea_table (
    data BYTEA
);

-- Insert a row with a bytea value
INSERT INTO bytea_table (data) VALUES ('Hello,world!');

-- Select and display the inserted bytea value.
SELECT * FROM bytea_table;
-- \x48656c6c6f2c776f726c6421
```

Examples of bytea_output configuration parameter

Set bytea_output to “hex” format

```sql
SET bytea_output = 'hex';

SELECT '\xDEADBEEF'::bytea;
-- \xdeadbeef
```

This will set the output format of `bytea` values to hexadecimal. When you query a `bytea` column, it will be displayed in hexadecimal format.

Set bytea_output to “escape” format

In “escape” format, `bytea` values are displayed as a series of octal numbers prefixed with `\` and escaped characters are displayed as they are.

```sql
SET bytea_output = 'escape';

SELECT 'abc \153\154\155 \052\251\124'::bytea;
-- abc klm *\251T
```

Note: Changing this setting affects the display format of `bytea` values in query results but does not affect the actual storage format in the database.

Operators

The following table shows the operators that are supported:

Operator	Description
&	Bitwise And
|	Bitwise Or
# or ^	Bitwise Xor
~	Bitwise Not
<<	Shift Left
>>	Logical Shift Right
||	Concat

Casting Rules

`BYTEA` does not support length type modifiers, it can only be shortened by calling the `SUBSTRING` function. For type casting, if `BYTEA` cannot be directly cast to a specified type, an error will occur as described in the sections below:

```sql
[42846] ERROR: cannot cast type bytea to <target type>
```

Also, attempting to cast from a `<source type>` to a `BYTEA` results in the following error:

```sql
[42846] ERROR: cannot cast type <source type> to bytea
```

To know how implicit casting is handled, please refer Implicit Casting.

BYTEA and UUID

Casting to and from the UUID type leads to an error. The recommended solution is to perform the casting through `VARCHAR`.

BYTEA and FLOAT4/FLOAT8

Casting to and from `FLOAT4` / `FLOAT8` types results in an error. The solution to this issue is to perform the casting through `VARCHAR`.

BYTEA and DECIMAL/NUMERIC

Casting to and from the `DECIMAL` type causes an error. You can avoid this by casting through `VARCHAR`. Alternatively, you can use the `GET_BYTE` function to extract each byte as an integer.

BYTEA and VARCHAR/TEXT

Casting `VARCHAR` / `TEXT` to `BYTEA` can take the following notations as input:

- Hex specified as `[0\\]x[a-fA-F0-9]`.
- Binary specified as `B'[01]+`.
- Octal Notation `E'\\\\000\\\\001\\\\002'`.

- String values, these are interpreted as the byte sequence of the string (see below).

String byte sequences

We require that strings in non-hex, binary, or octal prefixes can still be cast to `BYTEA`. The string value is interpreted as the `UTF8` byte representation of the `VARCHAR`.

Multibyte UTF8 sequence

```sql
SELECT 'и'::bytea::VARCHAR;
 varchar
---------
\xe0b897
```

Which is the `UTF8` binary representation of the "и" character. **Note:** Ensure that this process is run against a `UTF8` encoded database for proper handling of text. Additionally, be aware that SQL clients may interpret encodings differently, which can sometimes result in variations in the displayed output.

When casting a `BYTEA` to a `VARCHAR`, the output is the hex string representation of the contents of the `BYTEA`. This means that it is possible to overflow the max size of `VARCHAR` if you have a sufficiently long `BYTEA`. This results in the following error:

```sql
[220BF] - Overflow when casting '<context>' to VARCHAR
```

With `<context>` being the first 20 bytes of the `BYTEA`. If more than 20 bytes are present, an ellipsis is added.

BYTEA and CHAR

Casting directly to `CHAR` would lead to an error.

BYTEA and IP address/MAC data types

Casting to or from `BYTEA` and `MAC` / `IP` types results in an error. The workaround, again, is to use `GET_BYTE` and manually form a string.

Functions

The following functions are supported.

Note that the `Get_` style functions return an `INTEGER` type, not a smaller type.

Function	Description
`GET_BIT (BYTEA x, INT n) -> INT`	Returns the n^{th} bit in x
`GET_BYTE(BYTEA x, INT n) -> INT`	Returns the n^{th} byte
`LENGTH(BYTEA X)`	Returns length (in bytes) of the binary (i.e. the size of the value in the type)
`BIT_COUNT(BYTEA X)`	Returns the number of set bits in x
`SET_BIT(BYTEA x, INT n, INT v)`	Set the n^{th} bit to v
`SET_BYTE(BYTEA x, INT n, INT v)`	Set the n^{th} byte to v
`SUBSTR(BYTEA x, INT start [, INT count]) -> BYTEA`	Returns the byte as start offset up until count (default 1)

Function	Description
`ENCODE(BYTEA x, VARCHAR format) -> VARCHAR`	Formats x into a VARCHAR string which uses format 'encoding.' We will support base64, escape, and hex formats
`DECODE(VARCHAR x, VARCHAR format) -> BYTEA`	The reverse of encode. Turns x into a BYTEA(n) using format as encoding
`SHA1(BYTEA) -> BYTEA`	Calculate SHA1 and returns a BYTEA
`SHA256(BYTEA) -> BYTEA`	Calculates SHA256 and returns a BYTEA
`SHA512(BYTEA) -> BYTEA`	Calculates SHA512 and returns a BYTEA
`MD5(BYTEA) -> VARCHAR(32)`	Calculates MD5 and returns a hex respresentation as a VARCHAR. This is done for compatibility reasons

Aggregating and Ordering

`BYTEA` can be grouped and window aggregated. The only valid aggregate functions on `BYTEA` are `MIN`, `MAX` and `COUNT` (including `COUNT DISTINCT`).

`BYTEA` are ordered by their byte order (the same as `LATIN9`).

Joining and Equality

Two `BYTEA` types can only be equal if their lengths are the same and their byte patterns are the same. This means you can join on `BYTEA` and perform equality match by doing simple `memcmp` after length checks.

Distribution and Partitioning

The `BYTEA` type cannot be used for partitioning and distribution defined in the Data definition language (DDL). Attempting to do so results in the following error:

```sql
[42603] The column '<column name>' cannot be used as a {distribution | partition} key because it is of the BYTEA type.
```

BOOLEAN

The BOOLEAN data type stores 1-byte values that evaluate to `true`, `false`, or `unknown`. The actual values that you can insert into a BOOLEAN column (or use as Boolean literals) are as follows:

Accepted Values	Meaning	Return Value
`true`, `t`, `y`, `yes`, `1`	True	t
`false`, `f`, `n`, `no`, `0`	False	f
`null`	Unknown	null

Uppercase and lowercase values are accepted: `TRUE` or `true`, for example. Single quotes are required for all values except `true`, `false`, and `null`.

CHAR

The `CHAR` data type is *conceptually* a fixed-length character string that is padded with trailing blanks.

Trailing blanks are removed from strings in `CHAR` columns when the data is stored. Blanks are restored on output to the client. A `CHAR(5)` column, for example, returns 5 bytes to the client but may store fewer than 5 bytes in the database.

All `CHAR`-related functions and operators work on a compact (trimmed) representation of the input data. A `CHAR` string behaves exactly like a `VARCHAR` string without the trailing spaces. For example, `'abc '::char(10)` is exactly equivalent to `'abc'::varchar(10)`.

For more details and examples, see Trailing Blanks in Character Data.

An attempt to load a string that is longer than the specified length of a `CHAR` column results in a `value too long` error.

In terms of the data type hierarchy for character strings, `VARCHAR` has precedence over `CHAR`, and `CHAR` has precedence over `UNKNOWN`.

A `CHAR` column without a length specification defaults to a 1-byte column.

The maximum length of a `CHAR` column is 64000 bytes. The maximum length of a row in a table is 64231 bytes. See Maximum Row Size.

DATE

The DATE data type stores 4-byte calendar date values without any timestamp information.

The range of valid dates is from `01-01-0001` to `12-31-9999`. The default format for returning date values to the client is `YYYY-MM-DD`.

```
premdb=# create table dates(c1 date);
CREATE TABLE
premdb=# insert into dates values('01-01-0001');
INSERT 0 1
premdb=# insert into dates values('12-31-9999');
INSERT 0 1
premdb=# select * from dates;
    c1
------------
 0001-01-01
 9999-12-31
(2 rows)
```

See also Formats for Datetime Values.

You can load DATE columns with date values that contain timestamp information, but only the calendar date is stored and returned in query output. For example:

```
premdb=# create table datetest(c1 date);
CREATE TABLE
premdb=# insert into datetest values('2016-05-10 18:20:18.674 PDT');
INSERT 0 1
premdb=# select * from datetest;
    c1
------------
 2016-05-10
(1 row)
```

DOUBLE PRECISION

Yellowbrick Documentation > Reference > SQL Reference > SQL Data Types > DOUBLE PRECISION

Platforms: All platforms

Parent topic: SQL Data Types

REAL and DOUBLE PRECISION are floating-point data types. These data types store numeric values with variable precision. Some values may be stored as approximations; you may see slight discrepancies between a value that is loaded and the value that is returned. The DECIMAL data type provides more exact results.

The DOUBLE PRECISION data type stores 64-bit floating-point values (8 bytes). The precision of a DOUBLE PRECISION column is 15 digits.

FLOAT and FLOAT8 and are valid synonyms for DOUBLE PRECISION.

Note: `FLOAT8` and `FLOAT(8)` are not synonyms. Yellowbrick permits the `float(p)` syntax but will promote it to the next floating-point number into which it will fit. This behavior is different from PostgreSQL.

Leading zeros and whitespace characters are allowed. Trailing whitespace characters are also allowed.

Note: Yellowbrick supports `NaN` (not a number) and `Infinity` *as floating-point values*. The system can store and operate on these values.

- Conversion of `NaN`, `Infinity`, `+inf`, and `-inf` to string types produces the appropriate string.
- DECIMAL and INTEGER types do not support these values for storage or conversion.
- When `NaN` is compared to any non-NULL value, the result of the comparison is always `false`.
- When an ORDER BY clause sorts `NaN` with other values, `NaN` always sorts as greater than any non-NULL value.

The following example demonstrates the values that you can load into a DOUBLE PRECISION column. Note that floating-point values with excess digits on the right side of the decimal point are rounded. Values with excess digits on the left side of the decimal point are converted to scientific notation; these values are not rejected.

```
premdb=# create table floattest(c1 float);
CREATE TABLE
premdb=# \d floattest
      Table "public.floattest"
 Column |       Type       | Modifiers
--------+------------------+-----------
 c1     | double precision |

Distribution: Hash (c1)

premdb=# insert into floattest values(1000000000000.123);
INSERT 0 1

premdb=# select * from floattest;
       c1
------------------
 1000000000000.12
(1 row)

premdb=# insert into floattest values(1000000000000000);
INSERT 0 1
premdb=# select * from floattest;
       c1
------------------
 1000000000000.12
           1e+15
(2 rows)
```

extra_float_digits

The `extra_float_digits` parameter controls the number of extra significant digits that are included when a floating-point value is converted to text for output. The default value is `0`. Increasing the number to `1` or greater produces output that more accurately represents the stored value.

For example, note the difference in query results against a `REAL` column when `extra_float_digits` is set to `3` compared to `0`:

```
premdb=# reset extra_float_digits;
RESET
premdb=# create table extra_float as select avg_att::real from team where avg_att>0;
SELECT 20
premdb=# select * from extra_float;
 avg_att
---------
  35.776
  20.594
  24.631
  34.91
  59.944
  33.69
  11.189
  ...
(20 rows)

premdb=# set extra_float_digits to 3;
SET
premdb=# create table extra_float_3 as select avg_att::real from team where avg_att>0;
SELECT 20
premdb=# select * from extra_float_3;
  avg_att
------------
 35.776001
 20.5939999
 24.6310005
 34.9099998
 59.9440002
 33.6899986
 11.1890001
 ...
(20 rows)
```

GEOGRAPHY

Yellowbrick Documentation > Reference > SQL Reference > SQL Data Types > GEOGRAPHY

Platforms: All platforms

Parent topic: SQL Data Types

> **BETA FEATUREPrerequisites:**
>
> - To use the `GEOGRAPHY` type and its associated functions, you must enable the feature by setting the `enable_geospatial` configuration parameter to `ON`.
>
> **Warning:** This is a beta feature that:
>
> - Is subject to change
> - May cause system instability
> - May have incomplete functionality

The `GEOGRAPHY` data type stores geospatial data representing points, lines, and polygons on the Earth's surface. It uses a geodetic coordinate system (latitude/longitude) with SRID 4326 (WGS84). Calculations may approximate or fully account for the Earth's curvature, depending on whether the spherical or spheroidal model is used.

Spherical vs. Spheroidal Earth Models

When using `GEOGRAPHY` functions, it's important to understand which Earth model they employ. The **spherical model** assumes the Earth is a perfect sphere, which can lead to distance calculation errors of up to 0.3%, particularly over distances greater than 100 kilometers. This occurs because the Earth is actually a **spheroid**, not a perfect sphere. Functions like `ST_Distance` and `ST_Length` allow you to choose between models using the `use_spheroid` parameter. Other functions without a similar parameter use the spherical model unless stated otherwise.

Functions Using Planar Projection

Some functions are not working natively in spherical coordinates, but instead cast to planar geometry. As a result, they may fail to return correct results for objects with very large extents that cannot be cleanly converted to a planar representation.

For example, the `ST_Buffer(geography,distance)` function transforms the geography object into a “best” planar projection, buffers it, and then transforms it back to spherical geography. If there is no “best” projection (the object is too large), the operation can fail or return a malformed buffer.

The following functions use planar projection:

- `ST_Buffer`
- `ST_Intersection`
- `ST_Difference`
- `ST_Union`
- `ST_SimplifyPreserveTopology`
- `ST_Touches`
- `ST_Overlaps`
- `ST_Makevalid`
- `ST_IsValid`
- `ST_IsValidReason`

- `ST_Crosses`
- `ST_ReducePrecision`

Input Formats

Geography data can be input using two standard formats:

Well-Known Text (WKT)

WKT is a text-based format representing geographic shapes. Coordinates must be specified as longitude/latitude pairs.

Well-Known Binary (WKB)

WKB is a binary representation of geographic data that can be input using hexadecimal strings prefixed with `\x`. Refer to `st_asbinary` for more details about casting and conversion functions.

Examples

```sql
CREATE TABLE locations (
    id INTEGER,
    location `GEOGRAPHY`
);

-- Insert using WKT format
INSERT INTO locations VALUES
    (1, 'POINT(-122.27652 37.80574)');

-- Insert using WKB format
INSERT INTO locations VALUES
    (2, '\x0101000000282cf180b2915ec01e8a027d22e74240'::VARBINARY);

-- Both formats produce the same `GEOGRAPHY` object
SELECT * FROM locations;
--  id |          location
-- ----+---------------------------
--   1 | POINT(-122.27652 37.80574)
--   2 | POINT(-122.27652 37.80574)

-- Query distance between two points (in meters)
SELECT ST_Distance(
    'POINT(-122.27652 37.80574)'::GEOGRAPHY,
    'POINT(-122.27645 37.80577)'::GEOGRAPHY
);
-- 7.0062797239034
```

Known Limitations

1. **SRID Support:**
   - The only supported SRID is 4326 (WGS84).
2. **Type Support:**
   - Only the `GEOGRAPHY` type is supported.
   - The `GEOMETRY` type is not available.

3. **Size Limitations:**
   - The maximum object size is 64KB.

Known Issues

1. **Invalid Geometry Behavior:**
   - Current behavior for invalid geometries may change in future releases.
2. `ST_Union` **Non-Determinism**
   - Multiple calls to `ST_Union` can produce different but equivalent geography collections.
   - `ST_Union` might fail with `ERROR: lwgeom_from_ybgserialized: Expected size (...) did not match result size (...)` when aggregating a large collection of disjoined points.
3. **Aggregation Instability**
   - Aggregations on `GEOGRAPHY` (e.g., `GROUP BY`, `DISTINCT`) may produce inconsistent results for large datasets.

INTEGER

The INTEGER data type stores 4-byte integer values.

- The accepted range of values is from -2147483648 to +2147483647.
- The INT and INT4 keywords are synonyms for INTEGER.
- Leading zeros and whitespace characters are allowed. Trailing whitespace characters are also allowed.

IPV4

The `IPV4` data type stores 4-byte IP addresses in the standard format:

```
x.x.x.x
```

where *x* is an *octet* with a value between `0` and `255`. An `IPV4` address always contains three periods and four octets. For example:

```
10.10.0.1
255.255.253.0
19.120.51.251
```

See also IPV6, which is a separate data type, and Network Address Functions. Some network address functions accept netmask values for processing; however, only single-host specifications are stored in `IPV4` columns, assuming a `/32` netmask. If a source data value contains the `/n` CIDR notation, this notation is ignored when the value is loaded. For example, `1.2.3.4/8` is stored as `1.2.3.4`

To explicitly identify a string as an `IPV4` literal value, use the `IPV4` keyword. For example:

```
where ip_address > ipv4 '192.168.1.6'
```

`IPV4` columns do not accept empty strings.

IPV6

The `IPV6` data type stores 16-byte IP addresses in the standard format:

```
y:y:y:y:y:y:y:y
```

where each *y* segment contains any hexadecimal value between `0` and `FFFF`. Uppercase or lowercase letters are accepted. The eight segments are separated by colons. For example:

```
684D:1111:222:3333:4444:5555:6:77
2001:db8:3333:4444:cccc:dddd:eeee:ffff
0:0:aaaa:bbbb:ffff:1:1:0
```

See also IPV4, which is a separate data type, and Network Address Functions. Some network address functions accept netmask values for processing; however, only single-host specifications are stored in `IPV6` columns, assuming a `/128` netmask.

To explicitly identify a string as an `IPV6` literal value, use the `IPV6` keyword. For example:

```
where ip_address > ipv6 '0:0:aaaa:bbbb:ffff:1:1:0'
```

`IPV6` columns accept empty strings.

MACADDR

The `MACADDR` data type stores 6-byte MAC addresses. (Note that fully uncompressed `MACADDR` values occupy 8 bytes of storage in the database.)

You can insert MAC addresses into table columns in several different formats. All of the following 12-character formats are valid, using colons, periods, or hyphens as separators (or no separator). Upper and lower case is accepted for the characters `a` through `f`. All of these input values would result in the same MAC address being stored:

```
'08:00:2b:01:02:03'
'08-00-2b-01-02-03'
'08002b:010203'
'08002b-010203'
'0800.2b01.0203'
'0800-2b01-0203'
'08002b010203'
```

Output is always returned in the first format: six pairs of digits or characters, separated by colons. For example:

```
premdb=# create table macaddrs(c1 macaddr);
CREATE TABLE
premdb=# \d macaddrs
   Table "public.macaddrs"
 Column |  Type   | Modifiers
--------+---------+-----------
 c1     | macaddr |

Distribution: Hash (c1)

premdb=# insert into macaddrs
values('08:00:2b:01:02:03'),
     ('08-00-2b-01-02-03'),
     ('08002b:010203'),
     ('08002b-010203'),
     ('0800.2b01.0203'),
     ('0800-2b01-0203'),
     ('08002b010203');
INSERT 0 7
premdb=# select * from macaddrs;
       c1
-------------------
 08:00:2b:01:02:03
 08:00:2b:01:02:03
 08:00:2b:01:02:03
 08:00:2b:01:02:03
 08:00:2b:01:02:03
 08:00:2b:01:02:03
 08:00:2b:01:02:03
(7 rows)
```

To explicitly identify a string as a `MACADDR` literal value, use the `MACADDR` keyword. For example:

```
where mac_address > macaddr '08:00:2b:01:02:03'
```

`MACADDR` columns do not accept empty strings.

You can implicitly or explicitly cast a `MACADDR` value to the `MACADDR8` type; 6-byte addresses are stored with the 4th and 5th bytes set to `FF` and `FE`, respectively. For example:

```
premdb=# select * from macaddrs;
       mac        |          mac8
------------------+-------------------------
 08:00:2b:01:02:03 | 08:00:2b:01:02:03:04:05
(1 row)

premdb=# select mac::macaddr8 from macaddrs;
          mac
-------------------------
 08:00:2b:ff:fe:01:02:03
(1 row)
```

See also MACADDR8 and Network Address Functions.

MACADDR8

Yellowbrick Documentation > Reference > SQL Reference > SQL Data Types > MACADDR8

Platforms: All platforms

Parent topic: SQL Data Types

The `MACADDR8` data type stores 8-byte (EUI-64 format) MAC addresses.

This data type accepts both 6-byte and 8-byte MAC addresses and stores them in 8-byte format. 6-byte addresses are stored with the 4th and 5th bytes set to `FF` and `FE`, respectively. All of the following 16-digit or 12-digit formats are valid, using colons, periods, or hyphens as separators (or no separator). Upper and lower case is accepted for the characters `a` through `f`. Leading and trailing whitespace is ignored.

All of these input values would result in the same `MACADDR8` value being stored:

```
'08:00:2b:01:02:03:04:05'
'08-00-2b-01-02-03-04-05'
'08002b:0102030405'
'08002b-0102030405'
'0800.2b01.0203.0405'
'0800-2b01-0203-0405'
'08002b01:02030405'
'08002b0102030405'
```

Output is always returned in the first format (8 pairs of digits separated by colons):

```
08:00:2b:01:02:03:04:05
```

`MACADDR8` columns do not accept empty strings.

The following example shows a `MACADDR` value being inserted into a `MACADDR8` column. The 4th and 5th bytes are set to `FF` and `FE`.

```
premdb=# insert into macaddr8s values('08:00:2b:01:02:03');
INSERT 0 1
premdb=# select * from macaddr8s;
          c1
------------------------
 08:00:2b:ff:fe:01:02:03
(1 row)
```

To explicitly identify a string as a `MACADDR8` literal value, use the `MACADDR8` keyword. For example:

```
where mac_address > macaddr8 '08:00:2b:01:02:03:04:05'
```

See also Network Address Functions. To convert a 48-bit MAC address in EUI-48 format to modified EUI-64 format (as the host portion of an `IPV6` address), use the MACADDR8_SET7BIT function.

REAL

REAL and FLOAT are floating-point data types. These data types store numeric values with variable precision. Some values may be stored as approximations; you may see slight discrepancies between a value that is loaded and the value that is returned. The DECIMAL data type provides more exact results.

The REAL data type stores 32-bit floating-point values (4 bytes). The precision of a REAL column is 6 digits.

FLOAT4 is a valid synonym for REAL.

Note: `FLOAT4` and `FLOAT(4)` are not synonyms. Yellowbrick permits the `float(p)` syntax but will promote it to the next floating-point number into which it will fit. This behavior is different from the PostgreSQL behavior.

Leading zeros and whitespace characters are allowed. Trailing whitespace characters are also allowed.

Note: Yellowbrick supports `NaN` (not a number) and `Infinity` *as floating-point values*. The system can store and operate on these values.

- Conversion of `NaN`, `Infinity`, `+inf`, and `-inf` to string types produces the appropriate string.
- DECIMAL and INTEGER types do not support these values for storage or conversion.
- When `NaN` is compared to a number, the result of the comparison is always `false`.
- When an ORDER BY clause sorts `NaN` with numbers, `NaN` always sorts as greater than those numbers.

The following example demonstrates the values that you can load into a REAL column. Note that floating-point values with excess digits on the right side of the decimal point are rounded. Values with excess digits on the left side of the decimal point are converted to scientific notation; these values are not rejected.

```
premdb=# create table realtest(c1 real);
CREATE TABLE
premdb=# \d realtest
  Table "public.realtest"
 Column | Type | Modifiers
--------+------+-----------
 c1     | real |

Distribution: Hash (c1)

premdb=# insert into realtest values(100.1238);
INSERT 0 1
premdb=# select * from realtest;
   c1
---------
 100.124
(1 row)

premdb=# insert into realtest values(100000);
INSERT 0 1
premdb=# select * from realtest;
   c1
---------
 100.124
  100000
(2 rows)
```

```
premdb=# insert into realtest values(1000000);
INSERT 0 1
premdb=# select * from realtest;
   c1
---------
 100.124
  100000
   1e+06
(3 rows)
```

You can insert `Nan` and `Infinity` values into a REAL or DOUBLE PRECISION column (and convert them to character strings):

```
premdb=# insert into realtest values('-inf');
INSERT 0 1
premdb=# insert into realtest values('+inf');
INSERT 0 1
premdb=# insert into realtest values('NaN');
INSERT 0 1
premdb=# select * from realtest order by c1;
   c1
-----------
 -Infinity
   100.124
   100000
    1e+06
  Infinity
       NaN
(6 rows)

premdb=# select c1::varchar from realtest order by c1;
   c1
-----------
 -Infinity
 100.124
 100000
 1e+06
 Infinity
 NaN
(6 rows)
```

---

extra_float_digits

The `extra_float_digits` configuration parameter controls the number of extra significant digits that are included when a floating-point value is converted to text for output.

SMALLINT

The SMALLINT data type stores 2-byte integer values.

- The accepted range of values is from -32768 to +32767.
- The INT2 keyword is a synonym for SMALLINT.
- Leading zeros and whitespace characters are allowed. Trailing whitespace characters are also allowed.

TIME

Yellowbrick Documentation > Reference > SQL Reference > SQL Data Types > TIME

Platforms: All platforms

Parent topic: SQL Data Types

The TIME data type stores 8-byte time of day values that do not include the date or time zone. If input values contain dates and time zones, they are ignored.

You cannot define a TIME column with a specific precision for fractional seconds. All TIME values support up to six fractional seconds; if additional fractional digits exist in input values, the time is rounded up or down, as appropriate.

Values inserted with INSERT statements must fall between `00:00:00` and `24:00:00`. Values inserted with `ybload` bulk loads must fall between `00:00:00` and `23:59:59`. Time values may also include the `am` or `pm` designation.

The following table shows examples of values that can be inserted into a TIME column.

Inserted Value	Stored Value	Notes
`01:01:01`	`01:01:01`	
`23:59:59.123`	`23:59:59.123`	Three digits representing fractional seconds are inserted.
`00:00:00.999999`	`00:00:00.999999`	Six digits representing fractional seconds are inserted.
`01:00:00.999999555`	`01:00:01.000000`	The fractional seconds (9 digits) are rounded up.
`2017-05-25 00:00:00.9999999 GMT`	`00:00:01`	The date is ignored. The time zone is ignored. The fractional seconds (7 digits) are rounded up.
`24:00:00.123`	Value out of range	Returns an error.
`02:02:45.6789pm`	`14:02:45.6789`	`am` and `pm` can be specified in uppercase or lowercase.
`02:02:45.6789 am`	`02:02:45.6789`	The space before `am` or `pm` is optional.

Note: You cannot add two TIME values. However, you can subtract a TIME value from another TIME value, which produces an INTERVAL value. For example:

```
premdb=# select time '12:00:00' - time '3:30:30' as timeleft
from sys.const;
 timeleft
----------
 08:29:30
(1 row)
```

TIMESTAMP

The TIMESTAMP data type stores 8-byte date values that include timestamp information in UTC format but no time zone information. You can load TIMESTAMP columns with time zone information, but the time zone is not stored or returned in query output.

You cannot define a TIMESTAMP column with a specific precision for fractional seconds other than 6. A valid timestamp includes up to 6 digits of precision. If additional fractional digits exist in input values, the timestamp is rounded up or down, as appropriate.

For example, the following CREATE TABLE statements are allowed and produce the same table:

```
premdb=# create table timestamps6(c1 timestamp(6));
CREATE TABLE
```

```
premdb=# create table timestamps(c1 timestamp);
CREATE TABLE
```

The following CREATE TABLE statement is disallowed:

```
premdb=# create table timestamps(c1 timestamp(4));
ERROR:  TIMESTAMP with precision 4 is not allowed, only 6 is supported
```

The following examples show the behavior when different values are inserted into a TIMESTAMP column. The range of valid dates in timestamps is from `01-01-0001` to `12-31-9999`.

Inserted Value	Stored Value	Notes
`2016-05-10 18:20:18.674 PDT`	`2016-05-10 18:20:18.674`	The `PDT` time zone is removed.
`2016-05-11 2:30`	`2016-05-11 02:30:00`	By default, `:00` is added for seconds. A leading zero is added for the hours (`02`).
`2016-05-11 2:31:59.99999`	`2016-05-11 02:31:59.99999`	Loaded as is, with 5 digits of precision for fractional seconds. A leading zero is added for the hours (`02`).
`2016-05-11 2:31:59.999999`	`2016-05-11 02:31:59.999999`	Loaded as is, with a maximum 6 digits of precision for fractional seconds. A leading zero is added for the hours (`02`).
`2016-05-11 2:31:59.9999999`	`2016-05-11 02:32:00`	Too many digits (7) for fractional seconds. The timestamp is rounded up.

TIMESTAMP WITH TIME ZONE

The TIMESTAMP WITH TIME ZONE (or TIMESTAMPTZ) data type stores 8-byte date values that include timestamp and time zone information in UTC format.

You cannot define a TIMESTAMPTZ column with a specific precision for fractional seconds other than 6. A valid timestamp with time zone value includes up to 6 digits of precision. If additional fractional digits exist in input values, the value is rounded up or down, as appropriate.

For example, the following CREATE TABLE statements are allowed and produce the same table:

```
premdb=# create table timestamptz6(c1 timestamptz(6));
CREATE TABLE
```

```
premdb=# create table timestamptz0(c1 timestamptz);
CREATE TABLE
```

The following CREATE TABLE statement is disallowed:

```
premdb=# create table timestamptz4(c1 timestamptz(4));
ERROR:  TIMESTAMPTZ with precision 4 is not allowed, only 6 is supported
```

The range of valid dates in timestamps is from `01-01-0001` to `12-31-9999`.

The following examples show the behavior when values are inserted or loaded into a TIMESTAMPTZ column, and the current time zone for the database session is `US/Pacific` (or `07`).

Inserted Value	Stored Value	Notes
`2016-05-10 18:20:18.674 UTC`	`2016-05-10 11:20:18.674-07`	The time zone identifier is returned as `07`. A 7-hour offset is required for the timestamp itself (`11:20:18.674`).
`2016-05-10 18:20:18.674 +1:00`	`2016-05-10 10:20:18.674-07`	The time zone identifier is returned as `07` (the time zone of the current session). The timestamp is converted to UTC `+1:00` (an 8-hour offset).

Note: The `ybload` utility supports UTC time zones, as shown in these examples, and IANA time zone names, such as `US/Pacific` and `Europe/Dublin`. Yellowbrick recommends that you export `timestamptz` data in UTC format before running a bulk load. If you use a SQL INSERT command to load data, you can also use other standard time zone abbreviations, such as `PST` and `PDT`; `ybload` does not support these abbreviations.

Trailing Blanks in Character Data

Yellowbrick Documentation > Reference > SQL Reference > SQL Data Types > Trailing Blanks in Character Data

Platforms: All platforms

Parent topic: SQL Data Types

This section explains how `CHAR` and `VARCHAR` columns handle trailing blanks (space characters) when character data is loaded, stored, and queried.

A `CHAR` data type is conceptually a fixed-length, blank-padded string. The rules for handling trailing blanks in fixed-length character strings are as follows:

- `CHAR` columns are stored with any trailing blanks trimmed. For example, values loaded into a `CHAR(10)` column as `'London'` and `'London '` are stored with an equal length of 6 characters. The `LENGTH` function for both values returns 6.
- A comparison of values such as `'London'` and `'London '` (one trailing blank) or `'London'` and `'London '` (two trailing blanks) returns a match.
- Trailing blanks in `CHAR` columns are restored on output to the client. For example, a `CHAR(10)` column returns 10 characters per column value.

A `VARCHAR` data type is a variable-length character string. The rules for processing variable-length strings are as follows:

- `VARCHAR` columns are stored and processed with any trailing blanks preserved. For example, values loaded into a `VARCHAR(10)` column as `'London'` and `'London '` are stored with lengths of 6 and 8 characters, respectively.
- A comparison of values such as `'London'` and `'London '` does not return a match.
- `VARCHAR` columns are returned to the client as variable-length strings. For example, a `VARCHAR(10)` column returns anything from 0 to 10 characters.

The examples in this section use a three-column version of the `season` table, which has the following schema:

```
premdb=# \d season
            Table "public.season"
   Column    |         Type          | Modifiers
-------------+-----------------------+-----------
 seasonid    | smallint              |
 season_name | character(12)         |
 winners     | character varying(30) |

Distribution: Hash (seasonid)
```

This table contains the following rows:

```
premdb=# select * from season order by 1;
 seasonid | season_name  |      winners
----------+--------------+-------------------
       20 | 2011-2012    | Manchester City
       21 | 2012-2013    | Manchester United
       22 | 2013-2014    | Manchester City
       23 | 2014-2015    | Chelsea
       24 | 2015-2016    | Leicester City
(5 rows)
```

The following query returns the respective lengths of all of the character strings:

```
premdb=# select *, length(season_name) length_char, length(winners) length_varchar from season order by 1;
 seasonid | season_name  |      winners      | length_char | length_varchar
----------+--------------+-------------------+-------------+----------------
       20 | 2011-2012    | Manchester City   |           9 |             15
```

```
      21 | 2012-2013    | Manchester United |           9 |            17
      22 | 2013-2014    | Manchester City   |           9 |            15
      23 | 2014-2015    | Chelsea           |           9 |             7
      24 | 2015-2016    | Leicester City    |           9 |            14
(5 rows)
```

Note that the length of the `VARCHAR` strings (`winners`) varies. The length of the `CHAR` strings (`season_name`) is consistently `9` , regardless of the fact that the column was defined as `CHAR(12)` .

Now insert two more rows as follows, and note the trailing blanks in the strings:

```
premdb=# insert into season values(25, '2016-2017 ', 'Chelsea  ');
INSERT 0 1
premdb=# insert into season values(26, '2017-2018 ', '');
```

Run the length query again:

```
premdb=# select *, length(season_name) length_char, length(winners) length_varchar from season;
 seasonid | season_name  |      winners      | length_char | length_varchar
----------+--------------+-------------------+-------------+----------------
      20 | 2011-2012    | Manchester City   |           9 |            15
      21 | 2012-2013    | Manchester United |           9 |            17
      22 | 2013-2014    | Manchester City   |           9 |            15
      23 | 2014-2015    | Chelsea           |           9 |             7
      24 | 2015-2016    | Leicester City    |           9 |            14
      25 | 2016-2017    | Chelsea           |           9 |             9
      26 | 2017-2018    |                   |           9 |             0
(7 rows)
```

Note that the `season_name` column always returns a length of `9` . The trailing blanks for rows `25` and `26` were trimmed. Further, note that the lengths of the `Chelsea` values for rows `23` and `25` are different. In row `25` , the trailing blanks are preserved, making a length of `9` , not `7` .

Now run some queries that compare these character strings with literal values:

```
premdb=# select * from season where winners='Chelsea';
 seasonid | season_name  | winners
----------+--------------+---------
      23 | 2014-2015    | Chelsea
(1 row)

premdb=# select * from season where winners like 'Chelsea%';
 seasonid | season_name  |  winners
----------+--------------+-----------
      23 | 2014-2015    | Chelsea
      25 | 2016-2017    | Chelsea
(2 rows)
```

```
premdb=# select * from season where season_name ='2016-2017 ';
 seasonid | season_name  |  winners
----------+--------------+-----------
      25 | 2016-2017    | Chelsea
(1 row)
```

Now create a new version of the `season` table called `season_varchar` , in which both character columns are defined as `VARCHAR` columns.

```
premdb=# \d season_varchar
        Table "public.season_varchar"
```

```
   Column    |         Type          | Modifiers
-------------+-----------------------+-----------
 seasonid    | smallint              |
 season_name | character varying(12) |
 winners     | character varying(30) |

Distribution: Hash (seasonid)
```

Insert the same data in the new table, including the trailing blanks. You now have two tables that appear to return identical result sets:

```
premdb=# select * from season_varchar order by 1;
 seasonid | season_name |      winners
----------+-------------+-------------------
       20 | 2011-2012   | Manchester City
       21 | 2012-2013   | Manchester United
       22 | 2013-2014   | Manchester City
       23 | 2014-2015   | Chelsea
       24 | 2015-2016   | Leicester City
       25 | 2016-2017   | Chelsea
       26 | 2017-2018   |
(7 rows)

premdb=# select * from season order by 1;
 seasonid | season_name  |      winners
----------+--------------+-------------------
       20 | 2011-2012    | Manchester City
       21 | 2012-2013    | Manchester United
       22 | 2013-2014    | Manchester City
       23 | 2014-2015    | Chelsea
       24 | 2015-2016    | Leicester City
       25 | 2016-2017    | Chelsea
       26 | 2017-2018    |
(7 rows)
```

However, when you join these tables on `season_name`, note the results:

```
premdb=# select * from season s, season_varchar sv where s.season_name=sv.season_name order by s.seasonid;
 seasonid | season_name  |      winners      | seasonid | season_name |      winners
----------+--------------+-------------------+----------+-------------+-------------------
       20 | 2011-2012    | Manchester City   |       20 | 2011-2012   | Manchester City
       21 | 2012-2013    | Manchester United |       21 | 2012-2013   | Manchester United
       22 | 2013-2014    | Manchester City   |       22 | 2013-2014   | Manchester City
       23 | 2014-2015    | Chelsea           |       23 | 2014-2015   | Chelsea
       24 | 2015-2016    | Leicester City    |       24 | 2015-2016   | Leicester City
(5 rows)
```

Only `5` rows are returned, not `7`. The trailing blanks preserved in the `VARCHAR` version of the `season_name` column (rows `25` and `26`) cause a mismatch for those two rows.

UUID

Yellowbrick Documentation > Reference > SQL Reference > SQL Data Types > UUID

Platforms: All platforms

Parent topic: SQL Data Types

The UUID data type stores 128-bit Universally Unique Identifiers. See also sys.gen_random_uuid().

A UUID is a sequence of lower-case hexadecimal digits. Each UUID forms five groups of digits, and each group is separated by a hyphen: a group of 8 digits, three groups of 4 digits, and a group of 12 digits. This sequence makes a total of 32 digits representing 128 bits. For example:

```
653dfdba-f044-42a2-9000-199945f61dce
```

Alternative input forms are accepted: the use of uppercase digits, the standard format surrounded by braces, omission of some or all hyphens, an additional hyphen after any group of four digits. For example:

```
653DFDBA-F044-42A2-9000-199945F61DCE
{653dfdba-f044-42a2-9000-199945f61dce}
653dfdbaf04442a29000199945f61dce
653d-fdba-f044-42a2-9000-1999-45f6-1dce
653dfdba-f04442a2-90001999-45f61dce
```

The output is always in the standard form, regardless of the input format.

```
premdb=# \d uuidtest
  Table "public.uuidtest"
 Column  | Type | Modifiers
---------+------+-----------
 uuidcol | uuid |

Distribution: Hash (uuidcol)

premdb=# insert into uuidtest values('653dfdba-f04442a2-90001999-45f61dce');
INSERT 0 1

premdb=# select distinct uuidcol from uuidtest where uuidcol='653dfdba-f04442a2-90001999-45f61dce';
              uuidcol
--------------------------------------
 653dfdba-f044-42a2-9000-199945f61dce
(1 row)
```

Note: You cannot select UUID values based on a LIKE condition. Instead, use a condition with a comparison operator such as `>=` and complete the unknown digits with zeros. For example:

```
premdb=# select * from uuidtest where c1 >= '653dfdba-0000-0000-0000-000000000000';
                c1
--------------------------------------
 653dfdba-f044-42a2-9000-199945f61dce
(1 row)
```

VARCHAR

The `VARCHAR` data type is a variable-length character string. For example, a column declared as `VARCHAR(25)` holds up to 25 bytes in each row, depending on the length of the strings that are loaded. For details and examples, see Trailing Blanks in Character Data.

```sql
{ VARCHAR | CHARACTER VARYING } [ ( { bytes | MAX | ANY } ) ]
```

Note that `VARCHAR` columns can hold a given number of *bytes*, not characters.

The `MAX` and `ANY` keywords define the column with the maximum length of 64000 bytes (equivalent to `VARCHAR(64000)`). If you create a `VARCHAR` column without specifying a length, the default length is 256 bytes. For example:

```sql
create table var(c1 varchar, c2 varchar(max), c3 varchar(any), c4 varchar(100));

\d var
            Table "public.var"
 Column |           Type           | Modifiers
--------+--------------------------+-----------
 c1     | character varying(256)   |
 c2     | character varying(64000) |
 c3     | character varying(64000) |
 c4     | character varying(100)   |

Distribution: Hash (c1)
```

An attempt to load a string that is longer than the specified length of a `VARCHAR` column results in a `value too long` error.

Note: The maximum length of a row in a table is 64231 bytes. See Maximum Row Size.

In terms of the data type hierarchy for character strings, `VARCHAR` has precedence over `CHAR`, and `CHAR` has precedence over `UNKNOWN`.

SQL Functions and Operators

Platforms: All platforms

Parent topic: SQL Reference

In this section:

This section describes supported SQL functions and operators in several main categories, including a section on SQL conditions.

Aggregate Functions

Platforms: All platforms

Parent topic: SQL Functions and Operators

In this section:

This section describes standard SQL aggregate functions. For details about aggregate window functions (or *windowed aggregates*), see Window Functions.

AVG

Return the average (arithmetic mean) of the values for an expression.

```
AVG | AVERAGE (expression)
```

`AVG` and `AVERAGE` are both supported. The expression must be numeric data type or a time data type.

For example:

```
premdb=# SELECT AVG(capacity)::INT FROM team WHERE capacity >0;
  avg
-------
 31895
(1 row)
```

The following example groups by the `city` column to return average stadium capacities per city:

```
premdb=# SELECT city, AVG(capacity)::INT
FROM team WHERE capacity >0 GROUP BY city ORDER BY 2 DESC;
    city      |  avg
--------------+-------
 Manchester    | 65366
 Newcastle     | 52405
 Sunderland    | 49000
 Liverpool     | 42482
 Leeds         | 39460
 Sheffield     | 36217
...
```

Return Types

The return type of the AVG function depends on the input type:

- `AVG(INT2)` returns `DECIMAL(25,6)`
- `AVG(INT4)` returns `DECIMAL(25,6)`
- `AVG(INT8)` returns `DECIMAL(38,6)`
- `AVG(FLOAT4)` returns `FLOAT4`
- `AVG(FLOAT8)` returns `FLOAT8`
- `AVG(DECIMAL(p,s))` returns `DECIMAL(p,s)`
- `AVG(TIME)` returns `INTERVAL`

COUNT

Return the number of non-`NULL` values for an expression. The `ALL` keyword is optional. Use `COUNT(*)` to return a count of all of the rows in a table (regardless of `NULL` values).

```
COUNT( [ ALL ] expression)
COUNT(*)
```

The COUNT function works with all supported data types.

For example, count the rows in the `match` table:

```
premdb=# SELECT COUNT(*) FROM match;
 count
-------
  8606
(1 row)
```

The following query groups by the `winners` column in the `season` table to count the number of times a given team won the league:

```
premdb=# SELECT winners, COUNT(winners)
FROM season WHERE winners IS NOT NULL GROUP BY winners ORDER BY 2 DESC;
     winners      | count
------------------+-------
 Manchester United |    13
 Chelsea           |     4
 Arsenal           |     3
 Manchester City   |     2
 Blackburn Rovers  |     1
 Leicester City    |     1
(6 rows)
```

Note: The COUNT function always returns a BIGINT data type.

COUNT (DISTINCT)

Return the number of distinct non-`NULL` values for an expression.

```
COUNT(DISTINCT expression)
```

For example:

```
premdb=# SELECT COUNT(DISTINCT winners) FROM season;
 count
-------
     6
(1 row)
```

Aggregate functions are supported as both regular aggregate functions and aggregate window functions. See also COUNT (window function).

GROUP_CONCAT

Concatenate a set of character strings and insert a delimiter between each string. See STRING_AGG for details.

GROUPING

Platforms: All platforms

Parent topic: Aggregate Functions

Return an integer bit mask that identifies which set a row belongs to in the results of a query that uses `GROUP BY` extensions: `GROUPING SETS`, `CUBE`, or `ROLLUP`.

```
GROUPING(group_by_expression [ ,...])
```

The expressions for the `GROUPING` function must match, or be a subset of, the expressions used in the `GROUP BY GROUPING SETS`, `GROUP BY CUBE`, or `GROUP BY ROLLUP` clause.

For each grouped row in the result of the query, the `GROUPING` function assigns a bit to each `GROUP BY` expression. These bit values are converted to and displayed as their corresponding integer values (for example, `00 = 0`, `01 = 1`, `10 = 2`, `11 = 3`, and so on). A return value of `0` means that the expression in the `GROUPING` function is included in the grouping criteria for that row. A return value of `1` means that the expression in the `GROUPING` function is not included.

`GROUP BY CUBE` and `GROUP BY ROLLUP` queries return higher numbers, such as `3` (`11`) for their subtotal and grand total rows, given that none of the grouping criteria apply to those rows.

Because rows in queries that use `GROUP BY` extensions may be identified in this way, you can filter query results in the `HAVING` clause based on values returned by `GROUPING` functions. For example, you can return only rows with `0`, return only subtotal rows, or filter out subtotal and grand total rows.

Examples

The examples in this section are based on the following set of rows in the `player` table:

```
premdb=# select * from player;
 playerid | teamid | seasonid | firstname | lastname  | position |    dob     | weekly_wages | avg_mins_per_match | matches_played
----------+--------+----------+-----------+-----------+----------+------------+--------------+--------------------+---------------
       1 |      2 |       27 | NULL      | NULL      | NULL     | NULL       |         NULL |               NULL |          NULL
       2 |     41 |       27 | Harry     | Kane      | F        | 1993-07-28 |       250000 |            84.1567 |          36.1
       3 |     41 |       27 | Harry     | Winks     | M        | 1996-02-02 |        40000 |            72.3412 |          31.2
       4 |     24 |       27 | Kevin     | De Bruyne | M        | 1991-06-28 |       418000 |            86.0981 |          35.3
       5 |     25 |       27 | Paul      | Pogba     | M        | 1993-03-15 |       378000 |            76.5341 |          29.4
       6 |     25 |       27 | Alexis    | Sanchez   | F        | 1988-12-19 |       410000 |            71.2309 |          31.5
       7 |     13 |       27 | Gonzalo   | Higuain   | F        | 1987-12-10 |       352000 |            61.9803 |          26.6
       8 |      1 |       27 | Mesut     | Ozil      | M        | 1988-10-15 |       350000 |            64.4561 |          32.7
       9 |     13 |       27 | Eden      | Hazard    | M        | 1991-01-07 |       200000 |            81.2387 |          36.8
      10 |     24 |       27 | Sergio    | Aguero    | F        | 1988-06-02 |       220000 |            80.0032 |          33.9
      11 |     25 |       27 | Romelu    | Lukaku    | M        | 1993-05-13 |       250000 |            85.1897 |            34
      12 |     25 |       27 | David     | de Gea    | G        | 1990-11-07 |       200000 |            88.5764 |          35.1
      13 |     23 |       27 | Mo        | Salah     | F        | 1992-06-15 |       200000 |            82.2765 |          36.2
      14 |     24 |       27 | Riyad     | Mahrez    | M        | 1991-02-21 |       200000 |            55.1908 |          24.3
      15 |     23 |       27 | Virgil    | Van Dijk  | D        | 1991-07-08 |       180000 |            87.6345 |          37.4
      16 |     24 |       27 | David     | Silva     | M        | 1986-01-08 |       250000 |            79.6723 |          30.5
      17 |     22 |       27 | Jamie     | Vardy     | F        | 1987-01-11 |        80000 |            77.1986 |          31.6
      18 |     41 |       27 | Dele      | Alli      | M        | 1996-04-11 |       100000 |            75.9013 |          30.7
      19 |     41 |       27 | Marcus    | Rashford  | F        | 1997-10-31 |       250000 |            79.9912 |          32.8
      20 |     47 |       27 | NULL      | NULL      | NULL     | NULL       |         NULL |               NULL |          NULL
(20 rows)
```

This example uses the function on one column, `position`.

```
premdb=# select position, cob, grouping(position), sum(weekly_wages)
from player
group by grouping sets(position, cob)
order by 1, 2, 3;
 position |     cob     | grouping |   sum
----------+-------------+----------+---------
 D        | [NULL]      |        0 |  180000
 F        | [NULL]      |        0 | 1762000
 G        | [NULL]      |        0 |  200000
 M        | [NULL]      |        0 | 2186000
 [NULL]   | Algeria     |        1 |  200000
 [NULL]   | Argentina   |        1 |  572000
 [NULL]   | Belgium     |        1 |  250000
 [NULL]   | Chile       |        1 |  410000
 [NULL]   | Egypt       |        1 |  200000
 [NULL]   | England     |        1 |  720000
 [NULL]   | France      |        1 |  378000
 [NULL]   | Germany     |        1 |  350000
 [NULL]   | Netherlands |        1 |  798000
 [NULL]   | Spain       |        1 |  450000
 [NULL]   | [NULL]      |        0 |  [NULL]
 [NULL]   | [NULL]      |        1 |  [NULL]
(16 rows)
```

This example uses the function on one columns `position` and `cob`.

```
premdb=# select position, cob, grouping(position, cob), sum(weekly_wages)
from player
group by grouping sets(position, cob)
order by 1, 2, 3;
 position |     cob     | grouping |   sum
----------+-------------+----------+---------
 D        | [NULL]      |        1 |  180000
 F        | [NULL]      |        1 | 1762000
 G        | [NULL]      |        1 |  200000
 M        | [NULL]      |        1 | 2186000
 [NULL]   | Algeria     |        2 |  200000
 [NULL]   | Argentina   |        2 |  572000
 [NULL]   | Belgium     |        2 |  250000
 [NULL]   | Chile       |        2 |  410000
 [NULL]   | Egypt       |        2 |  200000
 [NULL]   | England     |        2 |  720000
 [NULL]   | France      |        2 |  378000
 [NULL]   | Germany     |        2 |  350000
 [NULL]   | Netherlands |        2 |  798000
 [NULL]   | Spain       |        2 |  450000
 [NULL]   | [NULL]      |        1 |  [NULL]
 [NULL]   | [NULL]      |        2 |  [NULL]
(16 rows)
```

This example uses the `GROUPING` function to identify results returned by `GROUP BY ROLLUP`. The rows that return 0 are those rows that can be grouped by both `position` and `cob`. The rows that return `1` are subtotal rows.

```
premdb=# select position, cob, grouping(position, cob), sum(weekly_wages)
from player
group by rollup(position, cob)
order by 1, 2, 3;
 position |     cob     | grouping |   sum
----------+-------------+----------+---------
 D        | Netherlands |        0 |  180000
```

```
 D        | NULL        |        1 |  180000
 F        | Argentina   |        0 |  572000
 F        | Chile       |        0 |  410000
 F        | Egypt       |        0 |  200000
 F        | England     |        0 |  580000
 F        | NULL        |        1 | 1762000
 G        | Spain       |        0 |  200000
 G        | NULL        |        1 |  200000
 M        | Algeria     |        0 |  200000
 M        | Belgium     |        0 |  250000
 M        | England     |        0 |  140000
 M        | France      |        0 |  378000
 M        | Germany     |        0 |  350000
 M        | Netherlands |        0 |  618000
 M        | Spain       |        0 |  250000
 M        | NULL        |        1 | 2186000
 NULL     | NULL        |        0 |    NULL
 NULL     | NULL        |        1 |    NULL
 NULL     | NULL        |        3 | 4328000
(20 rows)
```

LISTAGG

Platforms: All platforms

Parent topic: Aggregate Functions

Concatenate a set of character strings and insert a delimiter between each string. See STRING_AGG for details.

MAX

Return the maximum value for an expression (numeric, character, or datetime).

```
MAX(expression)
```

Note: The input `expression` for a MAX function cannot be a Boolean data type.

The following example returns the closing `matchday` for each season.

```
premdb=# SELECT seasonid, MAX(matchday)
FROM match WHERE matchday IS NOT NULL
GROUP BY seasonid ORDER BY seasonid;
 seasonid |         max
----------+---------------------
       2 | 1994-05-08 00:00:00
       3 | 1995-05-14 00:00:00
       4 | 1996-05-04 00:00:00
       5 | 1997-05-11 00:00:00
       6 | 1998-05-10 00:00:00
       7 | 1999-05-16 00:00:00
       8 | 2000-05-14 00:00:00
       9 | 2001-05-19 00:00:00
      10 | 2002-05-11 00:00:00
...
```

The following example returns the "maximum" `nickname` value from the `team` table:

```
premdb=# SELECT MAX(nickname) FROM team;
  max
--------
 Wolves
(1 row)
```

The following example returns the most goals scored in a single game per season. The MAX function is applied to substrings of the `ftscore` column (full-time score). The first substring returns the goals scored by the home team, and the second the goals scored by the away team.

```
premdb=# SELECT seasonid,
MAX(SUBSTR(ftscore,1,1)::SMALLINT + SUBSTR(ftscore,3,3)::SMALLINT) AS maxgoals_per_match
FROM matchcopy GROUP BY seasonid ORDER BY 1;
 seasonid | maxgoals_per_match
----------+--------------------
       1 |                  8
       2 |                  9
       3 |                  9
       4 |                  8
       5 |                  9
       6 |                  9
       7 |                  9
```

```
       8 |                9
       9 |                8
      10 |                8
      11 |                8
      12 |                8
      13 |                9
      14 |                7
      15 |                8
      16 |               11
      17 |                8
      18 |               10
      19 |                8
      20 |               10
      21 |               10
      22 |                9
...
```

MIN and MAX with NaN Values

Yellowbrick supports `NaN` (not a number) as a floating-point value. When MIN and MAX functions are applied to data that contains `NaN` values, `NaN` is returned as the *maximum* value. `NaN` sorts as greater than all other floating-point values, including `+inf` (infinity).

MEDIAN()

Return the median value from an ordered set, equivalent to `percentile_cont(0.5)`. The results of `median()` and `percentile_cont(0.5)` are always the same.

```
MEDIAN() WITHIN GROUP (ORDER BY sort_expression)
```

See PERCENTILE_CONT.

For example:

```
premdb=# select median() within group (order by avg_att) from team where avg_att>0;
 percentile_cont
-----------------
           34.3
(1 row)
```

MIN

Return the minimum value for an expression (numeric, character, or datetime).

```
MIN(expression)
```

Note: The input `expression` for a MIN function cannot be a Boolean data type.

The following example returns the starting `matchday` for each season.

```
premdb=# SELECT seasonid, MIN(matchday) FROM match
WHERE matchday IS NOT NULL GROUP BY seasonid ORDER BY seasonid;

 seasonid |         min
----------+---------------------
        2 | 1993-08-14 00:00:00
        3 | 1994-08-20 00:00:00
        4 | 1995-08-19 00:00:00
        5 | 1996-08-17 00:00:00
        6 | 1997-08-09 00:00:00
        7 | 1998-08-15 00:00:00
        8 | 1999-08-07 00:00:00
        9 | 2000-08-19 00:00:00
       10 | 2001-08-18 00:00:00
       11 | 2002-08-17 00:00:00
       12 | 2003-08-16 00:00:00
       13 | 2004-08-14 00:00:00
       14 | 2005-08-13 00:00:00
       15 | 2006-08-19 00:00:00
       16 | 2007-08-11 00:00:00
       17 | 2008-08-16 00:00:00
       18 | 2009-08-15 00:00:00
       19 | 2010-08-14 00:00:00
       20 | 2011-08-13 00:00:00
       21 | 2012-08-18 00:00:00
       22 | 2013-08-17 00:00:00
(21 rows)
```

The following example returns the "minimum" `nickname` value from the `team` table:

```
premdb=# SELECT MIN(nickname) FROM team;
   min
---------
 Addicks
(1 row)
```

MIN and MAX with NaN Values

Yellowbrick supports `NaN` (not a number) as a floating-point value. When MIN and MAX functions are applied to data that contains `NaN` values, `NaN` is returned as the *maximum* value. `NaN` sorts as greater than all other floating-point values, including `+inf` (infinity).

PERCENTILE_CONT

Platforms: All platforms

Parent topic: Aggregate Functions

Given an ordered set of values, this aggregate function returns the continuous percentile value, which is the value from the set that corresponds to the specified fraction. For example, `percentile_cont(0.5)` returns the median value. If necessary, the percentile value is interpolated in the set. See also MEDIAN() and PERCENTILE_DISC.

```
percentile_cont(fraction) WITHIN GROUP (ORDER BY sort_expression)
```

Specify a fraction between `0` and `1`. The sort expression data type must be numeric.

For example:

```
premdb=# select max(avg_att), min(avg_att),
percentile_cont(0.2) within group (order by avg_att) from team where avg_att>0;
  max   |  min   | percentile_cont
--------+--------+-----------------
 75.286 | 11.189 |          24.635
(1 row)
```

```
premdb=# select max(avg_att), min(avg_att),
percentile_cont(0.9) within group (order by avg_att) from team where avg_att>0;
  max   |  min   | percentile_cont
--------+--------+-----------------
 75.286 | 11.189 |         54.6313
(1 row)
```

PERCENTILE_DISC

Given an ordered set of values, this aggregate function returns the discrete percentile value, which is the first input value whose position in the ordering equals or exceeds the specified fraction. See also MEDIAN() and PERCENTILE_CONT.

```
percentile_disc(fraction) WITHIN GROUP (ORDER BY sort_expression)
```

Specify a fraction between `0` and `1`.

For example:

```
premdb=# select max(avg_att), min(avg_att),
percentile_disc(0.2) within group (order by avg_att) from team where avg_att>0;
  max   |  min   | percentile_disc
--------+--------+-----------------
 75.286 | 11.189 |          24.636
(1 row)
```

```
premdb=# select max(avg_att), min(avg_att),
percentile_disc(0.9) within group (order by avg_att) from team where avg_att>0;
  max   |  min   | percentile_disc
--------+--------+-----------------
 75.286 | 11.189 |          59.944
(1 row)
```

STDDEV_SAMP and STDDEV_POP

Given a set of integer, decimal, or floating-point numbers, return the sample standard deviation or the population standard deviation. The STDDEV_SAMP function returns a result that is equivalent to the square root of the sample variance of the same set of values.

```
STDDEV_SAMP | STDDEV (expression)
STDDEV_POP (expression)
```

The `expression` must be an integer, decimal, or floating-point number. The return type is either DOUBLE PRECISION (for floating-point inputs) or DECIMAL (for all other inputs).

STDDEV_SAMP and STDDEV are synonyms.

For example:

```
premdb=# select avg(avg_att), stddev_samp(avg_att) stddevsamp, stddev_pop(avg_att) stddevpop
from team where avg_att>0;
  avg   |    stddevsamp    |    stddevpop
--------+------------------+------------------
 36.461 | 15.0809117098404 | 14.6990544934019
(1 row)
```

In this example, the standard deviation results are rounded:

```
premdb=# select avg(avg_att), round(stddev_samp(avg_att)) stddevsamp, round(stddev_pop(avg_att)) stddevpop
from team where avg_att>0;
  avg   | stddevsamp | stddevpop
--------+------------+-----------
 36.461 |         15 |        15
(1 row)
```

Note: The STDDEV or STDDEV_SAMP function for an expression that consists of a single value returns `NULL`. The STDDEV_POP function for an expression that consists of a single value returns `0`).

STRING_AGG

Concatenate a set of character strings and insert a delimiter between each string. `GROUP_CONCAT` and `LISTAGG` are aliases for this function.

```
STRING_AGG([DISTINCT] expression, delimiter)
```

DISTINCT

Optionally, remove duplicates when concatenating the character strings. (The function alias `GROUP_CONCAT_DISTINCT` is also supported. If you use this alias, do not specify the `DISTINCT` keyword as well.)

expression

Specify a set of `CHAR` or `VARCHAR` values that will be concatenated.

delimiter

Specify a `CHAR` or `VARCHAR` literal value that is used as a separator.

Usage Notes

- This function returns a `VARCHAR(64000)` data type.
- This function does not support the `WITHIN GROUP (ORDER BY)` syntax; therefore, the results of this function are not deterministic. You should not expect the results to be returned in sorted order.
- `null` values for `expression` and `delimiter` are ignored.

Examples

Return a concatenated list of season names from the `season` table for specific seasons, using a comma and a space as the delimiter:

```
premdb=# select string_agg(season_name, ', ') from season where numteams=22;
         string_agg
--------------------------------
 1992-1993, 1993-1994, 1994-1995
(1 row)
```

Concatenate all the strings from the `season_name` column in the `season` table and return the length of the concatenated string:

```
premdb=# select length(listagg(season_name,'|')) from season;
 length
--------
    249
(1 row)

premdb=# select length(string_agg(season_name,'|')) from season;
 length
--------
    249
(1 row)
```

```
premdb=# select length(group_concat(season_name,'|')) from season;
 length
--------
    249
(1 row)
```

Note that all three function names return the same result.

Return a list of full-time scores for each home team in season 10:

```
premdb=# select team.htid, team.name, string_agg(ftscore, ', ')
from match join team on match.htid=team.htid
where seasonid=10
group by team.htid, team.name
order by team.htid;
 htid |       name        |                                                                                                  string_agg
------+-------------------+-------------------------------------------------------------------------------------------------------------------------------------------
    2 | Arsenal           | 3-2, 3-3, 1-1, 2-4, 2-1, 1-0, 4-3, 4-1, 2-0, 1-2, 4-0, 1-1, 3-1, 2-1, 1-3, 1-1, 3-0, 2-1, 2-0
    3 | Aston Villa       | 1-2, 2-0, 3-2, 1-0, 1-1, 2-1, 0-0, 2-0, 2-1, 0-1, 0-2, 1-2, 1-1, 0-0, 1-1, 2-1, 0-0, 1-1, 2-1
    6 | Blackburn Rovers  | 2-3, 3-0, 1-1, 4-1, 0-0, 0-1, 1-0, 3-0, 2-1, 1-2, 0-0, 1-1, 2-2, 0-1, 2-2, 2-0, 0-3, 2-1, 7-1
    8 | Bolton Wanderers  | 0-2, 3-2, 1-1, 0-0, 2-2, 1-3, 2-2, 0-0, 4-1, 0-3, 2-2, 2-1, 0-4, 1-0, 0-4, 0-1, 0-2, 1-1, 1-0
   13 | Charlton Athletic | 0-3, 1-2, 0-2, 1-2, 2-1, 1-0, 1-2, 1-1, 3-2, 0-2, 2-0, 0-2, 0-2, 0-0, 1-1, 1-1, 2-2, 3-1, 4-4
   14 | Chelsea           | 1-1, 1-3, 0-0, 5-1, 0-1, 2-1, 3-0, 3-2, 2-1, 2-0, 2-0, 4-0, 0-3, 2-2, 1-1, 2-4, 4-0, 4-0, 5-1
   17 | Derby County      | 0-2, 3-1, 2-1, 1-0, 1-1, 1-1, 3-4, 0-1, 1-3, 0-1, 2-3, 0-1, 2-2, 0-1, 2-3, 1-0, 0-1, 1-0, 0-0
   18 | Everton           | 0-1, 3-2, 1-2, 3-1, 0-3, 0-0, 1-0, 2-1, 1-2, 0-0, 2-2, 1-3, 0-2, 2-0, 1-3, 2-0, 1-0, 1-1, 5-0
   19 | Fulham            | 1-3, 0-0, 2-0, 3-0, 0-0, 1-1, 0-0, 2-0, 1-1, 0-0, 0-0, 0-2, 2-3, 2-1, 3-1, 2-1, 2-0, 0-2, 0-1
   21 | Ipswich Town      | 0-2, 0-0, 1-1, 1-2, 0-1, 0-0, 3-1, 0-0, 1-0, 1-2, 2-0, 0-6, 0-1, 1-0, 0-1, 1-3, 5-0, 2-1, 2-3
   22 | Leeds United      | 1-1, 1-1, 3-1, 0-0, 0-0, 0-0, 3-0, 3-2, 0-1, 2-0, 2-2, 0-4, 3-4, 1-0, 3-4, 2-0, 2-0, 2-1, 3-0
   23 | Leicester City    | 1-3, 2-2, 2-1, 0-5, 1-1, 2-3, 0-3, 0-0, 0-0, 1-1, 0-2, 1-4, 0-1, 1-2, 0-0, 0-4, 1-0, 2-1, 1-1
   24 | Liverpool         | 1-2, 1-3, 4-3, 1-1, 2-0, 1-0, 2-0, 1-1, 0-0, 5-0, 1-1, 1-0, 3-1, 2-0, 3-0, 1-1, 1-0, 1-0, 2-1
   26 | Manchester United | 0-1, 1-0, 2-1, 1-2, 0-0, 0-3, 5-0, 4-1, 3-2, 4-0, 1-1, 2-0, 0-1, 0-1, 3-1, 6-1, 4-1, 4-0, 0-1
   27 | Middlesbrough     | 0-4, 2-1, 1-3, 1-1, 0-0, 0-2, 5-1, 1-0, 2-1, 0-0, 2-2, 1-0, 1-2, 0-1, 1-4, 1-3, 2-0, 1-1, 2-0
   28 | Newcastle United  | 0-2, 3-0, 2-1, 3-2, 3-0, 1-2, 1-0, 6-2, 1-1, 2-2, 3-1, 1-0, 0-2, 4-3, 3-0, 3-1, 1-1, 0-2, 3-1
   37 | Southampton       | 0-2, 1-3, 1-2, 0-0, 1-0, 0-2, 2-0, 0-1, 1-1, 3-3, 0-1, 2-2, 2-0, 1-3, 1-1, 3-1, 2-0, 1-0, 2-0
   39 | Sunderland        | 1-1, 1-1, 1-0, 1-0, 2-2, 0-0, 1-1, 1-0, 1-1, 1-0, 2-0, 2-1, 0-1, 1-3, 0-1, 0-1, 1-1, 1-2, 1-0
   42 | Tottenham Hotspur | 1-1, 0-0, 1-0, 3-2, 0-1, 2-3, 3-1, 1-1, 4-0, 1-2, 2-1, 2-1, 1-0, 3-5, 2-1, 1-3, 2-0, 2-1, 1-1
   45 | West Ham United   | 1-1, 1-1, 2-0, 2-1, 2-0, 2-1, 4-0, 1-0, 0-2, 3-1, 0-0, 1-0, 1-1, 3-5, 1-0, 3-0, 2-0, 3-0, 0-1
(20 rows)
```

Run a similar query to the previous example, but specify that distinct scores are concatenated. The same score may only be listed once in the result of the function.

```
premdb=# select team.htid, team.name, string_agg(distinct ftscore, ', ')
from match join team on match.htid=team.htid
where seasonid=10
group by team.htid, team.name
order by team.htid;
 htid |       name        |                                string_agg
------+-------------------+---------------------------------------------------------------------------
    2 | Arsenal           | 1-0, 1-1, 3-2, 1-3, 4-3, 4-0, 3-3, 3-1, 1-2, 2-0, 4-1, 3-0, 2-1, 2-4
    3 | Aston Villa       | 3-2, 1-1, 1-0, 1-2, 0-1, 2-0, 0-0, 0-2, 2-1
    6 | Blackburn Rovers  | 1-2, 2-3, 0-1, 2-2, 2-0, 4-1, 0-0, 3-0, 2-1, 1-1, 0-3, 7-1, 1-0
    8 | Bolton Wanderers  | 0-1, 4-1, 2-2, 2-1, 0-2, 0-0, 0-4, 1-3, 3-2, 1-1, 1-0, 0-3
   13 | Charlton Athletic | 0-0, 0-2, 2-1, 2-0, 2-2, 0-3, 3-2, 1-1, 1-0, 4-4, 3-1, 1-2
   14 | Chelsea           | 2-2, 2-0, 5-1, 3-0, 0-0, 2-4, 2-1, 0-3, 1-3, 3-2, 1-1, 4-0, 0-1
   17 | Derby County      | 3-1, 0-1, 2-3, 2-1, 3-4, 2-2, 0-0, 0-2, 1-0, 1-3, 1-1
   18 | Everton           | 0-2, 0-0, 5-0, 2-1, 2-2, 2-0, 1-3, 3-2, 1-1, 1-0, 0-3, 0-1, 3-1, 1-2
   19 | Fulham            | 3-1, 0-1, 2-3, 2-1, 2-0, 0-2, 0-0, 3-0, 1-3, 1-1
   21 | Ipswich Town      | 1-2, 2-3, 0-1, 3-1, 5-0, 0-0, 0-2, 2-0, 2-1, 0-6, 1-1, 1-0, 1-3
   22 | Leeds United      | 0-1, 3-1, 3-0, 0-0, 2-1, 3-4, 2-2, 2-0, 0-4, 3-2, 1-1, 1-0
   23 | Leicester City    | 1-0, 0-3, 0-4, 1-3, 1-1, 0-5, 0-1, 1-4, 2-3, 1-2, 2-1, 2-2, 0-0, 0-2
   24 | Liverpool         | 1-2, 3-1, 5-0, 2-1, 2-0, 3-0, 0-0, 1-1, 1-0, 1-3, 4-3
   26 | Manchester United | 6-1, 4-0, 1-2, 0-1, 3-1, 5-0, 0-0, 4-1, 2-0, 2-1, 1-0, 0-3, 1-1, 3-2
   27 | Middlesbrough     | 1-0, 1-3, 0-4, 1-1, 1-4, 0-1, 1-2, 2-1, 2-2, 2-0, 5-1, 0-0, 0-2
```

```
    28 | Newcastle United  | 6-2, 2-2, 0-2, 3-0, 2-1, 3-2, 1-1, 1-0, 4-3, 1-2, 3-1
    37 | Southampton       | 3-1, 0-1, 3-3, 1-2, 2-2, 2-0, 0-2, 0-0, 1-3, 1-1, 1-0
    39 | Sunderland        | 1-0, 1-1, 1-3, 0-1, 1-2, 0-0, 2-1, 2-2, 2-0
    42 | Tottenham Hotspur | 3-1, 0-1, 2-3, 1-2, 2-0, 0-0, 2-1, 1-3, 3-2, 1-1, 1-0, 3-5, 4-0
    45 | West Ham United   | 1-1, 1-0, 4-0, 3-5, 0-1, 3-1, 2-1, 2-0, 3-0, 0-0, 0-2
(20 rows)
```

SUM

Yellowbrick Documentation > Reference > SQL Reference > SQL Functions and Operators > Aggregate Functions > SUM

Platforms: All platforms

Parent topic: Aggregate Functions

Return the sum of the values for a numeric expression.

```
SUM(expression)
```

The expression must be numeric.

The following example calculates the total capacity for all of the stadiums used by London teams.

```
premdb=# SELECT SUM(capacity) FROM team WHERE city='London';
  sum
--------
 296983
(1 row)
```

The following example returns the five cities that have the highest total stadium capacity:

```
premdb=# SELECT city, SUM(capacity) FROM team
GROUP BY city ORDER BY 2 desc LIMIT 5;
   city    |  sum
-----------+--------
 London    | 296983
 Manchester | 130732
 Birmingham |  99801
 Liverpool  |  84963
 Sheffield  |  72434
(5 rows)
```

Return Types

The return type of the SUM function depends on the input type. The result is promoted where possible to prevent overflow errors.

- `SUM(INT2)` returns `INT8`
- `SUM(INT4)` returns `INT8`
- `SUM(INT8)` returns `INT8`
- `SUM(FLOAT4)` returns `FLOAT8`
- `SUM(FLOAT8)` returns `FLOAT8`
- `AVG(DECIMAL(p,s))` returns `DECIMAL(38,s)`

VAR_SAMP and VAR_POP

Given a set of integer, decimal, or floating-point numbers, return the sample variance or population variance. The VAR_SAMP result is equivalent to the squared sample standard deviation of the same set of numbers. VAR_SAMP and VARIANCE are synonyms.

```
VAR_SAMP | VARIANCE (expression)
VAR_POP (expression)
```

The return type is either DOUBLE PRECISION (for floating-point inputs) or DECIMAL (for all other inputs).

For example:

```
premdb=# select avg(avg_att), var_samp(avg_att) varsamp, var_pop(avg_att) varpop
from team where avg_att>0;
  avg   |  varsamp   |   varpop
--------+------------+------------
 36.461 | 227.433898 | 216.062203
(1 row)
```

In this example, the variance results are rounded:

```
premdb=# select avg(avg_att), round(var_samp(avg_att)) varsamp, round(var_pop(avg_att)) varpop
from team where avg_att>0;
  avg   | varsamp | varpop
--------+---------+--------
 36.461 |     227 |    216
(1 row)
```

Note: The VARIANCE or VAR_SAMP function for an expression that consists of a single value returns `NULL`. The VAR_POP function for an expression that consists of a single value returns `0`).

Conditional Expressions

Platforms: All platforms

Parent topic: SQL Functions and Operators

In this section:

This section describes special SQL expressions that evaluate conditions.

CASE

Return specific values for different cases based on the evaluation of a series of WHEN conditions. CASE expressions take two forms: *simple* and *searched*.

Simple CASE expression:

```
CASE expression
WHEN value THEN result
[WHEN ...]
[ELSE result]
END
```

Searched CASE expression:

```
CASE
WHEN boolean condition THEN result
[WHEN ...]
[ELSE result]
END
```

Simple CASE example:

```
select season_name, winners,
case winners
when 'Leicester City' then 'One win'
when 'Blackburn Rovers' then 'One win'
when '' then 'Unknown'
else 'Two or more wins'
end
from season
order by 1;

 season_name |      winners      |       case
-------------+-------------------+------------------
 1992-1993   | Manchester United | Two or more wins
 1993-1994   | Manchester United | Two or more wins
 1994-1995   | Blackburn Rovers  | One win
 1995-1996   | Manchester United | Two or more wins
 1996-1997   | Manchester United | Two or more wins
 1997-1998   | Arsenal           | Two or more wins
 1998-1999   | Manchester United | Two or more wins
 1999-2000   | Manchester United | Two or more wins
 2000-2001   | Manchester United | Two or more wins
 2001-2002   | Arsenal           | Two or more wins
 2002-2003   | Manchester United | Two or more wins
 2003-2004   | Arsenal           | Two or more wins
 2004-2005   | Chelsea           | Two or more wins
 2005-2006   | Chelsea           | Two or more wins
 2006-2007   | Manchester United | Two or more wins
 2007-2008   | Manchester United | Two or more wins
 2008-2009   | Manchester United | Two or more wins
 2009-2010   | Chelsea           | Two or more wins
```

```
 2010-2011   | Manchester United | Two or more wins
 2011-2012   | Manchester City   | Two or more wins
 2012-2013   | Manchester United | Two or more wins
 2013-2014   | Manchester City   | Two or more wins
 2014-2015   | Chelsea           | Two or more wins
 2015-2016   | Leicester City    | One win
 2016-2017   |                   | Unknown
(25 rows)
```

Searched CASE example:

```
select seasonid, htid, atid, case
when ftscore='0-0' then 'Bore draw'
when ftscore='1-1' then 'Score draw'
when ftscore='2-2' then 'Score draw'
when ftscore='3-3' then 'Score draw'
when ftscore='4-4' then 'Score draw'
when ftscore='5-5' then 'Score draw'
else 'Not a draw'
end
from match
where seasonid=22 and htid=45
order by htid, atid;

 seasonid | htid | atid |    case
----------+------+------+------------
       22 |   45 |   51 | Not a draw
       22 |   45 |   52 | Bore draw
       22 |   45 |   61 | Not a draw
       22 |   45 |   63 | Not a draw
       22 |   45 |   65 | Not a draw
       22 |   45 |   67 | Not a draw
       22 |   45 |   68 | Not a draw
       22 |   45 |   69 | Not a draw
       22 |   45 |   73 | Not a draw
       22 |   45 |   74 | Not a draw
       22 |   45 |   75 | Not a draw
       22 |   45 |   77 | Not a draw
       22 |   45 |   78 | Not a draw
       22 |   45 |   86 | Not a draw
       22 |   45 |   87 | Not a draw
       22 |   45 |   88 | Bore draw
       22 |   45 |   89 | Not a draw
       22 |   45 |   91 | Not a draw
       22 |   45 |   93 | Score draw
(19 rows)
```

COALESCE

Return the value of the first expression in a list of one or more expressions that is not null. If all expressions are null, return null. As soon as a non-null value is found, do not evaluate any remaining expressions. `ISNULL` is an equivalent conditional expression, but it only accepts two arguments.

```
COALESCE( expression [, ...] )
```

For example, if the `city` column value is null, return `No City`:

```
premdb=# SELECT teamid, COALESCE(city,'No City') FROM team ORDER BY teamid DESC;
 teamid |   coalesce
--------+---------------
     50 | No City
     49 | No City
     48 | No City
     47 | Wolverhampton
     46 | London
     45 | Wigan
     44 | London
...
```

Make sure the list of expressions consists of compatible data types. For example, you cannot reference a SMALLINT and a VARCHAR in a COALESCE expression:

```
premdb=# select name,nickname,city,coalesce(teamid,name) from team order by teamid;
ERROR:  COALESCE types smallint and character varying cannot be matched
LINE 1: select name,nickname,city,coalesce(teamid,name) from team or...
```

DECODE

Replace given values for an expression with specified values or a default value. For example, you can use `DECODE` to replace numeric keys, IDs, and codes with meaningful character strings. This kind of conditional expression is similar to a simple CASE expression.

Syntax

```
DECODE ( expression, search, result [, search, result ] ... [ , default ] )
```

Parameters

expression
An expression that is the target of the search condition, such as a column name.

search
A value that may exist in the expression, such as a value in a column. This value is equivalent to a `when` value in a `CASE` expression. The search expression must evaluate to a single fixed value. You cannot specify an expression that evaluates to a range of values, such as a range of dates or ages. You must specify separate search and result pairs for each value that you want to replace. The data types of all instances of the search expression must be the same or compatible. The expression and search parameters must also be compatible.

result
A single value that may be replaced with the existing value when the search condition returns `true`. The `result` value is equivalent to a `then` value in a `CASE` expression. You must include at least one search and result pair in the DECODE expression. The data types of all instances of the result expression must be the same or compatible. The result and default parameters must also be compatible.

default
A single value that may be replaced with the existing value when the search condition fails (returns `false` or `unknown`). The `default` value is equivalent to the `else` value in a `CASE` expression. If no default value is specified and the search condition fails, the `DECODE` expression returns `null`.

Examples

The following simple examples demonstrate the logic of the `DECODE` syntax. In the first query, 1 = 1 so 3 is returned. In the second query, 1 != 2 so 4 (the default value) is returned.

```
premdb=# select decode(1,1,3,4);
 decode
--------
      3
(1 row)

premdb=# select decode(1,2,3,4);
 decode
--------
```

```
     4
(1 row)
```

In this example, several winning team names are replaced with nicknames. No default value is provided.

```
premdb=# select season_name,
decode(winners,
  'Manchester United', 'Red Devils',
  'Manchester City', 'Citizens',
  'Leicester City', 'Foxes',
  'Arsenal', 'Gunners',
  'Blackburn Rovers', 'Rovers',
  'Chelsea', 'Pensioners')
from season where winners is not null order by 1,2;
 season_name |   decode
-------------+------------
 1992-1993   | Red Devils
 1993-1994   | Red Devils
 1994-1995   | Rovers
 1995-1996   | Red Devils
 1996-1997   | Red Devils
 1997-1998   | Gunners
 1998-1999   | Red Devils
 1999-2000   | Red Devils
 2000-2001   | Red Devils
 2001-2002   | Gunners
 2002-2003   | Red Devils
 2003-2004   | Gunners
 2004-2005   | Pensioners
 2005-2006   | Pensioners
 2006-2007   | Red Devils
 2007-2008   | Red Devils
 2008-2009   | Red Devils
 2009-2010   | Pensioners
 2010-2011   | Red Devils
 2011-2012   | Citizens
 2012-2013   | Red Devils
 2013-2014   | Citizens
 2014-2015   | Pensioners
 2015-2016   | Foxes
(24 rows)
```

The following query provides a default value `Other` for winning teams that are not from London or Manchester.

```
premdb=# select season_name,
decode(winners,'Manchester United','Manchester',
'Manchester City','Manchester',
'Arsenal','London',
'Chelsea','London',
'Other') city from season where winners is not null order by 2,1;
 season_name |    city
-------------+------------
 1997-1998   | London
 2001-2002   | London
 2003-2004   | London
 2004-2005   | London
 2005-2006   | London
 2009-2010   | London
 2014-2015   | London
 1992-1993   | Manchester
 1993-1994   | Manchester
 1995-1996   | Manchester
 1996-1997   | Manchester
 1998-1999   | Manchester
```

```
  1999-2000   | Manchester
  2000-2001   | Manchester
  2002-2003   | Manchester
  2006-2007   | Manchester
  2007-2008   | Manchester
  2008-2009   | Manchester
  2010-2011   | Manchester
  2011-2012   | Manchester
  2012-2013   | Manchester
  2013-2014   | Manchester
  1994-1995   | Other
  2015-2016   | Other
(24 rows)
```

GREATEST and LEAST

Return the largest or smallest value from a list of expressions. The expressions must have compatible data types.

```
GREATEST | LEAST ( expression [, …] )
```

For example:

```
premdb=# select greatest(1,2,3) from sys.const;
 greatest
----------
        3
(1 row)

premdb=# select least(1,2,3) from sys.const;
 least
-------
     1
(1 row)

premdb=# select distinct least(current_date,matchday) from match order by 1 limit 5;
       least
---------------------
 1992-08-01 00:00:00
 1993-08-14 00:00:00
 1993-08-15 00:00:00
 1993-08-16 00:00:00
 1993-08-17 00:00:00
(5 rows)
```

If all expressions are `null`, the output is `null`. Otherwise, `null` values are ignored.

When the expressions are `DECIMAL` values:

- The output will be the largest (`p-s`) of the inputs and the largest `s` of the inputs.
- If `(p-s)` of the output is `>38`, the scale is reduced until `(p-s)` is `<=38`.

IIF

Return two specific conditional values for the same expression: one value if the expression evaluates to true and the other value if the expression evaluates to false.

```
IIF(test_expr, true_expr, false_expr)
```

test_expr

Valid Boolean expression.

true_expr

Value to return if `test_expr` evaluates to true.

false_expr

Value to return if `test_expr` evaluates to false.

Examples

For example, use `IIF` to state whether a team's average stadium attendance was above or below the 2018-19 season average of 38,000:

```
premdb=# SELECT name, stadium, IIF(avg_att > 38.000, 'Above 2018-19 season average', 'Below 2018-19 season average') AS Average_At
FROM team ORDER BY teamid LIMIT 5;
      name       |     stadium      |      average_attendance
-----------------+------------------+------------------------------
 Arsenal         | Emirates Stadium | Above 2018-19 season average
 Aston Villa     | Villa Park       | Below 2018-19 season average
 Barnsley        | Oakwell Stadium  | Below 2018-19 season average
 Birmingham City | St. Andrew's     | Below 2018-19 season average
 Blackburn Rovers | Ewood Park       | Below 2018-19 season average
(5 rows)
```

The `IIF` conditional expression is equivalent to the following simple CASE expression:

```
CASE
WHEN test_expr is true THEN true_expr
ELSE false_expr
END
```

Using the previous `IIF` example, the corresponding CASE statement returns the same results:

```
premdb=# SELECT name, stadium, IIF(avg_att > 38.000, 'Above 2018-19 season average', 'Below 2018-19 season average') AS Average_at
CASE WHEN avg_att > 38.000 is true THEN 'Above 2018-19 season average' ELSE 'Below 2018-19 season average' END AS Average_attendan
FROM team ORDER BY teamid LIMIT 5;
      name       |     stadium      |      average_attendance      |      average_attendance
-----------------+------------------+------------------------------+------------------------------
 Arsenal         | Emirates Stadium | Above 2018-19 season average | Above 2018-19 season average
 Aston Villa     | Villa Park       | Below 2018-19 season average | Below 2018-19 season average
 Barnsley        | Oakwell Stadium  | Below 2018-19 season average | Below 2018-19 season average
 Birmingham City | St. Andrew's     | Below 2018-19 season average | Below 2018-19 season average
```

```
 Blackburn Rovers | Ewood Park       | Below 2018-19 season average | Below 2018-19 season average
(5 rows)
```

ISNULL

ISNULL and NVL are equivalent conditional expressions. See NVL.

NULLIF

Return a `null` value if `value1` equals `value2`; otherwise return `value1`.

```
NULLIF(value1, value2)
```

For example:

```
premdb=# \pset null null
Null display is "null".

premdb=# premdb=# SELECT DISTINCT m.htid, h.htid, nullif(m.htid,h.htid)
FROM match m JOIN hometeam h ON m.htid=h.htid ORDER BY m.htid LIMIT 5;
 htid | htid | nullif
------+------+--------
    2 |    2 |   null
    3 |    3 |   null
    4 |    4 |   null
    5 |    5 |   null
    6 |    6 |   null
(5 rows)
```

```
premdb=# SELECT teamid, name, city, NULLIF(name,city)
FROM team ORDER BY 1;
 teamid |          name           |     city      |         nullif
--------+-------------------------+---------------+------------------------
      1 | Arsenal                 | London        | Arsenal
      2 | Aston Villa             | Birmingham    | Aston Villa
      3 | Barnsley                | Barnsley      | null
      4 | Birmingham City         | Birmingham    | Birmingham City
      5 | Blackburn Rovers        | Blackburn     | Blackburn Rovers
      6 | Blackpool               | Blackpool     | null
      7 | Bolton Wanderers        | Bolton        | Bolton Wanderers
      8 | Bournemouth             | Bournemouth   | null
      9 | Bradford City           | Bradford      | Bradford City
     10 | Burnley                 | Burnley       | null
     11 | Cardiff City            | Cardiff       | Cardiff City
...
```

NVL

Yellowbrick Documentation > Reference > SQL Reference > SQL Functions and Operators > Conditional Expressions > NVL

Platforms: All platforms

Parent topic: Conditional Expressions

Return the value of the first expression that is not `null` in a list of two expressions. If both expressions are `null`, NVL returns `null`. ISNULL and NVL are aliases for the same conditional expression. See also COALESCE, which is similar to NVL and ISNULL but allows more than two expressions in the list.

```
NVL( expression1, expression2 )
ISNULL( expression1, expression2 )
```

For example:

```
premdb=# select isnull(null,1) from sys.const;
 isnull
--------
      1
(1 row)

premdb=# select nvl(null,null) from sys.const;
  nvl
--------
 [NULL]
(1 row)
```

In this example, if the `city` column value is null, return `No City`:

```
premdb=# select teamid, nvl(city,'No City') from team order by teamid desc;
 teamid |      nvl
--------+---------------
     50 | No City
     49 | No City
     48 | No City
     47 | Wolverhampton
     46 | London
     45 | Wigan
     44 | London
...
```

Make sure the list of expressions consists of compatible data types. For example, you cannot reference a SMALLINT and a VARCHAR in an NVL expression.

NVL2

Return a specific value when an expression evaluates to not null and a specific value when the same expression evaluates to null.

```
NVL2( expression, not_null_return_value, null_return_value )
```

The second and third arguments to the function must have compatible data types.

For example, when the city column is not null, return the city name, and when it is null, return `No City`:

```
premdb=# select teamid, nvl2(city, city, 'No City') from team order by 1 desc;
 teamid |     city
--------+---------------
     50 | No City
     49 | No City
     48 | No City
     47 | Wolverhampton
     46 | London
     45 | Wigan
     44 | London
...
```

You can use the DECODE function to produce the same results. For example:

```
premdb=# select teamid, decode(city, null, 'No City', city) from team order by 1 desc;
 teamid |     city
--------+---------------
     50 | No City
     49 | No City
     48 | No City
     47 | Wolverhampton
     46 | London
     45 | Wigan
     44 | London
...
```

Datetime Functions

Platforms: All platforms

Parent topic: SQL Functions and Operators

In this section:

This section covers functions that work with datetime data types and literals.

ADD_MONTHS

Add some number of months to a date or timestamp expression. (You can also subtract months by using a negative number for the second argument.)

```
ADD_MONTHS(expression, number)
```

For example:

```
premdb=# select matchday, add_months(matchday,6) from match where htid=2 and atid=60;
     matchday       |     add_months
--------------------+---------------------
 2010-03-06 00:00:00 | 2010-09-06 00:00:00
(1 row)

premdb=# select matchday, add_months(matchday,-6) from match where htid=2 and atid=60;
     matchday       |     add_months
--------------------+---------------------
 2010-03-06 00:00:00 | 2009-09-06 00:00:00
(1 row)
```

If the month in the result has fewer days than the day specified in the input expression, the function returns the last day of the month. In other cases, the result contains the same day of month number as the input expression. However, when the input expression is the last day of the month, the function always returns the last day of the result month. For example, in a leap year, such as 2000:

```
premdb=# select add_months('2000-02-28',1)::date from sys.const;
 add_months
------------
 2000-03-28
(1 row)

premdb=# select add_months('2000-02-29',1)::date from sys.const;
 add_months
------------
 2000-03-31
(1 row)
```

If either argument is `null`, the result is `null`.

Note: You cannot use the `ADD_MONTHS` function in queries against system views.

AGE

Return the difference between two timestamps or the difference between the current timestamp and a future timestamp.

```
AGE(timestamp [, timestamp])
```

To specify timestamp literals with this function, use the TIMESTAMP keyword.

For example, calculate your age by comparing the current date with your birthdate. For example:

```
premdb=# SELECT AGE(TIMESTAMP '2016-05-23', TIMESTAMP '1990-09-04');
          age
-------------------------
 25 years 8 mons 19 days
(1 row)
```

You can also run this query by using the CURRENT_DATE or CURRENT_TIMESTAMP function as the first argument.

Find out how long ago a specific match was played:

```
premdb=# SELECT matchday, AGE(matchday)
FROM match WHERE seasonid=2 AND htid=2 AND atid=52;
     matchday       |           age
--------------------+-------------------------
 1993-11-06 00:00:00 | 22 years 6 mons 21 days
(1 row)
```

AT TIME ZONE

Yellowbrick Documentation > Reference > SQL Reference > SQL Functions and Operators > Datetime Functions > AT TIME ZONE

Platforms: All platforms

Parent topic: Datetime Functions

The `AT TIME ZONE` construct operates on timestamps (with and without time zones) and returns appropriate values based on standard UTC offsets. `AT TIME ZONE` does not support the `TIME` data type.

```
expression AT TIME ZONE zone
```

The behavior depends on the input type of the expression:

- `TIMESTAMP` (without time zone): Interpret the given value as located in the named time zone, but return the corresponding value in the time zone that is set for the client session (or database). For example, if the client time zone is `PST` and `AT TIME ZONE` specifies `MST`, return the `MST` value -1 hour.
- `TIMESTAMPTZ`: Convert the given value to its equivalent value in the named time zone, then return the converted value as a timestamp *without* a time zone designation. For example, if the client time zone is `PST` and `AT TIME ZONE` specifies `MST`, return the `PST` value +1 hour.

The expression must be a `TIMESTAMP` or `TIMESTAMPTZ` expression (such as a column name or a literal value). The zone must be a valid abbreviation, such as `GMT` or `PST`, or a long-form time zone name such as `US/Pacific` or `Asia/Shanghai`.

When the input type is `TIMESTAMP`, the return type is `TIMESTAMPTZ`. When the input type is `TIMESTAMPTZ`, the return type is `TIMESTAMP`.

You can specify the time zone either as a text string, such as `'GMT'`, or as an interval, such as `INTERVAL '-07:00'`).

For example, assume the `times` table contains a `TIMESTAMP` column and a `TIMESTAMPTZ` column with the following values.

```
premdb=# \d times
                Table "public.times"
   Column    |            Type             | Modifiers
-------------+-----------------------------+-----------
 start_time  | timestamp without time zone |
 start_timetz | timestamp with time zone    |

Distribution: Hash (start_time)

premdb=# select * from times;
    start_time      |      start_timetz
--------------------+-----------------------
 2017-03-02 11:00:00 | 2017-03-02 11:00:00-08
(1 row)
```

The client time zone that is in effect for these examples is `PST`.

In the first example, the `start_time` column is treated as a `GMT` timestamp, then converted to `PST` for display (11AM GMT is equivalent to 3AM PST).

```
premdb=# select start_time, start_time at time zone 'GMT' from times;
    start_time      |        timezone
--------------------+-----------------------
 2017-03-02 11:00:00 | 2017-03-02 03:00:00-08
(1 row)
```

The second example takes a `TIMESTAMPTZ` value specified in `PST` and converts it to local time in `GMT` (11AM PST is equivalent to 7PM GMT).

```
premdb=# select start_timetz, start_timetz at time zone 'GMT' from times;
     start_timetz      |      timezone
-----------------------+--------------------
 2017-03-02 11:00:00-08 | 2017-03-02 19:00:00
(1 row)
```

The following example selects the `matchday` column from the `match` table. The timestamps are treated as `GMT` values and the equivalent `PST` values that are displayed subtract 7 or 8 hours, depending on the time of year:

```
premdb=# select matchday, matchday at time zone 'GMT'
from match where htid=48 and atid=57 order by matchday;
     matchday       |        timezone
--------------------+------------------------
 2004-04-12 00:00:00 | 2004-04-11 17:00:00-07
 2009-12-05 00:00:00 | 2009-12-04 16:00:00-08
 2010-11-13 00:00:00 | 2010-11-12 16:00:00-08
 2012-03-31 00:00:00 | 2012-03-30 17:00:00-07
(4 rows)
```

CURRENT_DATE

Return the date when the current transaction starts. This value does not change during the course of a transaction and does not reflect the individual start dates of statements within a transaction.

You can use the result of this function to calculate intervals between the current date and other datetime values in the database. For example:

```
premdb=# select CURRENT_DATE, matchday, CURRENT_DATE-matchday numdays
FROM match WHERE seasonid=20 LIMIT 2;
   date    |      matchday       |  numdays
------------+---------------------+-----------
 2016-05-20 | 2012-03-24 00:00:00 | 1518 days
 2016-05-20 | 2012-02-04 00:00:00 | 1567 days
(2 rows)
```

CURRENT_TIMESTAMP

Return the date and time when the current transaction starts. This value does not change during the course of a transaction and does not reflect the individual start times of statements within a transaction.

```
CURRENT_TIMESTAMP[(6)]
```

The number `6` (for the fractional seconds part of the timestamp) is optional and does not affect the result. This function always returns a precision of 6. You may see results with fewer than 6 fractional seconds because trailing zeros are not returned. The return type is `TIMESTAMPTZ`.

For example:

```
premdb=# select current_timestamp;
              now
-------------------------------
 2018-03-25 22:47:36.725768-07
(1 row)

premdb=# select current_timestamp(6);
         timestamptz
-------------------------------
 2018-03-25 22:47:38.891822-07
(1 row)

premdb=# select current_timestamp from sys.const;
            now
-------------------------------
 2018-03-25 22:49:45.06732-07
(1 row)
```

You can extract supported date parts from the CURRENT_TIMESTAMP result:

```
premdb=# SELECT EXTRACT(week FROM CURRENT_TIMESTAMP);
 date_part
-----------
       20
(1 row)

premdb=# SELECT EXTRACT(w FROM CURRENT_TIMESTAMP);
 date_part
-----------
       20
(1 row)
```

You can use the result of this function to calculate intervals between the current timestamp and other datetime values in the database.

Note: The CURRENT_TIME function is not supported. Use LOCALTIME.

CURRENT_UTC_TIMESTAMP()

Return the UTC date and time when the current transaction starts. This value does not change during the course of a transaction and does not reflect the individual start times of statements within a transaction.

```
CURRENT_UTC_TIMESTAMP()
```

The parentheses are required. The return type is `TIMESTAMP`.

For example:

```
premdb=# select current_timestamp, current_utc_timestamp() from sys.const;
           now              |   current_utc_timestamp
-------------------------------+---------------------------
 2019-09-12 17:23:54.85683-07 | 2019-09-13 00:23:54.85683
(1 row)

premdb=# select current_utc_timestamp()-current_timestamp from sys.const;
 ?column?
----------
 07:00:00
(1 row)
```

You can extract supported date parts from the `CURRENT_UTC_TIMESTAMP` result:

```
premdb=# select extract(week from current_utc_timestamp()) from sys.const;
 date_part
-----------
        37
(1 row)
```

DATE_PART

Return a specific date part, such as the month or day, for a date, a timestamp (with or without time zone), or an interval literal.

```
DATE_PART('datepart', { date | timestamp | timestamptz | interval }
```

See Supported Date Parts. The DATE_PART function does not support all date parts in all cases. For example, if the second argument is an interval literal, which represents a duration of time, the date part cannot be `week`, which signifies a specific ISO week number in the calendar year.

For example:

```
premdb=# SELECT matchday, DATE_PART('week',matchday) FROM match LIMIT 2;
     matchday       | date_part
--------------------+-----------
 2006-10-28 00:00:00 |        43
 2006-09-16 00:00:00 |        37
(2 rows)
```

```
premdb=# SELECT DATE_PART('day',INTERVAL '2 weeks');
 date_part
-----------
        14
(1 row)
```

DATE_TRUNC

Truncate a datetime expression to the first hour, day, or month (for example), for that expression.

```
DATE_TRUNC('datepart', expression)
```

See Supported Date Parts. This function returns a `TIMESTAMP` or `TIMESTAMPTZ` data type. (If a `DATE` is passed as the second argument, that date is implicitly cast to `TIMESTAMPTZ`.)

For example, return `matchday` values truncated to the first of their respective months:

```
premdb=# SELECT matchday, DATE_TRUNC('month', matchday)
FROM match WHERE htid = 10 ORDER BY 1 LIMIT 3;
     matchday       |     date_trunc
--------------------+--------------------
 1999-08-14 00:00:00 | 1999-08-01 00:00:00
 1999-08-28 00:00:00 | 1999-08-01 00:00:00
 1999-09-12 00:00:00 | 1999-09-01 00:00:00
(3 rows)
```

For example, return the first day of a given week, then add 1 day to the result:

```
yellowbrick=# select date_trunc('week', '2020-10-16'::date), date_trunc('week', '2020-10-16'::date)+ 1 date_trunc_plus1;
      date_trunc       |    date_trunc_plus1
-----------------------+-----------------------
 2020-10-12 00:00:00-07 | 2020-10-13 00:00:00-07
(1 row)
```

DATEADD

Add a specified interval of time to a date or timestamp value.

```
DATEADD( datepart, interval, { date | timestamp } )
```

datepart

See Supported Date Parts.

interval

An *integer* that defines the number of days, weeks, or months, for example, to add to the specified date or timestamp. A negative integer subtracts the interval. You cannot use an interval literal; the interval value must be an integer.

date | timestamp

The date or timestamp must contain the specified date part. Use the `date` or `timestamp` keyword to specify a literal value. For example:

```
date '2016-02-28'
timestamp '2016-02-28 19:34:12.123'
```

Note: All `DATE` values implicitly have a time of `00:00:00`. Therefore, if you subtract a minute (or some other small amount of time) from a given date, the result is the previous day. Conversely, if you add less than 24 hours to a date, the result is the same day.

Examples

For example, add 1 month to the `matchday` column:

```
premdb=# select seasonid, matchday,
dateadd(month,1,matchday) matchday_plus_one
from match where htid=5 and atid=52 order by 1;
 seasonid |      matchday       |  matchday_plus_one
----------+---------------------+---------------------
       11 | 2002-09-16 00:00:00 | 2002-10-16 00:00:00
       12 | 2003-10-19 00:00:00 | 2003-11-19 00:00:00
       13 | 2005-03-20 00:00:00 | 2005-04-20 00:00:00
       14 | 2005-10-16 00:00:00 | 2005-11-16 00:00:00
       16 | 2007-11-11 00:00:00 | 2007-12-11 00:00:00
       18 | 2009-09-13 00:00:00 | 2009-10-13 00:00:00
       19 | 2011-01-16 00:00:00 | 2011-02-16 00:00:00
(7 rows)
```

This example adds 300 seconds to a given timestamp:

```
premdb=# select timestamp '2017-11-21 3:31:00', dateadd(seconds, 300, timestamp '2017-11-21 3:31:00') as plus300secs from sys.cons
     timestamp      |     plus300secs
```

```
---------------------+---------------------
 2017-11-21 03:31:00 | 2017-11-21 03:36:00
(1 row)
```

DATEDIFF

Given a datepart, return the difference between two dates or timestamps.

```
DATEDIFF ( datepart, { date | timestamp | timestamptz }, { date | timestamp | timestamptz } )
```

See Supported Date Parts.

Dates or timestamps must contain the specified datepart. The function returns `0` or a positive result if the second date or time is later than the first. The function returns `0` or a negative result if the second date or time is earlier than the first.

If one of the arguments is an infinite date or timestamp (`'infinity'`), the function returns an error.

Examples

Calculate the number of years since two teams played each other:

```
premdb=# select datediff(year, matchday, current_date)
from match where (htid=10 and atid=72) or (htid=72 and atid=10);
 datediff
----------
       18
       17
(2 rows)
```

Find the difference in seconds between two timestamps:

```
premdb=# select datediff(seconds, timestamp '2017-11-21 12:30:00', '2016-11-21 12:30:00')/60 from sys.const;
 ?column?
----------
   -525600
(1 row)
```

This function calculates the number of datepart boundaries that are crossed between the two expressions (it does not calculate an exact interval). For example, the difference in days between two dates that are 36 hours apart is 2 days.

```
premdb=# select datediff(days, current_date-interval '36 hours', current_date) from sys.const;
 datediff
----------
        2
(1 row)
```

DATENAME

Given a datepart, return a character string representation of a specified date, timestamp, or timestamptz.

```
DATENAME(part, expr)
```

part

See Supported Dateparts.

expr

Date, timestamp, or timestamptz containing the specified datepart.

Examples

For example, identify the day of the week a match took place:

```
premdb=# SELECT matchday, DATENAME('dow', matchday) FROM match LIMIT 5;
     matchday       | datename
--------------------+----------
 2014-05-11 00:00:00 | Sunday
 2013-09-21 00:00:00 | Saturday
 2013-09-01 00:00:00 | Sunday
 2014-04-12 00:00:00 | Saturday
 2014-04-26 00:00:00 | Saturday
(5 rows)
```

`DATENAME` is similar to the `DATE_PART` function but returns a character string rather than a `DOUBLE PRECISION` value. See DATE_PART. Using the previous example, the following compares how `DATE_PART` and `DATENAME` return the same information but in different formats:

```
premdb=# SELECT matchday, DATE_PART('dow', matchday), DATENAME('dow', matchday) FROM match LIMIT 5;
     matchday       | date_part | datename
--------------------+-----------+----------
 2014-05-11 00:00:00 |         0 | Sunday
 2013-09-21 00:00:00 |         6 | Saturday
 2013-09-01 00:00:00 |         0 | Sunday
 2014-04-12 00:00:00 |         6 | Saturday
 2014-04-26 00:00:00 |         6 | Saturday
(5 rows)
```

DAY

Platforms: All platforms

Parent topic: Datetime Functions

Return the day number (1 through 31) from a date or timestamp expression:

```
DAY(expression)
```

For example:

```
premdb=# select day(timestamp '11-30-2017 15:30:10.123456') from sys.const;
 day
-----
  30
(1 row)
```

See also MONTH and YEAR.

DAYS_BETWEEN

Return the number of days between two dates. The result may be positive or negative.

```
DAYS_BETWEEN ( date1, date2 )
```

The inputs may have `DATE`, `TIMESTAMP`, or `TIMESTAMPTZ` data types. If one of the inputs is `null`, the result is `null`.

If the first date is later than the second date, the result is a positive number; otherwise, the result is negative. For example, depending on the order of date expressions in this example, the function returns a positive number or a negative number:

```
premdb=# select matchday, days_between(current_timestamp, matchday) from match where htid=2 and atid=60;
     matchday       | days_between
--------------------+--------------
 2010-03-06 00:00:00 |         3055
(1 row)

premdb=# select matchday, days_between(matchday, current_timestamp) from match where htid=2 and atid=60;
     matchday       | days_between
--------------------+--------------
 2010-03-06 00:00:00 |        -3055
(1 row)
```

Fractional numbers of hours are ignored. This function always returns whole numbers. For example:

```
premdb=# select matchday, days_between(current_timestamp, matchday+interval '12 hours') from match where htid=3 and atid=53;
     matchday       | days_between
--------------------+--------------
 1998-03-11 00:00:00 |         7433
(1 row)
```

EOMONTH

`EOMONTH` is a synonym for LAST_DAY.

EXTRACT

Return a specific date part (such as the month, day, or hour) from a date, timestamp, timestamp with time zone, or time expression, or from an `INTERVAL` literal.

```
EXTRACT( datepart FROM { date | timestamptz | timestamp | time | interval } )
```

See Supported Date Parts.

Examples

Extract the day from a timestamp column:

```
premdb=# SELECT matchday, EXTRACT(day FROM matchday)
FROM match WHERE htid=26 AND seasonid=10 ORDER BY 1 LIMIT 5;
     matchday       | date_part
--------------------+-----------
 2001-08-19 00:00:00 |        19
 2001-09-08 00:00:00 |         8
 2001-09-22 00:00:00 |        22
 2001-10-20 00:00:00 |        20
 2001-10-27 00:00:00 |        27
(5 rows)
```

The following example extracts the `timezone` date part from the `matchday` column (a `timestamptz` column in the `matchtz` table):

```
premdb=# select extract(timezone from matchday),* from matchtz order by seasonid desc limit 5;
 date_part | seasonid |        matchday        | htid | atid | ftscore | htscore
-----------+----------+------------------------+------+------+---------+---------
   -28800 |       22 | 2013-11-09 00:00:00-08 |   14 |   93 | 2-2     | 1-0
   -28800 |       22 | 2014-01-29 00:00:00-08 |   14 |   94 | 0-0     | 0-0
   -25200 |       22 | 2014-04-12 00:00:00-07 |   16 |   52 | 1-0     | 0-0
   -25200 |       22 | 2013-10-26 00:00:00-07 |   16 |   51 | 0-2     | 0-0
   -28800 |       22 | 2013-12-07 00:00:00-08 |   16 |   61 | 2-0     | 1-0
(5 rows)
```

When you specify a literal expression, include the `DATE`, `TIMESTAMP`, `TIMESTAMPTZ`, `TIME`, or `INTERVAL` keyword. For example:

```
premdb=# SELECT EXTRACT(dow FROM TIMESTAMP '2016-04-17 12:00:00');
 date_part
-----------
         0
(1 row)

premdb=# SELECT EXTRACT(hr FROM TIME '11:21:15');
 date_part
-----------
        11
(1 row)
```

GETDATE

Return the current timestamp on any database.

```
GETDATE()
```

For example:

```
premdb=# select getdate();
         getdate
----------------------------
 2020-09-22 14:59:37.720194
(1 row)
```

`GETDATE` is similar to `CURRENT_TIMESTAMP` but returns a `TIMESTAMP` value rather than a `TIMESTAMPTZ` value. See CURRENT_TIMESTAMP. For example:

```
premdb=# select getdate(), current_timestamp as current_timestamp;
         getdate           |       current_timestamp
---------------------------+-------------------------------
 2020-09-22 15:06:07.179172 | 2020-09-22 15:06:07.179172-07
(1 row)
```

JUSTIFY_DAYS, _HOURS, _INTERVAL

Platforms: All platforms

Parent topic: Datetime Functions

Adjust intervals that are expressed in days or hours and return their duration in months or days.

```
JUSTIFY_DAYS(interval)
JUSTIFY_HOURS(interval)
JUSTIFY_INTERVAL(interval)
```

JUSTIFY_DAYS

Represent 30-day time periods as months.

JUSTIFY_HOURS

Represent 24-hour time periods as days.

JUSTIFY_INTERVAL

Represent days and hours as with the other JUSTIFY functions but apply additional plus or minus adjustments. For example:

```
interval '1 mon -1 hour'
```

interval

If you specify a literal, the INTERVAL keyword is required. Yellowbrick does not support INTERVAL columns, so this value must be a valid interval literal in single quotes or a function that evaluates to an interval (such as MAKE_INTERVAL).

JUSTIFY_DAYS Example

```
premdb=# select JUSTIFY_DAYS(INTERVAL '40 days'), matchday,
matchday+JUSTIFY_DAYS(INTERVAL '40 days') matchday_plus_40
FROM match WHERE seasonid=20 AND htid=2 AND atid=55;
 justify_days  |      matchday       |  matchday_plus_40
---------------+---------------------+---------------------
 1 mon 10 days | 2012-02-04 00:00:00 | 2012-03-14 00:00:00
(1 row)
```

JUSTIFY_HOURS Example

```
premdb=# select JUSTIFY_HOURS(INTERVAL '48 hours'), matchday,
matchday+JUSTIFY_DAYS(INTERVAL '48 hours') matchday_plus_48hrs
FROM match WHERE seasonid=20 AND htid=2 AND atid=55;
 justify_hours |      matchday       | matchday_plus_48hrs
---------------+---------------------+---------------------
 2 days        | 2012-02-04 00:00:00 | 2012-02-06 00:00:00
(1 row)
```

JUSTIFY_INTERVAL Example

```
premdb=# select JUSTIFY_INTERVAL(INTERVAL '40 days' + '8 hours'), matchday,
matchday+JUSTIFY_INTERVAL(INTERVAL '40 days' + '8 hours') matchday_plus_40days_plus_8hours
FROM match WHERE seasonid=20 AND htid=2 AND atid=55;
   justify_interval    |      matchday       | matchday_plus_40days_plus_8hours
-----------------------+---------------------+---------------------------------
 1 mon 10 days 08:00:00 | 2012-02-04 00:00:00 | 2012-03-14 08:00:00
(1 row)
```

LAST_DAY

Return the last day of the month, based on a given date or timestamp.

```
LAST_DAY | EOMONTH ( date | timestamp | timestamptz )
```

`EOMONTH` is a synonym for `LAST_DAY`. For example:

```
premdb=# select last_day(current_date) from sys.const;
  last_day
------------
 2017-11-30
(1 row)

premdb=# select eomonth(current_date) from sys.const;
  eomonth
------------
 2017-11-30
(1 row)
```

```
premdb=# select distinct last_day(matchday)
from match where seasonid=22 order by 1;
  last_day
------------
 2013-08-31
 2013-09-30
 2013-10-31
 2013-11-30
 2013-12-31
 2014-01-31
 2014-02-28
 2014-03-31
 2014-04-30
 2014-05-31
(10 rows)
```

LOCALTIME

Return the time of day when the current transaction starts. This value does not change during the course of a transaction and does not reflect the individual start times of statements within a transaction.

```
LOCALTIME[(6)]
```

The number `6` (for the fractional seconds part of the time) is optional and does not affect the result. This function always returns a precision of 6. You may see results with fewer than 6 fractional seconds because trailing zeros are not returned. The return type is TIME.

```
premdb=# select localtime from sys.const;
     time
-----------------
 23:03:37.860575
(1 row)

premdb=# select localtime(6) from sys.const;
     time
-----------------
 23:03:40.257122
(1 row)
```

LOCALTIMESTAMP

Return the date and time when the current transaction starts. This value does not change during the course of a transaction and does not reflect the individual start times of statements within a transaction.

```
LOCALTIMESTAMP[(6)]
```

The number `6` (for the fractional seconds part of the timestamp) is optional and does not affect the result. This function always returns a precision of 6. The return type is TIMESTAMP.

```
premdb=# SELECT LOCALTIMESTAMP;
        timestamp
----------------------------
 2016-05-23 16:01:23.347666
(1 row)

premdb=# SELECT LOCALTIMESTAMP, matchday FROM match LIMIT 2;
     timestamp      |      matchday
--------------------+---------------------
 2016-05-23 16:02:03 | 2006-10-28 00:00:00
 2016-05-23 16:02:03 | 2006-09-16 00:00:00
(2 rows)
```

MAKE_DATE

Make a DATE value from three integer values that represent the `year`, `month`, and `day`:

```
MAKE_DATE(year, month, day)
```

For example:

```
premdb=# CREATE TABLE season_dates
(seasonid INT, start_date DATE, end_date DATE);
CREATE TABLE
premdb=# INSERT INTO season_dates
VALUES(3, MAKE_DATE(1994, 8, 15), MAKE_DATE(1995, 5, 20));
INSERT 0 1
premdb=# SELECT * FROM season_dates;
 seasonid | start_date |  end_date
----------+------------+------------
        3 | 1994-08-15 | 1995-05-20
(1 row)
```

MAKE_INTERVAL

Make an interval value by *adding* a series of values that represent the year, months, weeks, days, hours, minutes, and seconds. (You can specify all or some of these values.)

```
make_interval(years, months, weeks, days, hours, mins, secs)
```

- The `year`, `month`, `day`, `hour`, and `minute` values are integers. All of these values default to `0`.
- The `seconds` value is a double precision number that defaults to `0.0`.

For example:

```
premdb=# SELECT MAKE_INTERVAL(1995,1,1,1,1,1,1);
         make_interval
---------------------------------
 1995 years 1 mon 8 days 01:01:01
(1 row)
```

```
premdb=# SELECT MAKE_INTERVAL(1995);
 make_interval
---------------
 1995 years
(1 row)
```

```
premdb=# SELECT matchday, MAKE_INTERVAL(1,1,1,1,1,1,1)+matchday matchday_plus_one
FROM match WHERE seasonid=3 LIMIT 2;
     matchday       |  matchday_plus_one
--------------------+---------------------
 1994-12-26 00:00:00 | 1996-02-03 01:01:01
 1994-08-31 00:00:00 | 1995-10-08 01:01:01
(2 rows)
```

MAKE_TIME

Make a complete TIME value from a series of input values that represent hours, minutes, and seconds.

```
MAKE_TIME(hour, minute, seconds)
```

- The `hour` and `minute` values are integers.
- The `seconds` value is a double precision number.

For example:

```
premdb=# select make_time(8, 15, 23.5) from sys.const;
 make_time
------------
 08:15:23.5
(1 row)
```

The `seconds` value supports up to 6 fractional digits of precision. Additional digits are rounded up. For example:

```
premdb=# select make_time(8, 15, 23.5555555) from sys.const;
   make_time
-----------------
 08:15:23.555556
(1 row)
```

The following example extracts the hour value from a `TIMESTAMP` column, adds 15 to it, and uses it as the first part of the `MAKE_TIME` result:

```
premdb=# select make_time(extract(hour from matchday)::int +15, 00, 00)
from match where seasonid=12 limit 1;
 make_time
-----------
 15:00:00
(1 row)
```

MAKE_TIMESTAMP

Make a complete TIMESTAMP value from a series of datetime values.

```
MAKE_TIMESTAMP(year, month, day, hour, minute, seconds)
```

- The `year`, `month`, `day`, `hour`, and `minute` values are integers.
- The `seconds` value is a double precision number.

For example:

```
premdb=# UPDATE match
SET matchday=MAKE_TIMESTAMP(2011,8,20,15,00,00)
WHERE matchday='2011-08-20';
UPDATE 6

premdb=# select * from match where matchday='2011-08-20 15:00:00';
 seasonid |      matchday       | htid | atid | ftscore | htscore
----------+---------------------+------+------+---------+---------
       20 | 2011-08-20 15:00:00 |    2 |   73 | 0-2     | 0-0
       20 | 2011-08-20 15:00:00 |    3 |   55 | 3-1     | 2-0
       20 | 2011-08-20 15:00:00 |   14 |   93 | 2-1     | 0-1
       20 | 2011-08-20 15:00:00 |   18 |   82 | 0-1     | 0-1
       20 | 2011-08-20 15:00:00 |   39 |   77 | 0-1     | 0-0
       20 | 2011-08-20 15:00:00 |   40 |   95 | 0-0     | 0-0
(6 rows)
```

MAKE_TIMESTAMPTZ

Make a TIMESTAMP WITH TIME ZONE value from a series of datetime values, including an optional text field for the time zone.

```
MAKE_TIMESTAMPTZ(year, month, day, hour, minute, seconds [, 'timezone'])
```

- The `year`, `month`, `day`, `hour`, and `minute` values are integers.
- The `seconds` value is a double precision number.
- The `timezone` is a string.

For example:

```
premdb=# INSERT INTO timezone VALUES
(MAKE_TIMESTAMPTZ(2013, 7, 15, 8, 15, 23.5, 'GMT'));
INSERT 0 1
premdb=# SELECT * FROM timezone;
          tstz
--------------------------
 2013-07-15 01:15:23.5-07
(1 row)
```

MONTH

Platforms: All platforms

Parent topic: Datetime Functions

Return the month number (1 through 12) from a date or timestamp expression:

```
MONTH(expression)
```

For example:

```
premdb=# select month(matchday), matchday from match where htid=2 and atid=60;
 month |      matchday
-------+---------------------
     3 | 2010-03-06 00:00:00
(1 row)

premdb=# select month(date '2017-12-31') from sys.const;
 month
-------
    12
(1 row)

premdb=# select month(timestamp '2017-11-30 15:30:10') from sys.const;
 month
-------
    11
(1 row)
```

See also YEAR and DAY.

MONTHS_BETWEEN

Return the number of months between two dates. The result may be positive or negative.

```
MONTHS_BETWEEN ( date1, date2 )
```

The inputs may have `DATE`, `TIMESTAMP`, or `TIMESTAMPTZ` data types. If one of the inputs is `null`, the result is `null`.

The function returns fractional values as required. The difference between the year and month values of the dates yields the whole number part of the result. The fractional part derives from the day and timestamp values of the dates, assuming a 31-day month.

If the first date is later than the second date, the result is a positive number; otherwise, the result is negative. For example, depending on the order of date expressions in this example, the function returns a positive number or a negative number:

```
premdb=# select months_between(matchday,current_date) num_months, matchday from match where htid=2 and atid=60;
   num_months     |      matchday
------------------+---------------------
 -97.3225806451613 | 2010-03-06 00:00:00
(1 row)

premdb=# select months_between(current_date,matchday) num_months, matchday from match where htid=2 and atid=60;
   num_months    |      matchday
-----------------+---------------------
 97.3225806451613 | 2010-03-06 00:00:00
(1 row)
```

If both dates fall on the same date within two different months, such as November 10th and September 10th, or the last day of the month, such as April 30th and February 28th (in a non-leap year), the result is a whole number based on the year and month values of the dates. Fractional numbers of hours are ignored. For example:

```
premdb=# select months_between(date '2017-09-10 12:15:00', date '2017-11-10 18:20:30') from sys.const;
 months_between
----------------
             -2
(1 row)

premdb=# select months_between(date '2021-04-30', date '2021-02-28') from sys.const;
 months_between
----------------
              2
(1 row)

premdb=# select months_between(date '2000-04-30', date '2000-02-28') from sys.const;
  months_between
------------------
 2.06451612903226
(1 row)
```

Note that the last query specifies dates in 2000, which was a leap year.

NEXT_DAY

Based on a specific date or timestamp, return the next date when a given day of the week falls.

```
NEXT_DAY ( { date | timestamp | timestamptz }, day )
```

Use the `date` or `timestamp` keyword if you want to specify a literal value for the date. For example:

```
date '2017-11-20'
timestamp '2017-11-20 12:15:01
```

The following example returns the date of the next Monday after the current date:

```
premdb=# select current_date, next_day(current_date,'Monday');
   date    |  next_day
-----------+------------
 2017-11-20 | 2017-11-27
(1 row)
```

The following example returns the date of the first Thursday in the year 2000:

```
premdb=# select next_day(date '1999-12-31','Th') from sys.const;
  next_day
------------
 2000-01-06
(1 row)
```

If the day of the week you specify is the same as the day of the week in the given date, the function returns the next occurrence of that day.

Valid `day` values include several abbreviations:

- `Sunday`, `Su`, `Sun`
- `Monday`, `M`, `Mo`, `Mon`
- `Tuesday`, `Tu`, `Tues`
- `Wednesday`, `W`, `We`, `Wed`
- `Thursday`, `Th`, `Thu`, `Thurs`
- `Friday`, `F`, `Fr`, `Fri`
- `Saturday`, `S`, `Sa`, `Sat`

NOW()

Return the current date and time (as of the start of the current transaction).

```
NOW()
```

The parentheses are required.

```
premdb=# select now();
              now
-------------------------------
 2016-05-23 15:10:23.717069-07
(1 row)
```

The following example casts the result of the `NOW()` function to the `TIME` data type:

```
premdb=# select now()::time;
      now
----------------
 17:09:56.469561
(1 row)
```

OVERLAPS

The SQL OVERLAPS operator returns a Boolean value:

- `true` when two time periods (defined by their endpoints) overlap
- `false` when they do not overlap

```
(start1, end1) OVERLAPS (start2, end2)
(start1, interval1) OVERLAPS (start2, interval2)
```

You can specify the endpoints as pairs of dates, times, or timestamps. Alternatively, you can specify a date, time, or timestamp followed by an interval literal. When you specify a pair of values, the order does not matter; OVERLAPS automatically takes the earlier value of the pair as the start.

The following example specifies two pairs of date literals.

```
premdb=# select
('17-May-2016'::date, '13-Oct-2016'::date)
overlaps
('1-jun-2016'::date, '1-jul-2016'::date)
from sys.const;
 overlaps
----------
 t
(1 row)
```

The following example specifies two pairs of time literals.

```
premdb=# select
(time '12:00:00', time '13:00:00')
overlaps
(time '12:15:00', time '12:45:00')
from sys.const;
 overlaps
----------
 t
(1 row)
```

The following example specifies a pair of timestamp columns and a pair of timestamp literal values.

```
premdb=# select
(min(matchday),max(matchday))
overlaps
('09-September-1990'::timestamp, '09-September-2000'::timestamp)
from match;
 overlaps
----------
 t
(1 row)
```

In this example, one endpoint of each pair of arguments is an interval (a count of days in this case):

```
premdb=# select
(min(matchday),interval '100 days')
overlaps
('09-September-1999'::timestamp,interval '100 days')
from match;
 overlaps
----------
 f
(1 row)
```

Supported Date Parts

Platforms: All platforms

Parent topic: Datetime Functions

You can use the date parts in the table as arguments to the following functions:

- DATE_PART
- DATE_TRUNC
- EXTRACT
- DATEADD
- DATENAME

Date parts can be specified in uppercase or lowercase.

Datepart	Description	Notes
epoch	Epoch	Not supported for DATEADD.
millennia, mil, mils	Millennium	
centuries, century, cent, c	Century	
decades, decade, decs	Decade	
years, year, yrs, yr, y	Year	See datestyle Support for Year Values.
quarter, qtr	Quarter	
months, month, mons, mon	Month	Represented as `1` through `12` with `1` being January. Returns in sentence case for DATENAME (for example, `January`).
weeks, week, w	Week	Returns ISO 8601 week numbers, `1` to `53`.
days, day, d	Days	
dow	Day of Week	Not supported for DATEADD. Represented as `0` through `6` with `0` being Sunday. Returns in sentence case for DATENAME (for example, `Monday`).
doy	Day of Year	Not supported for DATEADD.
isodow	ISO 8601 Day of Week	Not supported for DATEADD.
isoyear	ISO 8601 Day of Year	

Datepart	Description	Notes
hours, hour, hrs, hr, h	Hours	
minutes, minute, mins, min, m	Minutes	
seconds, second, secs, sec, s	Seconds	
useconds, usecond, usecs, usec, us	Microseconds	
millisecon, mseconds, msecond, msecs, msec, ms	Milliseconds	
timezone	Time zone offset from UTC, in seconds (the difference between local time and UTC). Positive values correspond to time zones east of UTC, negative values to zones west of UTC.	Supported only for `EXTRACT` and `DATE_PART`. The `timezone`, `timezone_hour`, and `timezone_minute` date parts operate only on `timestamptz` columns and expressions. For `Infinity`, `+inf`, and `-inf`, `EXTRACT` and `DATE_PART` return `0`.
timezone_hour	Hour component of the time zone offset.	
timezone_minute	Minute component of the time zone offset.	

datestyle Support for Year Values

Yellowbrick supports either a 4-digit year (`YYYY`) or a 2-digit year (`YY`). However, the default `datestyle` setting is `MDY`, which expects a *2-digit month* to be the first value in the date string. Therefore, queries that attempt to cast strings to dates but specify 2-digit years at the beginning of the string return "value out of range" errors. For example:

```
yellowbrick=> show datestyle;
 DateStyle
-----------
 ISO, MDY
(1 row)

yellowbrick=> select days_between(now(), '99-10-20');
ERROR:  The month value of 99 is out of range
LINE 1: select days_between(now(), '99-10-20');
                                   ^
yellowbrick=> set datestyle = 'ISO, YMD';
SET
yellowbrick=> select days_between(now(), '99-10-20');
 days_between
--------------
         8629
(1 row)
```

See also ybload Date Formats.

TIMEOFDAY()

Return a VARCHAR string that specifies the date, time, and weekday.

```
TIMEOFDAY()
```

For example:

```
premdb=# select timeofday() from sys.const;
          timeofday
-------------------------------
 Mon Oct 16 12:35:26.039702 2017
(1 row)
```

TIMEZONE

Returns a timestamp for the specified time zone and timestamp value.

```
TIMEZONE ('timezone', { timestamp | timestamptz )
```

Specify the time zone as a UTC abbreviation or an IANA time zone name. For example:

```
'GMT'
'PDT'
'Singapore'
'US/Pacific'
```

The second argument must be a timestamp (with or without time zone).

For example:

```
premdb=# select localtimestamp, timezone('GMT', localtimestamp);
        timestamp          |           timezone
---------------------------+----------------------------
 2017-11-21 16:56:11.649456 | 2017-11-21 08:56:11.649456-08
(1 row)

premdb=# select localtimestamp, timezone('US/Pacific', localtimestamp);
        timestamp          |           timezone
---------------------------+----------------------------
 2017-11-21 16:56:24.481238 | 2017-11-21 16:56:24.481238-08
(1 row)
```

YEAR

Return the 4-digit year number from a date or timestamp expression:

```
YEAR(expression)
```

For example:

```
premdb=# select year(timestamp '11-30-2017 15:30:10.123456') from sys.const;
 year
------
 2017
(1 row)
```

See also MONTH and DAY.

YEARS_BETWEEN

Return the full number of years between two dates. The result may be positive or negative.

```
YEARS_BETWEEN (date1, date2)
```

The inputs may have `DATE`, `TIMESTAMP`, or `TIMESTAMPTZ` data types. If one of the inputs is `null`, the result is `null`. The return type of this function is `DOUBLE PRECISION`.

The result of this function is equivalent to the result of `EXTRACT(year FROM AGE(date1, date2))`.

If the first date is later than the second date, the result is a positive number; otherwise, the result is negative.

For example, depending on the order of date expressions in this example, the function returns a positive number or a negative number:

```
premdb=> select years_between(matchday,current_date) num_years, matchday from match where htid=2 and atid=60;
 num_years |      matchday
-----------+---------------------
       -13 | 2010-03-06 00:00:00
(1 row)

premdb=> select years_between(current_date,matchday) num_years, matchday from match where htid=2 and atid=60;
 num_years |      matchday
-----------+---------------------
        13 | 2010-03-06 00:00:00
(1 row)
```

Note that the result of this function is always a whole number. These two dates are 2 years, 11 months apart, but the answer is `2`:

```
yellowbrick=> select years_between('2023-11-30','2020-12-31') from sys.const;
 years_between
---------------
             2
(1 row)
```

See also MONTHS_BETWEEN and DAYS_BETWEEN.

Formatting Functions

Platforms: All platforms

Parent topic: SQL Functions and Operators

In this section:

This section covers functions that are used to coerce values to specific data types and formats.

FLOAT4, FLOAT8

Platforms: All platforms

Parent topic: Formatting Functions

Coerce numeric expressions to floating-point numbers. See Data Type Casting.

Formats for Datetime Values

Platforms: All platforms

Parent topic: Formatting Functions

The formats in the table (datetime masks) are supported in arguments to the following functions:

- TO_CHAR
- TO_DATE
- TO_TIMESTAMP

Format	Description
HH, HH12	Hour of day (01-12)
HH24	Hour of day (00-23)
MI	Minute (00-59)
SS	Second (00-59)
MS	Millisecond (000-999)
US	Microsecond (000000-999999)
SSSS	Seconds past midnight (0-86399)
AM, am, PM, pm, A.M., a.m., P.M., p.m.	Meridiem indicator (with and without periods)
Y,YYY	4-digit year with comma
YYYY, YYY, YY, Y	4-digit year, last 3, last 2, last 1
IYYY, IYY, IY, I	ISO 8601 week-numbering year (4 digits, last 3, last 2 last 1)
BC, bc, AD, ad, B.C., b.c., A.D., a.d.	Era indicator (with and without periods)
MONTH, Month, month	Uppercase, capitalized, or lowercase month name (blank-padded to 9 characters)
MON, Mon, mon	Abbreviated uppercase, capitalized, or lowercase month name (3 characters in English)
MM	Month number (01-12)
DAY, Day, day	Uppercase, capitalized, or lowercase day name (blank-padded to 9 characters)
DY, Dy, dy	Abbreviated uppercase, capitalized, or lowercase day name (3 characters in English)
DDD	Day of year (001-366)

Format	Description
IDDD	Day of ISO 8601 week-numbering year (001-371; day 1 of the year is Monday of the first ISO week)
DD	Day of month (01-31)
D	Day of the week, Sunday (1) to Saturday (7)
FM	Fill mode prefix (a pattern modifier that suppresses trailing zeros and padding blanks)
FX	This prefix must be the first item in the format string. Normally, `TO_TIMESTAMP` and `TO_DATE` functions ignore multiple blank spaces. If you specify the `FX` prefix, these functions do not ignore blank spaces. For example: `FX Month DD Day`. This prefix is not supported with the `TO_CHAR` function.
ID	ISO 8601 day of the week, Monday (1) to Sunday (7)
W	Week of month (1-5) (the first week starts on the first day of the month)
WW	Week number of year (1-53) (the first week starts on the first day of the year)
IW	Week number of ISO 8601 week-numbering year (01-53; the first Thursday of the year is in week 1)
CC	Century (2 digits) (the twenty-first century starts on 2001-01-01). The `CC` pattern is ignored when the year is specified.
J	Julian Day (integer days since November 24, 4714 BC at midnight UTC) Not supported by TO_CHAR.
Q	Quarter (ignored by TO_DATE and TO_TIMESTAMP)
RM, rm	Month in uppercase or lowercase Roman numerals (I-XII and I=January or i-xii and i=January)Not supported by TO_CHAR.
TH, th	Ordinal number suffix (a pattern modifier). For example: `DDth` or `DDTH`
TZ, tz	Uppercase or lowercase time-zone name (the long form of the name. For example: `PST8PDT` is returned, not `PST`.
OF	Time-zone offsetNot supported by TO_CHAR.

To convert Julian dates to the `yyyy-mm-dd` format, store the Julian dates as VARCHAR with a `J` preceeding the value. For example:

```
premdb=# create table yb100 (julian varchar(10));
CREATE TABLE

premdb=# insert into yb100 values ('J2456783');
INSERT 0 1

premdb=# SELECT julian::DATE from yb100;
   julian
------------
 2014-05-05
(1 row)
```

Using the same table, Julian dates can be converted to the `yyyy-mm-dd` format using the `TO_DATE` function:

```
premdb=# SELECT TO_DATE(2456783::text, 'J');
  to_date
------------
 2014-05-05
(1 row)
```

Separators in Date Formats

The basic date formats work with or without separator characters and with uppercase and lowercase letters; however, the literal date values and the corresponding date masks must match. Do not specify separators in the mask that do not exist in the input string, and do not use additional separator characters, such as extra spaces or hyphens, between format elements unless the input string also contains those characters.

For example, the following date masks are valid for a date such as `30 May 2016`:

```
DD Mon YYYY
dd mon yyyy
```

However, the following masks will fail for the same input string:

```
DD-Mon-YYYY
DDMonYYYY
ddmonyyyy
dd  mon  yyyy
```

Year Formats

Do not use a four-digit year in the mask if the input string contains fewer digits. For example:

```
premdb=# SELECT TO_DATE('30 May 16 ', 'dd mon yyyy') from sys.const;
  to_date
------------
 0016-05-30
(1 row)
```

The correct mask in this case would be `dd mon yy`.

Conversely, if the input value contains a four-digit year, the mask may specify `YYYY`, `YYY`, `YY`, or `Y`. For example:

```
premdb=# SELECT TO_DATE('30-May-2016', 'DD-Mon-Y') from sys.const;
  to_date
------------
 2016-05-30
(1 row)
```

Handling of US Format (Microseconds)

The TO_TIMESTAMP function ignores any leading zeros that appear in the format for microseconds. For example:

```
premdb=# select to_timestamp('20180228 11 59 59 000999 AM', 'YYYYMMDD HH12 MI SS US AM') from sys.const;
       to_timestamp
----------------------------
 2018-02-28 11:59:59.999-08
(1 row)
```

Formats for Numeric Values

Platforms: All platforms

Parent topic: Formatting Functions

The following formats are supported in the second argument of the TO_NUMBER function.

Note: Any formats not listed here are not supported. You must use these formats as a template.

Format	Example	Description
`9`	`999`	Each `9` represents the position of a digit. For example, a format of `999` expects *up to three digits* in a row. The function returns a result if the input string has 0, 1, 2, or 3 digits. Digits formatted as `9` are required only when they appear before a dot (`.`) *and* there is a leading `0`. In all other cases, a `9` in the format does not have to map to a digit in the input string.
`0`	`000`	Each `0` also represents the position of a digit. However, `0` values must always map to a digit in the input string, and they cause `9` values that follow them to also behave as `0` values. For example, a format of `099` expects *exactly three digits* in a row. The `0` must map to a digit in the number string, and two additional digits must follow. A format of `9990000` expects a minimum of four digits: the three `9` values are optional but the four `0` values are required. An input string with less than four digits results in an error.
`.` (period) or `D`	`999.99` or `999D99`	Location of the decimal point. You can specify only one period character in the format string.
`,` (comma) or `G`	`9G999D99` or `9,999D99`	Location of the thousands separator in a number. A format string cannot begin with a comma, and a comma cannot appear to the right of a decimal character or period.
`PR`	`9G999D99PR`	The number string represents a negative value, as defined by the use of angle brackets (for example: `<123.456>`). `PR` must be the last element in the format string and cannot be used in combination with any other signs.
`MI`	`9G999D99MI`	Presence of a minus sign, for numbers less than 0. `MI` must be the last element in the format string. Only one sign element can be used in a format string.
`S`	`9G999D99S`	Presence of a sign anchored to the number, either `+` or `-` . `S` must be the first or last element in the format string. Only one sign element can be used in a format string.
`V`	`9G999V99`	This format divides the input values by `10^n` , where `n` is the number of digits following `V` . `V` cannot be combined with a decimal point (for example, `99.9V99`).

INT2, INT4, INT8

Platforms: All platforms

Parent topic: Formatting Functions

Coerce numeric expressions to integers. See Data Type Casting.

TO_CHAR

This function converts date, time, timestamp, numeric, and boolean expressions to character strings. TO_CHAR is intended for cases where explicit casting is not feasible.

```
TO_CHAR( { date | time | timestamp | timestamptz }, datetime_format)
TO_CHAR( { smallint | integer | bigint | real | float | decimal }, numeric_format [, decsep, groupsep ] )
TO_CHAR(boolean, boolean_format)
```

You can use column references or literal values for the first argument to the function. The second argument must be a format string that specifies how to interpret each input value.

For example, valid date formats for `2017-12-19` include `YYYY-MM-DD` and `DD-MM-YYYY`. The `INTERVAL` data type is not supported.

datetime_format

See Formats for Datetime Values.

Note: `TO_CHAR(time)` only supports a subset of the `datetime_format` patterns, specifically:

Pattern	Description
HH	Hour of day (01-12)
HH12	Hour of day (01-12)
HH24	Hour of day (00-23)
MI	Minute (00-59)
SS	Second (00-59)
MS	Millisecond (000-999)
US	Microsecond (000000-999999)
AM, am, PM, pm	Meridiem indicator (without periods)
A.M., a.m., P.M., p.m.	Meridiem indicator (with periods)

numeric_format

See Numeric Formatting.

decsep, groupsep

`TO_CHAR` with numeric data types supports optional characters for the decimal separator and group separator (`D` and `G` in the numeric format string). For example, you can use a comma for the decimal point and commas to separate groups. The default decimal separator is a period, and the default group separator is a comma. You must specify both or none of these separators. These arguments only apply if the appropriate characters ('`G`','`D`', '`,`', '`.`') appear in the format string.

boolean_format

See Boolean Formatting

In a format string, patterns are recognized and replaced with appropriately formatted data; any literal text in the string that is not recognized as a template pattern is copied into the result as is.

Examples with Dates and Timestamps

```
premdb=# select to_char(timestamp '1992-08-01 00:00:00', 'YYYY-MM-DD HH12:MI:SS')
from sys.const;
      to_char
--------------------
 1992-08-01 12:00:00
(1 row)
```

```
premdb=# select to_char(matchday, 'YYYY-MM-DD') from match
where seasonid=14 and htid=32 order by 1;
  to_char
------------
 2005-08-13
 2005-08-23
 2005-09-17
 2005-10-01
 2005-10-22
...
```

```
premdb1=# select matchday, to_char(matchday, 'FMDay, Month DD, YYYY') from match where seasonid=21 and htid=2 order by 1;
     matchday       |           to_char
--------------------+------------------------------
 2012-08-18 00:00:00 | Saturday, August 18, 2012
 2012-09-15 00:00:00 | Saturday, September 15, 2012
 2012-09-29 00:00:00 | Saturday, September 29, 2012
 2012-10-27 00:00:00 | Saturday, October 27, 2012
 2012-11-10 00:00:00 | Saturday, November 10, 2012
...
```

Examples with Numbers

Return character strings for `avg_att` values in the `team` table:

```
premdb=# select name, to_char(avg_att, '99.99') from team where avg_att>0.0 order by 2 desc;
        name         | to_char
---------------------+---------
 Manchester United   |  75.29
 Arsenal             |  59.94
 Manchester City     |  54.04
 Newcastle United    |  49.75
 Liverpool           |  43.91
 Sunderland          |  43.07
...
```

Return the top five `capacity` values from the `team` table as exponents:

```
premdb=# select capacity, to_char(capacity,'EEEE') from team order by 1 desc limit 5;
 capacity | to_char
----------+---------
   75635 |  8e+04
   60260 |  6e+04
```

```
   55097 |  6e+04
   52405 |  5e+04
   49000 |  5e+04
(5 rows)
```

Specify non-default decimal and group separators for a numeric constant. In this example, the decimal separator is a comma and the group separator is a space.

```
premdb=# select to_char(1234567.77, '9G999G999D99', ',', ' ');
   to_char
---------------
  1 234 567,77
(1 row)
```

Example with Boolean

The following example returns a `1` (`true`) or a `0` (`false`) for the values in a boolean column:

```
premdb=# select c1, to_char(c1, '1') from booltable;
 c1 | to_char
----+---------
 t  | 1
 f  | 0
 t  | 1
 f  | 0
(4 rows)
```

Numeric Formatting

Pattern	Description
`9`	Digit position (can be dropped if insignificant)
`0`	Digit position (will not be dropped, even if insignificant)
`.` (period)	Decimal point
`,` (comma)	Thousands separator
`PR`	Angle brackets for negative values. For example: `to_char(-12, '9999PR')` returns `<12>`.
`S`	Sign anchored to number (uses locale)
`D`	Decimal point (uses locale)
`FM`	Fill mode prefix (a pattern modifier). When toggled on, the leading zeros of all numbers will be suppressed until the modifier is used again. The modifier is not applied to fractional numbers, such as milliseconds and microseconds.
`G`	Group separator (uses locale)
`MI`	Minus sign in specified position (if number < 0). The `MI` and `PL` patterns cannot both be used in the same TO_CHAR function call.
`PL`	Plus sign in specified position (if number > 0). The `MI` and `PL` patterns cannot both be used in the same TO_CHAR function call.
`SG`	Plus/minus sign in specified position

Pattern	Description
`TH` or `th`	Ordinal number suffix (a pattern modifier).
`EEEE`	Exponent for scientific notation

Note: The currency symbol (`L`) is not supported.

Boolean Formatting

Pattern	Description
`T` or `F`	`T` for true, `F` for false
`t` or `f`	`t` for true, `f` for false
`TRUE` , `FALSE`	`TRUE` for true, `FALSE` for false
`true` , `false`	`true` for true, `false` for false
`1` or `0`	`1` for true, `0` for false
`Y` or `N`	`Y` for true, `N` for false
`y` or `n`	`y` for true, `n` for false
`YES` or `NO`	`YES` for true, `NO` for false
`yes` or `no`	`yes` for true, `no` for false

TO_DATE

Return a DATE value from a character string.

```
TO_DATE(string, format)
```

The input string is the first argument, and its format is the second argument. You must specify the format of the input string. See Formats for Datetime Values.

For example:

```
premdb=# SELECT TO_DATE('30 May 2016', 'DD Mon YYYY');
  to_date
------------
 2016-05-30
(1 row)
```

```
premdb=# SELECT TO_DATE('May 31 2016','Mon DD YYYY');
  to_date
------------
 2016-05-31
(1 row)
```

The following example converts a string to a DATE, then inserts it into a TIMESTAMP column:

```
premdb=# INSERT INTO match(matchday) SELECT TO_DATE('May 31 2016','Mon DD YYYY');
INSERT 0 1
premdb=# SELECT * FROM match WHERE matchday='2016-05-31';
 seasonid |      matchday       | htid | atid | ftscore | htscore
----------+---------------------+------+------+---------+---------
         | 2016-05-31 00:00:00 |      |      |         |
(1 row)
```

TO_NUMBER

Return a numeric value from a character string, based on a specified format.

```
TO_NUMBER(string, format)
```

string

The input string is the first argument, and its format is the second argument. The first argument must be a literal character string, an expression that evaluates to a character string, or a value that is implicitly cast to a character string.

Space characters are allowed only at the beginning of the input string, before any data character.

The input string must be shorter than or equal to the format string. For example, the following query returns an expected error because the input string has three digits but the format string has only two:

```
yellowbrick=# select to_number('123','99') from sys.const;
ERROR:  Invalid input number for TO_NUMBER
```

format

You must specify the format of the input string, using a supported list of elements. See Formats for Numeric Values for formatting rules and examples. The format cannot be an empty string (`''`).

The `DECIMAL` return type of the function matches the precision and scale of the format string. You cannot use the `TO_NUMBER` function to return a value with more than 38 digits. See DECIMAL for details about precision and scale rules for Yellowbrick `DECIMAL` values.

Note: This function is implemented for Oracle compatibility, not PostgreSQL compatibility.

Examples

The following `TO_NUMBER` examples demonstrate the behavior of the elements listed in Formats for Numeric Values.

Return a number that includes a decimal point:

```
premdb=# select to_number('123.45','999D99') from sys.const;
 to_number
-----------
    123.45
(1 row)

premdb=# select to_number('123.45','999.99') from sys.const;
 to_number
-----------
    123.45
(1 row)
```

Return a number that has a thousands separator and a decimal point:

```
premdb=# select to_number('1,123.45','9G999D99') from sys.const;
 to_number
-----------
   1123.45
(1 row)

premdb=# select to_number('1,123.45','9,999D99') from sys.const;
 to_number
-----------
   1123.45
(1 row)
```

Return a negative number that has angle brackets:

```
premdb=# select to_number('<1,123.45>','9G999D99PR') from sys.const;
 to_number
-----------
  -1123.45
(1 row)
```

Return a negative number that has a minus sign:

```
premdb=# select to_number('-1,123.45','S9G999D99') from sys.const;
 to_number
-----------
  -1123.45
(1 row)
```

Return a negative number that ends with a minus sign:

```
premdb=# select to_number('123-','999MI') from sys.const;
 to_number
-----------
     -123
(1 row)
```

Use a combination of a `0` and two `9` digits:

```
premdb=# select to_number('12.34', '990.99') from sys.const;
 to_number
-----------
    12.34
(1 row)
```

The following example shows the basic difference between `9` and `0` values:

```
premdb=# select to_number('12.34','9999D9999');
 to_number
-----------
   12.3400
(1 row)

premdb=# select to_number('12.34','0000D0000');
ERROR:  Invalid input number for TO_NUMBER
```

Attempt to use leading `0` values for numbers where an exact number of digits is required:

```
premdb=# select to_number('12.34', '099.99') from sys.const;
ERROR:  Invalid input number for TO_NUMBER
premdb=# select to_number('12.34', '0.99') from sys.const;
ERROR:  Invalid input number for TO_NUMBER
```

Attempt to convert a string that contains the `$` sign:

```
premdb=# select to_number('$12.34', '99.99') from sys.const;
ERROR:  Invalid input number for TO_NUMBER
```

The following example uses an expression on a character column (`ftscore`) in the first argument to the `TO_NUMBER` function:

```
premdb=# select ftscore, to_number(substr(ftscore,1,1), '9') from match order by 1 desc limit 3;
 ftscore | to_number
---------+-----------
 9-1     |         9
 9-0     |         9
 8-2     |         8
(3 rows)
```

```
premdb=# select ftscore, to_number(substr(ftscore,1,1), '9D9') from match order by 1 desc limit 3;
 ftscore | to_number
---------+-----------
 9-1     |       9.0
 9-0     |       9.0
 8-2     |       8.0
(3 rows)
```

The following example shows that the first argument may be implicitly cast to a string, then returned as a number:

```
premdb=# select seasonid, to_number(seasonid*10, '99999') from season order by seasonid desc limit 3;
 seasonid | to_number
----------+-----------
       25 |       250
       24 |       240
       23 |       230
(3 rows)
```

Note that the expression `seasonid*10` , where `seasonid` is an integer column, does not require an explicit cast.

TO_TIMESTAMP

Return a TIMESTAMPTZ value from either a character string or a UNIX epoch value.

```
TO_TIMESTAMP(string, format)
TO_TIMESTAMP(epoch)
```

When an input string is the first argument, its format is the second argument. You must specify the format of the input string. See Formats for Datetime Values.

For example:

```
premdb=# SELECT TO_TIMESTAMP('May 30 2016','Mon DD YYYY');
     to_timestamp
------------------------
 2016-05-30 00:00:00-07
(1 row)
```

The following example shows how to use this function with an epoch value (number of seconds since January 1st, 1970):

```
premdb=# SELECT TO_TIMESTAMP(1464700000) FROM sys.const;
     to_timestamp
------------------------
 2016-05-31 06:06:40-07
(1 row)
```

The following example converts a string to a TIMESTAMPTZ value, then inserts it into a TIMESTAMP column:

```
premdb=# INSERT INTO match(matchday) SELECT TO_TIMESTAMP('May 31 2016','Mon DD YYYY');
INSERT 0 1
premdb=# SELECT * FROM match WHERE matchday='2016-05-31';
 seasonid |      matchday       | htid | atid | ftscore | htscore
----------+---------------------+------+------+---------+---------
         | 2016-05-31 00:00:00 |      |      |         |
(1 row)
```

Note: You cannot repeat format elements. For example, because the function repeats `SS`, the following query returns an error:

```
premdb=# select to_timestamp('2018 02 01 12 15 15', 'YYYY MM DD HH MI SS SS') from sys.const;
ERROR:  Invalid datetime format YYYY MM DD HH MI SS SS
DETAIL:   [reason => Invalid datetime format string. Reason: Duplicate token SS found in format string. Duplicate tokens are not a
```

Geospatial Functions

Platforms: All platforms

Parent topic: SQL Functions and Operators

In this section:

This section covers geospatial functions. All functions are `NULL` strict unless stated otherwise.

ST_AREA

Calculates the area of a `GEOGRAPHY` object in square meters.

This function supports two calculation modes:

- **Spheroid-based (default):** Provides a more precise but slower calculation
- **Sphere-based:** Offers a faster but less accurate calculation

See also:

- `ST_LENGTH`

Syntax

```sql
ST_AREA(<input>, <use_spheroid> = true)
```

Arguments

- `<input>`: The `GEOGRAPHY` object to measure
- `<use_spheroid>`: An optional `BOOLEAN` to control the calculation mode:
  - `TRUE` (default): Uses the spheroidal model for higher precision
  - `FALSE`: Uses the spherical model for faster computation

Returns

Returns a `FLOAT8` value representing the area in square meters.

Returns `0` for invalid or non-polygonal inputs such as points and lines.

Example

```sql
SELECT ST_Area('POLYGON((0 0, 1 0, 1 1, 0 1, 0 0))'::GEOGRAPHY);
-- 12308778361.4695 -- area in square meters using spheroid

SELECT ST_Area('POLYGON((0 0, 1 0, 1 1, 0 1, 0 0))'::GEOGRAPHY, false);
-- 12364031798.5177 -- area in square meters using sphere
```

ST_ASBINARY

Converts a `GEOGRAPHY` object to its Well-Known Binary (WKB) representation.

This function is automatically invoked when casting a `GEOGRAPHY` to `VARBINARY`.

Note: The WKB format preserves full numeric precision but does not include the SRID. For an SRID-aware binary format, use `ST_ASEWKB`.

See also:

- `ST_ASEWKB`
- `ST_GEOGFROMWKB`

Syntax

```sql
ST_ASBINARY(<input>)
```

Arguments

- `<input>`: The `GEOGRAPHY` object to be converted

Returns

Returns a `VARBINARY` containing the WKB representation of the input `GEOGRAPHY`.

Example

```sql
SELECT ST_AsBinary('POINT(1 1)'::GEOGRAPHY);
-- '\x0101000000000000000000f03f000000000000f03f'

-- Casting works the same way
SELECT 'POINT(1 1)'::GEOGRAPHY::VARBINARY;
-- '\x0101000000000000000000f03f000000000000f03f'
```

ST_ASEWKB

Returns the Extended Well-Known Binary (EWKB) representation of a `GEOGRAPHY` object, including SRID metadata.

The EWKB format preserves full numeric precision, preventing rounding errors that can occur with text formats such as Well-Known Text (WKT).

See also:

- `ST_ASBINARY`
- `ST_GEOGFROMWKB`

Syntax

```sql
ST_ASEWKB(<input>)
```

Arguments

- `<input>`: The `GEOGRAPHY` object to be converted

Returns

Returns a `VARBINARY` containing the EWKB representation of the input `GEOGRAPHY`.

Example

```sql
SELECT ST_ASEWKB('POINT(1 1)'::GEOGRAPHY);
-- \x0101000020e6100000000000000000f03f000000000000f03f
```

ST_ASEWKT

Returns the Extended Well-Known Text (EWKT) representation of a `GEOGRAPHY` object, which includes the SRID prefix.

See also:

- `ST_ASTEXT`
- `ST_GEOGFROMTEXT`

Syntax

```sql
ST_ASEWKT(<input>)
```

Arguments

- `<input>`: The `GEOGRAPHY` object to be converted to EWKT format

Returns

A `VARCHAR` containing the EWKT representation of the input `GEOGRAPHY`, prefixed with its SRID.

Example

```sql
SELECT ST_AsEWKT(ST_Point(1, 1)::GEOGRAPHY);
-- 'SRID=4326;POINT(1 1)'
```

ST_ASGEOJSON

Converts a `GEOGRAPHY` object into a GeoJSON representation that follows RFC 7946 specifications. Returns the `GEOGRAPHY` as a GeoJSON `Geometry Object`.

See also:

- `ST_ASTEXT`
- `ST_ASBINARY`

Syntax

```sql
ST_ASGEOJSON(<input>)
```

Arguments

- `<input>`: A `GEOGRAPHY` object to convert

Returns

Returns a `JSON` value containing the GeoJSON representation of the input `GEOGRAPHY`.

Example

```sql
SELECT ST_AsGeoJSON('POINT(1 1)'::GEOGRAPHY);
-- {"type":"Point","coordinates":[1,1]}
```

Note: This function is equivalent to casting `GEOGRAPHY` to `JSON` using the `::JSON` syntax.

ST_ASGML

Returns the Geography Markup Language (GML) representation of a `GEOGRAPHY` object.

See also:

- `ST_ASGEOJSON`
- `ST_GEOGFROMGML`

Syntax

```sql
ST_ASGML(<input>)
```

Arguments

- `<input>`: A `GEOGRAPHY` object to be converted

Returns

Returns a `VARCHAR` containing the GML representation of the input `GEOGRAPHY`.

Example

```sql
SELECT ST_ASGML('POINT(1 1)'::GEOGRAPHY);
-- '<gml:Point srsName="EPSG:4326"><gml:coordinates>1,1</gml:coordinates></gml:Point>'
```

ST_ASTEXT

Returns the OGC Well-Known Text (WKT) representation of the `GEOGRAPHY`. This function is automatically invoked when casting `GEOGRAPHY` to `VARCHAR`.

See also:

- `ST_ASEWKT`
- `ST_GEOGFROMTEXT`

Syntax

```sql
ST_ASTEXT(<input>)
```

Arguments

- `<input>`: The `GEOGRAPHY` value to be converted to WKT

Returns

Returns `VARCHAR` containing the WKT representation of the input `GEOGRAPHY`.

Example

```sql
SELECT ST_AsText('0101000020e6100000000000000000f03f000000000000f03f'::GEOGRAPHY);
-- 'POINT(1 1)'
```

ST_BUFFER

Computes a `POLYGON` or `MULTIPOLYGON` representing all points within a specified distance from a `GEOGRAPHY` object.

See also:

- `ST_UNION`
- `ST_INTERSECTION`

Syntax

```sql
ST_BUFFER(<input>, <radius>, <num_seg_quarter_circle> = 8)
```

Arguments

- `<input>`: The `GEOGRAPHY` to calculate the buffer around
- `<radius>`: Distance in meters. Negative values shrink the `GEOGRAPHY` instead of expanding it
- `<num_seg_quarter_circle>`: Number of line segments used to approximate a quarter circle (default 8)

Returns

Returns a `GEOGRAPHY` representing the buffer area. Special cases include:

- A negative radius applied to points or lines returns an empty `GEOGRAPHY`
- A negative radius applied to polygons may result in an empty `GEOGRAPHY` if the shape is completely removed

Example

```sql
SELECT ST_AsText(ST_Buffer('POINT(0 0)'::GEOGRAPHY, 10, 2));
-- POLYGON((0.00009 0,0.00006 -0.00006,0 -0.00009,-0.00006 -0.00006,-0.00009 0,-0.00006 0.00006,0 0.00009,0.00006 0.00006,0.00009
```

ST_CENTROID

Computes the geometric center (centroid) of a `GEOGRAPHY` object.

- For `[MULTI]POINT` objects, the centroid is the arithmetic mean of the input coordinates
- For `[MULTI]LINESTRING` objects, the centroid is calculated using the weighted length of each line segment
- For `[MULTI]POLYGON` objects, the centroid is determined based on the area

Note: The centroid is not always guaranteed to be within the original shape

Syntax

```sql
ST_CENTROID(<input>)
```

Arguments

- `<input>`: A `GEOGRAPHY` value to compute the centroid of

Returns

Returns a `GEOGRAPHY` of subtype `ST_Point` representing the centroid.

For mixed-dimension input, the result is equal to the centroid of the component shapes of highest dimension (since the lower-dimension shapes contribute zero "weight" to the centroid).

Example

```sql
SELECT ST_AsText(ST_Centroid('POLYGON((0 0, 2 0, 2 2, 0 2, 0 0))'::GEOGRAPHY));
-- POINT(1 1)
```

ST_CONTAINS

Returns `TRUE` if one `GEOGRAPHY` object contains another. A `GEOGRAPHY` A contains B if and only if:

- All points of B lie inside A (either within the interior or on the boundary)
- The interiors of A and B share at least one point in common

This function has the following properties:

- **Reflexive:** Every `GEOGRAPHY` object contains itself
- **Antisymmetric:** If `ST_CONTAINS(A, B) = TRUE` and `ST_CONTAINS(B, A) = TRUE`, then A and B are topologically equal
- **Converse of `ST_WITHIN`:** `ST_CONTAINS(A, B) = ST_WITHIN(B, A)`

See also:

- `ST_WITHIN`
- `ST_COVERS`

Syntax

```sql
ST_CONTAINS(<a>, <b>)
```

Arguments

- `<a>` and `<b>`: The `GEOGRAPHY` objects to check for containment relationship

Returns

Returns `TRUE` if `<a>` contains `<b>`, `FALSE` otherwise.

Example

```sql
SELECT ST_CONTAINS('POLYGON((0 0, 0 2, 2 2, 2 0, 0 0))'::GEOGRAPHY, 'POINT(1 1)'::GEOGRAPHY);
-- TRUE

SELECT ST_CONTAINS('POLYGON((0 0, 0 2, 2 2, 2 0, 0 0))'::GEOGRAPHY, 'POINT(3 3)'::GEOGRAPHY);
-- FALSE
```

ST_COVERS

Returns `TRUE` if every point in `GEOGRAPHY` B lies inside (intersects the interior or boundary of) `GEOGRAPHY` A. That is, it ensures that no point of B lies in the exterior of A.

This function has the following properties:

- **Reflexive:** Every `GEOGRAPHY` object covers itself
- **Antisymmetric:** If `ST_COVERS(A, B) = TRUE` and `ST_COVERS(B, A) = TRUE`, then A and B are topologically equal
- **Converse of `ST_COVEREDBY`:** `ST_COVERS(A, B) = ST_COVEREDBY(B, A)`

Generally, this function should be used instead of `ST_CONTAINS` because it has a simpler definition that does not require the interiors to intersect.

See also:

- `ST_CONTAINS`
- `ST_WITHIN`

Syntax

```sql
ST_COVERS(<a>, <b>)
```

Arguments

- `<a>` and `<b>`: The `GEOGRAPHY` objects to check for a covers relationship

Returns

Returns `TRUE` if `<a>` covers `<b>`, otherwise returns `FALSE`

Example

```sql
SELECT ST_COVERS('POLYGON((0 0, 0 2, 2 2, 2 0, 0 0))'::GEOGRAPHY, 'POINT(1 1)'::GEOGRAPHY);
-- TRUE

SELECT ST_COVERS('POLYGON((0 0, 0 2, 2 2, 2 0, 0 0))'::GEOGRAPHY, 'POINT(3 3)'::GEOGRAPHY);
-- FALSE
```

ST_CROSSES

Returns `TRUE` if two `GEOGRAPHY` objects have intersecting interiors but do not share all interior points. The intersection must have a dimension less than the maximum dimension of the input `GEOGRAPHY` objects.

This function is **symmetric**, meaning:

- `ST_CROSSES(a, b)` = `ST_CROSSES(b, a)`

See also:

- `ST_INTERSECTS`

Syntax

```sql
ST_CROSSES(<a>, <b>)
```

Arguments

- `<a>` and `<b>`: The two `GEOGRAPHY` objects to compare

Returns

Returns a `BOOLEAN`:

- `TRUE` if the `GEOGRAPHY` objects cross
- `FALSE` if they do not cross, or in cases involving Point/Point or Area/Area comparisons

Behavior

Input Types	Result
Point/Line	`TRUE` if the point lies on the line's interior
Point/Area	`TRUE` if the point lies within the area's interior
Line/Line	`TRUE` if the lines intersect at one or more points
Line/Area	`TRUE` if the line intersects the area's interior
Area/Area	Always `FALSE`
Point/Point	Always `FALSE`

ST_DIFFERENCE

Returns a `GEOGRAPHY` representing the part of `GEOGRAPHY` A that does not intersect `GEOGRAPHY` B. This is equivalent to `A - ST_INTERSECTION(A, B)`.

If A is completely contained within B, an empty `GEOGRAPHY` of the appropriate type is returned.

See also:

- `ST_INTERSECTION`
- `ST_UNION`

Syntax

```sql
ST_DIFFERENCE(<a>, <b>)
```

Arguments

- `<a>`: The `GEOGRAPHY` object to subtract from
- `<b>`: The `GEOGRAPHY` object to subtract

Returns

Returns a `GEOGRAPHY` representing all parts of `<a>` that do not intersect with `<b>`

Behavior

`<a>` subtype	`<b>` subtype	Behavior
`ST_Point`	`ST_Point`	Returns empty if `<b>` equals `<a>`, otherwise returns `<a>`
`ST_Point`	`ST_Linestring`	Returns `<a>` if it is not on the line of `<b>`, otherwise empty
`ST_Point`	`ST_Polygon`	Returns `<a>` if it does not intersect `<b>`, otherwise empty
`ST_Linestring`	`ST_Point`	Always returns `<a>`
`ST_Linestring`	`ST_Linestring`	Returns an `ST_GeometryCollection` of parts of `<a>` that are not in `<b>`. May return a single `ST_Linestring` if the lines fully overlap, except in one case
`ST_Linestring`	`ST_Polygon`	Returns an `ST_GeometryCollection` of all areas of `<b>` that do not touch the polygon in `<b>`
`ST_Polygon`	`ST_Point`	Always returns `<a>`

`<a>` subtype	`<b>` subtype	Behavior
`ST_Polygon`	`ST_Linestring`	Returns an `ST_GeometryCollection` containing a potentially "cut up" set of `ST_polygon` sliced up by the linestring in `<b>`. Can also return `<a>` if no part of `<b>` is inside `<a>`. A part of a linestring touching the polygon does not subtract from `<b>`
`ST_Polygon`	`ST_Polygon`	Returns an `ST_GeometryCollection` slicing up `<a>` by the areas not in `<b>`. A potentially complex shape

ST_DISTANCE

Returns the minimum geodesic distance between two `GEOGRAPHY` objects in meters.

By default, the calculation is performed on the spheroid defined by the SRID. If `use_spheroid` is set to `FALSE`, a faster spherical calculation is used instead.

See also:

- `ST_DWITHIN`

Syntax

```sql
ST_DISTANCE(<a>, <b>, <use_spheroid> = true)
```

Arguments

- `<a>` and `<b>`: The two `GEOGRAPHY` objects to measure the distance between
- `<use_spheroid>`: An optional boolean to control the calculation mode:
  - `TRUE` (default): Uses the spheroidal model for higher precision
  - `FALSE`: Uses the spherical model for faster computation

Returns

Returns a `FLOAT8` value representing the distance in meters between the two `GEOGRAPHY` objects. Returns `NULL` if the calculation is not possible.

Example

```sql
SELECT ST_Distance('POINT(0 0)'::GEOGRAPHY, 'POINT(1 1)'::GEOGRAPHY);
-- 156899.56829134

SELECT ST_Distance('POINT(0 0)'::GEOGRAPHY, 'POINT(1 1)'::GEOGRAPHY, false);
-- 157249.597768505
```

ST_DWITHIN

Returns `TRUE` if the `GEOGRAPHY` objects are within the specified distance in meters.

See also:

- `ST_DISTANCE`

Syntax

```sql
ST_DWITHIN(<a>, <b>, <d>, <use_spheroid> = true)
```

Arguments

- `<a>` , `<b>` : The `GEOGRAPHY` objects to check for proximity
- `<d>` : The distance in meters (must be `>= 0`)
- `<use_spheroid>` : An optional boolean to control calculation accuracy:
  - `TRUE` (default): Uses the spheroidal model for higher precision
  - `FALSE` : Uses the spherical model for faster computation

Returns

Returns a `BOOLEAN` :

- `TRUE` if `<a>` and `<b>` are within the distance `<d>`
- `FALSE` if `<a>` and `<b>` are farther apart than `<d>` or if the inputs are invalid

Example

```sql
SELECT ST_DWithin('POINT(1 1)'::GEOGRAPHY, 'POINT(1.1 1.1)'::GEOGRAPHY, 16000);
-- TRUE
```

ST_ENDPOINT

Returns the last point of a `LINESTRING`.

See also:

- `ST_STARTPOINT`

Syntax

```sql
ST_ENDPOINT(<input>)
```

Arguments

- `<input>`: The `GEOGRAPHY` linestring object

Returns

Returns a `POINT` representing the last point of the input `LINESTRING`. Returns `NULL` if the input is not a `LINESTRING`.

Example

```sql
SELECT ST_AsText(ST_EndPoint('LINESTRING(1 1, 2 2, 3 3)'::GEOGRAPHY));
-- POINT(3 3)
```

ST_ENVELOPE

Returns a double-precision minimum bounding box for the supplied geography. The polygon is defined by the corner points `((MINX, MINY), (MINX, MAXY), (MAXX, MAXY), (MAXX, MINY), (MINX, MINY))`.

For degenerate cases (e.g.: vertical lines or points), returns a `GEOGRAPHY` of a lower dimension (`POINT` or `LINESTRING`).

Syntax

```sql
ST_ENVELOPE(<input>)
```

Arguments

- `<input>` : A `GEOGRAPHY` object to calculate the envelope

Returns

A `GEOGRAPHY` representing the bounding box. Return type depends on input:

- For most cases: returns a `POLYGON` defined by the corner points
- For vertical lines or horizontal lines: returns a `LINESTRING`
- For points: returns a `POINT`

Example

```sql
SELECT ST_AsText(ST_Envelope('LINESTRING(0 0, 1 1, 2 2)'::GEOGRAPHY));
-- POLYGON((0 0, 0 2, 2 2, 2 0, 0 0))

SELECT ST_AsText(ST_Envelope('POINT(1 1)'::GEOGRAPHY));
-- POINT(1 1)
```

ST_EQUALS

Returns `TRUE` if two `GEOGRAPHY` objects are **topologically equal**.

Topological equality means that the objects have the same dimension and their point sets occupy the same space, regardless of vertex order.

This function is **symmetric**, meaning:

- `ST_EQUALS(A, B)` = `ST_EQUALS(B, A)`

Mathematical properties:

- `ST_EQUALS(A, B)` ⇔ `A = B`
- `ST_EQUALS(A, B)` ⇔ `ST_WITHIN(A, B)` ∧ `ST_WITHIN(B, A)`

See also:

- `ST_WITHIN`

Syntax

```sql
ST_EQUALS(<a>, <b>)
```

Arguments

- `<a>` and `<b>`: The two `GEOGRAPHY` objects to compare

Returns

Returns `TRUE` if the `GEOGRAPHY` objects are topologically equal, otherwise returns `FALSE`

Example

```sql
SELECT ST_EQUALS('POINT(1 1)'::GEOGRAPHY, 'POINT(1 1)'::GEOGRAPHY);
-- TRUE

SELECT ST_EQUALS('LINESTRING(0 0, 1 1)'::GEOGRAPHY, 'LINESTRING(1 1, 0 0)'::GEOGRAPHY);
-- TRUE (same space, different vertex order)
```

ST_GEOGFROMGEOJSON

Constructs a `GEOGRAPHY` object from GeoJSON `Geometry Object`.

See also:

- `ST_ASGEOJSON`

Syntax

```sql
ST_GEOGFROMGEOJSON(<input>)
```

Arguments

- `<input>`: A `JSONB` containing GeoJSON geometry

Returns

Returns a `GEOGRAPHY` object representing the input GeoJSON geometry

Examples

```sql
SELECT ST_GEOGFROMGEOJSON('{"type":"Point","coordinates":[1,1]}');
-- POINT(1 1)

SELECT '
{
  "type": "Feature",
  "geometry": {
    "type": "Point",
    "coordinates": [125.6, 10.1]
  },
  "properties": {
    "name": "Dinagat Islands"
  }
}
'::JSONB:$.geometry::GEOGRAPHY;
-- POINT(125.6 10.1)

SELECT feature:$.geometry::GEOGRAPHY
FROM FLATTEN(
    '
{
    "type": "FeatureCollection",
    "features": [{
        "type": "Feature",
        "geometry": {
```

```
            "type": "Point",
            "coordinates": [102.0, 0.5]
        },
        "properties": {
            "prop0": "value0"
        }
    }, {
        "type": "Feature",
        "geometry": {
            "type": "LineString",
            "coordinates": [
                [102.0, 0.0],
                [103.0, 1.0],
                [104.0, 0.0],
                [105.0, 1.0]
            ]
        },
        "properties": {
            "prop0": "value0",
            "prop1": 0.0
        }
    }]
}
    ':$.features, $.value AS feature
) AS f;
```

```
             geometry
-----------------------------------
 POINT(102 0.5)
 LINESTRING(102 0,103 1,104 0,105 1)
(2 rows)
```

Note: This function is equivalent to casting `JSONB` to `GEOGRAPHY` using the `::GEOGRAPHY` syntax.

ST_GEOGFROMGML

Creates a `GEOGRAPHY` object from a Geography Markup Language (GML) representation.

See also:

- `ST_ASGML`

Limitations

Supported: Basic geometry types (`Point` , `LineString` , `Polygon` , `Multi*` , `MultiGeometry`) using `gml:coordinates` format with EPSG SRS codes.

Not supported: Custom namespaces, href links, alternative coordinate formats (`gml:pos` , `gml:posList`), or advanced GML features.

Only GML produced by `ST_ASGML` function is guaranteed to be imported successfully.

Syntax

```sql
ST_GEOGFROMGML(<input>)
```

Arguments

- `<input>` : A GML formatted `VARCHAR`

Returns

A `GEOGRAPHY` type represented by the input GML format

Example

```sql
SELECT ST_GEOGFROMGML('<gml:Point srsName="EPSG:4326"><gml:coordinates>1,1</gml:coordinates></gml:Point>');
-- POINT(1 1)
```

ST_GEOGFROMTEXT

Creates a `GEOGRAPHY` object from an OGC Well-Known Text (WKT) representation.

This function is automatically invoked when casting a `VARCHAR` to `GEOGRAPHY`.

See also:

- `ST_ASTEXT`
- `ST_GEOGFROMWKB`

Syntax

```sql
ST_GEOGFROMTEXT(<input>)
```

Arguments

- `<input>`: A WKT formatted VARCHAR

Returns

A `GEOGRAPHY` type represented by the input WKT format

Example

```sql
SELECT ST_GEOGFROMTEXT('POINT(1 1)');
-- POINT(1 1)
```

ST_GEOGFROMWKB

Creates a `GEOGRAPHY` object from a Well-Known Binary (WKB) or Extended WKB (EWKB) representation.

This function is automatically invoked when casting a `VARBINARY` to `GEOGRAPHY`.

See also:

- `ST_ASBINARY`
- `ST_ASEWKB`

Syntax

```sql
ST_GEOGFROMWKB(<input>)
```

Arguments

- `<input>`: A WKB or EWKB formatted type in VARBINARY

Returns

A `GEOGRAPHY` type represented by the input WKB/EWKB format

Example

```sql
SELECT ST_GeogFromWKB('\x0101000000000000000000f03f000000000000f03f'::VARBINARY);
-- 'POINT(1 1)'
```

ST_GEOMETRYTYPE

Returns the type of the `GEOGRAPHY` as a string with `'ST_'` prefix (e.g., `'ST_LineString'`, `'ST_Polygon'`, `'ST_Point'`).

Syntax

```sql
ST_GEOMETRYTYPE(<input>)
```

Arguments

- `<input>`: A `GEOGRAPHY` object to analyze

Returns

A `VARCHAR` containing the `GEOGRAPHY` type with `'ST_'` prefix

Example

```sql
SELECT ST_GeometryType('POINT(1 1)'::GEOGRAPHY);
-- 'ST_Point'

SELECT ST_GeometryType('POLYGON((0 0, 1 0, 1 1, 0 0))'::GEOGRAPHY);
-- 'ST_Polygon'
```

ST_INTERSECTION

Calculates the intersecting shapes of two `GEOGRAPHY` objects.

This function is **symmetric**, meaning:

- `ST_INTERSECTION(a, b)` = `ST_INTERSECTION(b, a)`

See also:

- `ST_DIFFERENCE`
- `ST_UNION`

Syntax

```sql
ST_INTERSECTION(<a>, <b>)
```

Arguments

- `<a>` and `<b>`: The two `GEOGRAPHY` objects to intersect

Returns

Returns a `GEOGRAPHY` object representing the portion of `<a>` that overlaps with `<b>`. If `<a>` and `<b>` do not intersect, an empty `GEOGRAPHY` object is returned.

If `<a>` and `<b>` are of different subtypes, the following rules apply:

`<a>` subtype	`<b>` subtype	Behavior
`ST_Point`	`ST_Point`	Returns `<a>` if `<b>` equals `<a>`, otherwise returns empty.
`ST_Point`	`ST_Linestring`	Returns `<a>` if it lies on the line of `<b>`, otherwise empty.
`ST_Point`	`ST_Polygon`	Returns `<a>` if it is contained within `<b>`, otherwise empty. If `<a>` touches `<b>`, `<a>` is returned.
`ST_Linestring`	`ST_Linestring`	Returns an `ST_GeometryCollection` of all overlapping areas of two linestrings. Can return an `ST_Point` if the lines only overlap in one place, or an `ST_Linestring` if they overlap exactly once.
`ST_Linestring`	`ST_Polygon`	Returns an `ST_GeometryCollection` containing overlapping parts of the `ST_Linestring`. If the polygon touches the linestring, the touching part of the linestring is returned.
`ST_Polygon`	`ST_Polygon`	Returns a multi-shape containing the intersection, or empty if there is no overlap. As in the other cases, touching parts are considered part of the intersection.

Example

```sql
SELECT ST_AsText(ST_Intersection('POINT(0 0)'::GEOGRAPHY, 'LINESTRING(2 0,0 2)'::GEOGRAPHY));
-- GEOMETRYCOLLECTION EMPTY

SELECT ST_AsText(ST_Intersection('POINT(0 0)'::GEOGRAPHY, 'LINESTRING(0 0,0 2 )'::GEOGRAPHY));
-- POINT(0 0)
```

ST_INTERSECTS

Returns `TRUE` if two `GEOGRAPHY` objects intersect. Two geographies intersect if they share any common point.

For `GEOGRAPHY`, a distance tolerance of `0.00001` meters is applied, meaning points that are very close are considered to intersect.

This function is **symmetric**, meaning:

- `ST_INTERSECTS(a, b)` = `ST_INTERSECTS(b, a)`

See also:

- `ST_CONTAINS`

Syntax

```sql
ST_INTERSECTS(<a>, <b>)
```

Arguments

- `<a>` and `<b>`: The two `GEOGRAPHY` objects to test for intersection

Returns

Returns a `BOOLEAN`:

- `TRUE` if the `GEOGRAPHY` objects share any common points
- `FALSE` otherwise

Example

```sql
SELECT ST_Intersects('POINT(0 0)'::GEOGRAPHY, 'LINESTRING(0 0, 1 1)'::GEOGRAPHY);
-- TRUE

SELECT ST_Intersects('POLYGON((0 0, 1 0, 1 1, 0 1, 0 0))'::GEOGRAPHY, 'POINT(2 2)'::GEOGRAPHY);
-- FALSE
```

ST_ISEMPTY

Returns `TRUE` if the `GEOGRAPHY` represents an empty geometry collection, polygon, point etc.

Syntax

```sql
ST_ISEMPTY(<input>)
```

Arguments

- `<input>`: A `GEOGRAPHY` object

Returns

Returns a `BOOLEAN`:

- `TRUE` if the `GEOGRAPHY` is empty
- `FALSE` otherwise

Example

```sql
SELECT ST_ISEMPTY('POLYGON EMPTY'::GEOGRAPHY);
-- TRUE

SELECT ST_ISEMPTY('POINT(1 1)'::GEOGRAPHY);
-- FALSE
```

ST_ISVALID

Checks if a `GEOGRAPHY` value is well-formed and valid in 2D according to OGC rules.

For `GEOGRAPHY` objects with 3 or 4 dimensions, validity is still tested only in 2D.

See also:

- `ST_ISVALIDREASON`

Syntax

```sql
ST_ISVALID(<input>)
```

Arguments

- `<input>`: The `GEOGRAPHY` object to be tested

Returns

Returns a `BOOLEAN`:

- `TRUE` if the input `GEOGRAPHY` is valid
- `FALSE` if the input `GEOGRAPHY` is invalid

Example

```sql
SELECT ST_IsValid('POLYGON((0 0, 1 1, 1 0, 0 0))'::GEOGRAPHY);
-- TRUE

SELECT ST_IsValid('POLYGON((0 0, 1 1, 1 1, 0 0))'::GEOGRAPHY);
-- FALSE
```

ST_ISVALIDREASON

Returns a text description indicating whether a `GEOGRAPHY` object is valid. If the object is invalid, the function provides a reason for the invalidity.

This function is useful in combination with `ST_ISVALID` to generate detailed reports on invalid `GEOGRAPHY` objects.

See also:

- `ST_ISVALID`

Syntax

```sql
ST_ISVALIDREASON(<input>)
```

Arguments

- `<input>`: The `GEOGRAPHY` object to be tested

Returns

Returns a `TEXT` value that either states `"Valid Geometry"` or provides a reason why the input is invalid.

Example

```sql
SELECT ST_ISVALIDREASON('POLYGON((0 0, 2 2, 2 0, 0 2, 0 0))'::GEOGRAPHY);
-- 'Self-intersection[1 1]'

SELECT ST_ISVALIDREASON('POINT(1 1)'::GEOGRAPHY);
-- 'Valid Geometry'
```

ST_LENGTH

Returns the length of a `GEOGRAPHY` linear feature in meters.

For non-linear types, such as points and polygons, the function returns `0`.

See also:

- `ST_DISTANCE`

Syntax

```sql
ST_LENGTH(<input>[, <use_spheroid>])
```

Arguments

- `<input>`: The `GEOGRAPHY` object for which to compute the length
- `<use_spheroid>`: An optional `BOOLEAN`. If `FALSE`, calculations use a spherical approximation instead of a spheroid. Defaults to `TRUE`

Returns

Returns a `FLOAT` value representing the length in meters:

- For linear features (`LineStrings`, `MultiLineStrings`): Returns the total length
- For points and polygons: Returns `0`

Example

```sql
SELECT ST_Length('LINESTRING(0 0, 1 1)'::GEOGRAPHY);
-- 156899.56829134

SELECT ST_Length('POINT(0 0)'::GEOGRAPHY);
-- 0
```

ST_LINESTRING

Creates a `LineString` geometry from the points of `Point`, `MultiPoint`, or `LineString` `GEOGRAPHY` objects.

See also:

- `ST_POINT`
- `ST_POLYGON`

Syntax

```sql
ST_LINESTRING(<a>, <b>)
```

`ST_MAKELINE` is an alias for `ST_LINESTRING`.

Arguments

- `<a>`, `<b>`: The `GEOGRAPHY` objects of type `ST_Point` or `ST_LineString`

Returns

Returns a `GEOGRAPHY` of type `ST_LineString` containing all points from the input geographies.

Examples

```sql
SELECT ST_AsText(ST_LINESTRING(ST_POINT(1,2), ST_POINT(3,4)));
-- 'LINESTRING (1 2, 3 4)'

SELECT ST_AsText(ST_LINESTRING(ST_LINESTRING(ST_POINT(1,2), ST_POINT(3,4)), ST_POINT(5, 6)));
-- 'LINESTRING (1 2, 3 4, 5 6)'
```

ST_MAKEVALID

Creates a valid representation of an invalid `GEOGRAPHY` while preserving all input vertices.

If the input `GEOGRAPHY` is already valid, it is returned unchanged.

See also:

- `ST_ISVALID`

Syntax

```sql
ST_MAKEVALID(<input>)
```

Arguments

- `<input>`: The `GEOGRAPHY` object to be validated

Returns

Returns a valid `GEOGRAPHY` object. The output may be:

- The same `GEOGRAPHY` if the input was already valid
- A collection of geographies with equal or lower dimensions
- A `GEOGRAPHY` of lower dimension in case of dimensional collapse
- A multi-geometry if self-intersections are present

Example

```sql
SELECT ST_AsText(ST_MakeValid('POLYGON((1 1, -1 1, 1 -1, -1 -1, 1 1))'::GEOGRAPHY));
-- MULTIPOLYGON(((-1 1,1 1,0 0,-1 1)),((-1 -1,0 0,1 -1,-1 -1)))
```

ST_MINX, ST_MAXX, ST_MINY, ST_MAXY

The `ST_MINX`, `ST_MAXX`, `ST_MINY`, and `ST_MAXY` functions return the minimum X, maximum X, minimum Y, and maximum Y coordinates of a `GEOGRAPHY` object, respectively. These values define the bounding box of the geometry.

See also:

- `ST_POINT`
- `ST_NPOINTS`

Syntax

```sql
ST_MINX(<input>)
ST_MAXX(<input>)
ST_MINY(<input>)
ST_MAXY(<input>)
```

Arguments

- `<input>`: The `GEOGRAPHY` object.

Returns

Returns a `FLOAT8` value representing the requested coordinate.

Example

```sql
SELECT ST_XMIN('LINESTRING(1 2, 3 4)'::GEOGRAPHY);
-- 1.0
SELECT ST_XMAX('LINESTRING(1 2, 3 4)'::GEOGRAPHY);
-- 3.0
SELECT ST_YMIN('LINESTRING(1 2, 3 4)'::GEOGRAPHY);
-- 2.0
SELECT ST_YMAX('LINESTRING(1 2, 3 4)'::GEOGRAPHY);
-- 4.0
```

ST_MULTI

Transform a `GEOGRAPHY` into a `MULTI*` collection. If the input `GEOGRAPHY` is already a collection, it is returned unchanged.

Syntax

```sql
ST_MULTI(<input>)
```

Arguments

- `<input>`: A `GEOGRAPHY` object to be converted to a multi-type

Returns

A `GEOGRAPHY` of corresponding `MULTI*` type

Example

```sql
SELECT ST_AsText(ST_MULTI('POINT(1 1)'::GEOGRAPHY));
-- 'MULTIPOINT((1 1))'

SELECT ST_AsText(ST_MULTI('MULTIPOINT((1 1))'::GEOGRAPHY));
-- 'MULTIPOINT((1 1))'
```

ST_NPOINTS

Returns the number of points in a `GEOGRAPHY` object.

Note: For polygons, the start and end points are counted separately, even if they represent the same location. For example, a triangle would have 4 points in total because it closes back to its starting point.

Syntax

```sql
ST_NPOINTS(<input>)
```

Arguments

- `<input>`: The `GEOGRAPHY` object

Returns

Returns an `INTEGER` representing the total number of points in the input `GEOGRAPHY`.

Examples

```sql
SELECT ST_NPOINTS('LINESTRING(0 0, 1 1, 2 2)'::GEOGRAPHY);
-- 3

SELECT ST_NPOINTS('POLYGON((0 0, 1 1, 2 0, 0 0))'::GEOGRAPHY);
-- 4
```

ST_OVERLAPS

Returns `TRUE` if two `GEOGRAPHY` objects "spatially overlap". Two geographies overlap if they have the same dimension, their interiors intersect in that dimension, and each has at least one point inside the other (or equivalently, neither one covers the other). The overlaps relation is symmetric and irreflexive.

In mathematical terms: `ST_Overlaps(A, B) ⇔ ( dim(A) = dim(B) = dim( Int(A) ⋂ Int(B) )) ∧ (A ⋂ B ≠ A) ∧ (A ⋂ B ≠ B)`

See also:

- `ST_CONTAINS`
- `ST_INTERSECTS`
- `ST_CROSSES`

Syntax

```sql
ST_OVERLAPS(<a>, <b>)
```

Arguments

- `<a>` and `<b>`: Two `GEOGRAPHY` objects to check for overlap

Returns

Returns `TRUE` if the geographies overlap, `FALSE` otherwise.

Example

```sql
SELECT ST_OVERLAPS('POLYGON((0 0, 1 0, 1 1, 0 1, 0 0))'::GEOGRAPHY,
                   'POLYGON((0.5 0.5, 1.5 0.5, 1.5 1.5, 0.5 1.5, 0.5 0.5))'::GEOGRAPHY);
-- TRUE
```

ST_PERIMETER

Returns the perimeter of a `GEOGRAPHY` surface in meters. Returns `0` for non-surface types (points, lines). For linear geographies, use `ST_LENGTH` instead.

See also:

- `ST_LENGTH`
- `ST_AREA`

Syntax

```sql
ST_PERIMETER(<input>[, <use_spheroid>])
```

Arguments

- `<input>`: The `GEOGRAPHY` object to compute the perimeter of
- `<use_spheroid>`: Optional `BOOLEAN`. If `FALSE`, calculations use spherical approximation instead of spheroid. Defaults to `TRUE`.

Returns

Returns a `FLOAT8` value representing the perimeter in meters:

- For surfaces (`POLYGON`, `MULTIPOLYGON`): returns the perimeter
- For non-surface types (`POINT`, `LINESTRING`): returns `0`

Example

```sql
SELECT ST_Perimeter('POLYGON((0 0, 1 0, 1 1, 0 1, 0 0))'::GEOGRAPHY);
-- 443770.917248302

SELECT ST_Perimeter('POINT(0 0)'::GEOGRAPHY);
-- 0
```

ST_POINT

Creates a `GEOGRAPHY` point from given `x` (longitude) and `y` (latitude) coordinates.

See also:

- `ST_LINESTRING`
- `ST_POLYGON`

Syntax

```sql
ST_POINT(<x>, <y>)
```

`ST_MAKEPOINT` is an alias for `ST_POINT`.

Arguments

- `<x>` : A `FLOAT8` or `FLOAT4` value representing the longitude
- `<y>` : A `FLOAT8` or `FLOAT4` value representing the latitude

Returns

Returns a `GEOGRAPHY` object of subtype `ST_Point`.

Example

```sql
SELECT ST_AsText(ST_POINT(1, 1));
-- 'POINT(1 1)'
```

ST_POINTN

Returns the Nth point in a single linestring. Negative values are counted backwards from the end of the line, so that -1 is the last point.

See also:

- `ST_STARTPOINT`
- `ST_ENDPOINT`
- `ST_NPOINTS`

Syntax

```sql
ST_POINTN(<linestring>, <n>)
```

Arguments

- `<input>`: The `LINESTRING` object
- `<n>`: `INTEGER` indicating which point to return

Returns

Returns a `GEOGRAPHY` point that is the Nth point in the linestring. Returns `NULL` if input is not of `LINESTRING` type, or index is invalid.

Example

```sql
SELECT ST_AsText(ST_POINTN('LINESTRING(0 0, 1 1, 2 2)'::GEOGRAPHY, 2));
-- POINT(1 1)
```

ST_POLYGON

Creates a `GEOGRAPHY` polygon from the given `LineString`.

See also:

- `ST_POINT`
- `ST_LINESTRING`

Syntax

```sql
ST_POLYGON(<linestring>)
```

`ST_MAKEPOLYGON` is an alias for `ST_POLYGON`.

Arguments

- `<linestring>`: A `GEOGRAPHY` object representing a closed `LineString` that forms the boundary of the polygon

Returns

Returns a `GEOGRAPHY` object of subtype `ST_Polygon` constructed from the input `LineString`.

Example

```sql
SELECT ST_AsText(ST_POLYGON('LINESTRING(0 0, 0 1, 1 1, 1 0, 0 0)'::GEOGRAPHY));
-- POLYGON((0 0, 0 1, 1 1, 1 0, 0 0))
```

ST_REDUCEPRECISION

Returns a `GEOGRAPHY` object with reduced precision, achieved by rounding all coordinates to a specified grid size. Features smaller than the grid size may be removed or altered, but the output will be a valid `GEOGRAPHY` object.

This function is generally more accurate near the equator and for smaller geographic scales. For large-scale projections, it may introduce significant distortions.

Syntax

```sql
ST_REDUCEPRECISION(<input>, <gridsize>)
```

Arguments

- `<input>`: A `GEOGRAPHY` object whose precision will be reduced.
- `<gridsize>`: A `FLOAT8` value representing the grid tolerance in meters.

Returns

Returns a `GEOGRAPHY` object with reduced precision. All coordinates are rounded to the specified grid size, and features smaller than this tolerance may be removed or altered. The resulting object will be a valid `GEOGRAPHY`.

Example

```sql
SELECT
  ROUND(ST_DISTANCE(p1, p2)::NUMERIC, 2) AS original_distance_m,
  ROUND(ST_DISTANCE(ST_REDUCEPRECISION(p1, 20), ST_REDUCEPRECISION(p2, 20))::NUMERIC, 2) AS distance_after_20m_grid,
  ROUND(ST_DISTANCE(ST_REDUCEPRECISION(p1, 100), ST_REDUCEPRECISION(p2, 100))::NUMERIC, 2) AS distance_after_100m_grid
FROM (
  SELECT
    ST_POINT(10.0001, 50.0001)::GEOGRAPHY AS p1,
    ST_POINT(10.0003, 50.0003)::GEOGRAPHY AS p2
) AS points;
-- original_distance_m | distance_after_20m_grid | distance_after_100m_grid
-- --------------------+-------------------------+-------------------------
--               26.00 |                   20.00 |                    0.00
```

ST_REVERSE

Reverses the order of vertices in a `GEOGRAPHY` object, producing an object with the same shape but opposite vertex order.

Syntax

```sql
ST_REVERSE(<input>)
```

Arguments

- `<input>`: The `GEOGRAPHY` object whose vertices will be reversed

Returns

Returns a `GEOGRAPHY` object with vertices in reversed order.

Example

```sql
SELECT ST_AsText(ST_Reverse('LINESTRING(1 1, 2 2, 3 3)'::GEOGRAPHY));
-- 'LINESTRING(3 3, 2 2, 1 1)'
```

ST_ROTATE

Rotates `GEOGRAPHY` counter-clockwise about the origin point. The rotation origin can be specified either as a `POINT`, or defaults to `POINT(0 0)`.

See also:

- `ST_SCALE`

Syntax

```sql
ST_ROTATE(<input>, <rot_radians>[, <origin>])
```

Arguments

- `<input>`: A `GEOGRAPHY` object to rotate
- `<rot_radians>`: Rotation angle in radians (counter-clockwise)
- `<origin>`: Optional `GEOGRAPHY` of type `POINT` specifying rotation center

Returns

Returns a `GEOGRAPHY` of the same type as the input, rotated by the specified angle.

Example

```sql
SELECT ST_AsText(ST_ROTATE('POINT(1 0)'::GEOGRAPHY, pi()));
-- 'POINT(-1 0)'

SELECT ST_AsText(ST_ROTATE('POINT(1 0)'::GEOGRAPHY, pi(), 'POINT(1 0)'::GEOGRAPHY));
-- 'POINT(1 0)'
```

ST_SCALE

Scales a `GEOGRAPHY` to a new size by multiplying the coordinates with the corresponding factor parameters.

See also:

- `ST_ROTATE`

Syntax

```sql
ST_SCALE(<input>, <xfactor>, <yfactor>)
```

Arguments

- `<input>`: A `GEOGRAPHY` type to scale
- `<xfactor>`: A `FLOAT8` scale to apply to the x-axis. Negative values are allowed
- `<yfactor>`: A `FLOAT8` scale to apply to the y-axis. Negative values are allowed

Returns

Returns a `GEOGRAPHY` of the same type as the input, scaled according to the provided factors. Using negative values will result in mirroring along that axis.

Example

```sql
SELECT ST_AsText(ST_Scale('POINT(1 1)'::GEOGRAPHY, 2, 3));
-- 'POINT(2 3)'

SELECT ST_AsText(ST_Scale('LINESTRING(0 0, 1 1)'::GEOGRAPHY, 2, -2));
-- 'LINESTRING(0 0, 2 -2)'
```

ST_SIMPLIFYPRESERVETOPOLOGY

Generates a simplified representation of a `GEOGRAPHY` object using a variant of the Douglas-Peucker algorithm while preserving topology.

Vertices within the specified tolerance distance are removed from the simplified geometry.

Syntax

```sql
ST_SIMPLIFYPRESERVETOPOLOGY(<input>, <tolerance>)
```

Arguments

- `<input>`: The `GEOGRAPHY` object to be simplified
- `<tolerance>`: A `FLOAT8` value representing the tolerance in meters

Returns

Returns a simplified `GEOGRAPHY` object that preserves the topology of the input.

Example

```sql
SELECT ST_AsText(ST_SimplifyPreserveTopology('LINESTRING(0 0, 0.5 0.5, 1 1)'::GEOGRAPHY, 1000));
-- 'LINESTRING(0 0, 1 1)'
```

ST_STARTPOINT

Returns the first point of a `LINESTRING`.

See also:

- `ST_ENDPOINT`

Syntax

```sql
ST_STARTPOINT(<input>)
```

Arguments

- `<input>`: The `LINESTRING` object

Returns

Returns a `POINT` representing the first point of the input `LINESTRING`. Returns `NULL` if input is not of `LINESTRING` type.

Example

```sql
SELECT ST_AsText(ST_StartPoint('LINESTRING(1 1, 2 2, 3 3)'::GEOGRAPHY));
-- POINT(1 1)
```

ST_TOUCHES

Returns `TRUE` if two `GEOGRAPHY` objects share at least one point on their boundaries, while their interiors do not intersect.

This function is symmetric:

- `ST_TOUCHES(A, B)` = `ST_TOUCHES(B, A)`

See also:

- `ST_INTERSECTS`
- `ST_CROSSES`

Syntax

```sql
ST_TOUCHES(<a>, <b>)
```

Arguments

- `<a>` and `<b>`: Two `GEOGRAPHY` objects to test

Returns

Returns a `BOOLEAN`:

- `TRUE` if the geographies touch (share boundary points but no interior points)
- `FALSE` if the geographies don't touch or if either input is a point (points have no boundary)

Example

```sql
SELECT ST_TOUCHES('POLYGON((0 0, 1 0, 1 1, 0 1, 0 0))'::GEOGRAPHY,
                  'POLYGON((1 0, 2 0, 2 1, 1 1, 1 0))'::GEOGRAPHY);
-- TRUE
```

ST_TRANSLATE

Translates a `GEOGRAPHY` object by applying specified offsets to its coordinates.

See also:

- `ST_ROTATE`
- `ST_SCALE`

Syntax

```sql
ST_TRANSLATE(<input>, <offset_x>, <offset_y>)
```

Arguments

- `<input>`: A `GEOGRAPHY` object to translate.
- `<offset_x>`: A `FLOAT8` value representing the offset to apply to the x-coordinates.
- `<offset_y>`: A `FLOAT8` value representing the offset to apply to the y-coordinates.

Returns

Returns a `GEOGRAPHY` of the same type as the input, with its coordinates translated by the specified delta values.

Example

```sql
SELECT ST_AsText(ST_TRANSLATE('POINT(1 1)'::GEOGRAPHY, 10, -5));
-- 'POINT(11 -4)'

SELECT ST_AsText(ST_TRANSLATE('LINESTRING(0 0, 1 1)'::GEOGRAPHY, 2, 3));
-- 'LINESTRING(2 3, 3 4)'
```

ST_UNION

Combines two `GEOGRAPHY` objects into a single shape without overlaps. This function is commonly used to merge complex `GEOGRAPHY` shapes into a simpler one.

There are two variants of this function:

- **Two-input variant:** Returns a `GEOGRAPHY` object representing the union of two input geographies. If either input is `NULL`, the result is `NULL`.
- **Aggregate variant:** Returns a `GEOGRAPHY` object representing the union of a rowset of geographies. `NULL` values are ignored in this variant.

This function is **symmetric**, meaning:

- `ST_UNION(A, B)` = `ST_UNION(B, A)`

See also:

- `ST_INTERSECTION`
- `ST_DIFFERENCE`

Syntax

```sql
-- Two-input variant:
ST_UNION(<a>, <b>)

-- Aggregate variant:
ST_UNION(<geography_column>)
```

Arguments

- `<a>` and `<b>`: The two `GEOGRAPHY` objects to be merged

Returns

Returns a `GEOGRAPHY` object representing the union of the input geographies.

The output type can be:

- A `GEOGRAPHY` object
- A `MultiGeometry`
- A `GeometryCollection`

Example

```sql
CREATE VIEW v AS
             SELECT 'POINT(1 1)' geo
    UNION ALL SELECT 'LINESTRING(1 1,2 2)'
    UNION ALL SELECT 'POINT(3 3)';

SELECT ST_AsText(ST_Union(geo::GEOGRAPHY)) from v;
-- 'GEOMETRYCOLLECTION (LINESTRING (1 1, 2 2), POINT (3 3))'
-- Note: POINT(1 1) was merged into LINESTRING(1 1,2 2)
```

ST_WITHIN

Returns `TRUE` if one `GEOGRAPHY` object is within another.

A `GEOGRAPHY` A is within B if and only if:

- All points of A lie inside B (either within the interior or on the boundary)
- The interiors of A and B share at least one common point

This function has the following properties:

- **Reflexive:** Every `GEOGRAPHY` object is within itself
- **Antisymmetric:** If `ST_WITHIN(A, B) = TRUE` and `ST_WITHIN(B, A) = TRUE`, then A and B are topologically equal
- **Converse of `ST_CONTAINS`:** `ST_WITHIN(A, B)` is equivalent to `ST_CONTAINS(B, A)`

See also:

- `ST_CONTAINS`
- `ST_COVERS`

Syntax

```sql
ST_WITHIN(<a>, <b>)
```

Arguments

- `<a>` and `<b>`: The `GEOGRAPHY` objects to check for the "within" relationship

Returns

Returns `TRUE` if `<a>` is within `<b>`, otherwise returns `FALSE`.

Example

```sql
SELECT ST_WITHIN('POINT(1 1)'::GEOGRAPHY, 'POLYGON((0 0, 0 2, 2 2, 2 0, 0 0))'::GEOGRAPHY);
-- TRUE

SELECT ST_WITHIN('POINT(3 3)'::GEOGRAPHY, 'POLYGON((0 0, 0 2, 2 2, 2 0, 0 0))'::GEOGRAPHY);
-- FALSE
```

ST_X and ST_Y

The `ST_X` and `ST_Y` functions return the X (longitude) and Y (latitude) coordinates of a `GEOGRAPHY` object of subtype `ST_Point`.

See also:

- `ST_POINT`
- `ST_POINTN`

Syntax

```sql
ST_X(<input>)

ST_Y(<input>)
```

Arguments

- `<input>`: A `GEOGRAPHY` object of subtype `ST_Point`.

Returns

Returns a `FLOAT8` value representing the X or Y coordinate of the input point. Returns `NULL` if the input is not a `GEOGRAPHY` object of subtype `ST_Point`.

Example

```sql
SELECT ST_X('POINT(1 2)'::GEOGRAPHY);
-- 1.0

SELECT ST_Y('POINT(1 2)'::GEOGRAPHY);
-- 2.0
```

Mathematical Functions

Platforms: All platforms

Parent topic: SQL Functions and Operators

In this section:

This section covers mathematical functions and operators.

ABS

Return the absolute value of a number.

```
ABS(number)
```

Alternatively, you can use `@ number` (no parentheses around the number).

For example:

```
premdb=# SELECT ABS(-5);
 abs
-----
   5
(1 row)
```

```
premdb=# SELECT @ -5;
 ?column?
----------
       5
(1 row)
```

The following example returns the absolute value for the result of an expression. The result represents the difference between the two numbers in the `ftscore` column.

```
premdb=# SELECT ftscore,
ABS((SUBSTR(ftscore,1,1)::INT)-(SUBSTR(ftscore,3,1)::INT))
FROM match LIMIT 25;
 ftscore | abs
---------+-----
 0-1     |   1
 0-1     |   1
 2-1     |   1
 3-0     |   3
 3-0     |   3
 2-0     |   2
 0-0     |   0
 0-0     |   0
 0-1     |   1
 1-0     |   1
 0-1     |   1
...
```

ACOS

Return the inverse cosine value of an input radian value.

```
ACOS(number)
```

The return type is a `DOUBLE PRECISION` number.

For example:

```
premdb=# select acos(0.10) from sys.const;
      acos
------------------
 1.47062890563334
(1 row)
```

```
premdb=# select acos(avg_att/100) from team order by teamid limit 2;
      acos
-------------------
 0.927995034405117
  1.22717385372596
(2 rows)
```

ASIN

Return the inverse sine value of an input radian value.

```
ASIN(number)
```

The return type is a `DOUBLE PRECISION` number.

For example:

```
premdb=# select asin(1) from sys.const;
     asin
----------------
 1.5707963267949
(1 row)
```

```
premdb=# select asin(avg_att/100) from team order by teamid limit 2;
      asin
------------------
  0.64280129238978
 0.343622473068933
(2 rows)
```

CBRT

Return the cube root of a number.

```
CBRT(number)
```

Alternatively, you can use `||/ number` (no parentheses around `number`).

```
premdb=# SELECT CBRT(10);
      cbrt
------------------
 2.15443469003188
(1 row)
```

```
premdb=# SELECT ||/ 10;
    ?column?
------------------
 2.15443469003188
(1 row)
```

```
premdb=# SELECT CBRT(seasonid) FROM season;
      cbrt
------------------
                1
 1.25992104989487
 1.44224957030741
  1.5874010519682
  1.7099759466767
 1.81712059283214
 1.91293118277239
                2
  2.0800838230519
...
```

CEIL

Round a number up to the next whole number (a positive or negative integer).

```
CEIL(number)
```

`CEILING` is a synonym for `CEIL`.

For example:

```
premdb=# SELECT CEIL(2.5);
 ceil
------
   3
(1 row)
```

```
premdb=# SELECT CEIL(-1.2567);
 ceil
------
   -1
(1 row)
```

The following example finds the "ceiling" for the results of an AVG function.

```
premdb=# SELECT seasonid, AVG(SUBSTR(ftscore,3,1)::INT),
CEIL(AVG(SUBSTR(ftscore,3,1)::INT)) ceilgoals
FROM match GROUP BY seasonid ORDER BY 2;
 seasonid |            avg            | ceilgoals
----------+---------------------------+-----------
       15 | 0.9973684210526315789473 6 |         1
       14 | 1.0236842105263157894736 8 |         2
       13 | 1.0657894736842105263157 8 |         2
        9 | 1.0657894736842105263157 8 |         2
        7 | 1.0684210526315789473684 2 |         2
...
```

COS

Return the cosine value of an input radian value.

```
COS(number)
```

The return type is a `DOUBLE PRECISION` number.

For example:

```
premdb=# select cos(1) from sys.const;
      cos
-----------------
 0.54030230586814
(1 row)
```

```
premdb=# select cos(seasonid) from season;
       cos
--------------------
   0.54030230586814
  -0.416146836547142
  -0.989992496600445
  -0.653643620863612
   0.283662185463226
   0.960170286650366
...
```

DEGREES

Convert a number from radians to degrees.

```
DEGREES(number)
```

The return type is a `DOUBLE PRECISION` number.

For example:

```
premdb=# select degrees(1) from sys.const;
    degrees
------------------
 57.2957795130823
(1 row)
```

```
premdb=# select degrees(avg_att) from team where avg_att>0;
    degrees
------------------
 3434.53820713221
 1930.29481179574
 641.082476971878
 2377.77484979292
  1411.5388240843
 2184.34429815675
 1844.98139610076
 2515.85767841944
 3096.32122066648
 4313.57005642192
  2850.6942138939
...
```

DIV

Return the integer quotient from the division of two DECIMAL values.

```
DIV(expr1, expr2)
```

where `expr1` and `expr2` are both DECIMAL values. The first value is divided by the second.

For example:

```
premdb=# create table divtest(c1 decimal(5,3));
CREATE TABLE
premdb=# insert into divtest values(99.9);
INSERT 0 1
premdb=# select c1, div(c1,33.3) from divtest;
   c1   | div
--------+-----
 99.900 |   3
(1 row)
```

ERF

Run the Gaussian error function on an expression that evaluates to a floating-point number or a compatible data type.

```
ERF(number)
```

For example:

```
premdb=# select erf(0.123456789) from sys.const;
       erf
-------------------
 0.138601545055623
(1 row)
```

```
premdb=# select avg_att, erf(avg_att) from team order by 1 desc limit 3;
 avg_att | erf
---------+-----
  75.286 |   1
  59.944 |   1
  54.041 |   1
(3 rows)
```

EXP

Return the exponential value for a number.

```
EXP(number)
```

For example:

```
premdb=# SELECT EXP(10);
      exp
------------------
 22026.4657948067
(1 row)
```

```
premdb=# SELECT seasonid, EXP(seasonid) FROM season;
 seasonid |       exp
----------+------------------
        1 | 2.71828182845905
        2 | 7.38905609893065
        3 | 20.0855369231877
        4 | 54.5981500331442
        5 | 148.413159102577
```

FLOOR

Round a number down to the next whole number (a positive or negative integer).

```
FLOOR(number)
```

For example:

```
premdb=# SELECT FLOOR(2.5);
 ceil
------
   2
(1 row)
```

```
premdb=# SELECT FLOOR(-1.2567);
 ceil
------
   -2
(1 row)
```

The following example finds the "floor" for the results of an AVG function.

```
premdb=# SELECT seasonid, AVG(SUBSTR(ftscore,3,1)::INT),
FLOOR(AVG(SUBSTR(ftscore,3,1)::INT)) floorgoals FROM match GROUP BY seasonid ORDER BY 2;
 seasonid |           avg            | floorgoals
----------+--------------------------+------------
       15 | 0.9973684210526315789473 |          0
       14 | 1.0236842105263157894736 |          1
       13 | 1.0657894736842105263157 |          1
        9 | 1.0657894736842105263157 |          1
        7 | 1.0684210526315789473684 |          1
...
```

INV_NORM

Return an inverse normal distribution x-value, given a known probability (or percentile) and optional mean and standard deviation values. This function uses the Beasley-Springer-Moro approximation; the return value is *not* an exact value.

The `INV_NORM` syntax is as follows:

```
INV_NORM(percentage, [ mean, stddev ] )
```

percentage

Any numeric value from `>0` to `<1`. If only the first argument is specified, `mean` = `0` and `stddev` = `1`. For example: `inv_norm(p) = inv_norm(p, 0, 1).`

mean

Any numeric value. If you specify `mean`, you must also specify `stddev`.

stddev

Any numeric value `>0`. When the `mean` and `stddev` arguments are specified, the output of the function is scaled, using the specified `mean` and `stddev` values.

When invalid values are supplied, the function returns `NULL`. However, when `NaN` is specified for any argument, the function returns `NaN`.

Examples

For example:

```
yellowbrick=# select inv_norm(0.90) from sys.const;
    inv_norm
------------------
 1.28155156327703
(1 row)

yellowbrick=# select inv_norm(0.90,70,4.5) from sys.const;
    inv_norm
------------------
 75.7669820347467
(1 row)
```

```
yellowbrick=# create table doubles(c1 float8, c2 float4, c3 real);
CREATE TABLE
yellowbrick=# insert into doubles values(1.1234,2.1234,3.1234);
INSERT 0 1
yellowbrick=# insert into doubles values(0.1234,1.1234,2.1234);
INSERT 0 1
yellowbrick=# select inv_norm(c1,c2,c3) from doubles;
    inv_norm
------------------
           [NULL]
```

```
 -1.33583043079482
(2 rows)
```

LN

Return the natural logarithm of a number.

```
LN(number)
```

```
premdb=# SELECT LN(101);
       ln
------------------
 4.61512051684126
(1 row)

premdb=# SELECT LN(seasonid) FROM season;
       ln
-------------------
                0
 0.693147180559945
  1.09861228866811
  1.38629436111989
   1.6094379124341
  1.79175946922805
  1.94591014905531
  2.07944154167984
...
```

LOG

Return the base 10 logarithm of a number, or specify the base to use in the first argument and the number in the second argument.

```
LOG([ base, ] number)
```

This function returns a double precision or decimal type, depending on the input type.

base

A positive number. If `base` is specified, both arguments must be decimal types. If necessary, use explicit casts.

number

A positive number.

For example:

```
premdb=# select log(101) from sys.const;
      log
-----------------
 2.00432137378264
(1 row)

premdb=# select log(2,101) from sys.const;
       log
-------------------
 6.6582114827517947
(1 row)

premdb=# select log(10.1) from sys.const;
        log
------------------------
 1.00432137378264257428
(1 row)
```

```
premdb=# select avg_att, log(2.0, avg_att) from team where avg_att>0 limit 1;
 avg_att |                  log
---------+----------------------------------------
  59.944 | 5.905543451471921012831147221451800576
(1 row)
```

```
ppremdb=# select avg_att, log(avg_att)::dec(3,2) from team where avg_att>0 limit 1;
 avg_att | log
---------+------
  59.944 | 1.78
(1 row)
```

Mathematical Operators

This section covers mathematical operators. Subsequent sections cover specific mathematical functions.

Operator	Function	Example	Result
+	Addition	99+2	101
-	Subtraction	99-2	97
*	Multiplication	99*2	198
/	Division	99/2.0	49.5
%	Modulo	99 % 2	1
^	Exponentiation	99 ^ 2	9801
|/	Square root. See SQRT	|/ 99	9.94
||/	Cube root. See CBRT	||/ 99	4.62
&	Bitwise AND	25 & 20	16
|	Bitwise OR	25 | 20	29
#	Bitwise XOR	25 # 20	13
~	Bitwise NOT	~ 25	-26
<<	Bitwise shift left	25 << 20	26214400
>>	Bitwise shift right	25 >> 20	0

Note that the division of integers produces different results from the division of decimal values.

The bitwise operators work only with SMALLINT, INTEGER, and BIGINT data types. The other operators work with all numeric data types.

MOD

Return the remainder for a division operation, where the first parameter is divided by the second.

```
MOD(number, number)
```

Both `number` expressions must evaluate to integer or DECIMAL values.

For example:

```
premdb=# select mod(22, 21.234) from sys.const;
  mod
-------
 0.766
(1 row)

premdb=# select name, mod(avg_att,capacity) from team where capacity>0 order by 2 desc;
         name           |  mod
------------------------+--------
 Manchester United      | 75.286
 Arsenal                | 59.944
 Manchester City        | 54.041
 Newcastle United       | 49.754
 Liverpool              | 43.910
 Sunderland             | 43.071
...

premdb=# select seasonid, season_name from season where mod(seasonid,2)=1 order by 1,2;
 seasonid | season_name
----------+-------------
        1 | 1992-1993
        3 | 1994-1995
        5 | 1996-1997
        7 | 1998-1999
        9 | 2000-2001
       11 | 2002-2003
       13 | 2004-2005
       15 | 2006-2007
       17 | 2008-2009
       19 | 2010-2011
       21 | 2012-2013
       23 | 2014-2015
       25 | 2016-2017
(13 rows)
```

POWER

Return a number raised to a specified power.

```
POWER(number, power)
```

`POW` is a synonym for `POWER`.

Both arguments can be any number, including integer, decimal, and double precision values.

Return Type

If the first argument is an integer or a double precision value, the result is a double precision value.

If the first argument is a decimal value, the result is a decimal value. Decimal values are coerced to floating-point values for the calculation, then converted back to decimals in the result. The scale and precision of the result depend on the rules described in Calculations with DECIMAL Values.

Note: When the POWER function is used with decimal input values and the integral part of the result is greater than 38 minus the scale of the first argument, the function may return an overflow error.

Examples

For example:

```
premdb=# select power(10.12345,10) from sys.const;
      power
--------------------
 11305386703.894196
(1 row)
```

```
premdb=# select capacity, pow(capacity,10) from team where avg_att>60;
 capacity |         pow
----------+----------------------
    75635 | 6.12672112065688e+48
(1 row)
```

```
premdb=# select htid, atid, power(htid,atid) from team order by 1,2 asc limit 10;
 htid | atid |        power
------+------+----------------------
    2 |   51 | 2.25179981368525e+15
    3 |   52 | 6.46108188922667e+24
    4 |   53 | 8.11296384146067e+31
    5 |   54 | 5.55111512312578e+37
    6 |   55 |   6.285195213566e+42
    7 |   56 | 2.11587613802425e+47
```

```
    8 |   57 | 2.99315535325369e+51
    9 |   58 | 2.21853123446226e+55
   10 |   59 |               1e+59
   11 |   60 | 3.04481639541418e+62
(10 rows)
```

See also Calculations with DECIMAL Values.

PROBNORM

Return the probability that an observation from the standard normal distribution is less than or equal to the specified `number`. This function runs the following calculation on an expression that evaluates to a floating-point number or a compatible data type:

```
0.5 * (ERF(number / POW(2, 0.5)) + 1)
```

The `PROBNORM` syntax is as follows:

```
PROBNORM(number)
```

For example:

```
premdb=# select probnorm(0.123456789) from sys.const;
    probnorm
------------------
 0.549127305078142
(1 row)
```

```
premdb=# select avg_att, probnorm(avg_att) from team order by 1 desc limit 3;
 avg_att | probnorm
---------+----------
  75.286 |        1
  59.944 |        1
  54.041 |        1
(3 rows)
```

See also ERF.

RADIANS

Convert a number from degrees to radians.

```
RADIANS(number)
```

The return type is a `DOUBLE PRECISION` number.

For example:

```
premdb=# select radians(1) from sys.const;
     radians
--------------------
 0.0174532925199433
(1 row)
```

```
premdb=# select radians(avg_att) from team where avg_att>0 order by teamid;
     radians
-------------------
  1.04622016681548
  0.58800142499689
 0.195284890005646
 0.724311639577647
 0.429979314521323
 0.665389324030318
 0.562013472434694
  0.76637407455071
 0.943193381070256
  1.31398858065645
 0.868371116037259
 0.470750205847911
 0.536706198280776
 0.480558956244119
 0.751730762126478
 0.361475141380546
 0.624408993193491
 0.359433106155712
 0.429892048058723
  0.60929444187122
(20 rows)
```

RANDOM()

Return a random double precision number that falls within a range.

```
RANDOM([n1,n2])
```

If you specify parentheses only, the result falls between `0.0` and `1.0`. Alternatively, you can specify the range, such as `(5,10)` or `(100,1000)`.

Examples

```
premdb=# select random() from sys.const;
      random
--------------------
 0.0963835035787144
(1 row)
```

```
premdb=# select random(),* from match order by 1;
       random        | seasonid |      matchday       | htid | atid | ftscore | htscore
---------------------+----------+---------------------+------+------+---------+---------
 0.000131850332641704 |       16 | 2008-03-16 00:00:00 |   46 |   57 | 1-0     | 1-0
 0.000151228328515424 |        2 | 1993-12-29 00:00:00 |    2 |   84 | 3-0     |
 0.000216605043567899 |       11 | 2003-01-29 00:00:00 |   24 |   51 | 2-2     | 0-1
 0.000481910348946516 |        1 | 1992-08-01 00:00:00 |    6 |   63 | 2-0     | -
 0.000567176363097615 |        6 | 1998-03-31 00:00:00 |   47 |   77 | 0-0     | 0-0
...
```

```
yellowbrick=# select random(5,10) from sys.const;
     random
-----------------
 8.32123883983625
(1 row)
```

```
yellowbrick=# select random(10000.1,10001.2) from sys.const;
     random
-----------------
 10000.2421522765
(1 row)
```

ROUND

Round a number (up or down) to the nearest integer. Optionally, specify the number of decimal places for the rounding.

```
ROUND(number [, places])
```

where `number` can be any numeric data type (a constant or an expression). If only `number` is specified, the function rounds to the nearest whole number. If `places` is also specified, the function rounds to the nearest number with that many decimal places.

The second argument for `ROUND` must be a constant. See also ROUND_VAR, which accepts non-constant values for the second argument.

The return type is a `DECIMAL` with its precision and scale adjusted based on the input type of the second argument, if specified. For example, if the type of the first argument is `DECIMAL(4,3)` and the second argument is `2`, the return type will be `DECIMAL(3,2)`:

```
ROUND(1.234, 2) returns 1.23
```

Examples

Round up to the nearest whole number:

```
premdb=# select round(2.5) from sys.const;
 round
-------
     3
(1 row)
```

Round up with a precision of two decimal places:

```
premdb=# select round(2.5,2) from sys.const;
 round
-------
  2.50
(1 row)
```

The following example rounds the results of an `AVG` function to a precision of three decimal places.

```
premdb=# SELECT seasonid, AVG(SUBSTR(ftscore,3,1)::INT),
ROUND(AVG(SUBSTR(ftscore,3,1)::INT),3) roundgoals FROM match GROUP BY seasonid ORDER BY 2;
 seasonid |            avg             | roundgoals
----------+----------------------------+------------
       15 | 0.9973684210526315789473 6 |      0.997
       14 | 1.0236842105263157894736 8 |      1.024
       13 | 1.0657894736842105263157 8 |      1.066
        9 | 1.0657894736842105263157 8 |      1.066
        7 | 1.0684210526315789473684 2 |      1.068
```

```
      18 | 1.0736842105263157894736 |      1.074
       4 | 1.0736842105263157894736 |      1.074
       3 | 1.0779220779220779220792 |      1.078
      17 | 1.0789473684210526315789 |      1.079
       1 | 1.0800865800865800865800 |      1.080
       5 | 1.0815789473684210526315 |      1.082
...
```

The following example rounds `DECIMAL` values to the nearest integer.

```
premdb=# SELECT atid, htid, atid/htid::DEC(7,2),
ROUND(atid/htid::DEC(7,2)) FROM team;
 atid | htid |  ?column?   | round
------+------+-------------+-------
   51 |    2 | 25.50000000 |    26
   52 |    3 | 17.33333333 |    17
   53 |    4 | 13.25000000 |    13
   54 |    5 | 10.80000000 |    11
   55 |    6 |  9.16666666 |     9
   56 |    7 |  8.00000000 |     8
   57 |    8 |  7.12500000 |     7
   58 |    9 |  6.44444444 |     6
   59 |   10 |  5.90000000 |     6
   60 |   11 |  5.45454545 |     5
   61 |   12 |  5.08333333 |     5
   62 |   13 |  4.76923076 |     5
   63 |   14 |  4.50000000 |     5
...
```

ROUND_VAR

Round a number (up or down) to the nearest integer. Optionally, specify the number of decimal places for the rounding. This function is equivalent to ROUND, except that it allows the second argument to be a column name or an expression, while `ROUND` enforces the use of a constant.

```
ROUND_VAR(number [, places])
```

The return type of the `ROUND_VAR` function is the `DECIMAL` type of the first argument. For example:

```
ROUND_VAR(1.234, 1*2) returns 1.230
```

Examples

Determine the precision and scale of a given `DECIMAL` value by using the integer values found in the `seasonid` column of the `season` table:

```
premdb=# select round_var(1.111567, seasonid), seasonid from season;
 round_var | seasonid
-----------+----------
  1.100000 |        1
  1.110000 |        2
  1.112000 |        3
  1.111600 |        4
  1.111570 |        5
  1.111567 |        6
  1.111567 |        7
  1.111567 |        8
  1.111567 |        9
...
(25 rows)
```

Note that the scale of the result is 6 decimal places, so the results for `seasonid 6` and higher are the same.

Enabling ROUND_VAR Behavior for ROUND

If you commonly supply a variable number of decimal places to the `ROUND` function, the configuration parameter `enable_alternative_round` will instruct the query planner to automatically translate `ROUND` to `ROUND_VAR` without modifying the original SQL text.

SIGN

Return the sign of a number:

- `0` = the number is 0.
- `1` = the number is a positive number
- `-1` = the number is a negative number

```
SIGN(number)
```

For example:

```
premdb=# SELECT SIGN(-52.3);
 sign
------
   -1
(1 row)
```

The following example finds the sign of the result of an expression.

```
premdb=# SELECT ftscore,
SIGN((SUBSTR(ftscore,1,1)::INT)-(SUBSTR(ftscore,3,1)::INT))
FROM match LIMIT 25;
 ftscore | sign
---------+------
 0-1     |   -1
 0-1     |   -1
 2-1     |    1
 3-0     |    1
 3-0     |    1
 2-0     |    1
 0-0     |    0
 0-0     |    0
 0-1     |   -1
 1-0     |    1
 0-1     |   -1
 1-1     |    0
 2-4     |   -1
...
```

SIN

Return the sine value of an input radian value.

```
SIN(number)
```

The return type is a `DOUBLE PRECISION` number.

For example:

```
premdb=# select sin(1) from sys.const;
       sin
------------------
 0.841470984807897
(1 row)
```

```
premdb=# select sin(seasonid) from season;
        sin
---------------------
   0.841470984807897
   0.909297426825682
   0.141120008059867
   -0.756802495307928
   -0.958924274663138
...
```

SQRT

Return the square root of a number (any positive number, including integer, decimal, and double precision values).

```
SQRT(number)
```

Alternatively, you can use `|/ number` (no parentheses around `number`).

```
premdb=# SELECT SQRT(10);
      sqrt
------------------
 3.16227766016838
(1 row)
```

```
premdb=# SELECT |/ 10;
    ?column?
------------------
 3.16227766016838
(1 row)
```

```
premdb=# SELECT SQRT(seasonid) FROM season;
      sqrt
------------------
                1
  1.4142135623731
 1.73205080756888
                2
 2.23606797749979
 2.44948974278318
 2.64575131106459
 2.82842712474619
                3
 3.16227766016838
...
```

TRUNC

Round down a number, either to the next whole number or to a specified decimal precision.

```
TRUNC(number [, places])
```

For example:

```
premdb=# SELECT TRUNC(2.5);
 trunc
-------
     2
(1 row)
```

```
premdb=# SELECT TRUNC(-1.2567);
 trunc
-------
    -1
(1 row)
```

The following example truncates the results of an AVG function to three decimal places.

```
premdb=# SELECT seasonid, AVG(SUBSTR(ftscore,3,1)::INT),
TRUNC(AVG(SUBSTR(ftscore,3,1)::INT),3) truncgoals FROM match GROUP BY seasonid ORDER BY 2;
 seasonid |            avg             | truncgoals
----------+----------------------------+------------
       15 | 0.9973684210526315789473 6 |      0.997
       14 | 1.0236842105263157894736 8 |      1.024
       13 | 1.0657894736842105263157 8 |      1.066
        9 | 1.0657894736842105263157 8 |      1.066
        7 | 1.0684210526315789473684 2 |      1.068
...
```

Network Address Functions

In this section:

This section covers network address functions and operators.

These functions are supported for use with the `IPV4`, `IPV6`, `MACADDR`, and `MACADDR8` data types.

The following operators work on network address data types:

Operator	Description
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
=	Equal to
<> or !=	Not equal to
~	Bitwise NOT
&	Bitwise AND
|	Bitwise OR

CONTAINS

These functions determine whether a specified IPV4 or IPV6 host address is contained by a network.

```
CONTAINS(network, integer, host_ip)
```

where `network` is the IPV4 or IPV6 address of the containing network, `integer` completes the network IP address by providing the length of the mask (number of leading bits set to 1), and `host_ip` is a single IPV4 or IPV6 host IP address. The `integer` must be in the range of `0` to `32` for IPV4 netmasks and `0` to `128` for IPV6 netmasks.

To specify a constant value for the `network` or `host_ip` arguments, use the `IPV4` or `IPV6` keyword.

These functions return a Boolean value (`true` , `false` , `unknown`).

For example:

```
premdb=# select hostip,contains(ipv4 '10.10.10.0',24,hostip) from myhosts;
   hostip    | contains
-------------+----------
 10.10.10.5  | t
 10.10.10.6  | t
 10.10.10.10 | t
 10.10.190.1 | f
(4 rows)

premdb=# select contains(ipv6 '684D:1111:222:3333:4444:5555:6:88',64,ipv6 '684D:1111:222:3333:4444:5555:6:77') from sys.const;
 contains
----------
 t
(1 row)

premdb=# select contains(ipv6 '684D:1111:222:3333:4444:5555:6:88',128,ipv6 '684D:1111:222:3333:4444:5555:6:77') from sys.const;
 contains
----------
 f
(1 row)
```

HOSTMASK

Construct a host mask for an IPV4 or IPV6 network.

```
HOSTMASK(address, integer)
HOSTMASK(address, integer)
```

where `address` is an IPV4 or IPV6 address or expression, and `integer` is the subnet mask. The `integer` values must be in the range of `0` to `32` for IPV4 and `0` to `128` for IPV6.

To specify a constant value for the `address` argument, use the `IPV4` or `IPV6` keyword.

For example:

```
premdb=# select hostip, hostmask(hostip,24) from myhosts;
   hostip    | hostmask
-------------+----------
 10.10.10.5  | 0.0.0.5
 10.10.10.6  | 0.0.0.6
 10.10.10.10 | 0.0.0.10
 10.10.190.1 | 0.0.0.1
(4 rows)

premdb=# select hostip, ipv4_hostmask(hostip,32) from myhosts;
   hostip    | ipv4_hostmask
-------------+---------------
 10.10.10.5  | 0.0.0.0
 10.10.10.6  | 0.0.0.0
 10.10.10.10 | 0.0.0.0
 10.10.190.1 | 0.0.0.0
(4 rows)
```

```
premdb=# select hostmask(ipv6 '2001:0db8:85a3:0000:0000:8a2e:0370:7334',24) from sys.const;
              hostmask
---------------------------------------
 0000:00b8:85a3:0000:0000:8a2e:0370:7334
(1 row)
```

INET_MERGE

Return the smallest network that includes both of the given IPV4 or IPV6 network IP addresses.

```
INET_MERGE(network1, integer1, network2, integer2)
```

where `integer1` and `integer2` are values for the subnet mask for each IP address. Integers must be in the range of `0` to `32` for IPV4 addresses and `0` to `128` for IPV6 addresses.

To specify a constant value for the network addresses, use the `IPV4` or `IPV6` keyword.

For example:

```
premdb=# select inet_merge(ipv4 '192.168.1.5', 24, ipv4 '192.168.2.5', 24) from sys.const;
   inet_merge
---------------
 192.168.0.0/22
(1 row)
```

```
premdb=# select ipv6_inet_merge('2001:0db8:85a3:0000:0000:8a2e:0370:0000', 128, '2001:0db8:85a3:0000:0000:8a2e:0370:7334', 128) fr
             ipv6_inet_merge
--------------------------------------------
 2001:0db8:85a3:0000:0000:8a2e:0370:0000/113
(1 row)
```

IPV4_GET_INET, IPV6_GET_INET

Extract a strongly typed IPV4 or IPV6 address from a text string and remove masks as needed.

Use the appropriate function for IPV4-typed strings and IPV6-typed strings.

```
IPV4_GET_INET(text)
IPV6_GET_INET(text)
```

For example:

```
premdb=# select ipv4_get_inet('192.168.1.0/10'),
ipv6_get_inet('2001:0db8:85a3:0000:0000:8a2e:0370:7334/20')
from sys.const;
 ipv4_get_inet |             ipv6_get_inet
---------------+----------------------------------------
 192.168.1.0   | 2001:0db8:85a3:0000:0000:8a2e:0370:7334
(1 row)
```

IPV4_GET_MASKBITS, IPV6_GET_MASKBITS

Platforms: All platforms

Parent topic: Network Address Functions

Return the number of netmask bits from a formatted IPV4 or IPV6 address string.

```
IPV4_GET_MASKBITS(text)
IPV6_GET_MASKBITS(text)
```

where `text` is a formatted string that represents an IP address. These functions return an integer. If there are no netmask bits in the input string, the result is `0`.

For example:

```
premdb=# select ipv4_get_maskbits('192.168.0.0/22') from sys.const;
 ipv4_get_maskbits
-------------------
                22
(1 row)
```

```
premdb=# select ipv6_get_maskbits('2001:0db8:85a3:0000:0000:8a2e:0370:7334') from sys.const;
 ipv6_get_maskbits
-------------------
                 0
(1 row)
```

IPV4_NETMASK, IPV6_NETMASK

Construct the netmask for an IPV4 or IPV6 network, given a number that specifies how many leading bits should be set to 1 in the result.

```
IPV4_NETMASK(integer)
IPV6_NETMASK(integer)
```

where `integer` is in the range of `0` to `32` for IPV4 netmasks and `0` to `128` for IPV6 netmasks.

Each octet in an IPV4 value occupies 8 bits. If the first 24 bits (from 32) are set to 1, the function returns `255.255.255.0`. Similarly, if the first 24 bits (from 128) in an IPV6 value are set to 1, the function returns `ffff:ff00:0000:0000:0000:0000:0000:0000`.

Examples

For example:

```
premdb=# select ipv4_netmask(24) from sys.const;
 ipv4_netmask
---------------
 255.255.255.0
(1 row)

premdb=# select ipv4_netmask(18) from sys.const;
 ipv4_netmask
---------------
 255.255.192.0
(1 row)

premdb=# select ipv6_netmask(18) from sys.const;
             ipv6_netmask
----------------------------------------
 ffff:c000:0000:0000:0000:0000:0000:0000
(1 row)

premdb=# select ipv6_netmask(128) from sys.const;
             ipv6_netmask
----------------------------------------
 ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff
(1 row)
```

MACADDR8_SET7BIT

Set the 7th bit of a `MACADDR8` value to one, also known as modified EUI-64 format, for inclusion in an `IPV6` address.

```
MACADDR8_SET7BIT | SET7BIT (macaddr8)
```

where `macaddr8` is a column, expression, or constant with that data type. For example:

```
premdb=# select macaddr8_set7bit(macaddr8 '00:34:56:ab:cd:ef') from sys.const;
   macaddr8_set7bit
-------------------------
 02:34:56:ff:ee:ab:cd:ef
(1 row)
```

```
premdb=# select c2, macaddr8_set7bit(c2) from ipaddrs;
          c2            |    macaddr8_set7bit
------------------------+------------------------
 10:20:30:aa:bb:cc:dd:ee | 12:20:30:aa:bb:cc:dd:ee
(1 row)
```

TEXT

Return an IPV4 or IPV6 address as a `VARCHAR` character string.

```
TEXT(address [,integer])
```

where `address` is an `IPV4` or `IPV6` column, expression, or constant. To specify a constant value for the address, use the `IPV4` or `IPV6` keyword.

Optionally, you can also specify a second `integer` argument that represents the subnet mask (in the range of `0` to `32` for `IPV4` or `0` to `128` for `IPV6`).

For example:

```
premdb=# select hostip, text(hostip) from myhosts;
   hostip    |    text
-------------+-------------
 10.10.10.5  | 10.10.10.5
 10.10.10.6  | 10.10.10.6
 10.10.10.10 | 10.10.10.10
 10.10.190.1 | 10.10.190.1
(4 rows)

premdb=# select hostip, text(hostip,16) from myhosts;
   hostip    |      text
-------------+----------------
 10.10.10.5  | 10.10.10.5/16
 10.10.10.6  | 10.10.10.6/16
 10.10.10.10 | 10.10.10.10/16
 10.10.190.1 | 10.10.190.1/16
(4 rows)
```

```
premdb=# select text(ipv6 '2001:0db8:85a3:0000:0000:8a2e:0370:0000') from sys.const;
                 text
---------------------------------------
 2001:0db8:85a3:0000:0000:8a2e:0370:0000
(1 row)

premdb=# select text(ipv6 '2001:0db8:85a3:0000:0000:8a2e:0370:0000',128) from sys.const;
                   text
-------------------------------------------
 2001:0db8:85a3:0000:0000:8a2e:0370:0000/128
(1 row)
```

TRUNC

This function sets the last 3 bytes of a `MACADDR` value to zero or the last 5 bytes of a `MACADDR8` value to zero.

```
TRUNC(macaddr | macaddr8)
```

where `macaddr` and `macaddr8` are either columns or expressions with those data types or literal values in a valid format. Literal values must be preceded with the `MACADDR` or `MACADDR8` keyword.

For example:

```
premdb=# \d ips
     Table "public.ips"
 Column |   Type   | Modifiers
--------+----------+-----------
 c1     | ipv4     |
 c2     | ipv6     |
 c3     | macaddr  |
 c4     | macaddr8 |

Distribution: Hash (c1)

premdb=# select c3, c4 from ips;
       c3         |           c4
------------------+------------------------
 10:20:30:40:00:00 | 10:20:30:40:00:00:00:00
(1 row)

premdb=# select trunc(c3), trunc(c4) from ips;
      trunc       |          trunc
------------------+------------------------
 10:20:30:00:00:00 | 10:20:30:00:00:00:00:00
(1 row)
```

```
premdb=# select trunc(macaddr '10:20:30:00:00:00') from sys.const;
      trunc
------------------
 10:20:30:00:00:00
(1 row)

premdb=# select trunc(macaddr8 '10:20:30:00:00:00:00:00') from sys.const;
         trunc
------------------------
 10:20:30:00:00:00:00:00
(1 row)
```

Pattern Matching

Platforms: All platforms

Parent topic: SQL Functions and Operators

In this section:

This section covers pattern matching operators and functions, including supported regular expression operators and syntax.

Regular Expression Details

Platforms: All platforms

Parent topic: Pattern Matching

In this section:

Yellowbrick supports a subset of the POSIX-style regular expressions, including support for:

- Character classes and escaped special characters
- Quantifiers
- Anchors
- Grouping constructs: Capturing and non-capturing groups and sub-expressions
- Alternation constructs
- Back references
- Flags

Yellowbrick does not support:

- Constraints and assertions

The following sections cover all of these constructs.

Alternation Constructs

Platforms: All platforms

Parent topic: Regular Expression Details

Yellowbrick supports the following constructs:

Operator	Description
|	OR operator **Note:** Either side of an OR operator (|) may contain an empty string.

Note: The Yellowbrick regular expression engine does not guarantee an order for matching expressions used in an alternation construct. The order that expressions are matched in an alternation construct is nondeterministic.

Anchors

Platforms: All platforms

Parent topic: Regular Expression Details

Anchors specify a position in the string before, after, or between characters where a match must occur.

The following anchors are supported:

Anchor	Description
`\m or \<`	Start of word
`\M or \>`	End of word
`\y`	Word boundary: start or end of word
`\Y`	Non-word boundary: neither start nor end of word
`^`	Start of string
`$`	End of string

Note: Unlike other regular expression engines, Yellowbrick does not support `\b` as a word-boundary anchor. Rather, Yellowbrick supports `\y` as a word-boundary anchor and `\b` as the backspace character.

Examples

```
premdb=> select * from team where stadium ~ '[Ll]a';
 teamid | htid | atid |       name        | nickname |    city    |     stadium     | capacity | avg_att
--------+------+------+-------------------+----------+------------+-----------------+----------+---------
      2 |    3 |   52 | Aston Villa       | Villains | Birmingham | Villa Park      |    42785 |  33.690
     21 |   22 |   71 | Leeds United      | Whites   | Leeds      | Elland Road     |    39460 |   0.000
     34 |   35 |   84 | Sheffield United  | Blades   | Sheffield  | Bramall Lane    |    32702 |   0.000
     41 |   42 |   91 | Tottenham Hotspur | Spurs    | London     | White Hart Lane |    36284 |  35.776
(4 rows)
```

```
premdb=> select * from team where stadium ~ '\y[Ll]a';
 teamid | htid | atid |       name        | nickname |   city    |     stadium     | capacity | avg_att
--------+------+------+-------------------+----------+-----------+-----------------+----------+---------
     34 |   35 |   84 | Sheffield United  | Blades   | Sheffield | Bramall Lane    |    32702 |   0.000
     41 |   42 |   91 | Tottenham Hotspur | Spurs    | London    | White Hart Lane |    36284 |  35.776
(2 rows)
```

```
premdb=> select * from team where stadium ~ '\Y[Ll]a';
 teamid | htid | atid |     name     | nickname |    city    |   stadium   | capacity | avg_att
--------+------+------+--------------+----------+------------+-------------+----------+---------
      2 |    3 |   52 | Aston Villa  | Villains | Birmingham | Villa Park  |    42785 |  33.690
     21 |   22 |   71 | Leeds United | Whites   | Leeds      | Elland Road |    39460 |   0.000
(2 rows)
```

Back References

Platforms: All platforms

Parent topic: Regular Expression Details

Back references identify sub-expressions that were previously matched by capturing groups within regular expressions and the replacement strings in the REGEXP_REPLACE function. Back references take the form `\n`, where `n` is the capturing group identifier, ranging from 1 to 9 (`\1` to `\9`).

Examples

The following example matches all cities which have a double character in the name. The `\1` refers to the previous capturing group, which holds a word character.

```
premdb=> select distinct(city) from team where city ~ '(\w)\1';
    city
---------------
 Blackpool
 Cardiff
 Liverpool
 Hull
 Sheffield
 Leeds
 Middlesbrough
 Nottingham
(8 rows)
```

The following example uses back references in REGEXP_REPLACE and the filter expression:

```
premdb=> select name, regexp_replace(name, '(\w+)\W+(\w+)(?:\W+)?', '[\1] [\2]') from team where name ~ '(\w)\1\w\M';
   name     | regexp_replace
------------+-----------------
 Aston Villa | [Aston] [Villa]
 Blackpool   | Blackpool
 Liverpool   | Liverpool
(3 rows)
```

Character Classes and Escaped Special Characters

Character Classes

Character classes specify the characters that will successfully match a single character from a single input string.

The following character classes are supported:

Short Form	Long Form	Description	Matches
\d	[[:digit:]]	Digits	0-9
\D	[^[:digit:]]	Non-digits	All characters except 0-9
\w	[[:alnum:]_]	Word	A-Z a-z 0-9 _
\W	[^[:alnum:]_]	Non-word	All characters except A-Z a-z 0-9 _
\s	[[:space:]]	All whitespace characters, including the line breaks	\t \r \n \v \f and the space character: ' '
\S	[^[:space:]]	Non-space	All characters except [[:space:]]
\xHH	[[:xdigit:]]	Hexadecimal digits	A-F a-f 0-9
	[[:ascii:]]	ASCII characters	0x00 - 0x7f
	[[:extend:]]	Extended characters	0x0 - 0xff
	[[:alpha:]]	Alphabetic characters	A-Z a-z
	[[:alnum:]]	Alphanumeric characters	A-Z a-z 0-9
	[[:lower:]]	Lowercase letters	a-z
	[[:upper:]]	Uppercase letters	A-Z
	[[:cntrl:]]	Control characters	0x00 - 0x1f and 0x7f
	[[:graph:]]	Visible characters	0x21 - 0x7e
	[[:print:]]	Visible characters and spaces	0x20 - 0x7e
	[[:punct:]]	Punctuation (and symbols)	! " # $ % & ' ( ) * + , - . / : ; < = > ? @ [\ ] ^ _ ` { | } ~
	[[:blank:]]	Space and tab	' ' \t

Note: Long-form expressions are intended to be used in a bracket expression. For example, [:digit:] without double brackets would match a single character from this set: ('d', 'i, 'g', 't', ':').

Character Classes Examples

The following example filters strings with two non-word characters:

```
premdb=> select name from team where name ~ '\W\w+\W';
        name
---------------------
 Queens Park Rangers
 West Bromwich Albion
 West Ham United
(3 rows)
```

The following example filters two spaces:

```
premdb=> select name from team where name ~ '[[:space:]]\w+[[:space:]]';
        name
---------------------
 Queens Park Rangers
 West Bromwich Albion
 West Ham United
(3 rows)
```

The following example demonstrates how omitting one set of brackets will match strings containing the characters `:`, `s`, `p`, `a`, `c`, `e` in addition to the word characters in between:

```
premdb=> select name from team where name ~ '[:space:]\w+[:space:]';
         name
-------------------------
 Arsenal
 Barnsley
 Blackburn Rovers
 Blackpool
 Bolton Wanderers
 Charlton Athletic
 Chelsea
 Crystal Palace
 Ipswich Town
 Leeds United
 Leicester City
 Liverpool
 Manchester City
 Manchester United
 Newcastle United
 Oldham Athletic
 Queens Park Rangers
 Sheffield United
 Sheffield Wednesday
 Southampton
 Sunderland
 Swansea City
 Tottenham Hotspur
 Wigan Athletic
 Wolverhampton Wanderers
(25 rows)
```

Escaped Special Characters

Escaped special non-alphanumeric characters are treated as ordinary characters.

The following escaped special characters are supported:

Escape	Description
`\a`	alert (bell) character, as in C
`\b`	backspace, as in C
`\B`	synonym for backslash (`\`) to help reduce the need for backslash doubling
`\e`	the character whose collating-sequence name is `ESC`
`\f`	form feed, as in C
`\n`	newline, as in C
`\r`	carriage return, as in C
`\t`	horizontal tab, as in C
`\v`	vertical tab, as in C
`\0`	the character whose value is 0 (the null byte)
`\xHH`	(where `hh` is exactly two digits) the character whose hexadecimal value is `0xhh`
`\xy`	(where `xy` is exactly two octal digits and is not a back reference) the character whose octal value is `0xy`
`\xyz`	(where `xyz` is exactly three octal digits and is not a back reference) the character whose octal value is `0xyz`

Bracket Expressions

Bracket expressions define a list of characters from which a match can be made. You can specify character ranges with the `-` operator. You can negate a set of characters by specifying the `^` operator as the first character inside a bracket expression.

For example:

```
[a-e]
[^a-e]
```

Equivalent classes and collations are not supported in bracket expressions. Constraint escapes are not supported in bracket expressions.

Bracket Expressions Examples

The following simple example uses a table called `words` that contains four strings:

```
premdb=# select * from words;
 c1
-------
thee
thou
thy
thine
(4 rows)
```

Pattern	Matches

Pattern	Matches
c1~'[iy]'	thy, thine
c1~'i|y'	thy, thine
c1~'(i|y)'	thy, thine
c1~'(?:i|y)'	thy, thine
c1~'ee'	thee
c1~'[a-e]'	thee, thine
c1~'[^a-e]'	thee, thou, thy, thine

Flags

Flags, also called modifiers, allow specific rules to be applied to the whole regular expression.

Flags may be passed to the REGEXP_REPLACE, REGEXP_INSTR, REGEXP_EXTRACT, and REGEXP_LIKE functions in the flags value.

Flag	Description
`c`	Sets matching to be case-sensitive
`i`	Sets matching to be case-insensitive

Note: Matching is case-sensitive by default.

Note: If contradicting flags are specified, the Yellowbrick engine uses the last option in the `flags` parameter.

Examples

```
premdb=# select REGEXP_REPLACE('Hello World','[hewo]','_','c') from sys.const;
 regexp_replace
----------------
 H_ll_ W_rld
(1 row)

premdb=# select REGEXP_REPLACE('Hello World','[hewo]','_','i') from sys.const;
 regexp_replace
----------------
 __ll_ __rld
(1 row)
```

Grouping Constructs

Platforms: All platforms

Parent topic: Regular Expression Details

Capturing groups create a sub-expression by placing an expression inside matching open and close parentheses. Quantifiers and optionality can be applied to sub-expressions. Each set of parentheses creates a capture number that is used to identify capturing groups, ranging from 1 to 9 (`\1` to `\9`). Capturing groups can be referenced later by their capture number in the regular expression or the replacement string in REGEXP_REPLACE.

Note: While only 9 can be referenced, Yellowbrick supports patterns that contain up to 16 capturing groups. Using more than 16 capturing groups will result in an error.

Non-capturing groups create a sub-expression without creating a capturing group. Non-capturing groups are indicated by including the question mark (`?`) and colon (`:`) after the opening parenthesis.

Expression	Description
(re)	re is a sub-expression and noted as a capturing group
(?:re)	re is a sub-expression in the non-capturing group

Note: It is recommended that capturing groups not be used when the group is the entire expression as this will only cause the engine to execute slower.

Examples

```
premdb=# select REGEXP_REPLACE('Hello World','(?:\w+) (?:\w+)','\& -> \1') from sys.const;
 regexp_replace
-----------------
 Hello World ->
(1 row)

premdb=# select REGEXP_REPLACE('Hello World','(\w+) (?:\w+)','\& -> \1') from sys.const;
   regexp_replace
----------------------
 Hello World -> Hello
(1 row)

premdb=# select REGEXP_REPLACE('Hello World','(?:\w+) (\w+)','\& -> \1') from sys.const;
   regexp_replace
----------------------
 Hello World -> World
(1 row)
```

Quantifiers

Platforms: All platforms

Parent topic: Regular Expression Details

Quantifiers specify the number of instances of a character that must be present in the input for a match to occur.

Yellowbrick supports both greedy and lazy quantifiers. Quantifiers are greedy by default and will try to match as many characters as possible. Alternatively, lazy quantifiers will try to match an element as few times as possible.

The following quantifiers are supported:

Quantifier	Matches
`*`	Sequence of 0 or more matches
`*?`	Lazy version of `*`
`+`	Sequence of 1 or more matches
`+?`	Lazy version of `+`
`?`	Sequence of 0 or 1 matches
`??`	Lazy version of `?`
`{m}`	Sequence of exactly `m` matches
`{m}?`	Lazy version of `{m}`
`{m,}`	Sequence of `m` or more matches
`{m,}?`	Lazy version of `{m,}`
`{m,n}`	Sequence of `m` through `n` (inclusive) matches where `m` cannot exceed `n`
`{m,n}?`	Lazy version of `{m,n}`

Note: Other regular expression engines apply the behavior of the first quantifier to all subsequent quantifiers in the expression. The Yellowbrick engine treats all quantifiers in the expression by their indicated behavior. For example, the Yellowbrick engine would evaluate `a+?b*` as a lazy quantifier followed by a greedy quantifier instead of treating both quantifiers as lazy.

Quantifier Examples

The `??` quantifier applies to the preceding character `e`. The string `Barnsley` matches because the pattern looks for 0 or 1 occurrences of `e` in that position in the string.

```
premdb=# select * from team where name ~ 'Barne??sley';
 teamid | htid | atid |   name   | nickname |   city   |     stadium     | capacity | avg_att
--------+------+------+----------+----------+----------+-----------------+----------+---------
      3 |    4 |   53 | Barnsley | Tykes    | Barnsley | Oakwell Stadium |    23009 |   0.000
(1 row)
```

The `{9,15}?` quantifier in this example looks for a sequence of between 9 and 15 characters anywhere in the string. The row with `teamid 43` qualifies because `Hawthorns` is a 9-character string.

```
premdb=# select * from team where stadium ~ '\w{9,15}?';
 teamid | htid | atid |         name         | nickname  |     city      |       stadium        | capacity | avg_att
--------+------+------+----------------------+-----------+---------------+----------------------+----------+---------
      6 |    7 |   56 | Blackpool            | Seasiders | Blackpool     | Bloomfield Road      |    17338 |   0.000
     26 |   27 |   76 | Middlesbrough        | Boro      | Middlesbrough | Riverside Stadium    |    34742 |   0.000
     35 |   36 |   85 | Sheffield Wednesday  | Owls      | Sheffield     | Hillsborough Stadium |    39732 |   0.000
     37 |   38 |   87 | Stoke City           | Potters   | Stoke         | Britannia Stadium    |    27902 |  27.534
     43 |   44 |   93 | West Bromwich Albion | Baggies   | Birmingham    | The Hawthorns        |    27000 |  24.631
(5 rows)
```

Again, the `{9,15}?` quantifier in this example looks for a sequence of between 9 and 15 characters anywhere in the string. However, the pattern only matches if a space character (`\s`) follows the string of alphanumeric characters.

```
premdb=# select * from team where stadium ~ '\w{9,15}?\s';
 teamid | htid | atid |        name         | nickname  |     city      |       stadium        | capacity | avg_att
--------+------+------+---------------------+-----------+---------------+----------------------+----------+---------
      6 |    7 |   56 | Blackpool           | Seasiders | Blackpool     | Bloomfield Road      |    17338 |   0.000
     26 |   27 |   76 | Middlesbrough       | Boro      | Middlesbrough | Riverside Stadium    |    34742 |   0.000
     35 |   36 |   85 | Sheffield Wednesday | Owls      | Sheffield     | Hillsborough Stadium |    39732 |   0.000
     37 |   38 |   87 | Stoke City          | Potters   | Stoke         | Britannia Stadium    |    27902 |  27.534
(4 rows)
```

```
premdb=> select * from team where city ~ 'po*';
 teamid | htid | atid |          name           | nickname  |     city      |       stadium       | capacity | avg_att
--------+------+------+-------------------------+-----------+---------------+---------------------+----------+---------
      6 |    7 |   56 | Blackpool               | Seasiders | Blackpool     | Bloomfield Road     |    17338 |   0.000
     17 |   18 |   67 | Everton                 | Toffees   | Liverpool     | Goodison Park       |    40221 |  38.124
     20 |   21 |   70 | Ipswich Town            | Blues     | Ipswich       | Portman Road        |    30311 |   0.000
     23 |   24 |   73 | Liverpool               | Reds      | Liverpool     | Anfield             |    44742 |  43.910
     36 |   37 |   86 | Southampton             | Saints    | Southampton   | St. Mary's Stadium  |    32505 |  30.751
     47 |   48 |   97 | Wolverhampton Wanderers | Wolves    | Wolverhampton | Molineux Stadium    |    31700 |   0.000
(6 rows)
```

```
premdb=> select * from team where city ~ 'po+';
 teamid | htid | atid |   name    | nickname  |   city    |     stadium     | capacity | avg_att
--------+------+------+-----------+-----------+-----------+-----------------+----------+---------
      6 |    7 |   56 | Blackpool | Seasiders | Blackpool | Bloomfield Road |    17338 |   0.000
     17 |   18 |   67 | Everton   | Toffees   | Liverpool | Goodison Park   |    40221 |  38.124
     23 |   24 |   73 | Liverpool | Reds      | Liverpool | Anfield         |    44742 |  43.910
(3 rows)
```

```
premdb=> select * from team where city ~ 'po{2,}';
 teamid | htid | atid |   name    | nickname  |   city    |     stadium     | capacity | avg_att
--------+------+------+-----------+-----------+-----------+-----------------+----------+---------
      6 |    7 |   56 | Blackpool | Seasiders | Blackpool | Bloomfield Road |    17338 |   0.000
     17 |   18 |   67 | Everton   | Toffees   | Liverpool | Goodison Park   |    40221 |  38.124
     23 |   24 |   73 | Liverpool | Reds      | Liverpool | Anfield         |    44742 |  43.910
(3 rows)
```

SQL Operators and Pattern Matching Functions

Platforms: All platforms

Parent topic: Pattern Matching

In this section:

Yellowbrick supports a subset of the POSIX-style regular expressions.

Within a Yellowbrick query, a pattern may be used in the following different ways:

- Preceded by one of the regular expression operators: ~, ~*, !~, !~*
- In a LIKE condition
- In a SIMILAR TO condition
- In a SUBSTRING function
- In one of the supported regular expression functions: REGEXP_REPLACE, REGEXP_INSTR, REGEXP_EXTRACT, REGEXP_LIKE

The following sections cover all of these constructs.

LIKE

Platforms: All platforms

Parent topic: SQL Operators and Pattern Matching Functions

Define a condition where an entire character string matches (or does not match) a pattern of characters.

Syntax

```
string [NOT] LIKE | ILIKE pattern [ESCAPE 'character']
```

string

A character string or an expression that evaluates to a character string.

LIKE | ILIKE

- Pattern matching with LIKE is case-sensitive. The operator `~~` is equivalent to LIKE.
- Pattern matching with ILIKE is case-insensitive. The operator `~~*` is equivalent to ILIKE.
- The `!~~` and `!~~*` operators are equivalent to NOT LIKE and NOT ILIKE, respectively.

pattern

The `pattern` for LIKE and ILIKE must contain constants or query parameters. You cannot apply LIKE and ILIKE to function expressions or table column names. The `pattern` is a character string, in quotes, that may include the following wildcard characters:

- A percent symbol (%) matches any set of characters.
- An underscore (_) matches any one character.

A pattern without wildcards behaves the same as a condition with the equals operator (=).

ESCAPE 'character'

Identifies a single escape character that you can use when you want pattern matching to find actual percent and underscore characters, rather than use these characters as wildcards. The character must be identified here, and it must be placed before the percent or underscore character in the `pattern` itself.

Examples

This example selects all the rows from the `match` table where full-time scores begin with `8`.

```
premdb=# SELECT * FROM match
WHERE ftscore LIKE '8-%'
ORDER BY seasonid;
 seasonid |      matchday       | htid | atid | ftscore | htscore
----------+---------------------+------+------+---------+---------
        8 | 1999-09-19 00:00:00 |   28 |   85 | 8-0     | 4-0
       16 | 2008-05-11 00:00:00 |   27 |   74 | 8-1     | 2-0
       18 | 2010-05-09 00:00:00 |   14 |   95 | 8-0     | 2-0
       20 | 2011-08-28 00:00:00 |   26 |   51 | 8-2     | 3-1
```

```
      21 | 2012-12-23 00:00:00 |   14 |   52 | 8-0     | 3-0
(5 rows)
```

The following example shows the behavior of the escape character. The third query returns only one value because it defines an escape character for the condition (`'/'`) and references it in the LIKE pattern (`LIKE '75/%'`). Therefore, `%` , in this case, does not represent a wildcard.

```
premdb=# SELECT * FROM escapetest;
    c1
------------
 75p
 75%
(2 rows)

premdb=# SELECT * FROM escapetest WHERE c1 LIKE '75%';
    c1
------------
 75p
 75%
(2 rows)

premdb=# SELECT * FROM escapetest WHERE c1 LIKE '75/%' ESCAPE '/';
    c1
------------
 75%
(1 row)
```

You cannot use LIKE to compare one named table column with another:

```
premdb=# select * from team where name like city;
ERROR:  Pattern found in LIKE must be a string literal
```

REGEXP_EXTRACT

Return the matching text item.

When the pattern does not contain capturing groups, the REGEXP_EXTRACT function is similar to the SUBSTRING function.

When the pattern does contain capturing groups, the REGEXP_EXTRACT function will still return the entire matched input while the SUBSTRING function will only return the results of the first capturing group.

Syntax

```
result = REGEXP_EXTRACT(input, pattern  [, start [, reference]] [, flags])
```

input value

Specify the value against which the regular expression is matched.

pattern value

Specify the regular expression to be matched against the `input`.

The `pattern` value must be a constant string and cannot be a column reference.

start value

The position in the `input` from where the `pattern` match starts. The default start position is 1.

Yellowbrick does not support special meanings for negative `start` values.

The `start` values can be column references.

reference value

Specify which `pattern` match is of interest. The default value is 1.

The `reference` values can be column references.

flags value

Can be either `i` for case-insensitive matching or `c` for case-sensitive matching. Matching is case-sensitive by default.

The `flags` value must be a constant string and cannot be a column reference.

Return

If the `pattern` matches the `input`, the function returns the matched value.

If the `pattern` does not match the `input`, the function returns `NULL`.

Example

```
premdb=> select season_name,REGEXP_EXTRACT(season_name,'(\d+)-(\d+)') from season where numteams=22;
 season_name | regexp_extract
-------------+----------------
 1992-1993   | 1992-1993
 1993-1994   | 1993-1994
 1994-1995   | 1994-1995
(3 rows)
```

REGEXP_INSTR

Platforms: All platforms

Parent topic: SQL Operators and Pattern Matching Functions

Return the index of the matching item.

Syntax

```
result = REGEXP_INSTR(input, pattern [, start [, reference]] [, flags])
```

input value

Specify the value against which the regular expression is matched.

pattern value

Specify the regular expression to be matched against the `input`.

The `pattern` value must be a constant string and cannot be a column reference.

start value

The position in the `input` from where the `pattern` match starts. The default start position is 1.

Yellowbrick does not support special meanings for negative `start` values.

The `start` values can be column references.

reference value

Specify which `pattern` match is of interest. The default value is 1.

The `reference` values can be column references.

flags value

Can be either `i` for case-insensitive or `c` for case-sensitive matching. Matching is case-sensitive by default.

The `flags` value must be a constant string and cannot be a column reference.

Return

If the `pattern` is matched, the function returns the position of the matching text item.

If the `pattern` is not matched, the function returns `0`.

Example

```
premdb=> select season_name,REGEXP_INSTR(season_name,'-') from season where numteams=22;
 season_name | regexp_instr
-------------+--------------
 1992-1993   |            5
 1993-1994   |            5
 1994-1995   |            5
(3 rows)
```

REGEXP_LIKE

Return `t` if the pattern matches at least one occurrence in the input.

Return `f` if there is no match.

Syntax

```
result = REGEXP_LIKE(input, pattern [, start [, reference]] [, flags])
```

input value

Specify the value against which the regular expression is matched.

pattern value

Specify the regular expression to be matched against the `input`.

The `pattern` value must be a constant string and cannot be a column reference.

start value

The position in the `input` from where the `pattern` match starts. The default start position is 1.

Yellowbrick does not support special meanings for negative `start` values.

The `start` values can be column references.

reference value

Specify which `pattern` match is of interest.

The `reference` values can be column references.

Note: The `reference` parameter for `REGEXP_LIKE` is supported by Yellowbrick but may not be supported by other databases.

flags value

Can be either `i` for case-insensitive matching or `c` for case-sensitive matching. Matching is case-sensitive by default.

The `flags` value must be a constant string and cannot be a column reference.

Return

If the `pattern` matches the `input` at least once, the function returns `t`.

If the `pattern` does not match the `input`, the function returns `f`.

Example

```
premdb=> select name from team where REGEXP_LIKE(name, '([a-z])\1\M');
    name
-------------
 Cardiff City
 Hull City
(2 rows)
```

REGEXP_REPLACE

Replace instances of a pattern in the input with the value in the replacement.

Back references can be used within the replacement string to match expressions within capturing groups. Capturing groups can range from `1` to `9` (`\1` to `\9`).

The entire match can be retrieved with `\&` .

Syntax

```
result = REGEXP_REPLACE(input, pattern, replacement [, start [, reference]] [, flags])
```

input value

Specify the value against which the regular expression is matched.

pattern value

Specify the regular expression to be matched against the `input` .

When the `pattern` matches, the matched string from the `input` will be replaced by the value specified in the `replacement` argument.

The `pattern` value must be a constant string and cannot be a column reference.

replacement value

Specify the value to substitute for each instance of the `pattern` .

start value

The position in the `input` from where the `pattern` match starts. The default start position is 1.

Yellowbrick does not support special meanings for negative `start` values.

The `start` values can be column references.

reference value

Specify which match of the `pattern` to replace.

To replace a specific instance of the `pattern` , use a `reference` value greater than 0 to indicate the occurrence. For example, a `reference` value of 1 would replace the first match of the `pattern` , a `reference` value of 2 would replace the second match of the `pattern` , and so on.

To replace all occurrences, omit a `reference` value or use a `reference` value of 0, both of which are interpreted similarly and result in a global replace.

The `reference` values can be column references.

flags value

Can be either `i` for case-insensitive or `c` for case-sensitive matching. Matching is case-sensitive by default.

The `flags` value must be a constant string and cannot be a column reference.

Return

If the `pattern` matches the `input`, the function applies the `replacement` and returns the result.

If the `pattern` doesn't match the `input`, the function returns the `input` string.

Example

```
premdb=> select season_name,REGEXP_REPLACE(season_name,'(\d{4,4})','year:\1', 1, 1) from season where numteams=22;
 season_name | regexp_replace
-------------+----------------
 1992-1993   | year:1992-1993
 1993-1994   | year:1993-1994
 1994-1995   | year:1994-1995
(3 rows)
```

SIMILAR TO

Platforms: All platforms

Parent topic: SQL Operators and Pattern Matching Functions

Define a condition where an entire character string matches (or does not match) a pattern of characters.

Syntax

```
string [NOT] SIMILAR TO pattern [ESCAPE 'character']
```

pattern

SIMILAR TO is akin to LIKE, in that the SIMILAR TO operator succeeds only if its pattern matches the entire string. A SIMILAR TO `pattern` may include POSIX regular expression notation, but the results are likely to differ from regular expression results where the `pattern` can match any part of the string. The `pattern` also supports the following wildcard characters:

- % (matches any set of characters)
- _ (matches any one character)

The `pattern` for SIMILAR TO must contain constants or query parameters. You cannot apply SIMILAR TO to function expressions, the results of subqueries, or names of table columns.

ESCAPE 'character'

Identifies a single escape character that you can use when you want pattern matching to find actual percent and underscore characters, rather than use these characters as wildcards. The character must be identified in the ESCAPE clause, and that character must be placed before the percent or underscore character in the `pattern` itself.

The `character` for SIMILAR TO must contain constants or query parameters, same as `pattern`.

For example:

```
premdb=# SELECT * FROM match
WHERE ftscore SIMILAR TO '8-%';
 seasonid |      matchday       | htid | atid | ftscore | htscore
----------+---------------------+------+------+---------+---------
       16 | 2008-05-11 00:00:00 |   27 |   74 | 8-1     | 2-0
       18 | 2010-05-09 00:00:00 |   14 |   95 | 8-0     | 2-0
       20 | 2011-08-28 00:00:00 |   26 |   51 | 8-2     | 3-1
       21 | 2012-12-23 00:00:00 |   14 |   52 | 8-0     | 3-0
        8 | 1999-09-19 00:00:00 |   28 |   85 | 8-0     | 4-0
(5 rows)
```

You can specify a regular expression-style array as part of a `SIMILAR TO` condition. For example, this query looks for `ftscore` values that end with 6, 7, 8, or 9:

```
premdb=# SELECT * FROM match
WHERE SUBSTR(ftscore,3,1) SIMILAR TO '%(6|7|8|9)'
ORDER BY seasonid,matchday;
 seasonid |      matchday       | htid | atid | ftscore | htscore
```

```
----------+---------------------+------+------+---------+---------
       3 | 1994-08-20 00:00:00 |   16 |   73 | 1-6     |
       3 | 1995-04-01 00:00:00 |   36 |   79 | 1-7     |
       4 | 1996-02-25 00:00:00 |    8 |   75 | 0-6     | 0-2
       6 | 1997-08-24 00:00:00 |    4 |   63 | 0-6     | 0-3
       6 | 1997-12-06 00:00:00 |   42 |   63 | 1-6     | 1-1
       6 | 1998-05-02 00:00:00 |   47 |   91 | 2-6     | 2-2
       7 | 1999-01-16 00:00:00 |   23 |   75 | 2-6     | 1-1
       7 | 1999-02-06 00:00:00 |   30 |   75 | 1-8     | 1-2
       7 | 1999-04-24 00:00:00 |   27 |   51 | 1-6     | 0-3
      10 | 2002-02-09 00:00:00 |   21 |   73 | 0-6     | 0-2
      11 | 2003-04-05 00:00:00 |   13 |   71 | 1-6     | 1-3
      11 | 2003-04-12 00:00:00 |   28 |   75 | 2-6     | 1-4
      11 | 2003-04-26 00:00:00 |   44 |   73 | 0-6     | 0-1
      16 | 2008-04-12 00:00:00 |   17 |   52 | 0-6     | 0-3
      16 | 2008-04-28 00:00:00 |   17 |   51 | 2-6     | 1-2
      18 | 2009-08-15 00:00:00 |   18 |   51 | 1-6     | 0-3
      18 | 2010-04-03 00:00:00 |   11 |   74 | 1-6     | 0-5
      19 | 2010-08-21 00:00:00 |   46 |   63 | 0-6     | 0-1
      20 | 2011-10-23 00:00:00 |   26 |   74 | 1-6     | 0-1
      20 | 2012-04-14 00:00:00 |   29 |   74 | 1-6     | 0-2
      21 | 2013-04-27 00:00:00 |   28 |   73 | 0-6     | 0-2
      22 | 2014-03-22 00:00:00 |   12 |   73 | 3-6     | 2-2
(22 rows)
```

You can run an equivalent query by using several LIKE conditions with OR.

SIMILAR TO: Best Practices

To improve performance for `SIMILAR TO` patterns, follow the tips in this section or consider using regular expressions directly. `SIMILAR TO` expressions are converted to regular expressions.

The syntax `SIMILAR TO 'pattern'` is converted to `regex('^(?:pattern)$')`, which alternatively can be written as `~ '^(?:pattern)$'`

For example, the following expression:

```
SIMILAR TO '(%one%|%two%)'
```

is converted to:

```
regex('^(?:.*one.*|.*two.*)$')
```

The wildcard `%` sign is converted to `.*` (any character repeated zero or more times) and `_` is converted to `.` (any single character). As an essential part of the conversion, regular expression special characters (such as `^`, `$`, and `.`) are properly escaped to preserve the original `SIMILAR TO` pattern behavior.

For example, consider the following query with `SIMILAR TO`, which returns `true` (a match):

```
premdb=# select '123.456$' similar to '123.456$' from sys.const;
 ?column?
----------
 t
(1 row)
```

To preserve the semantics of this expression when it is converted, the `.` and `$` characters must be escaped with the `\` character:

```
premdb=# select '123.456$' ~ '^(?:123\.456\$)$' from sys.const;
 ?column?
----------
 t
(1 row)
```

The following sections suggest some `SIMILAR TO` patterns that you should consider rewriting (either as more optimal `SIMILAR TO` expressions or as regular expressions).

Rewrite (%...%|%...%) as %(...|...)%

For example, rewrite this `SIMILAR TO` expression:

```
name SIMILAR TO '(%Manchester United%|%Man Utd%|%Man United%|%Man U%)'
```

as follows:

```
name SIMILAR TO '%(Manchester United|Man Utd|Man United|Man U)%'
```

Alternatively, you could rewrite the same expression as the following regular expression:

```
name ~ '(Manchester United|Man Utd|Man United|Man U)'
```

Both expressions will return the same result. For example:

```
premdb=# select * from team
where name SIMILAR TO '%(Manchester United|Man Utd|Man United|Man U)%';
 teamid | htid | atid |       name        |  nickname  |    city    |   stadium    | capacity | avg_att
--------+------+------+-------------------+------------+------------+--------------+----------+---------
     25 |   26 |   75 | Manchester United | Red Devils | Manchester | Old Trafford |    75635 |  75.286
(1 row)
premdb=# select * from team
where name ~ '(Manchester United|Man Utd|Man United|Man U)';
 teamid | htid | atid |       name        |  nickname  |    city    |   stadium    | capacity | avg_att
--------+------+------+-------------------+------------+------------+--------------+----------+---------
     25 |   26 |   75 | Manchester United | Red Devils | Manchester | Old Trafford |    75635 |  75.286
(1 row)
```

Rewrite '(%lower case pattern%|% upper case pattern%)' as ~* '(lower case pattern)'

For example, you could rewrite this `SIMILAR TO` expression:

```
c1 SIMILAR TO '(%trial%|%TRIAL%|%TRAIL%|%trail%|%Trail%|%Trial%)'
```

as follows:

```
c1 SIMILAR TO '%(trial|TRIAL|TRAIL|trail|Trail|Trial)%'
```

If you use a regular expression rewrite, you can also apply a case-insensitive match. For example:

```
c1 ~* '(?:trial|trail)'
```

However, remember that case-insensitive matching in a regular expression may find more matches than an equivalent `SIMILAR TO` expression, which may or may not be what you want. For example, note the different results from the following two queries:

```
premdb=# select distinct winners from season where winners ~* 'ch';
     winners
------------------
 Manchester United
 Chelsea
 Manchester City
(3 rows)

premdb=# select distinct winners from season where winners similar to '%ch%';
     winners
------------------
 Manchester United
```

```
 Manchester City
(2 rows)
```

Rewrite (...%|...%|...%) as (..[.]|...)%

For example, rewrite this `SIMILAR TO` expression:

```
c1 SIMILAR TO '(12%|13%|14%|15%|16%|17%|0B%|0S%|BU%|''|0Y%|0Z%|VT%)'
```

as follows:

```
c1 SIMILAR TO '(12|13|14|15|16|17|0B|0S|BU|0Y|0Z|VT)%' OR c1 = ''
```

Take the rewrite a step further by using character classes to reduce the number of optional sub-expressions. For example, a better rewrite in this case is as follows:

```
c1 SIMILAR TO '(1[234567]|0[BSYZ]|BU|VT)%' OR c1 = ''
```

This is an example where a `SIMILAR TO` expression is still being specified but it contains regular expression constructs (character classes in this case).

Simple Regular Expression Examples

The following examples are very simple regular expressions.

This example finds strings that contain "pool" (lowercase):

```
premdb=# SELECT name FROM team WHERE name ~ 'pool';
   name
-----------
 Blackpool
 Liverpool
(2 rows)
```

The following example finds strings that contain "POOL" (case-insensitive):

```
premdb=# SELECT name FROM team WHERE name ~* 'POOL';
   name
-----------
 Blackpool
 Liverpool
(2 rows)
```

The following example finds strings that do not contain "a" or "e":

```
premdb=# SELECT name FROM team WHERE name !~* 'a|e';
    name
--------------
 Hull City
 Ipswich Town
 Norwich City
 Portsmouth
 Swindon Town
(5 rows)
```

The following example finds strings that contain "pool," "ham," or "ton":

```
premdb=# SELECT name FROM team WHERE name ~ 'pool|ham|ton';
         name
------------------------
 Aston Villa
 Birmingham City
 Blackpool
 Bolton Wanderers
 Charlton Athletic
 Everton
 Fulham
 Liverpool
 Nottingham Forest
 Oldham Athletic
```

```
 Southampton
 Tottenham Hotspur
 Wolverhampton Wanderers
(13 rows)
```

The following example finds strings that contain "ee":

```
premdb=# SELECT name FROM team WHERE name ~ 'e{2}';
       name
--------------------
 Leeds United
 Queens Park Rangers
(2 rows)
```

The following examples finds strings that contain "200" followed by a single character:

```
premdb=# select * from season where season_name ~ '200.';
 seasonid | season_name | numteams |      winners
----------+-------------+----------+-------------------
       8 | 1999-2000   |       20 | Manchester United
       9 | 2000-2001   |       20 | Manchester United
      10 | 2001-2002   |       20 | Arsenal
      11 | 2002-2003   |       20 | Manchester United
...
```

The following example finds strings that contain a "3":

```
premdb=# select * from season where season_name ~ '3';
 seasonid | season_name | numteams |      winners
----------+-------------+----------+-------------------
       1 | 1992-1993   |       22 | Manchester United
       2 | 1993-1994   |       22 | Manchester United
      11 | 2002-2003   |       20 | Manchester United
      12 | 2003-2004   |       20 | Arsenal
      21 | 2012-2013   |       20 | Manchester United
      22 | 2013-2014   |       20 | Manchester City
(6 rows)
```

SQL Regular Expression Operators

Platforms: All platforms

Parent topic: SQL Operators and Pattern Matching Functions

Yellowbrick supports the following operators for pattern matching:

Operator	Description
~	Find strings that match (case-sensitive)
~*	Find strings that match (case-insensitive)
!~	Find strings that do not match (case-sensitive)
!~*	Find strings that do not match (case-insensitive)

These operators locate matches *anywhere* in the string.

Note: These operators are different from the operators that represent LIKE and ILIKE (`~~` , `~~*` , and so on).

SUBSTRING (Pattern Match)

Platforms: All platforms

Parent topic: SQL Operators and Pattern Matching Functions

Extract a substring that matches a POSIX regular expression pattern.

Return the portion of the text that matches the pattern. If the pattern contains capturing groups, then the portion of text that matched the first parenthesized expression is returned.

Return `null` if there is no match.

See also SUBSTR and SUBSTRING.

Syntax

```
SUBSTRING(string FROM posix_regex_pattern)
```

posix_regex_pattern

Specify the regular expression to be matched against the `input`.

The `posix_regex_pattern` must be a constant string and cannot be a column reference.

Examples

The following example selects names from the `team` table where anything followed by 'City' or 'United' appears in the name.

```
premdb=# select name, substring(name from 'City|United')
from team
where name like '%City' or name like '%United';
      name        | substring
------------------+-----------
 Birmingham City   | City
 Bradford City     | City
 Cardiff City      | City
 Coventry City     | City
 Hull City         | City
 Leeds United      | United
 Leicester City    | City
 Manchester City   | City
 Manchester United | United
 Newcastle United  | United
 Norwich City      | City
 Sheffield United  | United
 Stoke City        | City
 Swansea City      | City
 West Ham United   | United
(15 rows)
```

Pay special attention when using parenthesized expressions as they can match an empty string. If this subexpression is the first capturing group, then substring may return an empty string. Consider using a non-capturing group to avoid this behavior.

```
yellowbrick_test=# select substring('ab', '(z*)b');
 ?column?
----------

(1 row)
```

The following example uses a non-capturing group, which changes the result from the previous example. The empty string is matched in the example, but it is not taken into account when returning the result.

```
yellowbrick_test=# select substring('ab', '(?:z*)b');
 substring
-----------
 b
(1 row)
```

SQL Conditions

Platforms: All platforms

Parent topic: SQL Functions and Operators

In this section:

Yellowbrick supports simple and complex SQL search conditions that use the constructs described in this section.

LIKE and SIMILAR TO are covered in the section on pattern matching.

Note: Conditions defined with `ALL` subquery expressions are not supported.

ANY

Define a comparison condition that returns true when the comparison is true for at least one value returned by a subquery. This kind of condition is synonymous with the IN condition.

```
ANY | SOME (subquery)
```

where `subquery` returns one column with 0 or more rows. ANY and SOME are synonyms.

For example:

```
premdb=# SELECT seasonid FROM season
WHERE seasonid=ANY(SELECT seasonid FROM match);
 seasonid
----------
       1
       2
       3
       4
       5
       6
       7
       8
       9
...
```

This query is equivalent to:

```
SELECT seasonid FROM season
WHERE seasonid IN(SELECT seasonid FROM match);
```

BETWEEN

Define a condition that qualifies rows based on a range of numbers, datetime values, or character strings.

```
expression [ NOT ] BETWEEN [ SYMMETRIC ] expression AND expression
```

For example:

```
premdb=# SELECT * FROM match
WHERE matchday BETWEEN '2013-05-15' AND '2013-05-31' ORDER BY 1;
 seasonid |      matchday       | htid | atid | ftscore | htscore
----------+---------------------+------+------+---------+---------
       21 | 2013-05-19 00:00:00 |   14 |   67 | 2-1     | 1-1
       21 | 2013-05-19 00:00:00 |   24 |   82 | 1-0     | 1-0
       21 | 2013-05-19 00:00:00 |   25 |   78 | 2-3     | 1-1
       21 | 2013-05-19 00:00:00 |   28 |   51 | 0-1     | 0-0
       21 | 2013-05-19 00:00:00 |   37 |   87 | 1-1     | 0-0
       21 | 2013-05-19 00:00:00 |   40 |   68 | 0-3     | 0-1
       21 | 2013-05-19 00:00:00 |   42 |   88 | 1-0     | 0-0
       21 | 2013-05-19 00:00:00 |   44 |   75 | 5-5     | 1-3
       21 | 2013-05-19 00:00:00 |   45 |   83 | 4-2     | 2-0
       21 | 2013-05-19 00:00:00 |   46 |   52 | 2-2     | 2-1
(10 rows)
```

BETWEEN SYMMETRIC is a variation on the standard BETWEEN syntax that can swap the range boundaries. The first argument does not have to be less than or equal to the second argument. For example, the first query below finds no matches, but the second query swaps the range boundaries and finds 11 rows:

```
premdb=# select * from team where teamid between 60 and 40;
 teamid | htid | atid | name | nickname | city | stadium | capacity
--------+------+------+------+----------+------+---------+----------
(0 rows)

premdb=# select * from team where teamid between symmetric 60 and 40;
 teamid | htid | atid |          name           | nickname |     city      |     stadium      | capacity
--------+------+------+-------------------------+----------+---------------+------------------+----------
    40 |   41 |   90 | Swindon Town            | Robins   | Swindon       | County Ground    |    15728
    41 |   42 |   91 | Tottenham Hotspur       | Spurs    | London        | White Hart Lane  |    36284
    42 |   43 |   92 | Watford                 | Hornets  | Watford       | Vicarage Road    |    21577
    43 |   44 |   93 | West Bromwich Albion    | Baggies  | Birmingham    | The Hawthorns    |    27000
    44 |   45 |   94 | West Ham United         | Hammers  | London        | Upton Park       |    35016
    45 |   46 |   95 | Wigan Athletic          | Latics   | Wigan         | DW Stadium       |    25138
    46 |   47 |   96 | Wimbledon               | Dons     | London        | Selhurst Park    |    26255
    47 |   48 |   97 | Wolverhampton Wanderers | Wolves   | Wolverhampton | Molineux Stadium |    31700
    48 |   49 |   98 |                         |          |               |                  |        0
    49 |   50 |   99 |                         |          |               |                  |        0
    50 |   51 |  100 |                         |          |               |                  |        0
(11 rows)
```

Comparison Operators

Yellowbrick Documentation > Reference > SQL Reference > SQL Functions and Operators > SQL Conditions > Comparison Operators

Platforms: All platforms

Parent topic: SQL Conditions

Yellowbrick supports the following comparison operators. See also SQL Conditions.

Operator	Definition
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
=	Equal to
<> or !=	Not equal to
`BETWEEN`, `NOT BETWEEN`	Comparison with an inclusive range of values. `BETWEEN SYMMETRIC` is also supported.
`IS NULL`, `IS NOT NULL`	Comparison with `null` values. `ISNULL` and `NOTNULL` synonyms are also supported. See also ISNULL.
`IS TRUE`, `IS NOT TRUE`	Comparison with Boolean values
`IS FALSE`, `IS NOT FALSE`	Comparison with Boolean values
`IS UNKNOWN`, `IS NOT UNKNOWN`	Comparison with Boolean values (finds `null` values in Boolean columns)
`IS DISTINCT FROM`, `IS NOT DISTINCT FROM`	Identical to `!=` or `=` when neither of the inputs is `null`. These operators treat `null` as a normal data value, not as "unknown." `IS DISTINCT FROM` returns `false` when both inputs are `null` and returns `true` when one of the inputs is `null`. `IS NOT DISTINCT FROM` returns `true` when both inputs are `null` and returns `false` when one of the inputs is `null`.

Examples

```
premdb=# select * from season where winners is null;
 seasonid | season_name | numteams | winners
----------+-------------+----------+---------
       25 | 2016-2017   |       20 |
(1 row)
```

```
premdb=# select * from team where avg_att >=50;
 teamid | htid | atid |       name        |  nickname  |    city    |     stadium      | capacity | avg_att
--------+------+------+-------------------+------------+------------+------------------+----------+---------
      1 |    2 |   51 | Arsenal           | Gunners    | London     | Emirates Stadium |    60260 |  59.944
     24 |   25 |   74 | Manchester City   | Citizens   | Manchester | Etihad Stadium   |    55097 |  54.041
     25 |   26 |   75 | Manchester United | Red Devils | Manchester | Old Trafford     |    75635 |  75.286
(3 rows)
```

```
premdb=# select * from season where seasonid is not distinct from 1;
 seasonid | season_name | numteams |      winners
----------+-------------+----------+-------------------
        1 | 1992-1993   |       22 | Manchester United
(1 row)
```

EXISTS

Define a condition that tests whether a subquery returns rows. Return `true` if the subquery returns at least one row.

```
[ NOT ] EXISTS (subquery)
```

NOT EXISTS means that the condition returns `true` if the subquery returns no rows.

Examples

```
premdb=# SELECT seasonid FROM season
WHERE EXISTS(SELECT seasonid FROM match);
 seasonid
----------
       1
       2
       3
       4
       5
       6
       7
...
```

```
premdb=# SELECT * FROM awayteam
WHERE EXISTS(SELECT htid FROM hometeam WHERE htid=awayteam.atid);
 atid |  name
------+---------
   51 | Arsenal
(1 row)
```

IN

Yellowbrick Documentation > Reference > SQL Reference > SQL Functions and Operators > SQL Conditions > IN

Platforms: All platforms

Parent topic: SQL Conditions

Define a condition that compares an expression with a list of literal values or the results of a subquery.

```
expression [ NOT ] IN (list | subquery)
```

where `subquery` returns one column with 0 or more rows.

Note: An `IN` list can contain a maximum of 200000 values.

For example:

```
premdb=# SELECT name, city FROM team
WHERE city in('London','Liverpool','Birmingham') ORDER BY 2,1;
        name         |    city
---------------------+-----------
 Aston Villa         | Birmingham
 Birmingham City     | Birmingham
 West Bromwich Albion | Birmingham
 Everton             | Liverpool
 Liverpool           | Liverpool
 Arsenal             | London
 Charlton Athletic   | London
 Chelsea             | London
 Crystal Palace      | London
 Fulham              | London
 Queens Park Rangers | London
 Tottenham Hotspur   | London
 West Ham United     | London
 Wimbledon           | London
(14 rows)
```

The following example uses a subquery as the input to the IN condition:

```
premdb=# SELECT seasonid FROM season
WHERE seasonid IN(SELECT seasonid FROM match);
 seasonid
----------
       1
       2
       3
       4
       5
       6
       7
       8
       9
...
```

Note: In general, row comparisons with multi-value `IN` lists are supported. For example:

```
premdb=# select * from match where(htid, atid) not in(select htid, atid from team) limit 5;
 seasonid |      matchday       | htid | atid | ftscore | htscore
----------+---------------------+------+------+---------+---------
        1 | 1992-08-01 00:00:00 |    2 |   52 | 0-1     | -
        1 | 1992-08-01 00:00:00 |    2 |   55 | 0-1     | -
        1 | 1992-08-01 00:00:00 |    2 |   63 | 2-1     | -
        1 | 1992-08-01 00:00:00 |    2 |   64 | 3-0     | -
        1 | 1992-08-01 00:00:00 |    2 |   65 | 3-0     | -
(5 rows)
```

However, Yellowbrick does not support row comparisons for multi-value `NOT IN` lists when an expression on either side of the `IN` condition evaluates to `NULL`.

To work around this problem, either remove `null` values from the data or rewrite the query by using an `EXISTS` or `NOT EXISTS` subquery.

For example, you can rewrite this query:

```
select * from t1 where
(t1.c1, t1.c2) not in (select * from t2);
```

as follows:

```
select * from t1 where
not exists (select 1 from t2 where t1.c1=t2.c1 and t1.c2=t2.c2);
```

Logical Operators

Yellowbrick Documentation > Reference > SQL Reference > SQL Functions and Operators > SQL Conditions > Logical Operators

Platforms: All platforms

Parent topic: SQL Conditions

Yellowbrick supports the following logical SQL operators:

- `AND`
- `OR`
- `NOT`

Boolean logic is applied to these operators, returning `TRUE`, `FALSE`, or `UNKNOWN`.

`AND` and `OR` are commutative; an expression returns the same result regardless of the order of the operands. For example, the following expressions produce the same results:

```
where seasonid=1 and season_name='1992-1993'
where season_name='1992-1993' and seasonid=1
```

You can use parentheses to define how the logical operators should be applied when multiple operators are used in the same complex expression.

The following simple queries use the logical operators in search conditions:

```
premdb=# select * from season where seasonid=1 or seasonid=2;
 seasonid | season_name | numteams |      winners
----------+-------------+----------+-------------------
       1 | 1992-1993   |       22 | Manchester United
       2 | 1993-1994   |       22 | Manchester United
(2 rows)

premdb=# select * from season where seasonid=1 and not seasonid=2;
 seasonid | season_name | numteams |      winners
----------+-------------+----------+-------------------
       1 | 1992-1993   |       22 | Manchester United

premdb=# select * from season where seasonid >1 and not numteams=20;
 seasonid | season_name | numteams |      winners
----------+-------------+----------+-------------------
       2 | 1993-1994   |       22 | Manchester United
       3 | 1994-1995   |       22 | Blackburn Rovers
(2 rows)
```

The following two queries show how a use of parentheses changes the logic of the search condition:

```
premdb=# select * from season where seasonid<10 or numteams=22 and winners='Manchester United';
 seasonid | season_name | numteams |      winners
----------+-------------+----------+-------------------
       1 | 1992-1993   |       22 | Manchester United
       2 | 1993-1994   |       22 | Manchester United
       3 | 1994-1995   |       22 | Blackburn Rovers
       4 | 1995-1996   |       20 | Manchester United
       5 | 1996-1997   |       20 | Manchester United
       6 | 1997-1998   |       20 | Arsenal
       7 | 1998-1999   |       20 | Manchester United
```

```
       8 | 1999-2000   |       20 | Manchester United
       9 | 2000-2001   |       20 | Manchester United
(9 rows)

premdb=# select * from season where (seasonid<10 or numteams=22) and winners='Manchester United';
 seasonid | season_name | numteams |      winners
----------+-------------+----------+-------------------
       1 | 1992-1993   |       22 | Manchester United
       2 | 1993-1994   |       22 | Manchester United
       4 | 1995-1996   |       20 | Manchester United
       5 | 1996-1997   |       20 | Manchester United
       7 | 1998-1999   |       20 | Manchester United
       8 | 1999-2000   |       20 | Manchester United
       9 | 2000-2001   |       20 | Manchester United
(7 rows)
```

Order of Evaluation

In general, the order of evaluation of conditions with `AND` and `OR` is non-deterministic. Yellowbrick does not support a "short circuit" order of evaluation for conditions, except in CASE WHEN expressions.

For example, the following `WHERE` clause condition:

```
where col1 !=0 and col1 / col2 > 5
```

is not evaluated from left to right, which would remove the need to evaluate the right side of the expression when `col1=0`. Yellowbrick evaluates the right side of this expression first and, depending on the data in the table, may return a divide-by-zero error:

```
premdb=# create table t1(c1 int, c2 int);
CREATE TABLE
premdb=# insert into t1 values (0,0), (1,0), (0,1), (1,1);
INSERT 0 4
premdb=# select * from t1;
 c1 | c2
----+----
  0 |  0
  1 |  0
  0 |  1
  1 |  1
(4 rows)

premdb=# select * from t1 where c1 !=0 and c1/c2>5;
ERROR:  division by zero - division by zero
```

SQL User Defined Function (UDF)

Yellowbrick Documentation > Reference > SQL Reference > SQL Functions and Operators > SQL User Defined Function (UDF)

Platforms: All platforms

Parent topic: SQL Functions and Operators

In this section:

- SQL UDF Create Function
- SQL UDF Function Overloading
- Troubleshooting: Common Errors and Solutions

A UDF evaluates an arbitrary SQL expression and returns the result(s) of the expression. Yellowbrick supports creating SQL user defined functions from version 5.4. Users can create custom UDF using SQL statements. SQL UDFs are meant to provide a handy mechanism to roll up expressions into a function-name and are expanded inline by the front-end database during parsing or planning.

Things to know before creating SQL UDFs:

1. SQL UDFs cannot be created in the `pg_catalog`, `sys`, and `yb_stat` schema.
2. Yellowbrick Data Warehouse SQL functions can only be "scalar functions". They operate on a single row and return exactly one value that is a base database data type. They cannot return a result set or record.
3. CREATE FUNCTION does not support `TRANSFORM`, `WINDOW`, `LEAKPROOF`, `SET`, `COST`, and `ROWS` clauses.
4. Function definition requirements:

- Single table-free select statement: The `SELECT` statement contained within the function must not include a `FROM` clause.
- Single Target Expression: The `SELECT` statement within the function must return exactly one column from the current row only.
- Non-Recursive: SQL functions cannot call themselves.
- Non-null and Non-empty: The function must perform an operation and cannot be a no operation.

5. SQL UDFs cannot be referenced in a cross-database query.

   Example:

```sql
SELECT one, one AS two, yellowbrick.public.add_numbers( one, two ) FROM sys.const;

ERROR: cross-database references are not allowed: yellowbrick.public.add_numbers
```

6. CREATE FUNCTION VOLATILITY constraints:

- If the function is defined to be Immutable, then the function definition can only contain immutable functions.
- If the function is defined to be Stable, then the function definition cannot contain volatile functions.
- If the function is defined to be Strict, then the function can only contain strict functions and cannot have unused parameters.
- If the function uses volatile built-in functions it must be volatile (which is the default).

SQL UDF Create Function

In this section:

`CREATE FUNCTION` defines a new function. `CREATE OR REPLACE FUNCTION` will either create a new function, or replace an existing definition. The user that creates the function becomes the owner of the function.

Enabling SQL UDF

Prerequisites for enabling SQL UDF feature are:

- The configuration parameter `ybd_allow_udf_creation` must be set.
- The current user should have `EXECUTE` privilege on the function. To know how to grant permission using `GRANT EXECUTE` statement, Click Here.

Syntax

```sql
CREATE [ OR REPLACE ] FUNCTION
    name ( [ [ argmode ] [ argname ] argtype [ { DEFAULT | = } default_expr ] [, ...] ] )
    [ RETURNS rettype ]
LANGUAGE SQL
{
    | IMMUTABLE | STABLE | VOLATILE
    | CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT
    | AS 'definition'
}
```

name

The name of the function to create.

argmode

The mode of an argument: `IN`, `OUT`, `INOUT`, or `VARIADIC`. If omitted, the default is `IN`. Only an `OUT` argument can follow a `VARIADIC` one.

argname

The name of an argument. The name of an output argument is significant because it defines the column name in the result row type. If the output argument's name is omitted, the system will assign a default column name.

argtype

The data type of the argument. The argument types can be base data types.

default_expr

An expression to be used as default value if the parameter is not specified. The expression has to be coercible to the argument type of the parameter. Only input (including `INOUT`) parameters can have a default value. All input parameters following a parameter with a default value must have default values as well.

rettype

The return data type (optionally with a schema). It can be an integer or reference a table column's type. If the function doesn't return a value, use `void` as the return type.

When there are `OUT` or `INOUT` parameters, the `RETURNS` clause can be omitted. If present, it must agree with the result type implied by the output parameters.

`IMMUTABLE` `STABLE` `VOLATILE` These attributes inform the query optimizer about the behavior of the function. At most one choice can be specified. If none of these appear, `VOLATILE` is the default assumption.

`IMMUTABLE` indicates that the function cannot modify the database and always returns the same result when given the same argument values; that is, it does not do database lookups or otherwise use information not directly present in its argument list. If this option is given, any call of the function with all-constant arguments can be immediately replaced with the function value.

`STABLE` indicates that the function does not change the database. It will always give the same result for the same inputs during a single table scan. However, the result could vary between different SQL statements. This is the appropriate selection for functions whose results depend on database lookups, parameter variables (such as the current time zone), etc. Also note that the `current_timestamp` family of functions qualify as stable, since their values do not change within a transaction.

`VOLATILE` indicates that the function value can change even within a single table scan, so no optimizations can be made. Relatively few database functions are volatile in this sense; some examples are `random()`, `timeofday()`.

`CALLED ON NULL INPUT` `RETURNS NULL ON NULL INPUT` `STRICT` `CALLED ON NULL INPUT` (the default) indicates that the function will be called normally when some of its arguments are null. It is then the function author's responsibility to check for null values if necessary and respond appropriately.

`RETURNS NULL ON NULL INPUT` or `STRICT` indicates that the function always returns null whenever any of its arguments are null. If this parameter is specified, the function is not executed when there are null arguments; instead a null result is assumed automatically.

Definition

A string constant defining the function. It can be an internal function name or an SQL command.

It is often helpful to use dollar quoting (`$$`) to write the function definition string, rather than the normal single quotation mark syntax. Without dollar quoting, any single quotes or backslashes in the function definition must be escaped by doubling them.

For SQL UDF examples, Click Here.

Function Overloading

Function overloading is a feature in Yellowbrick that allows you to define multiple functions with the same name but different parameters. For more details, refer SQL UDF Function Overloading.

GRANT EXECUTE statement

Platforms: All platforms

Parent topic: SQL UDF Create Function

In Yellowbrick, you can grant execution rights on functions to a user (or role) using the `GRANT EXECUTE` statement. This is useful when you want to give a user the ability to run a function without granting them broader access to the database.

Syntax

- For granting the execute permission on a function:

```sql
GRANT EXECUTE
ON FUNCTION your_function_name(parameter_type1, parameter_type2, ...)
TO user_name;
```

- For all functions in a schema:

```sql
GRANT EXECUTE
ON ALL FUNCTIONS
IN SCHEMA schema_name TO user_name;
```

Examples

Grant Execute on a Specific Function

```sql
--Grant execute permission on a function named `add_numbers(INT4, INT4)` to a user named `john_smith`
GRANT EXECUTE
ON FUNCTION add_numbers(INT4, INT4)
TO john_smith;
```

Grant Execute on All Functions in a Schema

```sql
-- Grant a user execute permissions on all functions within a specific schema, such as `public`
GRANT EXECUTE ON ALL FUNCTIONS IN SCHEMA public TO john_smith;
```

After executing the `GRANT EXECUTE` command, `john_smith` will have the necessary permissions to execute the specified function(s) in your Yellowbrick database.

SQL UDF Examples

Examples for SQL UDFs using various input parameters are given below:

Use only input parameters (argmode `IN`)

The example given below, adds numbers and returns an integer.

```sql
CREATE SCHEMA if not exists myschema;

CREATE OR REPLACE FUNCTION myschema.add_numbers( INT4, INT4 )
RETURNS INT4
AS
   $$ SELECT $1 + $2; $$
LANGUAGE SQL
IMMUTABLE
RETURNS NULL ON NULL INPUT
;

SELECT myschema.add_numbers( 2, 5 );
 add_numbers
-------------
 7
```

Use output parameter (argmode `OUT`)

The example given below, adds numbers and returns an integer.

```sql
CREATE SCHEMA if not exists myschema;

CREATE OR REPLACE FUNCTION add_numbers( IN INT4, IN INT4, OUT INT4 )
AS
   'SELECT $1 + $2;'
LANGUAGE SQL
IMMUTABLE
RETURNS NULL ON NULL INPUT
;

SELECT myschema.add_numbers( 2, 5 ) as sum;
 sum
-----
   7
(1 row)
```

Use `VARIADIC` argmode

The example given below, retrieves the upper bound of its dimensions or the highest index of the array in integer data type.

```sql
CREATE SCHEMA if not exists myschema;

create function dfunc(a variadic int[])
returns int as
$$ select array_upper($1, 1) $$ language sql;

-- select dfunc() fails because no default value
select dfunc();

ERROR:  function dfunc() does not exist
HINT:  No function matches the given name and argument types. You might need to add explicit type casts.
```

The correct way to use variadic argmode is given below:

```
create or replace function dfunc(a variadic int[] default array[]::int[])
returns int as
$$ select array_upper($1, 1) $$ language sql;

-- Now it does not give error
select dfunc();

 dfunc
-------
<<[NULL]>>
(1 row)

select dfunc(10);
 dfunc
-------
     1
(1 row)

select dfunc(10,20);
 dfunc
-------
     2
(1 row)
```

Use `INOUT` argmode

The example given below, increments input value by the increment value.

```sql
CREATE SCHEMA if not exists myschema;

CREATE OR REPLACE FUNCTION increment_value(input_val INOUT INTEGER, increment_val INTEGER)
RETURNS INTEGER AS $$
    select input_val + increment_val;
$$ LANGUAGE sql;

select increment_value(2, 3);
 increment_value
-----------------
               5
(1 row)
```

Use parameters `IMMUTABLE`, `STABLE` and `VOLATILE`

The example given below, checks functions with different volatility. And checks they have the right value set in the system catalog.

```sql
CREATE SCHEMA if not exists myschema;

-- CREATE FUNCTIONS WITH DIFFERENT VOLATILITY
CREATE FUNCTION functest_B_1(int) RETURNS bool LANGUAGE 'sql'
       AS 'SELECT $1 > 0';

CREATE FUNCTION functest_B_2(int) RETURNS bool LANGUAGE 'sql'
       IMMUTABLE AS 'SELECT $1 > 0';

CREATE FUNCTION functest_B_3(int) RETURNS bool LANGUAGE 'sql'
       STABLE AS 'SELECT $1 = 0';

CREATE FUNCTION functest_B_4(int) RETURNS bool LANGUAGE 'sql'
       VOLATILE AS 'SELECT $1 < 0';

SELECT proname, provolatile FROM pg_proc
       WHERE oid in ('functest_B_1'::regproc,
                     'functest_B_2'::regproc,
                     'functest_B_3'::regproc,
                     'functest_B_4'::regproc) ORDER BY proname;

   proname    | provolatile
--------------+-------------
 functest_b_1 | v
 functest_b_2 | i
 functest_b_3 | s
 functest_b_4 | v
(4 rows)
```

Use parameter `RETURNS NULL ON NULL INPUT` / `STRICT`

Use parameter `RETURNS NULL ON NULL INPUT`

In this example, if the input parameter `IN` is null, function example_function() returns null.

```sql
CREATE SCHEMA if not exists myschema;

CREATE OR REPLACE FUNCTION example_function(input_value INTEGER)
RETURNS INTEGER
LANGUAGE sql
RETURNS NULL ON NULL INPUT
AS $$
SELECT CASE
         WHEN input_value IS NULL THEN 100
         ELSE input_value * 3
       END;
$$;

-- Example call with null input
select example_function(NULL);
 example_function
------------------
 <<Null>>
 (1 row)
```

Use parameter `STRICT`

In this example, if the input parameter `IN` is null, function example_function() returns null.

```sql
CREATE SCHEMA if not exists myschema;
```

```
CREATE OR REPLACE FUNCTION example_function(input_value INTEGER)
RETURNS INTEGER
LANGUAGE sql
STRICT
AS $$
SELECT CASE
         WHEN input_value IS NULL THEN 100
         ELSE input_value * 3
       END;
$$;

-- Example call with null input
select example_function(NULL);
 example_function
------------------
 <<NULL>>
 (1 row)
```

Use parameter `CALLED ON NULL INPUT`

`CALLED ON NULL INPUT` means the function f() will execute normally even if any argument is null, as opposed to automatically returning null without running the function body, which is what happens when the function is `STRICT`.

sql

```
CREATE OR REPLACE FUNCTION f(input_value INTEGER)
RETURNS Boolean
LANGUAGE sql
called on null input
AS 'SELECT $1 IS NULL';

select f(3); -- returns false
 f
---
 f

select f(NULL); -- returns true because we ran the function body
 f
---
 t
```

SQL UDF Function Overloading

Function overloading allows you to create multiple functions with the same name. The input arguments to the functions can vary in number and can be of different data types. `OUT` parameters are excluded from the function signature during overloading. Therefore, functions cannot be overloaded based solely on differing return types when they have the same number and types of arguments. The following declaration is valid:

```
int calculate_value(int, int)
int calculate_value(varchar)
```

However the following declaration returns an error:

```
int calculate_value(int, int)
varchar calculate_value(int, int)
-- ERROR: function "calculate_value" already exists with same argument types
```

The following declarations conflict:

```
CREATE FUNCTION calculate_price(int) ...
CREATE FUNCTION calculate_price(int, out text) ...
```

Functions with different argument type lists will not be considered to conflict at creation time. However, if defaults are provided, they might conflict in use. For example:

```
CREATE FUNCTION calculate_price(int) ...
CREATE FUNCTION calculate_price(int, int default 42) ...
```

A call to calculate_price(10) will fail due to ambiguity about which function should be called.

Function Overloading Criteria

Yellowbrick resolves which function to execute based on the following criteria:

1. **Number of Parameters**: Functions with the same name can differ by the number of input parameters.
2. **Data Types of Parameters**: Functions can have the same name but operate on different data types (For example, integers, strings, arrays).
3. **Parameter Modes**: `IN` parameters can distinguish overloaded functions.

SQL UDF Function Overloading Examples

Overloading by Number of Parameters

```sql
-- Function with one parameter
CREATE OR REPLACE FUNCTION calculate_area(radius NUMERIC) RETURNS NUMERIC AS
$$ SELECT 3.14159 * radius * radius $$ LANGUAGE SQL;
```

```
-- Overloaded function with two parameters
CREATE OR REPLACE FUNCTION calculate_area(length NUMERIC, width NUMERIC) RETURNS NUMERIC AS
$$ SELECT length * width $$ LANGUAGE SQL;
```

In this example, `calculate_area()` is overloaded to handle calculations for both a circle (one parameter) and a rectangle (two parameters).

Overloading by Data Type

```sql
-- Function takes an integer
CREATE OR REPLACE FUNCTION check_value(val INT) RETURNS NUMERIC AS
$$ select val::numeric(16, 1); $$ LANGUAGE SQL;

-- Function takes a float
CREATE OR REPLACE FUNCTION check_value(val FLOAT) RETURNS NUMERIC AS
$$ select val::numeric(16, 1); $$ LANGUAGE SQL;

-- Function takes a numeric
CREATE OR REPLACE FUNCTION check_value(val NUMERIC) RETURNS NUMERIC AS
$$ select val::numeric(16, 1); $$ LANGUAGE SQL;
```

In this example, `check_value()` is overloaded to accept an integer, float or a numeric value.

```sql
select check_value(4::int);
 check_value
-------------
 4.0

select check_value(4.3::float);
 check_value
-------------
 4.3

select check_value(4.3::numeric(2, 1));
 check_value
-------------
 4.3
```

Troubleshooting: SQL UDF Common Errors and Solutions

1. ERROR: User-defined functions are not supported This error is returned if the `ybd_allow_udf_creation` configuration parameter is not set.

```sql
CREATE SCHEMA if not exists myschema;

CREATE OR REPLACE FUNCTION myschema.add_numbers( INT4, INT4 )
RETURNS INT4
AS
   $$ SELECT $1 + $2; $$
LANGUAGE SQL
IMMUTABLE
RETURNS NULL ON NULL INPUT
;
```

2. ERROR: SQL UDF *name* is too complex This error may occur if the UDF is too complex for the planner to process. Specifically, the planner cannot inline the UDF. A scalar function call will be inlined if **all** of the following conditions are met:

   - the function is `LANGUAGE SQL`.
   - the function is not `RETURNS SETOF` (or `RETURNS TABLE`).
   - the function is not `RETURNS RECORD`.
   - the function has no `SET` clauses in its definition.
   - the function body consists of a single, simple `SELECT expression`.
   - the body contains no aggregate or window function calls, no subqueries, no CTEs, no FROM clause or references to any table or table-like object, none of GROUP BY, HAVING, ORDER BY, DISTINCT, LIMIT, OFFSET, UNION, INTERSECT, EXCEPT.
   - the body query must return exactly one column.
   - the type of the body expression must match the declared return type of the function.
   - the expression must not return multiple rows. For example, from calling set-returning functions such as `unnest()` or `generate_series()`.
   - if the function is declared `IMMUTABLE`, then the expression must not call any non-immutable function or operator.
   - if the function is declared `STABLE`, then the expression must not call any volatile function or operator.
   - if the function is declared `STRICT`, then the planner must be able to prove that the body expression necessarily returns `NULL` if any parameter is null. At present, this condition is only satisfied if: every parameter is referenced at least once, and all functions, operators and other constructs used in the body are themselves `STRICT`.
   - if an actual argument to the function call is a volatile expression, then it must not be referenced in the body more than once.
   - if an actual argument is an "expensive" expression, defined as costing more than 10 operator costs or containing any subquery, then it must not be referenced in the body more than once.

3. ERROR: cross-database references are not allowed: <*database*>.<*schema*>.<*name*> You cannot reference a UDF in a foreign database. Doing so will result in this error.

String Functions

Platforms: All platforms

Parent topic: SQL Functions and Operators

In this section:

This section describes functions that operate on character strings.

ENCRYPT_KS

In this section:

Given an input character string, return an encrypted value using a key. See also DECRYPT_KS.

```
ENCRYPT_KS(input_expression, key [, algorithm [, ivec]])
```

This function returns a Base64-encoded `VARCHAR` value that is slightly larger than the input expression. The formula depends on the algorithm specified:

- If algorithm `1`, `2`, or `3` is explicitly specified as a constant in the function:

```
VARCHAR(CEIL((input * 8) / 6))
```

- Otherwise (algorithm = constant > `3`, not specified, or not a constant, such as the result of an expression):

```
VARCHAR(CEIL((512 + input) / 3) * 4)
```

Parameters

input_expression

A `VARCHAR` expression, such as a character column in a table, a substring of a column, or a concatenation of character strings from multiple columns. This expression represents the sensitive data that you want to protect and encrypt on output.

key

The name of a key created with the CREATE KEY command.

algorithm

The specific encryption algorithm that you want to use (optional). Valid entries are `1` through `9`. The default is `1`. All of these algorithms use Output Feedback Mode (OFB). For more details, see Encryption and Decryption Algorithms.

The algorithm that you select determines the required size of the key value: 128, 192, or 256 bits. The size of the initialization vector (`ivec`) for all three algorithms is 128 bits.

When the input key or `ivec` is too short, the input data is expanded to produce the required length by appending the input provided. When the input key or the `ivec` is too long, the extra input is folded into the required input, starting from the beginning, using `XOR` logic.

ivec

An initialization vector for the algorithm (optional). You must specify a hexadecimal string or an expression that evaluates to a hexadecimal string. If you specify a vector, you must also specify an algorithm. Changing the vector but using the same `key` has the effect of re-scrambling the output for a given input expression. See Encrypting

Sensitive Data. See also the description of the algorithm parameter for information about the required length of the `ivec` value.

ENCRYPT_KS Examples

Encrypt the `nickname` column from the `team` table using key `yb100key` and the default algorithm:

```
premdb=# select nickname, encrypt_ks(nickname, yb100key) from team order by teamid limit 5;
 nickname | encrypt_ks
----------+-------------
 Gunners  | nmFhVu67X3
 Villains | W0khTes1yFE
 Tykes    | Y0VgKW7
 Blues    | qK{jKW7
 Rovers   | aOEjKaN7
(5 rows)
```

Run the same query but change the algorithm:

```
premdb=# select nickname, encrypt_ks(nickname, yb100key, 3) from team order by teamid limit 5;
 nickname | encrypt_ks
----------+-------------
 Gunners  | 5pOQhIRyw1
 Villains | K3vQf2hwd5E
 Tykes    | M38RWAA
 Blues    | 0NfSWAA
 Rovers   | GRPSWEAy
(5 rows)
```

Encryption and Decryption Algorithms

Platforms: All platforms

Parent topic: ENCRYPT_KS

This section explains the specific algorithms that are used when values 1 through 9 are specified for the `ENCRYPT_KS` and `DECRYPT_KS` functions. Values 4 through 9 only are supported for `ENCRYPTED WITH` column constraints in `CREATE TABLE` statements, using the same algorithms.

Function Parameter	Algorithm	Output Length (Encryption)	Output Length (Decryption)
1	Initialize AES with a 128-bit key and encrypt (or decrypt) using Output Feedback Mode (OFB).	`floor((n1 * 8 + 5) / 6)`	`floor((n1 * 6) / 8))`
2	Initialize AES with a 192-bit key and encrypt (or decrypt) using OFB.	`floor((n1 * 8 + 5) / 6)`	`floor((n1 * 6) / 8))`
3	Initialize AES with a 256-bit key and encrypt (or decrypt) using OFB.	`floor((n1 * 8 + 5) / 6)`	`floor((n1 * 6) / 8))`
4	Use 128-bit key deterministic encryption (or deterministic decryption).	`ceil((17 + n1) / 3) * 4`	`(floor(n1 / 4) * 3) - 17`
5	Use 192-bit key deterministic encryption (or deterministic decryption).	`ceil((25 + n1) / 3) * 4`	`(floor(n1 / 4) * 3) - 25`
6	Use 256-bit key deterministic encryption (or deterministic decryption).	`ceil((33 + n1) / 3) * 4`	`(floor(n1 / 4) * 3) - 33`
7	Use 128-bit key randomized encryption (or randomized decryption).	`ceil((17 + n1) / 3) * 4`	`(floor(n1 / 4) * 3) - 17`
8	Use 192-bit key randomized encryption (or randomized decryption).	`ceil((25 + n1) / 3) * 4`	`(floor(n1 / 4) * 3) - 25`
9	Use 256-bit key randomized encryption (or randomized decryption).	`ceil((33 + n1) / 3) * 4`	`(floor(n1 / 4) * 3) - 33`

Randomized Encryption

The algorithm for randomized (non-deterministic) encryption follows these steps:

1. Accept (binary) input string `plaintext` with length `plaintext_length` bytes.
2. Accept (binary) input key with length `key_length` bytes.
3. Accept (integer) input `aes_keybits` (must be 128, 192, or 256).
4. Reuse/fold key to `aes_keybits`. The actual size of the key required depends on the algorithm selected. The actual size of the initialization vector for modes 4 to 9 is 128 bits.

- When the input `key` or `ivec` is shorter than expected, reuse the input until the required size is reached.
- When the input `key` or `ivec` is longer than expected, *fold* the extra input with the required input via `XOR`.

5. Generate a cryptographically random salt with `aes_keybits` (likely generated via Fortuna).
6. `ciphertext = ofb_aes(plaintext, key = (key XOR salt), iv = (salt >> (aes_keybits - 128))`
7. `ciphertext_length = plaintext_length`
8. `outputtext = version (1 byte) + salt (aes_keybits/8) + ciphertext (ciphertext_length bytes)`
9. `outputtext_length = 1 + (aes_keybits/8) + ciphertext_length`

Note:

- This encryption algorithm is specifically designed to be secure against any kind of *chosen plain text attacks* (including dictionary-based attacks for low-cardinality data sets).
- This algorithm is also resistant to *length-extension* attacks; that is, an attacker with *encrypt* access cannot pad a chosen (binary) string to an already encrypted value.
- Use an HMAC on the `outputtext` separately if you need to protect against *random* padding.
- `ciphertext_length` mirrors `plaintext_length`, and therefore leaks information about the input. For example, `ciphertext` for `'m'` will be indistinguishable (in length and randomness) from `ciphertext` for `'f'`; however, `ciphertext_length` for `'male'` will be different from `ciphertext_length` for `'female'`. To guard against this leakage, pad all input to the same length.

Randomized Decryption

The algorithm for randomized (non-deterministic) decryption follows these steps:

1. Accept (binary) input string `outputtext` with length `outputtext_length` bytes.
2. Accept (binary) input key with length `key_length` bytes.
3. Accept (integer) input `aes_keybits` (must 128, 192, or 256).
4. Reuse/fold key to `aes_keybits`. The actual size of the key required depends on the algorithm selected. The actual size of the initialization vector for modes 4 to 9 is 128 bits.

- When the input `key` or `ivec` is shorter than expected, reuse the input until the required size is reached.
- When the input `key` or `ivec` is longer than expected, *fold* the extra input with the required input via `XOR`.

5. `version = outputtext[0:0]`
6. `salt = outputtext[1:(aes_keybits/8)]`
7. `ciphertext = outputtext[(aes_keybits/8):(outputtext_length-1)]`
8. `plaintext = ofb_aes(ciphertext, key = (key XOR salt), iv = (salt >> (aes_keybits - 128))`
9. `plaintext_length = outputtext_length - (1 + (aes_keybits/8))`

Note:

- This decryption algorithm is not susceptible to *padding-oracle* attacks.
- OFB enables decryption in a stream-cipher mode instead of requiring PKCS-like padding.
- An attacker who has *decrypt* access but not *encrypt* access may be able to encrypt chosen plain texts. This may be a problem if a user erroneously relies on an encryption system for message-authentication; use a proper HMAC if needed to prevent message forgery.

Deterministic Encryption

The algorithm is the same as in randomized encryption, except for the choice of `salt` in step 5:

```
salt = hmac_sha256(key, plaintext) >> (256 - aes_keybits);
```

Note:

- Identical `plaintext` strings encrypt to identical `outputtext` strings when you use the same `key`; this method enables equality comparison.

- The same `plaintext` value encrypted with two different keys generates two different `outputtext` values.

Deterministic Decryption

The algorithm for deterministic decryption works exactly the same as the algorithm for randomized decryption.

Note:

- *padding-oracle* attacks are not possible.
- Given that deterministic decryption uses an `hmac_sha256` instead of `sha256`, it is possible to detect and prevent *random* length extensions. However, for compatibility with randomized encryption, Yellowbrick decryption instead returns an undefined result.

A Warning about "Symmetric" Algorithms

Encryption algorithms `1`, `2`, and `3` are considered "symmetric." Users need to be careful about choosing these algorithms if the default initialization vector (IV) is being used, if the IV is well-known, or if the IV may have been leaked. In these scenarios, roles who have `ENCRYPT` but not `DECRYPT` access may be able to decipher strings by reverse engineering the bit-stream that was used to protect those strings.

For example, assume that encryption key `k0` and role `r0` exist in your system, that `r0` is neither a superuser nor the owner of `k0`, and that `r0` has been granted `ENCRYPT` but not `DECRYPT` access on `k0`.

In turn, `r0` can run a query where:

```
cipher-text = encrypt(plain-text, k0, {1 | 2 | 3}, iv)
```

Although `r0` cannot run an equivalent `DECRYPT` query, the following property holds true for binary data:

```
plain-text = encrypt(encrypt(plain-text, k0, {1 | 2 | 3}, iv), k0, {1 | 2 | 3}, iv)
```

Therefore, if `r0` happens to know the `iv` in use, `r0` can recover the plain text from the ciphertext by reverse-engineering the bit-stream used to protect the plain text string.

ASCII

Given a character string, return the ASCII code value, an integer, for the first character in the string.

```
ASCII (string)
```

For UTF-8 characters, the function returns an integer that corresponds to the Unicode code point.

To get the ASCII code value or Unicode code point of an integer, use the CHR function.

Examples

Return the code value for the exclamation mark character:

```
premdb=# select ascii('!') from sys.const;
 ascii
-------
    33
(1 row)
```

Return code values for a set of character strings in a table:

```
premdb=# select *, ascii(nickname) from team where city='London' order by ascii;
 teamid | htid | atid |        name         |  nickname  |  city  |     stadium      | capacity | avg_att | ascii
--------+------+------+---------------------+------------+--------+------------------+----------+---------+-------
     12 |   13 |   62 | Charlton Athletic   | Addicks    | London | The Valley       |    27111 |   0.000 |    65
     18 |   19 |   68 | Fulham              | Cottagers  | London | Craven Cottage   |    25700 |   0.000 |    67
     46 |   47 |   96 | Wimbledon           | Dons       | London | Selhurst Park    |    26255 |   0.000 |    68
     15 |   16 |   65 | Crystal Palace      | Eagles     | London | Selhurst Park    |    26255 |  24.636 |    69
      1 |    2 |   51 | Arsenal             | Gunners    | London | Emirates Stadium |    60260 |  59.944 |    71
     44 |   45 |   94 | West Ham United     | Hammers    | London | Upton Park       |    35016 |  34.910 |    72
     32 |   33 |   82 | Queens Park Rangers | Hoops      | London | Loftus Road      |    18439 |   0.000 |    72
     13 |   14 |   63 | Chelsea             | Pensioners | London | Stamford Bridge  |    41663 |  41.500 |    80
     41 |   42 |   91 | Tottenham Hotspur   | Spurs      | London | White Hart Lane  |    36284 |  35.776 |    83
(9 rows)
```

BIT_LENGTH

Return the length in bits of a character string.

```
BIT_LENGTH(string)
```

For example, what is the maximum length, in bits, of the values in the `nickname` column:

```
premdb=# SELECT MAX(BIT_LENGTH(nickname)) FROM team;
 max
-----
  80
(1 row)
```

BTRIM

Remove leading or trailing characters based on a specified list. By default, remove leading and trailing spaces.

```
BTRIM(string [ ,string ])
```

```
premdb=# SELECT BTRIM(' Bob ');
 btrim
-------
 Bob
(1 row)

premdb=# SELECT BTRIM(' Bob ','Bb');
 btrim
-------
  Bob
(1 row)

premdb=# SELECT BTRIM(' Bob ','Bb ');
 btrim
-------
 o
(1 row)
```

This example removes the trailing lowercase `s` from the nickname column:

```
premdb=# SELECT nickname, BTRIM(nickname,'s') FROM team;
  nickname  |   btrim
------------+-----------
 Gunners    | Gunner
 Villains   | Villain
 Tykes      | Tyke
 Blues      | Blue
 Rovers     | Rover
 Seasiders  | Seasider
 Trotters   | Trotter
 Cherries   | Cherrie
...
```

CHAR_LENGTH

Return the number of characters in a string. (The LENGTH function also returns the number of characters.)

```
CHAR_LENGTH | CHARACTER_LENGTH (string)
```

For example:

```
premdb=# SELECT LENGTH(nickname), CHAR_LENGTH(nickname) FROM team WHERE city='London';
 length | char_length
--------+-------------
      7 |           7
      7 |           7
     10 |          10
      6 |           6
      9 |           9
      5 |           5
      5 |           5
      7 |           7
      4 |           4
(9 rows)
```

```
premdb=# SELECT MAX(CHARACTER_LENGTH(name)) FROM team;
 max
-----
  23
(1 row)
```

CHR

Return the character for a given code point (Unicode code points for UTF8).

```
CHR(int)
```

For example:

```
premdb=# select chr(90);
 chr
-----
 Z
(1 row)

premdb=# select seasonid, chr(seasonid)
from season order by seasonid;

 seasonid |   chr
----------+----------
        1 | \x01
        2 | \x02
        3 | \x03
        4 | \x04
        5 | \x05
...
```

CONCAT_WS

Concatenate a series of strings, using the first argument as a separator between the strings. This function also accepts values that are not character strings, such as integers and timestamps.

`NULL` arguments are ignored.

```
CONCAT_WS(separator, string [, string [ , … ] ])
```

For example, combine two character columns and separate them with a hyphen (-):

```
premdb=# SELECT CONCAT_WS('-',nickname,city) FROM team LIMIT 5;
     concat_ws
---------------------
 Gunners-London
 Villains-Birmingham
 Tykes-Barnsley
 Blues-Birmingham
 Rovers-Blackburn
(5 rows)
```

Combine two integer columns and separate them with a colon:

```
premdb=# select concat_ws(':',htid,atid) from team order by htid;
 concat_ws
-----------
 2:51
 3:52
 4:53
 5:54
 6:55
 7:56
...
```

CONCAT, ||

Concatenate a series of strings.

```
CONCAT(string [, string [ , … ] ])
```

You can also use the `||` operator to concatenate a series of strings. `CONCAT` and `||` also accept values that are not character strings, such as integers and timestamps.

The `CONCAT` function and the `||` operator have exactly the same behavior.

Trailing spaces in `CHAR` strings are ignored (trimmed) when they are concatenated. Trailing spaces in `VARCHAR` strings are preserved.

Examples

The following example concatenates four strings (two `VARCHAR` columns (`name` and `nickname`) and two literal values).

```
premdb=# SELECT CONCAT(name,', The ',nickname) AS names
FROM team LIMIT 5;
            names
-------------------------------
 Arsenal, The Gunners
 Aston Villa, The Villains
 Birmingham City, The Blues
 Blackpool, The Seasiders
 Bolton Wanderers, The Trotters
(5 rows)
```

The following example concatenates three character columns:

```
premdb=# SELECT name||nickname||city AS names FROM team LIMIT 5;
            names
-------------------------------
 ArsenalGunnersLondon
 Aston VillaVillainsBirmingham
 Birmingham CityBluesBirmingham
 BlackpoolSeasidersBlackpool
 Bolton WanderersTrottersBolton
(5 rows)
```

The following example concatenates two `SMALLINT` columns and a colon character:

```
premdb=# select concat(htid,':',atid) from team order by htid;
 concat
--------
 2:51
 3:52
 4:53
```

```
 5:54
 6:55
 7:56
...
```

Compare the results of concatenated strings with trailing spaces in `CHAR` and `VARCHAR` expressions:

```
premdb=> select 'abc '::char(5) || 'zzz '::char(5) as concat_char, length(concat_char) from sys.const;
 concat_char | length
-------------+--------
 abczzz      |      6
(1 row)

premdb=> select 'abc '::varchar(5) || 'zzz '::varchar(5) as concat_varchar, length(concat_varchar) from sys.const;
 concat_varchar | length
----------------+--------
 abc zzz        |      8
(1 row)
```

DECRYPT (deprecated)

Given an encrypted character string, return the decrypted value. See also ENCRYPT (deprecated).

```
DECRYPT(encrypted_string, key [, algorithm [, ivec]])
```

This function returns a `VARCHAR` value that is slightly smaller than the input expression. The formula depends on the algorithm specified:

- If algorithm `1`, `2`, or `3` is explicitly specified as a constant in the function:

```
VARCHAR(FLOOR((input * 6) / 8))
```

- Otherwise (algorithm = constant > `3`, not specified, or not a constant, such as the result of an expression):

```
VARCHAR(CEIL(input / 4) * 3)
```

Parameters

encrypted_string

An encrypted character string (Base64-encoded), as produced by the `ENCRYPT` function. This string represents the data that you want to decrypt on output.

key

A hexadecimal string or an expression that evaluates to a hexadecimal string. For example, you can specify a character string inside an MD5 function. This key is like a password, providing a level of security for your use of the function and a guarantee of consistent encryption and decryption. To produce consistent results when encrypting and decrypting the same input expression, users must provide the same key value. See the description of the `algorithm` parameter for information about the required length of the key. See also Encrypting Sensitive Data.

algorithm

The specific encryption algorithm that you want to use. Valid entries are `1`, `2`, and `3`. The default is `1`. All of these algorithms use Output Feedback Mode (OFB).

Function parameter	Algorithm
1	Initialize AES with a 128-bit key
2	Initialize AES with a 192-bit key
3	Initialize AES with a 256-bit key

The algorithm that you select determines the required size of the key value: 128, 192, or 256 bits. The size of the initialization vector (`ivec`) for all three algorithms is 128 bits.

When the input key or `ivec` is too short, the input data is expanded to produce the required length by appending the input provided. When the input key or the `ivec` is too long, the extra input is folded into the required input, starting from the beginning, using `XOR` logic.

ivec

An initialization vector for the algorithm (optional). You must specify a hexadecimal string or an expression that evaluates to a hexadecimal string. If you specify a vector, you must also specify an algorithm. See Encrypting Sensitive Data.

Examples

Decrypt the `nickname` column, which is stored encrypted in a table named `encrypted_team`. The first query shows the encrypted `nickname` values:

```
premdb=# select * from encrypted_team order by 1 limit 5;
 teamid | htid | atid |       name       |  nickname   |    city    |     stadium      | capacity
--------+------+------+------------------+-------------+------------+------------------+----------
      1 |    2 |   51 | Arsenal          | A8jstaVuQ0  | London     | Emirates Stadium |    60260
      2 |    3 |   52 | Aston Villa      | RuCsrql{7K2 | Birmingham | Villa Park       |    42785
      3 |    4 |   53 | Barnsley         | PuztyyE     | Barnsley   | Oakwell Stadium  |    23009
      4 |    5 |   54 | Birmingham City  | FiSmyyE     | Birmingham | St. Andrew's     |    30016
      5 |    6 |   55 | Blackburn Rovers | VWimyuEu    | Blackburn  | Ewood Park       |    31367
(5 rows)
```

The second query shows the decrypted `nickname` values. Note that the key provided here must be exactly the same as the key that was used when the column was encrypted. If you do not know this key, you will not be able to decrypt the values correctly.

```
premdb=# select name, decrypt(nickname, md5('We are such stuff as dreams are made on'))
from encrypted_team order by teamid limit 5;
      name       | decrypt
-----------------+----------
 Arsenal         | Gunners
 Aston Villa     | Villains
 Barnsley        | Tykes
 Birmingham City | Blues
 Blackburn Rovers | Rovers
(5 rows)
```

DECRYPT_KS

Given an encrypted character string, return the decrypted value using a key. See also ENCRYPT_KS.

```
DECRYPT_KS(encrypted_string, key [, algorithm [, ivec]]);
```

This function returns a `VARCHAR` value that is slightly smaller than the input expression. The formula depends on the algorithm specified:

- If algorithm `1`, `2`, or `3` is explicitly specified as a constant in the function:

```
VARCHAR(FLOOR((input * 6) / 8))
```

- Otherwise (algorithm = constant > `3`, not specified, or not a constant, such as the result of an expression):

```
VARCHAR(CEIL(input / 4) * 3)
```

Parameters

encrypted_string

An encrypted character string (Base64-encoded), as produced by the `ENCRYPT_KS` function. This string represents the data that you want to decrypt on output.

key

The name of a key created with the CREATE KEY command.

algorithm

The specific encryption algorithm that you want to use (optional). Valid entries are `1` through `9`. The default is `1`. All of these algorithms use Output Feedback Mode (OFB). For more details, see Encryption and Decryption Algorithms.

The algorithm that you select determines the required size of the key value: 128, 192, or 256 bits. The size of the initialization vector (`ivec`) for all three algorithms is 128 bits.

When the input key or `ivec` is too short, the input data is expanded to produce the required length by appending the input provided. When the input key or the `ivec` is too long, the extra input is folded into the required input, starting from the beginning, using `XOR` logic.

ivec

An initialization vector for the algorithm (optional). You must specify a hexadecimal string or an expression that evaluates to a hexadecimal string. If you specify a vector, you must also specify an algorithm. See Encrypting Sensitive Data.

Examples

The following subquery encrypts the `nickname` column, then the outer query decrypts it. The query verifies that the decrypted value matches the original value. Because similar `nickname` values, such as `Blues`, are the same, the `encrypted` column in the result proves that the same string is encrypted the same way.

```
premdb=# select nickname, encrypted,
decrypt_ks(encrypted,yb100key,3) as decrypted
from(select nickname, encrypt_ks(nickname,yb100key,3) as encrypted from team) t1
order by 1;
  nickname  |   encrypted    | decrypted
------------+----------------+-----------
 Addicks    | 3tvOiABww1     | Addicks
 Baggies    | 0Z9OYYhvw1     | Baggies
 Bantams    | 0ZPQn2hxw1     | Bantams
 Black Cats | 0NfPcgxeAD|NI3 | Black Cats
 Blades     | 0NfPXIBy       | Blades
 Blues      | 0NfSWAA        | Blues
 Blues      | 0NfSWAA        | Blues
...
```

The following example decrypts the values of an encrypted expression using key `yb100key`:

```
premdb=# select teamid, decrypt_ks(team_name,yb100key) as full_name, avg_att from encrypted_team_names limit 5;
 teamid |        full_name        | avg_att
--------+-------------------------+---------
      1 | Arsenal:Gunners         |  59.944
      2 | Aston Villa:Villains    |  33.690
      3 | Barnsley:Tykes          |   0.000
      4 | Birmingham City:Blues   |   0.000
      5 | Blackburn Rovers:Rovers |   0.000
(5 rows)
```

DLE_DST

Return the difference between two character strings in terms of the number of changes required to convert the first string into the second. The comparison of the strings is case-sensitive. This function uses the Damerau-Levenshtein edit distance algorithm to compute the "distance" between the strings. With this algorithm, a character transposition change such as `'ab'` to `'ba'` counts as *one* change, not *two*.

See also LE_DST.

Usage Notes

This function returns an integer.

Two configuration parameters affect the performance and memory usage of this function. See `max_levenshtein_distance` and `max_levenshtein_string_size` for more information.

Examples

The following example requires two changes, one for the case of the letter `C`, and one for the transposition of `kc`:

```
yellowbrick=# select dle_dst('Chekcing','checking') from sys.const;
 le_dst
--------
      2
(1 row)
```

Calculate the distance between London team names and their nicknames in the `team` table:

```
premdb=# select name, nickname, dle_dst(name,nickname) dledst from team where city='London' and name is not null order by 3 desc;
       name         |  nickname  | dledst
--------------------+------------+--------
 Queens Park Rangers | Hoops      |     18
 Charlton Athletic   | Addicks    |     16
 Tottenham Hotspur   | Spurs      |     15
 West Ham United     | Hammers    |     12
 Crystal Palace      | Eagles     |     12
 Wimbledon           | Dons       |      8
 Chelsea             | Pensioners |      8
 Fulham              | Cottagers  |      8
 Arsenal             | Gunners    |      6
(9 rows)
```

See also the examples for LE_DST.

ENCODE

Encode a hexadecimal string and return a base64 value.

```
ENCODE('hex_string','HEX','BASE64')
```

'hex_string'
Specify a valid hexadecimal string or an expression that evaluates to a hexadecimal string.

'HEX'
Specify `HEX` (or `hex`) as the input. No other input types are supported.

'BASE64'
Specify `BASE64` (or `base64`) as the output. No other output types are supported.

Examples

For example:

```
premdb=> select encode('f5e4d3c2b1a0','HEX','BASE64') from sys.const;
  encode
----------
 9eTTwrGg
(1 row)
```

In this example, the result of the `HMAC_KS` function produces the input to the `ENCODE` function:

```
premdb=> select encode(hmac_ks(nickname,playerkey,2),'hex','base64') from team order by teamid limit 3;
                 encode
--------------------------------------------
 so9Ir6BcSA219q3jbWx5Y2ugrXsM/YG39F3hOdp5J+g=
 99oaUrPWoGE6DuZUw6Ar2QDMFaCsUtatXk24TGi5I4A=
 W8Apz44L4MrcAm4Bz8iCCbunokywHuvtQAzSnTI9+z8=
(3 rows)
```

ENCRYPT (deprecated)

Platforms: All platforms

Parent topic: String Functions

Given an input character string, return an encrypted value. See also DECRYPT (deprecated).

```
ENCRYPT(input_expression, key [, algorithm [, ivec]])
```

This function returns a Base64-encoded `VARCHAR` value that is slightly larger than the input expression. The formula depends on the algorithm specified:

- If algorithm `1`, `2`, or `3` is explicitly specified as a constant in the function:

```
VARCHAR(FLOOR((input * 8 + 5) / 6))
```

- Otherwise (algorithm = constant > `3`, not specified, or not a constant, such as the result of an expression):

```
VARCHAR(CEIL((512 + input) / 3) * 4)
```

Parameters

input_expression

A `VARCHAR` expression, such as a character column in a table, a substring of a column, or a concatenation of character strings from multiple columns. This expression represents the sensitive data that you want to protect and encrypt on output.

key

A hexadecimal string or an expression that evaluates to a hexadecimal string. For example, you can specify a character string inside an MD5 function. This key is like a password, providing a level of security for your use of the function and a guarantee of consistent encryption. To produce consistent results when encrypting and decrypting the same input expression, users must provide the same key value. See the description of the `algorithm` parameter for information about the required length of the key. See also Encrypting Sensitive Data.

algorithm

The specific encryption algorithm that you want to use. Valid entries are `1`, `2`, and `3`. The default is `1`. All of these algorithms use Output Feedback Mode (OFB).

Function parameter	Algorithm
1	Initialize AES with a 128-bit key
2	Initialize AES with a 192-bit key
3	Initialize AES with a 256-bit key

The algorithm that you select determines the required size of the key value: 128, 192, or 256 bits. The size of the initialization vector (`ivec`) for all three algorithms is 128 bits.

When the input key or `ivec` is too short, the input data is expanded to produce the required length by appending the input provided. When the input key or the `ivec` is too long, the extra input is folded into the required input, starting from the beginning, using `XOR` logic.

ivec

An initialization vector for the algorithm (optional). You must specify a hexadecimal string or an expression that evaluates to a hexadecimal string. If you specify a vector, you must also specify an algorithm. Changing the vector but using the same `key` has the effect of re-scrambling the output for a given input expression. See Encrypting Sensitive Data. See the description of the algorithm parameter for information about the required length of the `ivec` value.

Examples

Encrypt the `nickname` column from the `team` table, using the default algorithm:

```
premdb=# select nickname, encrypt(nickname, md5('Testing 1-2-3'))
from team order by teamid limit 5;
 nickname |   encrypt
----------+-------------
 Gunners  | NchTADqJV2
 Villains | 6MAT8T4L2Y5
 Tykes    | 4MxS1L5
 Blues    | I2QR1L5
 Rovers   | 2EgR1HbJ
(5 rows)
```

Run the same query, using the same key, but change the algorithm:

```
premdb=# select nickname, encrypt(nickname, md5('Testing 1-2-3'), 3) from team order by teamid limit 5;
 nickname |   encrypt
----------+-------------
 Gunners  | Z14y6h|cS3
 Villains | onby4xFW1B6
 Tykes    | mnKzDpE
 Blues    | cbrwDpE
 Rovers   | sf5wDtkc
(5 rows)
```

Add an `ivec` value:

```
premdb=# select nickname, encrypt(nickname, md5('Testing 1-2-3'), 3, md5('IVEC Test')) from team order by teamid limit 5;
 nickname |   encrypt
----------+-------------
 Gunners  | l91kL6ChL0
 Villains | {vWkNMyj8K5
 Tykes    | yvHlUUD
 Blues    | gjmeUUD
 Rovers   | wX0eUQTh
(5 rows)
```

Encrypt the result of a `CONCAT` function. Change all of the parameters.

```
premdb=# select concat(name,', ',nickname), encrypt(concat(name,', ',nickname), md5('Testing concat'), 2, md5('IVEC concat'))
from team order by teamid limit 5;
         concat          |             encrypt
-------------------------+---------------------------------
 Arsenal, Gunners        | jgbZQ1oAIRgYY1HF9QoAc0
 Aston Villa, Villains   | jkLYG1YQeFkn9HmV7N1Cvy12vTgV
 Barnsley, Tykes         | kcqZHrZ9RFlX54ZACQYA
```

```
 Birmingham City, Blues   | k6qZIT29PB{o8L44EU38vDrAynBQB1
 Blackburn Rovers, Rovers | kI4dSL2ABZFn5S3FHQoAcyLI2PwUTXAU
(5 rows)
```

Encrypt the `nickname` column by using a key that is selected from another table:

```
premdb=> select encrypt(nickname,
(select key from userkeys where whoami=current_user))
from team order by teamid;
      encrypt
--------------------
 CyftgWky|C2f10
 V8zzguku5qYmwD8
 T8jpjKkwi0
 BuyvqKkwi0
 R0VtqKkviC2
...
```

HASH

Apply a hashing algorithm to a character string and return a hexadecimal string.

```
HASH(string [, algorithm ])
```

string

Specify a character string that is the input to the function.

algorithm

Optionally, specify one of the following algorithms. The default is `2` (`SHA-2`).

Algorithm	Mode	Output Length (Bits)	Output Type
0	MD5	128	VARCHAR(128)
1	SHA-1	160	
2	SHA-2	256	
3	SHA-2	512	

The return type of this function is `VARCHAR(128)` .

Examples

For example:

```
premdb=# select nickname, hash(nickname,2) hashname
from team
where htid <10
order by nickname;
 nickname  |                            hashname
-----------+-----------------------------------------------------------------
 Blues     | a87be8077f00caa60dc4522640674d48d898fc6ad747513b137c8754800b5532
 Cherries  | 4bb92af3df886cdd3dfb96f1341fdec9c8d1ead626fb00bfc9c0f652fe7170ea
 Gunners   | 3d2424adb81588b55e21cfd2c3edd704a57a0b97006644f6ed18dabcacd30118
 Rovers    | 8104dacaef7ff8a603b1df60dc3725f17fa822dfcdb9b54f65618c16fe728627
 Seasiders | e098d41f26ac73ac9c95589d00de0a243cac8254cbab6f8b7fe54cedbc497e70
 Trotters  | 3521b2a0a33a5bcc8fcc749c4a8ceb509a5eb0bc1959d4307cf546374cf7c5f2
 Tykes     | 42f5e399370e0f2fad34deb312a66fa5b0599cd28c382806b530ffcd5cbda58d
 Villains  | 72c005cb8fad1826419e3f98582d67386682368ad61ff11c1514df48cdff85fb
(8 rows)
```

```
premdb=# select hash('We are such stuff as dreams are made on') from sys.const;
                            hash
```

```
------------------------------------------------------------------
 ca8e75531ae20c229cb7b6c996c5d2d7a9906f18564f1c60679dc76dddc2d41c
(1 row)
```

HASH4

Apply the `xxHash64` (32-bit) hashing algorithm to a character string.

```
HASH4(string [, algorithm ])
```

string

Specify a character string that is the input to the function.

algorithm

Optionally, specify `0` for the algorithm (`xxHash64`, 32-bit). No other algorithm values are valid.

The return type of this function is `INTEGER` (`INT4`).

Examples

For example:

```
premdb=# select nickname, hash4(nickname,0) hashname
from team
where htid <10
order by nickname;
 nickname  |  hashname
-----------+-------------
 Blues     |  -387055223
 Cherries  |  1443820713
 Gunners   |  1253095579
 Rovers    |  1003253655
 Seasiders | -2087951145
 Trotters  |  1669174631
 Tykes     |   866713967
 Villains  | -1467766758
(8 rows)
```

```
premdb=# select hash4('We are such stuff as dreams are made on')
from sys.const;
   hash4
-----------
 917459684
(1 row)
```

HASH8

Apply the `xxHash64` (64-bit) hashing algorithm to a character string.

```
HASH8(input [, algorithm ])
```

string

Specify a character string that is the input to the function.

algorithm

Optionally, specify `0` for the algorithm (`xxHash64`, 64-bit). No other algorithm values are valid.

The return type of this function is `BIGINT` (`INT8`).

Examples

For example:

```
premdb=# select nickname, hash8(hash(nickname)) hashname from team where htid <10 order by nickname;
 nickname  |       hashname
-----------+---------------------
 Blues     | -7564071997630541332
 Cherries  | -3383144761345514982
 Gunners   | -5149427805081569408
 Rovers    | -7647635251682027314
 Seasiders |  8780416543413982452
 Trotters  | -5729782585505407849
 Tykes     | -5365449622118805153
 Villains  |  4746666626073332151
(8 rows)
```

```
premdb=# select hash8('We are such stuff as dreams are made on') from sys.const;
       hash8
--------------------
 6254556390689297124
(1 row)
```

HMAC

Use a secret defined by a hexadecimal string to return an HMAC (hash-based message authentication code).

```
HMAC(string, hex_string [, algorithm ])
```

string

Specify an input character string, or an expression that evaluates to a character string.

hex_string

Specify a hexadecimal string that provides a secret for running the function.

algorithm

Optionally, specify one of the following algorithms. The default is `2` (`SHA-2`).

Algorithm	Mode	Output Length (Bits)	Output Type
0	MD5	128	VARCHAR(128)
1	SHA-1	160	
2	SHA-2	256	
3	SHA-2	512	

The return type of this function is `VARCHAR(128)` .

Examples

For example, using `SHA-1` mode and two different "secret" values:

```
premdb=> select hmac(nickname,'f5e4d3c2b1a0',1) from team order by teamid limit 3;
                hmac
-----------------------------------------
 be4a4a1100473015d41a658ab01040e23f0cccbe
 56156fa4e1c30e890849323bef6ca23f272bbc3f
 068516ffb71a4c585cc16fce3e5afa418ac17016
(3 rows)

premdb=> select hmac(nickname,'d3c2b1a0',1) from team order by teamid limit 3;
                hmac
-----------------------------------------
 22957e439e9c79b6fa790f9665a44f5847a00cf1
 f40feb8474bb96832bf6469fd6df7832133a7cef
 f3ff8dcf59cfdc8b70607f45e2c1ba0e3213a500
(3 rows)
```

In this example, the same secrets are used with `SHA-2` mode (with a length of 256 bits):

```
premdb=> select hmac(nickname,'f5e4d3c2b1a0',2) from team order by teamid limit 3;
                              hmac
------------------------------------------------------------------
 b28f48afa05c480db5f6ade36d6c79636ba0ad7b0cfd81b7f45de139da7927e8
 f7da1a52b3d6a0613a0ee654c3a02bd900cc15a0ac52d6ad5e4db84c68b92380
 5bc029cf8e0be0cadc026e01cfc88209bba7a24cb01eebed400cd29d323dfb3f
(3 rows)

premdb=> select hmac(nickname,'d3c2b1a0',2) from team order by teamid limit 3;
                              hmac
------------------------------------------------------------------
 403ea5b09d9f29a145a2373806b41d8f3651f7e2ac0e6275cb01d7cb93247d15
 052f4b11b93de81b0707f1a1fa1c9d991a64c704c2bc5c0a6e068c5aab8b0c35
 7f06ac46b2c99d398a5624d7cece79a4517e87cc491bf837cc81929b942993e6
(3 rows)
```

This example uses `SHA-2` mode (with a length of 512 bits):

```
premdb=# select hmac(nickname,'d3c2b1a0',3) from team order by teamid limit 3;
                                                                  hmac
------------------------------------------------------------------------------------------------------------------------------------
 5a724fdd70367f56a2d290f6dbbd729782821846d7e4cfa7c99606b16446fc76b8c3012fe6bfc6ae43b0c65ca25193a5e61b184cd9b3a678de6b6d347571615d
 c23caf7fd7029b3a5d14ecfd38947b87a8a0dbc51bbfaa10c72dd5138f97b201aa171b425f24999a1ab046c83726cca67a4bfb93fa26ca17f681a71ac1d2afaa
 16a5c1e96d75b3f3d99df35530ba13877cd29accae35799f3c6a1b2779679857423085553bb41ff904b2b27ccfe56a9dc2e98fcf4519a9eef7129eb243a89eec
(3 rows)
```

This example uses `MD5` mode:

```
premdb=# select hmac(nickname,'d3c2b1a0',0) from team order by teamid limit 3;
             hmac
----------------------------------
 d84f1cf1380d1e2d44c330096e141ce6
 2c7e526250cf6a584bbcdd269b8bc620
 59a11ef3c6f4f86a8ae40c503653938c
(3 rows)
```

HMAC_KS

Use a secret defined by an encryption key to run the `HMAC` function.

```
HMAC_KS(string, key [, algorithm])
```

string

Specify a character string or an expression that evaluates to a character string.

key

Specify the name of an encryption key, as defined with the CREATE KEY command. The secret associated with this key will be used to run the function. Users of this function must be granted `HMAC` privilege on the key. See ON KEY.

algorithm

Optionally, specify mode `0`, `1`, `2`, or `3`. The default is `2`. See HMAC for mode definitions.

The return type of this function is `VARCHAR(128)`.

Examples

For example:

```
premdb=# grant hmac on key playerkey to bobr;
GRANT
premdb=# \c premdb bobr
You are now connected to database "premdb" as user "bobr".
premdb=> select hmac_ks(nickname,playerkey,2) from team order by teamid limit 3;
                            hmac_ks
------------------------------------------------------------------
 3a2a310e0fe61e9c2c9fad2dfde4943354a18a24c888571c9721b8f0f1b93bed
 1cc33a31a00f002fdcc598b181715acbf6979ba5059054f102cfb917a8bdd4ab
 3213bc9c7217952b5a87c5d4a6fa807b01ada59f4cf4cfa98351a6c3b66880fc
(3 rows)
```

Here is the same query, using the `MD5` mode, then the `SHA-2` mode (with a length of 512 bits):

```
premdb=> select hmac_ks(nickname,playerkey,0) from team order by teamid limit 3;
            hmac_ks
----------------------------------
 d9f47dbfb2733dd2c8c301ad12f34a4e
 61d8ac8a9868ef8c29385a9c8faf94df
 45e7977a441970afae90af4523497b8b
(3 rows)

premdb=> select hmac_ks(nickname,playerkey,3) from team order by teamid limit 3;
                                                              hmac_ks
----------------------------------------------------------------------------------------------------------------------------------
 c000eb5a52cbb8452ab36e84856a5d4fe9d29d12c841eed6cc19f4824a5a1f56c058e78b15ea2736b0cd87b825439a992d697ef07e4ddf74fb61df63402f6218
 d3e3b3e354b8b6eb3c927476f33c22c0aba497e34d972a4643bf0fec3eb012dc707178ff71c4bc0d87cddc7df6b6f6c07a30b32f92045a100553b0909f9888b4
```

```
 86617d840fd40ebd627790d24495cce28ec06851d3255e4517680ac89fab6284f93e02a8a376a2fc2d397eea2bd966b0ff1c52110603f36a9705235081d504c1
(3 rows)
```

INITCAP

Capitalize the first letter of each word in a character string.

A word in this context is any alphanumeric string. Non-alphanumeric characters, such as underscores, apostrophes, punctuation marks, and dollar signs, are all treated as word separators.

```
INITCAP(string)
```

For example:

```
premdb=# select initcap('hello my name is Bob') from sys.const;
      initcap
----------------------
 Hello My Name Is Bob
(1 row)
```

```
yellowbrick=# select initcap('hello_my_name_is_bob') from sys.const;
      initcap
----------------------
 Hello_My_Name_Is_Bob
(1 row)
```

INSTR

Return the numeric position of the specified substring within a character string. Optionally, specify a starting position and an occurrence number. For details, see STRPOS.

LE_DST

Return the difference between two character strings in terms of the number of changes required to convert the first string into the second. The comparison of the strings is case-sensitive. This function uses the Levenshtein edit distance algorithm to compute the "distance" between the strings. With this algorithm, a character transposition change such as `'ab'` to `'ba'` counts as *two* changes.

This function returns an integer.

Note: If one of the inputs is an empty string, the function simply returns the number of characters in the other argument, regardless of the current settings of these configuration parameters.

See also DLE_DST.

Configuration

Two configuration parameters affect the performance and memory usage of this function. See `max_levenshtein_distance` and `max_levenshtein_string_size` for more information.

Examples

The following example requires three changes, one for the case of the letter `C`, and two for the transposition of `kc`:

```
yellowbrick=# select le_dst('Chekcing','checking') from sys.const;
 le_dst
--------
      3
(1 row)
```

Calculate the distance between the London team names and their nicknames in the `team` table:

```
premdb=# select name, nickname, le_dst(name,nickname) ledst from team
where city='London' and name is not null order by 3 desc;
        name         |  nickname  | ledst
---------------------+------------+-------
 Queens Park Rangers | Hoops      |    18
 Charlton Athletic   | Addicks    |    16
 Tottenham Hotspur   | Spurs      |    15
 West Ham United     | Hammers    |    12
 Crystal Palace      | Eagles     |    12
 Wimbledon           | Dons       |     8
 Chelsea             | Pensioners |     8
 Fulham              | Cottagers  |     8
 Arsenal             | Gunners    |     7
(9 rows)
```

Calculate the distance between two specific strings with both `LE_DST` and `DLE_DST`. Note that the results are different for these particular strings because the functions count character transpositions differently.

```
premdb=# select le_dst('Arsenal','Gunners'),dle_dst('Arsenal','Gunners') from sys.const;
 le_dst | dle_dst
--------+---------
      7 |       6
(1 row)
```

The following example shows the behavior of this function when `max_levenshtein_string_size` is set to `10`. Because of the length of the specified strings, both queries return an expected error:

```
yellowbrick=# set max_levenshtein_string_size to 10;
SET
yellowbrick=# show max_levenshtein_string_size;
 max_levenshtein_string_size
-----------------------------
 10
(1 row)
yellowbrick=# create table abc(c1 varchar);
CREATE TABLE
yellowbrick=# insert into abc values('abcdefghijklmnopqrstuvwxyz');
INSERT 0 1
yellowbrick=# insert into abc values('abcdefghijklm');
INSERT 0 1
yellowbrick=# select le_dst(c1,'abcdefghijklm') from abc;
ERROR:  LE_DST  argument exceeds maximum length of 10 bytes
HINT:   Change the max value with the 'max_levenshtein_string_size' configuration parameter
yellowbrick=# select le_dst('abcdefghijklm', c1) from abc;
ERROR:  LE_DST  argument exceeds maximum length of 10 bytes
HINT:   Change the max value with the 'max_levenshtein_string_size' configuration parameter
```

Also note that if the empty string is passed as one of the arguments in these queries, the result is the length of the other string:

```
yellowbrick=# select le_dst('', c1) from abc;
 le_dst
--------
    13
    26
(2 rows)

yellowbrick=# select le_dst(c1, '') from abc;
 le_dst
--------
    26
    13
(2 rows)
```

The following example shows the behavior when the `max_levenshtein_distance` parameter is set to `5` (and `max_levenshtein_string_size` is reset to its default value of `255`):

```
yellowbrick=# reset max_levenshtein_string_size;
RESET
yellowbrick=# set max_levenshtein_distance to 5;
SET
yellowbrick=# select c1, le_dst('abcdefghijklm', c1) from abc;
           c1             | le_dst
--------------------------+--------
 abcdefghijklmnopqrstuvwxyz |      6
```

```
 abcdefghijklm             |      0
(2 rows)
```

The first row returns 6 (`max_levenshtein_string_size` + 1) for the first row and `0` for the second row. The function returns `6` for the first value that has a distance greater than `5`, then stops processing.

LEFT

Return the first *n* characters in a string. If *n* is a negative number, return all but the last *n* characters. (See also RIGHT.)

```
LEFT(string, number)
```

For example, the following query uses the LEFT and RIGHT functions to extract goals scored from the `ftscore` column and find out which matches featured 10 or more goals:

```
premdb=# SELECT * FROM match
WHERE LEFT(ftscore,1)::INT+RIGHT(ftscore,1)::INT >=10
ORDER BY seasonid, matchday;
 seasonid |      matchday       | htid | atid | ftscore | htscore
----------+---------------------+------+------+---------+---------
       16 | 2007-09-29 00:00:00 |   32 |   83 | 7-4     | 2-1
       16 | 2007-12-29 00:00:00 |   42 |   83 | 6-4     | 1-1
       18 | 2009-11-22 00:00:00 |   42 |   95 | 9-1     | 1-0
       20 | 2011-08-28 00:00:00 |   26 |   51 | 8-2     | 3-1
       21 | 2012-12-29 00:00:00 |    2 |   77 | 7-3     | 1-1
       21 | 2013-05-19 00:00:00 |   44 |   75 | 5-5     | 1-3
(6 rows)
```

Note that the sum of the LEFT and RIGHT results is only possible because these values are cast from character strings to integers first.

LENGTH

Return the length of a character string, in terms of the number of characters. (The CHAR_LENGTH function also returns the number of characters.)

```
LENGTH(string)
```

```
premdb=# SELECT LENGTH(nickname), CHAR_LENGTH(nickname) FROM team WHERE city='London';
 length | char_length
--------+-------------
      7 |           7
      7 |           7
     10 |          10
      6 |           6
      9 |           9
      5 |           5
      5 |           5
      7 |           7
      4 |           4
(9 rows)
```

LOWER

Return a character string in all lowercase letters.

```
LOWER(string)
```

For example:

```
premdb=# SELECT LOWER(name) FROM team;
         lower
-------------------------
 arsenal
 aston villa
 barnsley
 birmingham city
 blackburn rovers
...
```

LPAD

Fill a character string to a specified length on the left side with designated characters (spaces by default). If the string is already longer than the specified length, it is truncated on the right side. See also RPAD.

```
LPAD(string, length [ , characters ] )
```

For example, add a sequence of zeros to the left side of the `teamid` column, for a length of 10 characters:

```
premdb=# SELECT LPAD(teamid::char(10),10,'0') FROM team;
   lpad
------------
 0000000001
 0000000002
 0000000003
 0000000004
 0000000005
 0000000006
 0000000007
 0000000008
 0000000009
 0000000010
...
```

LTRIM, RTRIM

Remove leading or trailing space characters, or other specified characters, from the start or end of a string.

```
LTRIM(string [ ,string ])
RTRIM(string [ ,string ])
```

where the first argument is the character string that you want to trim.

The second, optional argument is a list of characters that you want to trim from the start (`LTRIM`) or end (`RTRIM`) of the first string. If you do not specify the second argument, the default behavior is to trim leading or trailing space characters.

The individual characters listed in the second argument are removed without respect to the order in which they are specified. For example, `ytic` will remove the string `city` . The listed characters are case-sensitive. For example, `ytic` will not remove the `C` from `City` , but it will remove `ity` .

Examples

The following examples trim leading and trailing spaces by default:

```
premdb=# select ltrim('    City') from sys.const;
 ltrim
-------
 City
(1 row)

premdb=# select rtrim('    City     ') from sys.const;
  rtrim
----------
    City
(1 row)
```

The following example trims the characters `ytic` from team names that end with `City` . Note that the capital `C` of `City` is not trimmed.

```
premdb=# select name, rtrim(name, 'ytic') from team where name like '%City' order by teamid;
     name       |    rtrim
----------------+-------------
 Birmingham City | Birmingham C
 Bradford City   | Bradford C
 Cardiff City    | Cardiff C
 Coventry City   | Coventry C
 Hull City       | Hull C
 Leicester City  | Leicester C
 Manchester City | Manchester C
 Norwich City    | Norwich C
 Stoke City      | Stoke C
 Swansea City    | Swansea C
(10 rows)
```

The following example trims the characters `yticC` from team names that end with `City`. Note that the capital `C` of `City` is now trimmed.

```
premdb=# select name, rtrim(name, 'yticC') from team where name like '%City' order by teamid;
     name       |    rtrim
----------------+-------------
 Birmingham City | Birmingham
 Bradford City   | Bradford
 Cardiff City    | Cardiff
 Coventry City   | Coventry
 Hull City       | Hull
 Leicester City  | Leicester
 Manchester City | Manchester
 Norwich City    | Norwich
 Stoke City      | Stoke
 Swansea City    | Swansea
(10 rows)
```

MD5

Calculate the MD5 hash value for a character string and return the result in hexadecimal format.

```
MD5(string)
```

For example:

```
premdb=# SELECT DISTINCT MD5(ftscore) FROM match ORDER BY 1;
              md5
----------------------------------
 0299c06aed970473ae41d986b308cd09
 0dd954ea204f19a1b391b7828491927b
 12426c956d1bc5017082b12a969b0b7c
 13cee27a2bd93915479f049378cffdd3
 17380ddb842e984302034e1bb66c24e4
 1981e4a762b39858dc33f9ea28ed065a
 19a1de167122a18af369c749f4e40a48
 20fdcfb87379c9a34b5ecbe0dda37ced
 2db0874cc1843a7520d8d5fc2f8e3f37
 2dd6b4185ffaf931647b896faa2467dd
 2f5bb7747efda0546636fb385a3fa593
...
```

NYSIIS

Return the phonetic representation of a string by applying the New York State Identification and Intelligence System (NYSIIS) algorithm.

```
NYSIIS(string)
```

This function returns a `VARCHAR(6)` data type.

For example, find names in a table that match the phonetic representation of some constants:

```
premdb=# select * from player
where nysiis(firstname)=nysiis('Moe') or nysiis(firstname)=nysiis('Alex');
 playerid | teamid | seasonid | firstname | lastname | position |    dob     | weekly_wages | avg_mins_per_match | matches_played
----------+--------+----------+-----------+----------+----------+------------+--------------+--------------------+----------------
       6 |     25 |       27 | Alexis    | Sanchez  | F        | 1988-12-19 |       410000 |            71.2309 |           31.5 |
      13 |     23 |       27 | Mo        | Salah    | F        | 1992-06-15 |       200000 |            82.2765 |           36.2 |
(2 rows)
```

Find rows in the `team` table where the phonetic representations of values in three different columns all match:

```
premdb=# select * from team where nysiis(name)=nysiis(city) and nysiis(city)=nysiis(stadium);
 teamid | htid | atid |     name     | nickname |  city   |       stadium        | capacity | avg_att
--------+------+------+--------------+----------+---------+----------------------+----------+---------
    11 |   12 |   61 | Cardiff City | Clarets  | Cardiff | Cardiff City Stadium |    33280 |   0.000
(1 row)
```

OCTET_LENGTH

Return the number of bytes in a character string.

```
OCTET_LENGTH(string)
```

For example:

```
premdb=# SELECT season_name, OCTET_LENGTH(season_name) FROM season;
 season_name | octet_length
-------------+--------------
 1992-1993   |            9
 1993-1994   |            9
 1994-1995   |            9
 1995-1996   |            9
...
```

```
multibyte=# create table multibyte(c1 varchar(10));
CREATE TABLE
multibyte=# insert into multibyte values('谢谢');
INSERT 0 1
multibyte=# select length(c1), octet_length(c1) from multibyte;
 length | octet_length
--------+--------------
      2 |            6
(1 row)
```

POSITION

Return the starting position of a character string within another string. See also STRPOS.

```
POSITION(substring IN string)
```

```
premdb=# SELECT name, POSITION('City' IN name) pos
FROM team WHERE POSITION('City' IN name)>0;
     name       | pos
----------------+-----
 Birmingham City |  12
 Bradford City   |  10
 Cardiff City    |   9
 Coventry City   |  10
 Hull City       |   6
 Leicester City  |  11
 Manchester City |  12
 Norwich City    |   9
 Stoke City      |   7
 Swansea City    |   9
(10 rows)
```

For example, which `team` names start with their respective `city` names? The position column returns `1` for several team names because the city name starts at position 1 in the team name.

```
premdb=# SELECT name, city, POSITION(city IN name) FROM team
WHERE city IS NOT NULL ORDER BY city;
         name           |     city      | position
------------------------+---------------+----------
 Barnsley               | Barnsley      |        1
 Aston Villa            | Birmingham    |        0
 Birmingham City        | Birmingham    |        1
 West Bromwich Albion   | Birmingham    |        0
 Blackburn Rovers       | Blackburn     |        1
 Blackpool              | Blackpool     |        1
 Bolton Wanderers       | Bolton        |        1
 Bournemouth            | Bournemouth   |        1
 Bradford City          | Bradford      |        1
 Burnley                | Burnley       |        1
 Cardiff City           | Cardiff       |        1
 Coventry City          | Coventry      |        1
 Derby County           | Derby         |        1
 Hull City              | Hull          |        1
 Ipswich Town           | Ipswich       |        1
 Leeds United           | Leeds         |        1
 Leicester City         | Leicester     |        1
 Everton                | Liverpool     |        0
 Liverpool              | Liverpool     |        1
 Arsenal                | London        |        0
 Charlton Athletic      | London        |        0
 Chelsea                | London        |        0
...
```

REPEAT

Return a character string that is repeated *n* times.

```
REPEAT(string, number)
```

The first argument is any character string, or an expression that evaluates to a character string.

The second argument must be a literal number value (or an expression that evaluates to a literal number). You cannot specify a column name. A negative number is interpreted as zero. This function returns either a `CHAR` or a `VARCHAR` data type, depending on the input.

For example:

```
premdb=# select distinct repeat(city,3) from team where capacity>50000 order by 1;
           repeat
--------------------------------
 LondonLondonLondon
 ManchesterManchesterManchester
 NewcastleNewcastleNewcastle
(3 rows)
```

The following example shows the behavior of the function when character strings have trailing spaces. The `season` table contains four columns:

```
premdb=# \d season
             Table "public.season"
   Column    |         Type          | Modifiers
-------------+-----------------------+-----------
 seasonid    | smallint              |
 season_name | character(9)          |
 numteams    | smallint              |
 winners     | character varying(30) |
```

Two new rows are inserted:

```
premdb=# insert into season values(30, '2022-2023   ', 20, 'Unknown   ');
INSERT 0 1
premdb=# insert into season values(31, '2023     ', 20, 'Unknown   ');
INSERT 0 1
```

Trailing blanks are present in the values inserted into the `season_name` (`CHAR`) and `winners` (`VARCHAR`) columns. When the `REPEAT` function is run on these columns, the trailing spaces are not preserved for `season_name` , but they are preserved for `winners` :

```
premdb=> select repeat(season_name,2), repeat(winners,2) from season where seasonid in(30,31);
      repeat       |        repeat
-------------------+----------------------
 20232023          | Unknown   Unknown
```

```
 2022-20232022-2023 | Unknown   Unknown
(2 rows)
```

See also Trailing Blanks in Character Data.

REPLACE

Platforms: All platforms

Parent topic: String Functions

Replace all occurrences of an exact sequence of characters in a string with another exact sequence of characters. (Characters are not replaced individually.) See also TRANSLATE.

```
REPLACE(string, FROM characters, TO characters)
```

The following examples show the different behavior of the REPLACE and TRANSLATE functions.

```
premdb=# SELECT nickname, REPLACE(nickname, 'ue','00000')
FROM team WHERE teamid BETWEEN 1 AND 5;
 nickname | replace
----------+----------
 Gunners  | Gunners
 Villains | Villains
 Tykes    | Tykes
 Blues    | Bl00000s
 Rovers   | Rovers
(5 rows)

premdb=# SELECT nickname, TRANSLATE(nickname, 'ue','00000')
FROM team WHERE teamid between 1 and 5;
 nickname | translate
----------+-----------
 Gunners  | G0nn0rs
 Villains | Villains
 Tykes    | Tyk0s
 Blues    | Bl00s
 Rovers   | Rov0rs
(5 rows)
```

REVERSE

Return a character string with its characters in reverse order.

```
REVERSE(string)
```

For example:

```
premdb=# SELECT REVERSE(name) FROM team WHERE city='Manchester';
     reverse
-------------------
 ytiC retsehcnaM
 detinU retsehcnaM
(2 rows)
```

RIGHT

Return the last *n* characters in a string. If *n* is a negative number, return all but the first *n* characters. (See also LEFT.)

```
RIGHT(string, number)
```

For example, return the last 3 characters in the `capacity` column:

```
premdb=# SELECT stadium, RIGHT(capacity::CHAR(5),3)
FROM team WHERE capacity>0 ORDER BY 2;
      stadium        | right
---------------------+-------
 The Hawthorns       | 000
 Stadium of Light    | 000
 Oakwell Stadium     | 009
 St. Andrew's        | 016
 Upton Park          | 016
 Etihad Stadium      | 097
 Fratton Park        | 100
 The Valley          | 111
...
```

Note that the column has to be cast to a CHAR column before the RIGHT function is applied.

RPAD

Fill a character string to a specified length on the right side with designated characters (spaces by default). If the string is already longer than the specified length, it is truncated on the right side. See also LPAD.

```
RPAD(string, length [, characters])
```

For example, add a sequence of zeros to the right side of the `teamid` column, for a length of 10 characters:

```
premdb=# SELECT RPAD(teamid::CHAR(10),10,'0') FROM team;
   rpad
------------
 1000000000
 2000000000
 3000000000
 4000000000
 5000000000
 6000000000
 7000000000
 8000000000
 9000000000
 1000000000
...
```

SPLIT_PART

Split a character string where a delimiter string falls, then return one part of the string according to its numeric position.

```
SPLIT_PART(string, delimiter, number)
```

For example, the following string has five parts delimited by commas:

```
10,20,30,40,50
```

The first part of this string is the value `10`, the second is `20`, and so on.

In the following example, the query splits the `ftscore` column on the hyphen character (`-`), then returns the values on either side as home and away goals:

```
premdb=# SELECT ftscore, SPLIT_PART(ftscore,'-',1) homegoals, SPLIT_PART(ftscore,'-',2) awaygoals
FROM match LIMIT 5;
 ftscore | homegoals | awaygoals
---------+-----------+-----------
 0-1     | 0         | 1
 0-1     | 0         | 1
 2-1     | 2         | 1
 3-0     | 3         | 0
 3-0     | 3         | 0
(5 rows)
```

STRPOS

Return the numeric position of the specified substring within a character string. Optionally, specify a starting position and an occurrence number.

```
STRPOS(string, substring, [, position [, occurrence] ])
```

`INSTR` is an alias for `STRPOS`.

Tip: The two-argument version of this function is equivalent to POSITION(substring IN string), but the order of the arguments is reversed.

Parameters

string

Specify a character string or a string expression, such as a column name, that will be searched.

substring

Specify a substring to search for within the first `string`. If you specify an empty string, the function returns `1`.

position

Optionally, specify a starting position in `substring`. This argument must be an integer data type. The purpose of this argument is to provide a way for the evaluation of the function to be based on only the part of the string that starts at the specified position. However, regardless of the `position` value you specify, the function result always counts from the beginning of `string`.

Position `0` is treated the same as `1`.

occurrence

Optionally, specify which occurrence of `substring` to evaluate, assuming that `substring` occurs multiple times within `string`. This value must be an integer greater than `0`.

Examples

For example, find team names that contain `City` and return the position in the string where `City` begins. Return 0 if `City` is not found. (If `name` is `null`, return `null`.)

```
premdb=# select name, strpos(name, 'City') from team;
         name           | strpos
------------------------+--------
 Arsenal                |      0
 Aston Villa            |      0
 Barnsley               |      0
 Birmingham City        |     12
 Blackburn Rovers       |      0
 Blackpool              |      0
 Bolton Wanderers       |      0
 Bournemouth            |      0
```

```
 Bradford City           |     10
 Burnley                 |      0
 Cardiff City            |      9
...
```

Return the position of the third instance of the string `er` in team names. Start searching from position 1 in each name. Return only those rows where `er` appears for a third time.

```
premdb=# select name, strpos(name, 'er', 1, 3) instring from team where instring>0;
         name           | instring
------------------------+----------
 Wolverhampton Wanderers |       21
(1 row)
```

Adjust the previous query to return the position of the second instance of the string `er`.

```
premdb=# select name, strpos(name, 'er', 1, 2) instring from team where instring>0;
         name           | instring
------------------------+----------
 Bolton Wanderers        |       14
 Wolverhampton Wanderers |       19
(2 rows)
```

Adjust the previous query to return the position of the second instance of the string `er`, starting from position `15` in each team name.

```
premdb=# select name, strpos(name, 'er', 15, 2) instring from team where instring>0;
         name           | instring
------------------------+----------
 Wolverhampton Wanderers |       21
(1 row)
```

SUBSTR

Return a substring from a character string, based on start and end values. See also SUBSTRING.

```
SUBSTR(string, start [, count])
```

Note: `SUBSTRING` and `SUBSTR` are not synonymous. `SUBSTRING` returns a `VARCHAR` data type, but `SUBSTR` returns the data type of its input string.

string

An expression that evaluates to a character string. The data type of this string determines the return type of the function.

start

Starting position of the substring (an integer).

count

Number of characters in the substring.

The following example returns two substrings of the `ftscore` column, which contains 3-character strings such as `1-1`, `2-1`, `0-0`, `3-2`, and so on. The first substring returns only the first character, and the second substring returns only the third character.

```
premdb=# SELECT ftscore, SUBSTR(ftscore,1,1) homegoals, SUBSTR(ftscore,3,1) awaygoals
FROM match WHERE seasonid=2 AND htid=2 AND atid=52;
 ftscore | homegoals | awaygoals
---------+-----------+-----------
 1-2     | 1         | 2
(1 row)
```

SUBSTRING

Return an exact set of characters from a string. See also SUBSTR and SUBSTRING (Pattern Match).

```
SUBSTRING(string [FROM start] [FOR count])
```

Note: `SUBSTRING` and `SUBSTR` are not synonymous. `SUBSTRING` returns a `VARCHAR` data type, but `SUBSTR` returns the data type of its input string.

string
An expression that evaluates to a character string.

FROM start
Starting position for the substring (an integer).

FOR count
Number of characters in the substring.

For example, select the first 10 characters from each team name:

```
premdb=# SELECT name, SUBSTRING(name FROM 1 FOR 10) FROM team;
         name           | substring
------------------------+------------
 Arsenal                | Arsenal
 Aston Villa            | Aston Vill
 Barnsley               | Barnsley
 Birmingham City        | Birmingham
 Blackburn Rovers       | Blackburn
 Blackpool              | Blackpool
...
```

For example, select the last four characters from the `season_name` column:

```
premdb=# SELECT SUBSTRING(season_name FROM 6 FOR 4) FROM season;
 season_name | substring
-------------+-----------
 1992-1993   | 1993
 1993-1994   | 1994
 1994-1995   | 1995
 1995-1996   | 1996
 1996-1997   | 1997
 1997-1998   | 1998
 1998-1999   | 1999
 1999-2000   | 2000
 2000-2001   | 2001
...
```

TO_ASCII

Return the ASCII value for a character string stored in a `LATIN9` database.

```
TO_ASCII(string [, 'latin9'])
```

For example:

```
premdb=# SELECT TO_ASCII('µ') from sys.const;
 to_ascii
----------
 u
(1 row)

premdb=# premdb=# SELECT TO_ASCII('µ', 'latin9') from sys.const;
 to_ascii
----------
 u
(1 row)
```

This function runs only on `LATIN9` databases and accepts only the `LATIN9` encoding as the second argument. Encoding conversion from UTF8 to ASCII is not supported.

For example, the following query returns an error:

```
premdb=# SELECT TO_ASCII('µ','UTF8') from sys.const;
ERROR:  encoding conversion from UTF8 to ASCII not supported
```

TO_HEX

Return the hexadecimal value for a number.

```
TO_HEX(number)
```

where `number` is a BIGINT or INTEGER value or expression. This function does not work on SMALLINT columns.

For example:

```
premdb=# SELECT capacity, TO_HEX(capacity)
FROM team ORDER BY teamid LIMIT 5;
 capacity | to_hex
----------+--------
   60260 | eb64
   42785 | a721
   23009 | 59e1
   30016 | 7540
   31367 | 7a87
(5 rows)
```

TRANSLATE

Replace individual characters in a string, given *corresponding* lists of "from" and "to" characters. See also REPLACE

```
TRANSLATE(string, from_chars, to_chars)
```

This example removes all of the vowels from the nickname column and replaces them with an empty string (no characters):

```
premdb=# SELECT nickname, TRANSLATE(nickname,'aeiou','') FROM team;
  nickname  | translate
------------+-----------
 Gunners    | Gnnrs
 Villains   | Vlls
 Tykes      | Tyks
 Blues      | Bls
 Rovers     | Rvrs
 Seasiders  | Ssdrs
...
```

This example removes all of the vowels and replaces each one with a zero. Note that for this example to work, 5 zeros must be listed, one corresponding to each vowel.

```
premdb=# SELECT nickname, TRANSLATE(nickname,'aeiou','00000') FROM team;
  nickname  | translate
------------+------------
 Gunners    | G0nn0rs
 Villains   | V0ll00ns
 Tykes      | Tyk0s
 Blues      | Bl00s
 Rovers     | R0v0rs
 Seasiders  | S00s0d0rs
 Trotters   | Tr0tt0rs
 Cherries   | Ch0rr00s
...
```

TRIM

Remove specific leading or trailing characters from a string. Remove spaces by default. See also BTRIM.

```
TRIM ([ LEADING | TRAILING | BOTH ] [characters] FROM string)
```

By default, `TRIM` without `BOTH` trims both sides of the string.

Examples

Trim leading and trailing spaces:

```
premdb=# select trim(from '    City') from sys.const;
 btrim
-------
 City
(1 row)
```

```
premdb=# select trim(from '    City     ') from sys.const;
 btrim
-------
 City
(1 row)
```

```
premdb=# select trim(trailing from '    City     ') from sys.const;
  rtrim
----------
    City
(1 row)
```

Remove a specific set of leading and trailing characters:

```
premdb=# select nickname, trim('aeiousy' from nickname) from team where nickname like '%s';
  nickname  |   btrim
------------+-----------
 Gunners    | Gunner
 Villains   | Villain
 Tykes      | Tyk
 Blues      | Bl
 Rovers     | Rover
 Seasiders  | Seasider
 Trotters   | Trotter
 Cherries   | Cherr
...
```

Remove a specific set of trailing characters:

```
premdb=# select season_name, trim(trailing '0123456789' from season_name) from season;
 season_name |   rtrim
-------------+-----------
 1992-1993   | 1992-
 1993-1994   | 1993-
 1994-1995   | 1994-
 1995-1996   | 1995-
 1996-1997   | 1996-
...
```

UPPER

Return a character string in all uppercase letters.

```
UPPER(string)
```

For example:

```
premdb=# SELECT UPPER(name) FROM team;
         upper
-------------------------
 ARSENAL
 ASTON VILLA
 BARNSLEY
 BIRMINGHAM CITY
 BLACKBURN ROVERS
...
```

System Functions

Platforms: All platforms

Parent topic: SQL Functions and Operators

In this section:

This section covers functions that return information about the system or generate data.

CURRENT_CLUSTER()

Return the name of the cluster that is currently connected.

```
CURRENT_CLUSTER()
```

For example:

```
premdb=# select current_cluster();
  current_cluster
--------------------
 bobr-rc5-cluster-01
(1 row)
```

CURRENT_DATABASE()

Return the database for the current session.

```
CURRENT_DATABASE()
CURRENT_CATALOG
```

These functions are synonyms. `CURRENT_DATABASE()` requires the parentheses. `CURRENT_CATALOG` does not work with parentheses.

```
premdb=# select CURRENT_DATABASE();
 current_database
------------------
 premdb
(1 row)
```

```
premdb=# SELECT CURRENT_CATALOG;
 current_database
------------------
 premdb
(1 row)
```

CURRENT_QUERY()

Return the text of the query that is currently running.

```
CURRENT_QUERY()
```

The parentheses are required. For example:

```
premdb=# create table q as select current_query() from sys.const;
SELECT 1
premdb=# select * from q;
                    current_query
-----------------------------------------------------------
 create table q as select current_query() from sys.const;
(1 row)
```

CURRENT_SCHEMA

Return the name of the first schema in the search path (or a null value if the search path is empty).

```
CURRENT_SCHEMA[()]
```

The parentheses are optional. For example:

```
yellowbrick=# SET SCHEMA 'stage';
SET

yellowbrick=# SELECT CURRENT_SCHEMA();
current_schema
----------------
stage
(1 row)
```

See also CURRENT_SCHEMAS().

---

search_path (configuration variable)

See the `search_path` configuration parameter to see how database objects are searched for in schemas.

Alternatively, you can use a `set schema` command with single quotes around the schema name to change the current schema.

```
yellowbrick=# set schema 'neilschema';
SET
```

Note: `set schema 'value'` is equivalent to `set search_path to 'value'`, but you can only specify one schema in the `set schema` syntax.

A qualified schema name consists of the schema name and table name separated by dots. You may also include the database name before the schema name.

```
database.schema.table
```

New objects that are not schema-qualified, consisting of just the table name, are always created in the first schema of the search path. To access any object that is not in a schema listed in the search path, you must provide the schema name.

CURRENT_SCHEMAS()

Return all schemas in the search path (or a null value if the search path is empty).

```
CURRENT_SCHEMAS()
```

For example:

```
premdb=# set search_path to public, sys, bobr;
SET
premdb=# select current_schemas();
current_schemas
-----------------
public,sys,bobr
(1 row)
```

See also CURRENT_SCHEMA.

CURRENT_SETTING

Return the current setting of a specific configuration parameter.

```
CURRENT_SETTING('parameter')
```

For example:

```
premdb_utf8=# set application_name='YBSQL_CLIENT';
SET
premdb_utf8=# select current_setting('application_name');
 current_setting
-----------------
 YBSQL_CLIENT
(1 row)

premdb_utf8=# select current_setting('search_path') from sys.const;
     current_setting
---------------------------
 public,premdb_tables,bobr
(1 row)
```

See also SHOW.

CURRENT_USER

Return the user currently subject to permission checking, which may or may not be the same user who initiated the session.

```
CURRENT_USER | USER
```

For example:

```
premdb=# \c premdb yellowbrick
You are now connected to database "premdb" as user "yellowbrick".
premdb=# select current_user;
 current_user
--------------
 yellowbrick
(1 row)
```

`USER` is a synonym for `CURRENT_USER`.

DEFAULT_CLUSTER()

Platforms: All platforms

Parent topic: System Functions

Return the name of the default cluster for an instance.

```
DEFAULT_CLUSTER()
```

For example:

```
yellowbrick=# select default_cluster();
   system_cluster
--------------------
 eng-rc9-cluster-01
(1 row)
```

See also ALTER USER SET DEFAULT_CLUSTER and ALTER SYSTEM SET CLUSTER.

GETDATABASEENCODING()

Return the database encoding for the current database.

```
GETDATABASEENCODING()
```

For example:

```
premdb=# select getdatabaseencoding();
 getdatabaseencoding
---------------------
 LATIN9
(1 row)

premdb=# create database premdb_utf8 encoding 'utf8';
CREATE DATABASE
premdb=# \c premdb_utf8
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)
You are now connected to database "premdb_utf8" as user "yellowbrick".
premdb_utf8=# select getdatabaseencoding();
 getdatabaseencoding
---------------------
 UTF8
(1 row)
```

See also CREATE DATABASE.

HAS_CLUSTER_PRIVILEGE

Return `true` if the user has the specified privilege on the specified cluster.

```
HAS_EXTERNAL_STORAGE_PRIVILEGE([role,] object, privilege)
```

role

If the `role` (or user) name is omitted, the function returns results for the current user.

object

Name of an existing cluster.

privilege

Specify one of the following privileges:

- `USAGE`
- `USAGE WITH GRANT OPTION`
- `ALTER CLUSTER`
- `ALTER CLUSTER WITH GRANT OPTION`
- `DROP CLUSTER`
- `DROP CLUSTER WITH GRANT OPTION`

For example:

```
yellowbrick=# select has_cluster_privilege('uat-rc5-cluster-01','usage');
 has_cluster_privilege
-----------------------
 t
(1 row)

yellowbrick=# select has_cluster_privilege('hkane@thfc.com','uat-rc5-cluster-01', 'usage');
 has_cluster_privilege
-----------------------
 f
(1 row)
```

HAS_DATABASE_PRIVILEGE

Return `true` if the user has the specified privilege for the specified database.

```
HAS_DATABASE_PRIVILEGE([role,] database, privilege)
```

If the `role` name is omitted, the function returns results for the current user.

Valid privilege values are:

- `ALTER ANY SCHEMA`
- `BACKUP`
- `BULK LOAD`
- `CONNECT`
- `CONTROL`
- `CREATE`
- `DROP ANY SCHEMA`
- `EXPLAIN QUERY`
- `RESTORE`
- `TEMP` or `TEMPORARY`
- `TRACE QUERY`
- `VIEW QUERY TEXT`

Note: This function cannot be used in a query that has a FROM clause (with table references).

For example:

```
premdb=# select has_database_privilege('bobr','premdb','create');
 has_database_privilege
------------------------
 f
(1 row)
premdb=# select has_database_privilege('premdb','create');
 has_database_privilege
------------------------
 t
(1 row)

premdb=# select has_database_privilege('bobr','premdb','bulk load');
 has_database_privilege
------------------------
 t
(1 row)
```

HAS_EXTERNAL_*_PRIVILEGE

Note: These functions cannot be used in a query that has a `FROM` clause (with table references).

HAS_EXTERNAL_STORAGE_PRIVILEGE

Return `true` if the user has the specified privilege for the specified external storage object.

```
HAS_EXTERNAL_STORAGE_PRIVILEGE([role,] object, privilege)
```

role

If the `role` (or user) name is omitted, the function returns results for the current user.

object

Name of an existing external storage object.

privilege

Specify one of the following privileges:

- `DROP EXTERNAL STORAGE`
- `USAGE EXTERNAL STORAGE`

For example:

```
premdb=> select has_external_storage_privilege('bobr','premdbs3','drop external storage');
 has_external_storage_privilege
--------------------------------
 t
(1 row)
```

HAS_EXTERNAL_FORMAT_PRIVILEGE

Return `true` if the user has the specified privilege for the specified external format object.

```
HAS_EXTERNAL_FORMAT_PRIVILEGE([role,] object, privilege)
```

role

If the `role` (or user) name is omitted, the function returns results for the current user.

object

Name of an existing external format object.

privilege

Specify one of the following privileges:

- `DROP EXTERNAL FORMAT`
- `USAGE EXTERNAL FORMAT`

For example:

```
premdb=> select has_external_format_privilege('backup1','premfiles','usage external format');
 has_external_format_privilege
-------------------------------
 f
(1 row)
```

HAS_EXTERNAL_LOCATION_PRIVILEGE

Return `true` if the user has the specified privilege for the specified external location object.

```
HAS_EXTERNAL_LOCATION_PRIVILEGE([role,] object, privilege)
```

role

If the `role` (or user) name is omitted, the function returns results for the current user.

object

Name of an existing external location object.

privilege

Specify the following privilege: `DROP EXTERNAL LOCATION`.

For example:

```
premdb=> select has_external_location_privilege('bobr','premdbs3data','drop external location');
 has_external_location_privilege
---------------------------------
 t
(1 row)
```

HAS_FUNCTION_PRIVILEGE

Return `true` if the user has the specified privilege for the specified stored procedure.

```
HAS_FUNCTION_PRIVILEGE([role,] function, privilege)
```

If the `role` name is omitted, the function returns results for the current user. In this context, `function` is the name of a stored procedure created with CREATE PROCEDURE.

The only valid privilege value is `EXECUTE`.

Note: This function cannot be used in a query that has a `FROM` clause (with table references).

For example:

```
premdb=# create procedure addteam(name varchar(30)) as
$$ begin
insert into team(name) values(name);
end;
$$ language plpgsql;
CREATE PROCEDURE

premdb=# select has_function_privilege('bobr','addteam(varchar)','execute');
 has_function_privilege
------------------------
 f
(1 row)

premdb=# grant execute on procedure addteam(varchar) to bobr;
GRANT
premdb=# select has_function_privilege('bobr','addteam(varchar)','execute');
 has_function_privilege
------------------------
 t
(1 row)
```

HAS_KEY_PRIVILEGE

Return `true` if the specified user (or current user) has the `HMAC` privilege on the encryption key. See also GRANT ON KEY.

```
HAS_KEY_PRIVILEGE(role, key, privilege)
```

This function returns results for the current user if no role is specified. Specify the name of the encryption key and one of the following privileges:

- `ENCRYPT`
- `DECRYPT`
- `HMAC`

Instead of the key name, you can specify the key ID, as returned by the sys.key view.

Note: This function cannot be used in a query that has a FROM clause (with table references).

For example:

```
premdb=# grant hmac on key yb100key to yb100;
GRANT
premdb=# select has_key_privilege('yb100','yb100key','hmac');
 has_key_privilege
-------------------
 t
(1 row)
premdb=# \c premdb yb100
You are now connected to database "premdb" as user "yb100".
premdb=> select has_key_privilege('yb100key','hmac');
 has_key_privilege
-------------------
 t
(1 row)
```

HAS_ROLE_PRIVILEGE

Return `true` if a role (or user) has the specified `ROLE` privilege on another specified role.

```
HAS_ROLE_PRIVILEGE([role1,] [role2,] privilege)
```

`role1`, `role2`, or both must be specified. The first role specified is the role being checked, and the second is the role that the first role may have privileges to alter, drop, or control. If only one role is specified, the function checks the privileges of the current user.

Valid privilege values are:

- `ALTER ROLE`
- `DROP ROLE`
- `CONTROL`

Note: This function cannot be used in a query that has a `FROM` clause (with table references).

For example, in this session `backup1` is the logged in user (current user). The results show that `backup1` has privilege to alter role `bar`, but `bar` does not have privilege to alter `backup1`.

```
premdb=# \c premdb backup1
You are now connected to database "premdb" as user "backup1".
premdb=> select has_role_privilege('bar','alter role');
 has_role_privilege
--------------------
 t
(1 row)

premdb=> select has_role_privilege('bar','backup1','alter role');
 has_role_privilege
--------------------
 f
(1 row)
```

HAS_SCHEMA_PRIVILEGE

Return `true` if the user has the specified privilege for the specified schema.

```
HAS_SCHEMA_PRIVILEGE([role,] schema, privilege)
```

If the `role` name is omitted, the function returns results for the current user.

Valid privilege values are:

- `CREATE`
- `USAGE`

Note: This function cannot be used in a query that has a FROM clause (with table references).

For example:

```
premdb=# select has_schema_privilege('ybd','bobr','create');
 has_schema_privilege
----------------------
 t
(1 row)

premdb=# select has_schema_privilege('bobr','usage');
 has_schema_privilege
----------------------
 t
(1 row)
```

HAS_SYSTEM_PRIVILEGE

Return `true` if the user has the specified privilege on the system.

```
HAS_SYSTEM_PRIVILEGE([role,], privilege)
```

If the `role` name is omitted, the function returns results for the current user.

Valid privilege values are:

- `ALTER ANY DATABASE`
- `ALTER ANY ROLE`
- `BACKUP ANY DATABASE`
- `CONTROL ANY SESSION`
- `CONTROL LDAP`
- `CREATE DATABASE`
- `CREATE ROLE`
- `DROP ANY DATABASE`
- `DROP ANY ROLE`
- `EXPLAIN QUERY`
- `RESTORE ANY DATABASE`
- `TRACE QUERY`
- `VIEW QUERY TEXT`

Note: This function cannot be used in a query that has a FROM clause (with table references).

For example:

```
premdb=# select has_system_privilege('bobr','restore any database');
 has_system_privilege
----------------------
 f
(1 row)
premdb=# grant restore any database on system to bobr;
GRANT
premdb=# select has_system_privilege('bobr','restore any database');
 has_system_privilege
----------------------
 t
(1 row)
```

```
premdb=# grant create database on system to yb100;
GRANT
premdb=# select has_system_privilege('yb100','create database');
```

```
 has_system_privilege
----------------------
 t
(1 row)
```

HAS_TABLE_PRIVILEGE

Return `true` if the user has the specified privilege for the specified table.

```
HAS_TABLE_PRIVILEGE([role,] table, privilege)
```

If the `role` name is omitted, the function returns results for the current user. The value `public` is allowed as a role name. The table name may also specify the schema (`schema.table`).

Valid privilege values are:

- `SELECT`
- `INSERT`
- `UPDATE`
- `DELETE`
- `TRUNCATE`
- `REFERENCES`

Note: This function cannot be used in a query that has a FROM clause (a query that has table references).

For example:

```
premdb=# select has_table_privilege('bobr','match','select');
 has_table_privilege
---------------------
 f
(1 row)

premdb=# select has_table_privilege('match','select');
 has_table_privilege
---------------------
 t
(1 row)
```

PG_ADVISORY_LOCK

Obtain an exclusive session-level advisory lock. If another session already holds a lock on the same resource key, this function will wait until the resource becomes available. The lock is exclusive.

Advisory locks are session-level locks on application-defined resources, identified by a single `BIGINT` key value. The system does not enforce the use of advisory locks; the application must handle them correctly. Once acquired, an advisory lock is held until it is explicitly released or the session ends. A lock can be acquired multiple times by its owning process; for each completed lock request there must be a corresponding unlock request before the lock is actually released. If a session already holds a given advisory lock, additional requests by it will always succeed, even if other sessions are waiting for the lock.

```
PG_ADVISORY_LOCK(key)
```

where `key` is a `BIGINT` value.

For example:

```
premdb=# select pg_advisory_lock(32451239);
 pg_advisory_lock
------------------

(1 row)
```

This function returns a `VOID` data type.

Note: Advisory lock functions are not supported in queries with table references in the `FROM` clause.

PG_ADVISORY_UNLOCK

Release an exclusive session-level advisory lock. Multiple lock requests stack; if the same resource is locked three times, it must be unlocked three times to be released for use by other sessions.

```
PG_ADVISORY_UNLOCK(lock_id)
```

where `lock_id` is a `BIGINT` value.

This function returns a `BOOLEAN` data type. For example, when a lock is released successfully, the function returns `true`:

```
premdb=# select pg_advisory_unlock(32451239);
 pg_advisory_unlock
--------------------
 t
(1 row)
```

If a lock with the specified key does not exist, the function returns false:

```
premdb=# select pg_advisory_unlock(32451240);
WARNING:  you don't own a lock of type ExclusiveLock
 pg_advisory_unlock
--------------------
 f
(1 row)
```

Note: Advisory lock functions are not supported in queries with table references in the `FROM` clause.

PG_HAS_ROLE

Return `true` if a user can access a role with `MEMBER` or `USAGE` privilege.

```
PG_HAS_ROLE([user,] role, privilege)
```

If the `user` name is omitted, the function returns results for the current user.

Valid privilege values are:

- `MEMBER` : whether the user has direct or indirect membership in the role (the right to do SET ROLE).
- `USAGE` : whether the privileges of the role are immediately available (without doing SET ROLE).

Note: This function cannot be used in a query that has a FROM clause (that is, a query that selects from tables).

```
premdb=# select pg_has_role('bobr','member');
 pg_has_role
-------------
 t
(1 row)
premdb=# select pg_has_role('bobr','bobr','member');
 pg_has_role
-------------
 t
(1 row)
```

PG_RELOAD_CONF()

Reload the system-level configuration after modifying one or more parameters.

```
pg_reload_conf();
```

For example:

```
yellowbrick=# select pg_reload_conf();
 pg_reload_conf
----------------
 t
(1 row)
```

This function returns a `BOOLEAN` data type.

Typically, you would run this function after modifying one or more parameters that do not require a database restart. For example:

```
premdb=# show enable_alternative_round;
 enable_alternative_round
--------------------------
 off
(1 row)

premdb=# alter system set enable_alternative_round to on;
WARNING:  Will be effective after the next server configuration reload, or after the next server restart in the case of parameters
ALTER SYSTEM
premdb=# select pg_reload_conf();
 pg_reload_conf
----------------
 t
(1 row)

premdb=# \c premdb bobr
Password for user bobr:
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)
You are now connected to database "premdb" as user "bobr".
premdb=> show enable_alternative_round;
 enable_alternative_round
--------------------------
 on
(1 row)
```

See also ALTER SYSTEM, SHOW, and Configuration Parameters.

Note: This reload function is not supported in queries with table references in the `FROM` clause.

PG_TRY_ADVISORY_LOCK

Obtain an exclusive session-level advisory lock without waiting for the lock to become available. This function either obtains the lock immediately and returns `true`, or returns `false` if the lock cannot be acquired immediately.

```
PG_TRY_ADVISORY_LOCK(lock_id)
```

where `lock_id` is a `BIGINT` value.

This function returns a `BOOLEAN` data type. For example:

```
premdb=# select pg_try_advisory_lock(32451240);
 pg_try_advisory_lock
----------------------
 t
(1 row)
```

Note: Advisory lock functions are not supported in queries with table references in the `FROM` clause.

SESSION_USER

Return the user who initiated the session.

```
SESSION_USER
```

```
% ybsql yellowbrick bobr -W
Password for user bobr:
ybsql (6.1.0)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)
Type:  \h for help with SQL commands
       \? for help with ybsql commands
       \g or terminate with semicolon to execute query
       \q to quit
yellowbrick=> select session_user;
 session_user
--------------
 bobr
(1 row)
```

Typically, the session user is the user who opened the current database connection; however, superusers can change the session user by running the SET SESSION AUTHORIZATION command.

sys.execution_id()

Return the execution ID of the running query.

```
sys.execution_id()
```

To return a meaningful result, always use this function in a query with a table reference, such as a reference to sys.const.

For example:

```
premdb=# select sys.execution_id() from sys.const;
 execution_id
--------------
      6051039
(1 row)
```

sys.filesys_number_from_rowid

Given a row, return its file system location (its SSD).

```
sys.filesys_number_from_rowid(rowid)
```

See System Columns in Tables.

This function accepts either the string `rowid` as its input or a specific `rowid` value. In most cases you will use the string `rowid` to return the specific row ID for each row in a set of rows. (If you specify an individual row ID as the input to the function, make sure it is valid; the system does not detect row IDs that are not valid or do not exist.)

For example:

```
yellowbrick_test=# select distinct sys.filesys_number_from_rowid(rowid)
from matchstats order by 1;
 filesys_number_from_rowid
---------------------------
                         0
                         1
                         4
                         5
                         6
                         7
(6 rows)
```

```
premdb=# select rowid, sys.filesys_number_from_rowid(rowid) from match order by 1 limit 5;
     rowid       | filesys_number_from_rowid
-----------------+---------------------------
 5630521744818176 |                         5
 5630521744818177 |                         5
 5630521744818178 |                         5
 5630521744818179 |                         5
 5630521744818180 |                         5
(5 rows)
```

```
premdb=# select sys.filesys_number_from_rowid(5630521744818176) from match limit 1;
 filesys_number_from_rowid
---------------------------
                         5
(1 row)
```

sys.gen_random_uuid()

Generate a random `UUID` value (version 4).

```
SYS.GEN_RANDOM_UUID()
```

This function does not have any parameters. The parentheses are required. Include `sys.` before the function name if the `sys` schema is not in your search path. For information about the `UUID` data type, see UUID.

For example:

```
premdb=# select sys.gen_random_uuid();
          gen_random_uuid
--------------------------------------
 f280eb64-a88f-40d8-9fc0-f345fc4d2726
(1 row)
```

Add a `UUID` column to a table called `newmatchstats`, then generate unique values for it:

```
premdb=# alter table newmatchstats add column matchid uuid;
ALTER TABLE
premdb=# \d newmatchstats;
       Table "public.newmatchstats"
  Column  |         Type         | Modifiers
----------+----------------------+-----------
 seasonid | smallint             |
 matchday | date                 |
 htid     | smallint             |
 atid     | smallint             |
 moment   | character varying(5) |
 matchid  | uuid                 |

Distribution: Hash (seasonid)

premdb=# update newmatchstats set matchid=sys.gen_random_uuid();
UPDATE 524966

premdb=# select count(distinct matchid) from newmatchstats;
 count
--------
 524966
(1 row)
```

The `COUNT(DISTINCT)` query verifies the uniqueness of the generated `UUID` values.

sys.inode_number_from_rowid

Given a row, return its inode number (its shard location).

```
sys.inode_number_from_rowid(rowid)
```

See System Columns in Tables.

This function accepts either the string `rowid` as its input or a specific `rowid` value. In most cases you will use the string `rowid` to return the specific row ID for each row in a set of rows. (If you specify an individual row ID as the input to the function, make sure it is valid; the system does not detect row IDs that are not valid or do not exist.)

For example, you can find out which shards are used to store rows for a given table:

```
premdb=# select distinct sys.inode_number_from_rowid(rowid) from newmatchstats order by 1;
 inode_number_from_rowid
-------------------------
                    4097
                    8193
                   19457
                   19458
                   46082
                   68609
                  130049
                  146433
                  149505
                  196609
                  250881
(11 rows)
```

sys.ldap_sync

Synchronize LDAP users and groups on an appliance.

```
premdb=# select sys.ldap_sync();
              ldap_sync
-----------------------------------------
 LDAP authorization data was synchronized+

(1 row)
```

Based on your current LDAP configuration, the Yellowbrick appliance synchronizes with the users and groups in the LDAP directory and creates users and roles with the same credentials. See also Synchronizing Users and Groups, which explains how to use the SMC for this purpose.

CONTROL LDAP privilege is required. Unauthorized users will see this message:

```
premdb=> select sys.ldap_sync();
                                ldap_sync
--------------------------------------------------------------------------------
 permission denied to query sys.ldap_sync(). CONTROL LDAP privilege is needed.
(1 row)
```

Note: This function cannot be used in a query that has a `FROM` clause (with table references).

sys.oldest_rollback_point_id_in_replica

Return the ID of the oldest rollback point for a replica. This function is useful when you need to roll back a database to a valid snapshot created during database replication. See also Rolling Back a Replicated Database and ROLLBACK DATABASE TO SNAPSHOT.

```
sys.oldest_rollback_point_id_in_replica('replica_name')
```

For example:

```
premdb=# select sys.oldest_rollback_point_id_in_replica('premdb_replica1');
 oldest_rollback_point_id_in_replica
-------------------------------------
 premdb_replica1_19_12_16_23_24_17
(1 row)
```

This function returns a `VARCHAR` data type.

To check the name of a replica before using this function, query the `sys.replica` view. For example:

```
premdb=# select * from sys.replica;
-[ RECORD 1 ]------------------+----------------------------
replica_id                    | 17605
database_id                   | 17563
remote_server_id              | 16417
backup_chain_id               | premrep1
name                          | premrep1
last_replication_time         | 2020-01-24 17:45:06.456424-08
status                        | PAUSED
frequency                     | 300
alias                         | premrep1db
bandwidth_limit_bytes         | [NULL]
exclude_filter                | [NULL]
security_mode                 | all
user_resolution_mode          | [NULL]
user_duplicate_resolution_mode | [NULL]
is_sync_users                 | [NULL]
is_sync_roles                 | [NULL]
is_sync_grants                | [NULL]
is_sync_acls                  | [NULL]
is_sync_superusers            | [NULL]
reverse_replica               | [NULL]
```

sys.row_number_from_rowid

Return the row number for a row (starting from 0). Row numbers are relative values that are unique to a given combination of compute node, SSD, and shard.

```
sys.row_number_from_rowid(rowid)
```

See System Columns in Tables.

This function accepts either the string `rowid` as its input or a specific `rowid` value. In most cases you will use the string `rowid` to return the specific row ID for each row in a set of rows. (If you specify an individual row ID as the input to the function, make sure it is valid; the system does not detect row IDs that are not valid or do not exist.)

For example:

```
yellowbrick_test=# select sys.worker_id_from_rowid(rowid) worker,
sys.filesys_number_from_rowid(rowid) ssd,
sys.inode_number_from_rowid(rowid) shard,
sys.row_number_from_rowid(rowid) rownum
from matchstats order by 1 limit 10;
 worker | ssd | shard  | rownum
--------+-----+--------+--------
      0 |   7 | 168961 | 338055
      0 |   7 | 168961 | 338056
      0 |   7 | 168961 | 338057
      0 |   7 | 168961 | 338058
      0 |   7 | 168961 | 338059
      0 |   7 | 168961 | 338060
      0 |   7 | 168961 | 338061
      0 |   7 | 168961 | 338062
      0 |   7 | 168961 | 338063
      0 |   7 | 168961 | 338064
(10 rows)
```

sys.worker_id_from_rowid

Return the compute node ID for either all rows returned by the query or a specific row.

```
sys.worker_id_from_rowid(rowid)
```

See System Columns in Tables.

This function accepts either the string `rowid` as its input or a specific `rowid` value. In most cases you will use the string `rowid` to return the specific row ID for each row in a set of rows. (If you specify an individual row ID as the input to the function, make sure it is valid; the system does not detect row IDs that are not valid or do not exist.)

To return a meaningful result, always use this function in a query with a table reference, as shown in the following examples.

For example, the row ID specified here is stored on compute node 12:

```
premdb=# select sys.worker_id_from_rowid(221804300291014656) from match limit 1;
 worker_id_from_rowid
----------------------
                   12
(1 row)

premdb=# select sys.worker_id_from_rowid(221804300291014656) from sys.const;
 worker_id_from_rowid
----------------------
                   12
(1 row)
```

sys.worker_id()

Return the logical ID of the compute node that runs this function.

```
sys.worker_id()
```

To return a meaningful result, always use this function in a query with a table reference, such as a reference to sys.const.

The following repeated queries return a different compute node ID, depending on which compute node runs the query in each case.

```
premdb=# select sys.worker_id() from sys.const;
 worker_id
-----------
         9
(1 row)

premdb=# select sys.worker_id() from sys.const;
 worker_id
-----------
         3
(1 row)

premdb=# select sys.worker_id() from sys.const;
 worker_id
-----------
         8
(1 row)
```

The query counts the number of rows by the compute node that *processed* the row (not the compute node where the rows are stored).

```
premdb=# select sys.worker_id(), count(*) from newmatchstats group by 1 order by 1;
 worker_id | count
-----------+--------
         0 |  68400
         1 |  34200
         4 |  41580
         5 |  75780
         6 |  34200
         7 |  34200
         8 | 102600
         9 | 102600
        10 | 144180
        12 |  68400
        13 |  68400
(11 rows)
```

sys.worker_uuid

Given a logical compute node ID, return the UUID of that compute node if it was involved in the query where the function is used. If the compute node was not involved, the function returns `NULL`.

```
sys.worker_uuid(worker_id)
```

To return a meaningful result, always use this function in a query with a table reference, such as a reference to sys.const.

For example, the first two queries return UUIDs, and the third returns `NULL`:

```
premdb=# select sys.worker_uuid(9) from sys.const;
           worker_uuid
--------------------------------------
 00000000-0000-0000-0000-38b8ebd00393
(1 row)

premdb=# select sys.worker_uuid(3) from sys.const;
           worker_uuid
--------------------------------------
 00000000-0000-0000-0000-38b8ebd002a3
(1 row)

premdb=# select sys.worker_uuid(16) from sys.const;
 worker_uuid
-------------

(1 row)
```

SYSTEM_CLUSTER()

Return the name of the system cluster for an instance.

```
SYSTEM_CLUSTER()
```

For example:

```
yellowbrick=# select system_cluster();
   system_cluster
--------------------
 eng-rc9-cluster-01
(1 row)
```

See also ALTER USER SET DEFAULT_CLUSTER and ALTER SYSTEM SET CLUSTER.

The system cluster is the default cluster for all system work, including automatic `ANALYZE` operations, flushing of rows from the row store, and "GC" operations. Only one cluster per instance can be the system cluster, and all system work runs against it.

When you create a cluster and specify `WITH (DEFAULT_CLUSTER)`, this syntax implies that it is the system cluster, at least initially. Subsequently, you can change the system cluster as follows:

```
alter system set system_cluster "cluster_name"
```

VERSION()

Return the version of the Yellowbrick software that is installed.

```
VERSION()
```

For example:

```
premdb=# select version();
                version
----------------------------------------
 Yellowbrick Database version 5.2.0-43109
(1 row)
```

yb_get_view_table_usage()

Return information about views and the tables they depend on.

```
yb_get_view_table_usage()
```

This function returns a six-column result set, with the name of the view, its schema, and its database, followed by each table that the view depends on, its schema, and its database.

For example, the view `join_all_tables` depends on five tables, and the view `vteam` depends only on the `team` table:

```
premdb=# select * from yb_get_view_table_usage() where view_database='premdb';
 view_database | view_schema |    view_name    | table_database | table_schema | table_name
---------------+-------------+-----------------+----------------+--------------+------------
 premdb        | public      | join_all_tables | premdb         | public       | season
 premdb        | public      | join_all_tables | premdb         | public       | team
 premdb        | public      | join_all_tables | premdb         | public       | hometeam
 premdb        | public      | join_all_tables | premdb         | public       | awayteam
 premdb        | public      | join_all_tables | premdb         | public       | match
 premdb        | public      | vteam           | premdb         | public       | team
(6 rows)
```

In this example, the function returns a view and a table that have the same name and database but different schemas:

```
premdb=# create view premdb.bobr.season as select * from premdb.public.season;
CREATE VIEW
premdb=# select * from yb_get_view_table_usage() where view_name=table_name;
 view_database | view_schema | view_name | table_database | table_schema | table_name
---------------+-------------+-----------+----------------+--------------+------------
 premdb        | bobr        | season    | premdb         | public       | season
(1 row)
```

Note: You cannot create views and tables that store the results of this function.

yb_terminate_session

Terminate the specified user session.

```
yb_terminate_session(session_id)
```

This function returns `t` (success), `f` (failure), or `NULL` (unknown session ID). You can get session IDs from the sys.session view. The session ID must be an `INT` or `BIGINT` value.

For example:

```
yellowbrick=# select * from sys.session where application_name='ybsql';
 session_id | application_name | client_hostname | client_ip_address | database_id | user_id |          start_time           | pro
------------+------------------+-----------------+-------------------+-------------+---------+-------------------------------+----
     18120 | ybsql            | [NULL]          | [NULL]            |        4400 |      10 | 2018-06-04 13:14:16.212301-07 |
     18126 | ybsql            | [NULL]          | [NULL]            |       16388 |   16399 | 2018-06-04 13:17:21.43458-07  |
(2 rows)

yellowbrick=# select yb_terminate_session(18126);
 yb_terminate_session
----------------------
 t
(1 row)
```

Note: If you try to terminate an internal system user session (or the current user session), the function returns `NULL` and does not terminate the session.

Type-Safe Casting Functions

In this section:

The following SQL functions are supported for compatibility with applications developed in IBM Netezza and other databases. These functions implement type-safe casting from `VARCHAR` strings to `DATE`, `TIMESTAMP`, and `DOUBLE` data types.

Note: These functions are disabled by default. To enable them system-wide, set the `enable_safe_compat_functions_with_nz` configuration parameter.

SAFE_TO_DATE

Platforms: All platforms

Parent topic: Type-Safe Casting Functions

Return a `DATE` data type from a `VARCHAR` string that represents a date in `YYYYMMDD` format.

```
SAFE_TO_DATE(input_date varchar, format varchar)
```

The input `VARCHAR` string is the first argument, and its format is the second argument.

Note the specific requirements of this function:

- The specified format must be exactly `YYYYMMDD`. If you specify any other format, the function returns `NULL`.
- Each part of the date must be present in the input string (year, month, and day).
- The input string cannot contain delimiter characters between the date parts, such as `-` or `/` characters.
- The range of supported dates is from `0001-01-01` to `9999-12-31`. Dates that fall outside this range cannot be cast and return `NULL`.

For example:

```
yellowbrick=# select safe_to_date('19600309','YYYYMMDD');
 safe_to_date
--------------
 1960-03-09
(1 row)
```

The following example returns `NULL` because the date format is incorrect.

```
yellowbrick_test=# select safe_to_date('1960-03-09','MDY');
 safe_to_date
--------------
 [NULL]
(1 row)
```

The following example returns `NULL` because the input string contains delimiters between the date parts:

```
yellowbrick_test=# select safe_to_date('1960-03-09','YYYYMMDD');
 safe_to_date
--------------
 [NULL]
(1 row)
```

To enable this function, set the `enable_safe_compat_functions_with_nz` configuration parameter.

SAFE_TO_DOUBLE

Platforms: All platforms

Parent topic: Type-Safe Casting Functions

Return a `DOUBLE PRECISION` value from a `VARCHAR` string that represents a number.

```
SAFE_TO_DOUBLE(input_number varchar)
```

The input `VARCHAR` string is the only argument to this function. If the input is `NULL`, the function returns `NULL`. If any parsing or conversion errors occur, this function returns `0.0`.

For example:

```
premdb=# select safe_to_double('39.23789');
 safe_to_double
----------------
      39.23789
(1 row)

premdb=# select safe_to_double('1000000000000000');
 safe_to_double
----------------
         1e+15
(1 row)
```

To enable this function, set the `enable_safe_compat_functions_with_nz` configuration parameter.

SAFE_TO_TIMESTAMP

Return a `TIMESTAMP` value from a `VARCHAR` string that represents a timestamp.

```
SAFE_TO_TIMESTAMP(input_timestamp varchar, format varchar)
```

The input `VARCHAR` string is the first argument, and its format is the second argument.

Note the specific requirements of this function:

- The specified format must be exactly `YYYYMMDDHH24MISS`. If you specify any other format, the function returns `NULL`.
- Each part of the timestamp must be present in the input string, except for seconds, which may be omitted. The function assumes 0 seconds if seconds are omitted. If any other date part is missing, the function returns `NULL`.
- The input string cannot contain delimiter characters between the date and time parts, such as `-` or `:` characters.
- The range of supported dates is from `0001-01-01` to `9999-12-31`. Dates that fall outside this range cannot be cast and return `NULL`.
- The ranges of supported hours, minutes, and seconds are as follows. If values fall outside these ranges, the function returns `NULL`:
- Hours: `0-23`
- Minutes: `0-59`
- Seconds: `0-59`

For example:

```
premdb=# select safe_to_timestamp('19991231235959', 'YYYYMMDDHH24MISS');
  safe_to_timestamp
---------------------
 1999-12-31 23:59:59
(1 row)
```

In this example, the input string does not contain any seconds, so the function returns `00` for the seconds:

```
premdb=# select safe_to_timestamp('202301012359', 'YYYYMMDDHH24MISS');
  safe_to_timestamp
---------------------
 2023-01-01 23:59:00
(1 row)
```

The following example returns `NULL` because the timestamp format is incorrect:

```
premdb=# select safe_to_timestamp('202301012359', 'YYYYMMDDHH24');
 safe_to_timestamp
-------------------
 [NULL]
(1 row)
```

The following example returns `NULL` because the input string contains delimiters between some of the date parts:

```
premdb=# select safe_to_timestamp('202301012359', 'YYYYMMDDHH24:MI:SS');
 safe_to_timestamp
-------------------
 [NULL]
(1 row)
```

The following example returns `NULL` because the input string contains an invalid hours value (`24`):

```
premdb=# select safe_to_timestamp('202301012401', 'YYYYMMDDHH24MISS');
 safe_to_timestamp
-------------------
 [NULL]
(1 row)
```

To enable this function, set the `enable_safe_compat_functions_with_nz` configuration parameter.

Window Functions

Yellowbrick Documentation > Reference > SQL Reference > SQL Functions and Operators > Window Functions

Platforms: All platforms

Parent topic: SQL Functions and Operators

In this section:

AVG (window function)
COUNT (window function)
CUME_DIST
DENSE_RANK
FIRST_VALUE, LAST_VALUE
LEAD, LAG
MAX (window function)
MEDIAN (window function)
MIN (window function)
NTILE
PERCENT_RANK
PERCENTILE_CONT (window function)
PERCENTILE_DISC (window function)
RANK
ROW_NUMBER
STDDEV_SAMP and STDDEV_POP (window functions)
SUM (window function)
Syntax for Window Functions
VAR_SAMP and VAR_POP (window functions)

Window functions are specialized SQL analytic functions that come in several forms: *windowed aggregates* and *ranking functions* are the most common forms.

A windowed aggregate is similar to a standard aggregate function, but is more flexible. Windowed aggregates can produce result sets over a *window frame*. A window frame defines a moving target for the evaluation of the function. For example, you can compute moving averages and sums over a set of ordered rows. You can also reset these calculations within a result set by defining *window partitions*.

Yellowbrick supports the following windowed aggregates:

- SUM
- COUNT
- AVG
- MIN
- MAX

COUNT(DISTINCT) is not supported as a windowed aggregate. The other aggregate functions in this list are supported as both regular aggregate functions and aggregate window functions.

The following ranking and row numbering window functions operate on ordered and partitioned sets of rows.

- CUME_DIST
- RANK
- DENSE_RANK
- PERCENT_RANK
- ROW_NUMBER

- NTILE

Note: The ORDER BY clause is not required as part of the window definition for the ranking window functions, but the absence of this clause is likely to produce meaningless results.

Yellowbrick also supports two other types of window functions:

- LAG and LEAD functions
- FIRST_VALUE and LAST_VALUE functions

Window frames with RANGE and ROWS are supported.

Window function definitions with a FILTER clause are not supported.

You can use the WINDOW clause in the SELECT statement as a shortcut to define the same behavior for multiple window functions in the same query.

AVG (window function)

Compute the average for an expression based on a window definition that contains an optional frame clause.

```
AVG | AVERAGE ([ expression ]) OVER { window_name | ( [ window_definition ] ) }
```

`AVG` and `AVERAGE` are synonyms for the same function. For the full `window_definition` syntax, see Syntax for Window Functions.

For example:

```
premdb=# select name, city, capacity, avg(capacity)
over(partition by city order by name)::dec(6,1) as avg_cap
from team where city in('London','Birmingham','Liverpool') order by 2,1;
        name         |    city    | capacity | avg_cap
---------------------+------------+----------+---------
 Aston Villa         | Birmingham |    42785 | 42785.0
 Birmingham City     | Birmingham |    30016 | 36400.5
 West Bromwich Albion | Birmingham |    27000 | 33267.0
 Everton             | Liverpool  |    40221 | 40221.0
 Liverpool           | Liverpool  |    44742 | 42481.5
 Arsenal             | London     |    60260 | 60260.0
 Charlton Athletic   | London     |    27111 | 43685.5
 Chelsea             | London     |    41663 | 43011.3
 Crystal Palace      | London     |    26255 | 38822.3
 Fulham              | London     |    25700 | 36197.8
 Queens Park Rangers | London     |    18439 | 33238.0
 Tottenham Hotspur   | London     |    36284 | 33673.1
 West Ham United     | London     |    35016 | 33841.0
 Wimbledon           | London     |    26255 | 32998.1
(14 rows)
```

Add an explicit window frame to this example:

```
premdb=# select name, city, capacity, avg(capacity)
over(partition by city order by name rows between 2 preceding and 2 following)::dec(6,1) as avg_cap
from team where city in('London','Birmingham','Liverpool') order by 2,1;
        name         |    city    | capacity | avg_cap
---------------------+------------+----------+---------
 Aston Villa         | Birmingham |    42785 | 33267.0
 Birmingham City     | Birmingham |    30016 | 33267.0
 West Bromwich Albion | Birmingham |    27000 | 33267.0
 Everton             | Liverpool  |    40221 | 42481.5
 Liverpool           | Liverpool  |    44742 | 42481.5
 Arsenal             | London     |    60260 | 43011.3
 Charlton Athletic   | London     |    27111 | 38822.3
 Chelsea             | London     |    41663 | 36197.8
 Crystal Palace      | London     |    26255 | 27833.6
 Fulham              | London     |    25700 | 29668.2
 Queens Park Rangers | London     |    18439 | 28338.8
 Tottenham Hotspur   | London     |    36284 | 28338.8
 West Ham United     | London     |    35016 | 28998.5
```

```
 Wimbledon           | London     |    26255 | 32518.3
(14 rows)
```

COUNT (window function)

Compute the count over an expression, based on a window definition and an optional frame clause. Three variations are supported:

- `COUNT(*)` (all rows)
- `COUNT([ALL] expression)` (all non-NULL values, including duplicates)
- `COUNT(DISTINCT expression)` (all distinct non-NULL values)

```
COUNT ( * | [ DISTINCT | [ ALL ] ] expression ]) OVER { window_name | ( [ window_definition ] ) }
```

`ALL` is an optional keyword; `ALL` is implied when `DISTINCT` is not used. For the full `window_definition` syntax, see Syntax for Window Functions.

Note: `COUNT(DISTINCT)` window functions cannot have an `ORDER BY` clause in the window definition or a non-default window frame.

For example:

```
premdb=# select name, city, capacity, count(capacity)
over(partition by city order by name) as count_cap
from team where city in('London','Birmingham','Liverpool') order by 2,1;
        name         |    city    | capacity | count_cap
---------------------+------------+----------+-----------
 Aston Villa         | Birmingham |    42785 |         1
 Birmingham City     | Birmingham |    30016 |         2
 West Bromwich Albion | Birmingham |    27000 |         3
 Everton             | Liverpool  |    40221 |         1
 Liverpool           | Liverpool  |    44742 |         2
 Arsenal             | London     |    60260 |         1
 Charlton Athletic   | London     |    27111 |         2
 Chelsea             | London     |    41663 |         3
 Crystal Palace      | London     |    26255 |         4
 Fulham              | London     |    25700 |         5
 Queens Park Rangers | London     |    18439 |         6
 Tottenham Hotspur   | London     |    36284 |         7
 West Ham United     | London     |    35016 |         8
 Wimbledon           | London     |    26255 |         9
(14 rows)
```

Add an explicit window frame to this example:

```
premdb=# select name, city, capacity, count(capacity)
over(partition by city order by name rows between 2 preceding and 2 following) as count_cap
from team where city in('London','Birmingham','Liverpool') order by 2,1;
        name         |    city    | capacity | count_cap
---------------------+------------+----------+-----------
 Aston Villa         | Birmingham |    42785 |         3
 Birmingham City     | Birmingham |    30016 |         3
 West Bromwich Albion | Birmingham |    27000 |         3
 Everton             | Liverpool  |    40221 |         2
 Liverpool           | Liverpool  |    44742 |         2
 Arsenal             | London     |    60260 |         3
```

```
 Charlton Athletic    | London     |    27111 |         4
 Chelsea              | London     |    41663 |         5
 Crystal Palace       | London     |    26255 |         5
 Fulham               | London     |    25700 |         5
 Queens Park Rangers  | London     |    18439 |         5
 Tottenham Hotspur    | London     |    36284 |         5
 West Ham United      | London     |    35016 |         4
 Wimbledon            | London     |    26255 |         3
(14 rows)
```

The following example uses `COUNT(DISTINCT expression)`. The query counts the number of distinct full-time scores that start with `7` for rows partitioned by half-time scores.

```
premdb=# select htscore, ftscore, count(distinct ftscore) over(partition by htscore) from match where ftscore like '7%' and htscor
 htscore | ftscore | count
---------+---------+-------
 1-1     | 7-3     |     1
 2-0     | 7-0     |     1
 2-0     | 7-1     |     2
 2-1     | 7-1     |     1
 2-1     | 7-4     |     2
 3-0     | 7-0     |     2
 3-0     | 7-0     |     2
 3-0     | 7-1     |     4
 3-0     | 7-1     |     4
 3-1     | 7-1     |     4
 3-1     | 7-1     |     4
 3-1     | 7-1     |     4
 3-1     | 7-1     |     4
 4-0     | 7-0     |     3
 4-0     | 7-0     |     3
 4-0     | 7-0     |     3
 4-0     | 7-2     |     4
 4-2     | 7-2     |     1
 5-1     | 7-1     |     1
 5-1     | 7-2     |     2
(20 rows)
```

CUME_DIST

Calculate the cumulative distribution of each value in an ordered set.

```
CUME_DIST() OVER { window_name | ( [ window_definition ] ) }
```

This function returns values in the range of 0 to 1. Tie values always evaluate to the same cumulative distribution value. The `ORDER BY` clause is not required; however, if it is omitted, the return value is `1` for all rows. The frame clause, if specified, is ignored. For the full `window_definition` syntax, see Syntax for Window Functions.

Examples

Return the cumulative distribution for teams based on an ascending order of average attendance `avg_att` values.

```
premdb=# select name, avg_att,
cume_dist() over(order by avg_att)::dec(3,2)
from team
where avg_att>0
order by avg_att;
        name         | avg_att | cume_dist
---------------------+---------+-----------
 Bournemouth         |  11.189 |      0.05
 Watford             |  20.594 |      0.10
 Swansea City        |  20.711 |      0.15
 West Bromwich Albion |  24.631 |      0.20
 Crystal Palace      |  24.636 |      0.25
 Norwich City        |  26.972 |      0.30
 Stoke City          |  27.534 |      0.35
 Southampton         |  30.751 |      0.40
 Leicester City      |  32.201 |      0.45
 Aston Villa         |  33.690 |      0.50
 West Ham United     |  34.910 |      0.55
 Tottenham Hotspur   |  35.776 |      0.60
 Everton             |  38.124 |      0.65
 Chelsea             |  41.500 |      0.70
 Sunderland          |  43.071 |      0.75
 Liverpool           |  43.910 |      0.80
 Newcastle United    |  49.754 |      0.85
 Manchester City     |  54.041 |      0.90
 Arsenal             |  59.944 |      0.95
 Manchester United   |  75.286 |      1.00
(20 rows)
```

Return the cumulative distribution for teams in each of three cities, based on an ascending order of `capacity` values.

```
premdb=# select name, city, capacity,
cume_dist() over(partition by city order by capacity)::dec(3,2)
from team
where city in('London','Birmingham','Manchester')
order by city;
```

```
         name         |    city    | capacity | cume_dist
----------------------+------------+----------+-----------
 West Bromwich Albion | Birmingham |    27000 |      0.33
 Birmingham City      | Birmingham |    30016 |      0.67
 Aston Villa          | Birmingham |    42785 |      1.00
 Queens Park Rangers  | London     |    18439 |      0.11
 Fulham               | London     |    25700 |      0.22
 Crystal Palace       | London     |    26255 |      0.44
 Wimbledon            | London     |    26255 |      0.44
 Charlton Athletic    | London     |    27111 |      0.56
 West Ham United      | London     |    35016 |      0.67
 Tottenham Hotspur    | London     |    36284 |      0.78
 Chelsea              | London     |    41663 |      0.89
 Arsenal              | London     |    60260 |      1.00
 Manchester City      | Manchester |    55097 |      0.50
 Manchester United    | Manchester |    75635 |      1.00
(14 rows)
```

DENSE_RANK

Rank a result set based on an ordered set of rows. Unlike RANK, do not skip any ranking values when ties occur in the result.

```
DENSE_RANK() OVER { window_name | ( window_definition ) }
```

The `ORDER BY` clause in the `window_definition` is required. The frame clause, if specified, is ignored. For the full `window_definition` syntax, see Syntax for Window Functions.

Compare the results of this example with the equivalent example for the RANK function.

```
premdb=# SELECT season_name, h.name, a.name, ftscore,
DENSE_RANK() OVER(ORDER BY ftscore DESC)
FROM season s, hometeam h, awayteam a, match m
WHERE s.seasonid=m.seasonid AND h.htid=m.htid AND a.atid=m.atid
ORDER BY 5,1 LIMIT 25;
 season_name |       name        |        name         | ftscore | dense_rank
-------------+-------------------+---------------------+---------+------------
 2009-2010   | Tottenham Hotspur | Wigan Athletic      | 9-1     |          1
 1994-1995   | Manchester United | Ipswich Town        | 9-0     |          2
 2011-2012   | Manchester United | Arsenal             | 8-2     |          3
 2007-2008   | Middlesbrough     | Manchester City     | 8-1     |          4
 1999-2000   | Newcastle United  | Sheffield Wednesday | 8-0     |          5
 2009-2010   | Chelsea           | Wigan Athletic      | 8-0     |          5
 2012-2013   | Chelsea           | Aston Villa         | 8-0     |          5
 2007-2008   | Portsmouth        | Reading             | 7-4     |          6
 2012-2013   | Arsenal           | Newcastle United    | 7-3     |          7
 1997-1998   | Blackburn Rovers  | Sheffield Wednesday | 7-2     |          8
 1999-2000   | Tottenham Hotspur | Southampton         | 7-2     |          8
 2009-2010   | Chelsea           | Sunderland          | 7-2     |          8
 1992-1993   | Blackburn Rovers  | Norwich City        | 7-1     |          9
 1993-1994   | Newcastle United  | Swindon Town        | 7-1     |          9
 1994-1995   | Aston Villa       | Wimbledon           | 7-1     |          9
 1996-1997   | Everton           | Southampton         | 7-1     |          9
 1996-1997   | Newcastle United  | Tottenham Hotspur   | 7-1     |          9
 1998-1999   | Liverpool         | Southampton         | 7-1     |          9
 1999-2000   | Manchester United | West Ham United     | 7-1     |          9
 2001-2002   | Blackburn Rovers  | West Ham United     | 7-1     |          9
 2007-2008   | Everton           | Sunderland          | 7-1     |          9
 2009-2010   | Chelsea           | Aston Villa         | 7-1     |          9
 2010-2011   | Manchester United | Blackburn Rovers    | 7-1     |          9
 2011-2012   | Arsenal           | Blackburn Rovers    | 7-1     |          9
 1995-1996   | Blackburn Rovers  | Nottingham Forest   | 7-0     |         10
(25 rows)
```

FIRST_VALUE, LAST_VALUE

Return the first or last value associated with a window frame, given an ordered set of rows.

```
FIRST_VALUE | LAST_VALUE ( expression [ RESPECT NULLS | IGNORE NULLS ] )
OVER { window_name | ( [ window_definition ] ) }
```

RESPECT NULLS | IGNORE NULLS

When `NULL` values appear in `expression`, respect or ignore them as potential first and last values. `RESPECT NULLS` is the default.

If `IGNORE NULLS` is used, return the first value or last value in the window frame that is not `NULL`. If `RESPECT NULLS` is used, return the first value or last value whether it is `NULL` or not. If all values in the frame are `NULL`, `NULL` always qualifies as the first or last value.

window_definition

See Syntax for Window Functions. The `ORDER BY` clause is not required but is recommended for determining "first" and "last" values based on an ordered set of rows.

Examples

In this example, the "first value" in the frame is `1` for `London`, `2` for `Birmingham`, and so on (the `teamid` for an ordered list of cities).

```
premdb=# select teamid, city,
first_value(teamid) over(partition by city order by city)
from team order by 3;
 teamid |     city      | first_value
--------+---------------+-------------
      1 | London        |           1
     12 | London        |           1
     13 | London        |           1
     15 | London        |           1
     18 | London        |           1
     32 | London        |           1
     41 | London        |           1
     44 | London        |           1
     46 | London        |           1
      2 | Birmingham    |           2
      4 | Birmingham    |           2
     43 | Birmingham    |           2
      3 | Barnsley      |           3
      5 | Blackburn     |           5
      6 | Blackpool     |           6
...
```

The following two queries demonstrate the use of the `IGNORE NULLS` option. The `FIRST_VALUE` function operates on the `capacity` column, which contains some `NULL` values. The first query ignores `NULL` values in the `capacity` column when they are the first value in the window frame (defined as `1 preceding and 1 following`). The only time `NULL` is returned for the `FIRST_VALUE` function in this case is when the frame consists of three consecutive rows with `NULL` capacity values.

```
premdb=# select teamid, name, city, stadium, capacity,
first_value(capacity ignore nulls) over(partition by city order by teamid rows between 1 preceding and 1 following)
from team where city in ('London','Birmingham','Manchester');
 teamid |         name         |    city    |     stadium      | capacity | first_value
--------+----------------------+------------+------------------+----------+-------------
      2 | Aston Villa          | Birmingham | Villa Park       |    42785 |       42785
      4 | Birmingham City      | Birmingham | St. Andrew's     |    30016 |       42785
     43 | West Bromwich Albion | Birmingham | The Hawthorns    |    27000 |       30016
      1 | Arsenal              | London     | Emirates Stadium |    60260 |       60260
     12 | Charlton Athletic    | London     | The Valley       |    27111 |       60260
     13 | Chelsea              | London     | Stamford Bridge  |    41663 |       27111
     15 | Crystal Palace       | London     | Selhurst Park    |   [NULL] |       41663
     18 | Fulham               | London     | Craven Cottage   |   [NULL] |      [NULL]
     32 | Queens Park Rangers  | London     | Loftus Road      |   [NULL] |       36284
     41 | Tottenham Hotspur    | London     | White Hart Lane  |    36284 |       36284
     44 | West Ham United      | London     | Upton Park       |    35016 |       36284
     46 | Wimbledon            | London     | Selhurst Park    |   [NULL] |       35016
     24 | Manchester City      | Manchester | Etihad Stadium   |    55097 |       55097
     25 | Manchester United    | Manchester | Old Trafford     |    75635 |       55097
(14 rows)
```

The second query uses `RESPECT NULLS` (which is equivalent to the default behavior and could be omitted from the syntax). Everything else in the query is the same as the first query. Note that the `FIRST_VALUE` results are different. Because `NULL` values are respected, three rows return `NULL` for the `FIRST_VALUE` function.

```
premdb=# select teamid, name, city, stadium, capacity,
first_value(capacity respect nulls) over(partition by city order by teamid rows between 1 preceding and 1 following)
from team where city in ('London','Birmingham','Manchester');
 teamid |         name         |    city    |     stadium      | capacity | first_value
--------+----------------------+------------+------------------+----------+-------------
      2 | Aston Villa          | Birmingham | Villa Park       |    42785 |       42785
      4 | Birmingham City      | Birmingham | St. Andrew's     |    30016 |       42785
     43 | West Bromwich Albion | Birmingham | The Hawthorns    |    27000 |       30016
      1 | Arsenal              | London     | Emirates Stadium |    60260 |       60260
     12 | Charlton Athletic    | London     | The Valley       |    27111 |       60260
     13 | Chelsea              | London     | Stamford Bridge  |    41663 |       27111
     15 | Crystal Palace       | London     | Selhurst Park    |   [NULL] |       41663
     18 | Fulham               | London     | Craven Cottage   |   [NULL] |      [NULL]
     32 | Queens Park Rangers  | London     | Loftus Road      |   [NULL] |      [NULL]
     41 | Tottenham Hotspur    | London     | White Hart Lane  |    36284 |      [NULL]
     44 | West Ham United      | London     | Upton Park       |    35016 |       36284
     46 | Wimbledon            | London     | Selhurst Park    |   [NULL] |       35016
     24 | Manchester City      | Manchester | Etihad Stadium   |    55097 |       55097
     25 | Manchester United    | Manchester | Old Trafford     |    75635 |       55097
(14 rows)
```

LEAD, LAG

Return values for a row at a given position before or after the current row in the partition (*before* the current row=LAG, *after* the current row=LEAD). The position is defined as an *offset* value.

```
{ LEAD | LAG } (expression [, offset [, default] ])
OVER { window_name | ( [ window_definition ] ) }
```

Parameters

offset

The `offset` is an optional integer value or numeric expression. The default `offset` is `1`.

default

If `offset` is specified, a `default` value may also be specified, and this value must have the same data type as the first argument to the function.

This `default` provides a value when the function does not return a LEAD or LAG value for the specified `offset` (because no such row exists with respect to the current row). If `default` is not specified, the function returns `null`.

window_definition

For the full `window_definition` syntax, see Syntax for Window Functions. The `ORDER BY` clause is not required for this function. The frame clause, if specified, is ignored.

Examples

For example, use the LEAD function with an `offset` of 1 row:

```
premdb=# select name, city, capacity,
lead(capacity,1) over(partition by city order by name) as lead_cap
from team where city='London' order by 2,1;
       name         |  city  | capacity | lead_cap
--------------------+--------+----------+----------
 Arsenal            | London |    60260 |    27111
 Charlton Athletic  | London |    27111 |    41663
 Chelsea            | London |    41663 |    26255
 Crystal Palace     | London |    26255 |    25700
 Fulham             | London |    25700 |    18439
 Queens Park Rangers | London |    18439 |    36284
 Tottenham Hotspur  | London |    36284 |    35016
 West Ham United    | London |    35016 |    26255
 Wimbledon          | London |    26255 |
(9 rows)
```

Here is the same query with the LAG function instead of LEAD:

```
premdb=# select name, city, capacity,
lag(capacity,1) over(partition by city order by name) as lag_cap
from team where city='London' order by 2,1;
       name         |  city  | capacity | lag_cap
--------------------+--------+----------+---------
 Arsenal            | London |    60260 |
 Charlton Athletic  | London |    27111 |   60260
 Chelsea            | London |    41663 |   27111
 Crystal Palace     | London |    26255 |   41663
 Fulham             | London |    25700 |   26255
 Queens Park Rangers | London |    18439 |   25700
 Tottenham Hotspur  | London |    36284 |   18439
 West Ham United    | London |    35016 |   36284
 Wimbledon          | London |    26255 |   35016
(9 rows)
```

In the following example, there is a lag of 5 rows. A `default` lag value of `0` is supplied for rows that would otherwise return `null`:

```
premdb=# select name, city, capacity,
lag(capacity,5,0) over(partition by city order by name) as lag_cap
from team where city='London' order by 2,1;
       name         |  city  | capacity | lag_cap
--------------------+--------+----------+---------
 Arsenal            | London |    60260 |       0
 Charlton Athletic  | London |    27111 |       0
 Chelsea            | London |    41663 |       0
 Crystal Palace     | London |    26255 |       0
 Fulham             | London |    25700 |       0
 Queens Park Rangers | London |    18439 |   60260
 Tottenham Hotspur  | London |    36284 |   27111
 West Ham United    | London |    35016 |   41663
 Wimbledon          | London |    26255 |   26255
(9 rows)
```

MAX (window function)

Compute the maximum value for an expression based on a window definition and an optional frame clause.

```
MAX ([ expression ]) OVER { window_name | ( [ window_definition ] ) }
```

For the full `window_definition` syntax, see Syntax for Window Functions.

Note: The input `expression` for a MAX function cannot be a CHAR, VARCHAR, or BOOLEAN data type.

Example

For example, return the maximum `seasonid` from the `season` table and partition the results by the `winners` column. (In effect, this query shows when each winning team last won the league, alongside a list of all the other seasons when they were the winners.)

```
premdb=# select season_name, winners,
max(seasonid) over(partition by winners)
from season
where winners is not null
order by max desc;
 season_name |      winners      | max
-------------+-------------------+-----
 2015-2016   | Leicester City    |  24
 2004-2005   | Chelsea           |  23
 2005-2006   | Chelsea           |  23
 2009-2010   | Chelsea           |  23
 2014-2015   | Chelsea           |  23
 2011-2012   | Manchester City   |  22
 2013-2014   | Manchester City   |  22
 1992-1993   | Manchester United |  21
 1993-1994   | Manchester United |  21
 1995-1996   | Manchester United |  21
 1996-1997   | Manchester United |  21
 1998-1999   | Manchester United |  21
 1999-2000   | Manchester United |  21
 2000-2001   | Manchester United |  21
 2002-2003   | Manchester United |  21
 2006-2007   | Manchester United |  21
 2007-2008   | Manchester United |  21
 2008-2009   | Manchester United |  21
 2010-2011   | Manchester United |  21
 2012-2013   | Manchester United |  21
 1997-1998   | Arsenal           |  12
 2001-2002   | Arsenal           |  12
 2003-2004   | Arsenal           |  12
 1994-1995   | Blackburn Rovers  |   3
(24 rows)
```

MEDIAN (window function)

Return the median value from an ordered set, equivalent to `percentile_cont(0.5)`.

```
MEDIAN() WITHIN GROUP (ORDER BY sort_expression) OVER { window_name | ( [ window_definition ] ) }
```

The sort expression data type must be numeric.

The `window_definition` for this function can have a `PARTITION BY` clause but no `ORDER BY` clause or frame clause. For the full `window_definition` syntax, see Syntax for Window Functions.

For example:

```
premdb=# select city, median() within group (order by avg_att) over(partition by city)
from team where avg_att>0 order by 1;
   city     | percentile_cont
------------+-----------------
 Birmingham |         29.1605
 Birmingham |         29.1605
 Bournemouth |          11.189
 Leicester  |          32.201
 Liverpool  |          41.017
 Liverpool  |          41.017
 London     |          35.776
 London     |          35.776
 London     |          35.776
 London     |          35.776
 London     |          35.776
 Manchester |         64.6635
 Manchester |         64.6635
 Newcastle  |          49.754
 Norwich    |          26.972
 Southampton |          30.751
 Stoke      |          27.534
 Sunderland |          43.071
 Swansea    |          20.711
 Watford    |          20.594
(20 rows)
```

MIN (window function)

Compute the minimum value for an expression based on a window definition and an optional frame clause.

```
MIN ([ expression ]) OVER { window_name | ( [ window_definition ] ) }
```

For the full `window_definition` syntax, see Syntax for Window Functions.

Note: The input `expression` for a MIN function cannot be a CHAR, VARCHAR, or BOOLEAN data type.

For example, return the minimum `capacity` from the `team` table and partition the results by the `city` column:

```
premdb=# select city, capacity,
min(capacity) over(partition by city order by capacity)
from team
where city in('London','Birmingham','Liverpool')
order by city;
   city    | capacity |  min
-----------+----------+-------
 Birmingham |    27000 | 27000
 Birmingham |    30016 | 27000
 Birmingham |    42785 | 27000
 Liverpool  |    40221 | 40221
 Liverpool  |    44742 | 40221
 London     |    18439 | 18439
 London     |    25700 | 18439
 London     |    26255 | 18439
 London     |    26255 | 18439
 London     |    27111 | 18439
 London     |    35016 | 18439
 London     |    36284 | 18439
 London     |    41663 | 18439
 London     |    60260 | 18439
(14 rows)
```

Add an explicit window frame to this example:

```
premdb=# select city, capacity, min(capacity)
over(partition by city order by capacity rows between 2 preceding and 2 following)
from team where city in('London','Birmingham','Liverpool') order by city;
   city    | capacity |  min
-----------+----------+-------
 Birmingham |    27000 | 27000
 Birmingham |    30016 | 27000
 Birmingham |    42785 | 27000
 Liverpool  |    40221 | 40221
 Liverpool  |    44742 | 40221
 London     |    18439 | 18439
 London     |    25700 | 18439
 London     |    26255 | 18439
 London     |    26255 | 25700
```

```
 London     |    27111 | 26255
 London     |    35016 | 26255
 London     |    36284 | 27111
 London     |    41663 | 35016
 London     |    60260 | 36284
(14 rows)
```

NTILE

Rank a result set into a specific number of groups ("tiles"), based on an ordered set of rows. For example, `NTILE(100)` ranks results into percentiles.

```
NTILE(expression) OVER { window_name | ( window_definition ) }
```

The `expression` argument must be a positive whole number or an expression that evaluates to a whole number, such as 3, 10, or 50. Decimal values and negative values are not allowed. This expression may not contain window functions.

The `ORDER BY` clause in the `window_definition` is technically optional but highly recommended for all `NTILE` queries. Meaningful ranking values depend on an ordered result set. The frame clause, if specified, is ignored.

For the full `window_definition` syntax, see Syntax for Window Functions.

Usage Notes

Note the following expected behavior with this function:

- `NTILE` ranks rows with an even distribution across the groups. The number of rows that fall into each group may differ by no more than 1 row. Remainder rows are placed in each group, starting with group 1. For example, an `NTILE(3)` function over a partition of 10 rows will place 4 rows in group 1 and 3 rows in groups 2 and 3.
- If the `expression` for the function is greater than the number of rows in the partition, only the first `n` ranking values are used, starting with `1`. For example, `NTILE(100)` over a partition of 66 rows assigns rankings from `1` to `66` only.
- If the `expression` for the function is not a constant, the first value that the expression returns defines the number of groups. For example: `ntile(seasonid*10)`, where `seasonid` is a column name, computes 20 groups if the first value found in the column is `2`.

Examples

The following example ranks teams into three groups (numbered 1, 2, and 3), based on stadium capacity:

```
premdb=# select name, city, stadium, capacity,
ntile(3) over(order by capacity desc) ranking
from team
where capacity>0 and city in('London','Manchester','Birmingham')
order by ranking;
        name         |    city    |     stadium      | capacity | ranking
---------------------+------------+------------------+----------+---------
 Manchester United   | Manchester | Old Trafford     |    75635 |       1
 Arsenal             | London     | Emirates Stadium |    60260 |       1
 Manchester City     | Manchester | Etihad Stadium   |    55097 |       1
 Aston Villa         | Birmingham | Villa Park       |    42785 |       1
 Chelsea             | London     | Stamford Bridge  |    41663 |       1
 Tottenham Hotspur   | London     | White Hart Lane  |    36284 |       2
 West Ham United     | London     | Upton Park       |    35016 |       2
 Birmingham City     | Birmingham | St. Andrew's     |    30016 |       2
 Charlton Athletic   | London     | The Valley       |    27111 |       2
 West Bromwich Albion | Birmingham | The Hawthorns    |    27000 |       2
 Crystal Palace      | London     | Selhurst Park    |    26255 |       3
```

```
 Wimbledon           | London     | Selhurst Park    |    26255 |       3
 Fulham              | London     | Craven Cottage   |    25700 |       3
 Queens Park Rangers | London     | Loftus Road      |    18439 |       3
(14 rows)
```

The following query partitions the `NTILE` results by the `city` column:

```
premdb=# select name, city, stadium, capacity,
ntile(3) over(partition by city order by capacity desc) ranking
from team
where capacity>0 and city in('London','Manchester','Birmingham')
order by city;
        name         |    city    |     stadium      | capacity | ranking
---------------------+------------+------------------+----------+---------
 Aston Villa         | Birmingham | Villa Park       |    42785 |       1
 Birmingham City     | Birmingham | St. Andrew's     |    30016 |       2
 West Bromwich Albion | Birmingham | The Hawthorns    |    27000 |       3
 Arsenal             | London     | Emirates Stadium |    60260 |       1
 Chelsea             | London     | Stamford Bridge  |    41663 |       1
 Tottenham Hotspur   | London     | White Hart Lane  |    36284 |       1
 West Ham United     | London     | Upton Park       |    35016 |       2
 Charlton Athletic   | London     | The Valley       |    27111 |       2
 Crystal Palace      | London     | Selhurst Park    |    26255 |       2
 Wimbledon           | London     | Selhurst Park    |    26255 |       3
 Fulham              | London     | Craven Cottage   |    25700 |       3
 Queens Park Rangers | London     | Loftus Road      |    18439 |       3
 Manchester United   | Manchester | Old Trafford     |    75635 |       1
 Manchester City     | Manchester | Etihad Stadium   |    55097 |       2
(14 rows)
```

The following example clearly shows the expected behavior when the result set is not evenly divisible by the `NTILE` expression, which is `10` in this case. Because the query returns 14 rows, the first four rankings (1-4) are assigned twice each, then the last six appear once each (5-10). Remainder rows are allocated evenly over the groups, starting from 1, until they are exhausted. Note that duplicate values in the ORDER BY column (`capacity` in this case) may be assigned different rankings.

```
premdb=# select name, city, stadium, capacity,
ntile(10) over(order by capacity desc) ranking
from team
where capacity>0 and city in('London','Manchester','Birmingham')
order by ranking;
        name         |    city    |     stadium      | capacity | ranking
---------------------+------------+------------------+----------+---------
 Manchester United   | Manchester | Old Trafford     |    75635 |       1
 Arsenal             | London     | Emirates Stadium |    60260 |       1
 Manchester City     | Manchester | Etihad Stadium   |    55097 |       2
 Aston Villa         | Birmingham | Villa Park       |    42785 |       2
 Chelsea             | London     | Stamford Bridge  |    41663 |       3
 Tottenham Hotspur   | London     | White Hart Lane  |    36284 |       3
 West Ham United     | London     | Upton Park       |    35016 |       4
 Birmingham City     | Birmingham | St. Andrew's     |    30016 |       4
 Charlton Athletic   | London     | The Valley       |    27111 |       5
 West Bromwich Albion | Birmingham | The Hawthorns    |    27000 |       6
 Crystal Palace      | London     | Selhurst Park    |    26255 |       7
 Wimbledon           | London     | Selhurst Park    |    26255 |       8
 Fulham              | London     | Craven Cottage   |    25700 |       9
 Queens Park Rangers | London     | Loftus Road      |    18439 |      10
(14 rows)
```

PERCENT_RANK

Calculate the percent ranking of each value in an ordered set.

```
PERCENT_RANK() OVER { window_name | ( [ window_definition ] ) }
```

The following formula is used to calculate percent rankings for each partition:

```
(rank - 1) / (count - 1)
```

This function returns values in the range of 0 to 1. The first row in any set returns a ranking of 0. The `ORDER BY` clause is not required; however, if it is omitted, the ranking is `0` for all rows. The frame clause, if specified, is ignored. For the full `window_definition` syntax, see Syntax for Window Functions.

Examples

Return the percent rankings for teams based on an ascending order of average attendance `avg_att` values.

```
premdb=# select name, avg_att,
percent_rank() over(order by avg_att)::dec(3,2)
from team
where avg_att>0
order by avg_att;
        name         | avg_att | percent_rank
---------------------+---------+--------------
 Bournemouth         |  11.189 |         0.00
 Watford             |  20.594 |         0.05
 Swansea City        |  20.711 |         0.11
 West Bromwich Albion |  24.631 |         0.16
 Crystal Palace      |  24.636 |         0.21
 Norwich City        |  26.972 |         0.26
 Stoke City          |  27.534 |         0.32
 Southampton         |  30.751 |         0.37
 Leicester City      |  32.201 |         0.42
 Aston Villa         |  33.690 |         0.47
 West Ham United     |  34.910 |         0.53
 Tottenham Hotspur   |  35.776 |         0.58
 Everton             |  38.124 |         0.63
 Chelsea             |  41.500 |         0.68
 Sunderland          |  43.071 |         0.74
 Liverpool           |  43.910 |         0.79
 Newcastle United    |  49.754 |         0.84
 Manchester City     |  54.041 |         0.89
 Arsenal             |  59.944 |         0.95
 Manchester United   |  75.286 |         1.00
(20 rows)
```

Return percent rankings for teams in each of three cities, based on an ascending order of `capacity` values.

```
premdb=# select name, city, capacity,
percent_rank() over(partition by city order by capacity)::dec(3,2)
from team
where city in('London','Birmingham','Manchester')
order by city;
        name         |    city    | capacity | percent_rank
---------------------+------------+----------+--------------
 West Bromwich Albion | Birmingham |    27000 |         0.00
 Birmingham City      | Birmingham |    30016 |         0.50
 Aston Villa          | Birmingham |    42785 |         1.00
 Queens Park Rangers  | London     |    18439 |         0.00
 Fulham               | London     |    25700 |         0.13
 Crystal Palace       | London     |    26255 |         0.25
 Wimbledon            | London     |    26255 |         0.25
 Charlton Athletic    | London     |    27111 |         0.50
 West Ham United      | London     |    35016 |         0.63
 Tottenham Hotspur    | London     |    36284 |         0.75
 Chelsea              | London     |    41663 |         0.88
 Arsenal              | London     |    60260 |         1.00
 Manchester City      | Manchester |    55097 |         0.00
 Manchester United    | Manchester |    75635 |         1.00
(14 rows)
```

PERCENTILE_CONT (window function)

Given an ordered set of values, this aggregate window function returns the continuous percentile value, which is the value from the set that corresponds to the specified fraction. For example, `percentile_cont(0.5)` returns the median value. If necessary, the percentile value is interpolated in the set.

```
PERCENTILE_CONT(fraction) WITHIN GROUP (ORDER BY sort_expression) OVER { window_name | ( [ window_definition ] ) }
```

Specify a fraction between `0` and `1`. The sort expression data type must be numeric.

The `window_definition` for this function can have a `PARTITION BY` clause but no `ORDER BY` clause or frame clause. For the full `window_definition` syntax, see Syntax for Window Functions.

For example:

```
premdb=# select city,
percentile_cont(0.5) within group(order by avg_att) over(partition by city)
from team where avg_att>0 order by 2 desc;
   city     | percentile_cont
------------+-----------------
 Manchester |         64.6635
 Manchester |         64.6635
 Newcastle  |          49.754
 Sunderland |          43.071
 Liverpool  |          41.017
 Liverpool  |          41.017
 London     |          35.776
 London     |          35.776
 London     |          35.776
 London     |          35.776
 London     |          35.776
 Leicester  |          32.201
 Southampton|          30.751
 Birmingham |         29.1605
 Birmingham |         29.1605
 Stoke      |          27.534
 Norwich    |          26.972
 Swansea    |          20.711
 Watford    |          20.594
 Bournemouth|          11.189
(20 rows)
```

PERCENTILE_DISC (window function)

Platforms: All platforms

Parent topic: Window Functions

Given an ordered set of values, this aggregate window function returns the discrete percentile value, which is the first input value whose position in the ordering equals or exceeds the specified fraction.

```
PERCENTILE_DISC(fraction) WITHIN GROUP (ORDER BY sort_expression) OVER { window_name | ( [ window_definition ] ) }
```

Specify a fraction between `0` and `1`. The sort expression data type must be numeric.

The `window_definition` for this function can have a `PARTITION BY` clause but no `ORDER BY` clause or frame clause. For the full `window_definition` syntax, see Syntax for Window Functions.

For example:

```
premdb=# select city,
percentile_disc(0.9) within group(order by avg_att) over(partition by city)
from team where avg_att>0 order by 2 desc;
   city     | percentile_disc
------------+-----------------
 Manchester |          75.286
 Manchester |          75.286
 London     |          59.944
 London     |          59.944
 London     |          59.944
 London     |          59.944
 London     |          59.944
 Newcastle  |          49.754
 Liverpool  |          43.910
 Liverpool  |          43.910
 Sunderland |          43.071
 Birmingham |          33.690
 Birmingham |          33.690
 Leicester  |          32.201
 Southampton |         30.751
 Stoke      |          27.534
 Norwich    |          26.972
 Swansea    |          20.711
 Watford    |          20.594
 Bournemouth |         11.189
(20 rows)
```

RANK

Rank a result set based on an ordered set of rows.

```
RANK() OVER { window_name | ( window_definition ) }
```

The `RANK()` function itself does not take any arguments, but the empty parentheses are required. The `ORDER BY` clause in the `window_definition` is required. The frame clause, if specified, is ignored. For the full `window_definition` syntax, see Syntax for Window Functions.

The following example returns the 25 highest-scoring home wins, based on an ordered set of `ftscore` values. (Note that this query orders on the first number in the `ftscore` column so a score of 7-4 ranks lower than a score of 8-0, for example.)

```
SELECT season_name, h.name, a.name, ftscore,
RANK() OVER(ORDER BY ftscore DESC)
FROM season s, hometeam h, awayteam a, match m
WHERE s.seasonid=m.seasonid AND h.htid=m.htid AND a.atid=m.atid
ORDER BY 5,1 LIMIT 25;
```

```
 season_name |       name        |        name         | ftscore | rank
-------------+-------------------+---------------------+---------+------
 2009-2010   | Tottenham Hotspur | Wigan Athletic      | 9-1     |    1
 1994-1995   | Manchester United | Ipswich Town        | 9-0     |    2
 2011-2012   | Manchester United | Arsenal             | 8-2     |    3
 2007-2008   | Middlesbrough     | Manchester City     | 8-1     |    4
 1999-2000   | Newcastle United  | Sheffield Wednesday | 8-0     |    5
 2009-2010   | Chelsea           | Wigan Athletic      | 8-0     |    5
 2012-2013   | Chelsea           | Aston Villa         | 8-0     |    5
 2007-2008   | Portsmouth        | Reading             | 7-4     |    8
 2012-2013   | Arsenal           | Newcastle United    | 7-3     |    9
 1997-1998   | Blackburn Rovers  | Sheffield Wednesday | 7-2     |   10
 1999-2000   | Tottenham Hotspur | Southampton         | 7-2     |   10
 2009-2010   | Chelsea           | Sunderland          | 7-2     |   10
 1992-1993   | Blackburn Rovers  | Norwich City        | 7-1     |   13
 1993-1994   | Newcastle United  | Swindon Town        | 7-1     |   13
 1994-1995   | Aston Villa       | Wimbledon           | 7-1     |   13
 1996-1997   | Everton           | Southampton         | 7-1     |   13
 1996-1997   | Newcastle United  | Tottenham Hotspur   | 7-1     |   13
 1998-1999   | Liverpool         | Southampton         | 7-1     |   13
 1999-2000   | Manchester United | West Ham United     | 7-1     |   13
 2001-2002   | Blackburn Rovers  | West Ham United     | 7-1     |   13
 2007-2008   | Everton           | Sunderland          | 7-1     |   13
 2009-2010   | Chelsea           | Aston Villa         | 7-1     |   13
 2010-2011   | Manchester United | Blackburn Rovers    | 7-1     |   13
 2011-2012   | Arsenal           | Blackburn Rovers    | 7-1     |   13
 1995-1996   | Blackburn Rovers  | Nottingham Forest   | 7-0     |   25
(25 rows)
```

You can partition this result by season, for example:

```
SELECT season_name, h.name, a.name, ftscore,
RANK() OVER(PARTITION BY season_name ORDER BY ftscore DESC)
FROM season s, hometeam h, awayteam a, match m
WHERE s.seasonid=m.seasonid AND h.htid=m.htid AND a.atid=m.atid
ORDER BY 1,5;
```

ROW_NUMBER

Generate a sequence of row numbers based on an ordered set of rows.

```
ROW_NUMBER() OVER { window_name | ( window_definition ) }
```

The `ORDER BY` clause in the `window_definition` is required. The frame clause, if specified, is ignored. For the full `window_definition` syntax, see Syntax for Window Functions.

For example:

```
premdb=# SELECT ROW_NUMBER() OVER(ORDER BY name NULLS LAST),
name FROM team ORDER BY name;
 row_number |          name
------------+------------------------
          1 | Arsenal
          2 | Aston Villa
          3 | Barnsley
          4 | Birmingham City
          5 | Blackburn Rovers
          6 | Blackpool
          7 | Bolton Wanderers
          8 | Bournemouth
          9 | Bradford City
         10 | Burnley
...
```

The following example partitions the results by `city` and constrains the result to `London`.

```
premdb=# SELECT ROW_NUMBER() OVER(PARTITION BY city ORDER BY city NULLS LAST),
city, name FROM team WHERE city='London' ORDER BY city;
 row_number |  city  |        name
------------+--------+--------------------
          1 | London | Arsenal
          2 | London | Charlton Athletic
          3 | London | Chelsea
          4 | London | Crystal Palace
          5 | London | Fulham
          6 | London | Queens Park Rangers
          7 | London | Tottenham Hotspur
          8 | London | West Ham United
          9 | London | Wimbledon
(9 rows)
```

STDDEV_SAMP and STDDEV_POP (window functions)

Platforms: All platforms

Parent topic: Window Functions

Given a set of integer, decimal, or floating-point numbers, return the sample standard deviation or the population standard deviation, based on a window definition with an optional frame clause. The STDDEV_SAMP function returns a result that is equivalent to the square root of the sample variance of the same set of values. STDDEV_SAMP and STDDEV are synonyms.

```
STDDEV_SAMP | STDDEV (expression)
STDDEV_POP (expression)
OVER { window_name | ( [ window_definition ] ) }
```

The `expression` must be an integer, decimal, or floating-point number. The return type is either DOUBLE PRECISION (for floating-point inputs) or DECIMAL (for all other inputs).

For the full `window_definition` syntax, see Syntax for Window Functions.

For example:

```
premdb=# select teamid, avg_att,
stddev_pop(capacity) over (partition by city order by avg_att rows between 1 preceding and 2 following) as stddevpop
from team where avg_att>0 order by 1,2;
 teamid | avg_att |  stddevpop
--------+---------+-------------
      1 |  59.944 |  9298.500000
      2 |  33.690 |  7892.500000
      8 |  11.189 |     0.000000
     13 |  41.500 | 10272.025798
     15 |  24.636 |  4458.995726
     17 |  38.124 |  2260.500000
     22 |  32.201 |     0.000000
     23 |  43.910 |  2260.500000
     24 |  54.041 | 10269.000000
     25 |  75.286 | 10269.000000
     27 |  49.754 |     0.000000
     28 |  26.972 |     0.000000
     36 |  30.751 |     0.000000
     37 |  27.534 |     0.000000
     38 |  43.071 |     0.000000
     39 |  20.711 |     0.000000
     41 |  35.776 | 10101.610623
     42 |  20.594 |     0.000000
     43 |  24.631 |  7892.500000
     44 |  34.910 |  5530.972451
(20 rows)
```

Note: The STDDEV or STDDEV_SAMP function for an expression that consists of a single value returns `NULL`. The STDDEV_POP function for an expression that consists of a single value returns `0`).

SUM (window function)

Compute the sum for an expression based on a window definition that contains an optional frame clause.

```
SUM ([ expression ]) OVER { window_name | ( [ window_definition ] ) }
```

For the full `window_definition` syntax, see Syntax for Window Functions.

For example:

```
premdb=# select name, city, capacity, sum(capacity) over(partition by city order by name) as sum_cap
from team where city in('London','Birmingham','Liverpool') order by 2,1;
        name         |    city    | capacity | sum_cap
---------------------+------------+----------+---------
 Aston Villa         | Birmingham |    42785 |   42785
 Birmingham City     | Birmingham |    30016 |   72801
 West Bromwich Albion | Birmingham |    27000 |   99801
 Everton             | Liverpool  |    40221 |   40221
 Liverpool           | Liverpool  |    44742 |   84963
 Arsenal             | London     |    60260 |   60260
 Charlton Athletic   | London     |    27111 |   87371
 Chelsea             | London     |    41663 |  129034
 Crystal Palace      | London     |    26255 |  155289
 Fulham              | London     |    25700 |  180989
 Queens Park Rangers | London     |    18439 |  199428
 Tottenham Hotspur   | London     |    36284 |  235712
 West Ham United     | London     |    35016 |  270728
 Wimbledon           | London     |    26255 |  296983
(14 rows)
```

Syntax for Window Functions

This section is an overview of the syntax supported for window functions. For more details, see the individual function descriptions.

Function Definition

```
function_name ([ expression ]) OVER { window_name | ( window_definition ) }
```

Some window functions support special syntax after the `expression`. See the individual function descriptions.

The `window_name` is a reusable window definition that is defined in the WINDOW clause of the query.

Window Definition

The window definition may consist of a `PARTITION BY` clause, an `ORDER BY` clause, and a frame clause. At a minimum, the `OVER` clause must have parentheses: `OVER()` without any parameters is valid syntax.

```
[ PARTITION BY expression [, ...] ]
[ ORDER BY expression [ ASC | DESC ] [ NULLS { FIRST | LAST } ] [, ...] ]
[ frame_clause ]
```

Frame Clause

A window frame is a subset of rows in a partition (or the whole data set if no `PARTITION BY` clause is defined). The specific value that a window function returns for a given row depends on an evaluation of all the rows in the frame. For example, the frame may refer to a moving set of three rows, defined as `rows between 1 preceding and 1 following`, where the "current row" being evaluated is in the middle of the frame.

The `frame_clause` supports `RANGE` and `ROWS` modes with different options. If no frame clause is specified, the default frame for all window functions is as follows:

```
RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
```

which is equivalent to `RANGE UNBOUNDED PRECEDING`.

Range-Based Frames

The `RANGE` options are as follows:

```
RANGE UNBOUNDED PRECEDING
RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING
RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
```

`CURRENT ROW` in `RANGE` mode refers to the first or last peer row in the partition. A peer is a repeating value in the `ORDER BY` column for the window function. (Note that in `ROWS` mode, `CURRENT ROW` simply means the current row.)

Rows-Based Frames

The `ROWS` options are as follows:

```
ROWS frame_start
ROWS BETWEEN frame_start AND frame_end
```

where `frame_start` is one of the following:

```
UNBOUNDED PRECEDING
value PRECEDING
CURRENT ROW
value FOLLOWING
```

and `frame_end` is one of the following:

```
value PRECEDING
CURRENT ROW
value FOLLOWING
UNBOUNDED FOLLOWING
```

and `value` is an integer that defines the row count that precede or follow the current row.

In `ROWS` mode, `CURRENT ROW` simply means the current row.

VAR_SAMP and VAR_POP (window functions)

Given a set of integer, decimal, or floating-point numbers, return the sample variance or population variance, based on a window definition with an optional frame clause. The VAR_SAMP result is equivalent to the squared sample standard deviation of the same set of numbers. VAR_SAMP and VARIANCE are synonyms.

```
VAR_SAMP | VARIANCE (expression)
VAR_POP (expression)
OVER { window_name | ( [ window_definition ] ) }
```

The return type is either `DOUBLE PRECISION` (for floating-point inputs) or `DECIMAL` (for all other inputs).

For the full `window_definition` syntax, see Syntax for Window Functions.

For example:

```
premdb=# select teamid, avg_att,
variance(capacity) over (partition by city order by avg_att rows between 1 preceding and 2 following) as varsamp
from team where avg_att>0 order by 1,2;
 teamid | avg_att |     varsamp
--------+---------+-----------------
      1 |  59.944 | 172924204.500000
      2 |  33.690 | 124583112.500000
      8 |  11.189 |          [NULL]
     13 |  41.500 | 158271771.000000
     15 |  24.636 |  29823964.333333
     17 |  38.124 |  10219720.500000
     22 |  32.201 |          [NULL]
     23 |  43.910 |  10219720.500000
     24 |  54.041 | 210904722.000000
     25 |  75.286 | 210904722.000000
     27 |  49.754 |          [NULL]
     28 |  26.972 |          [NULL]
     36 |  30.751 |          [NULL]
     37 |  27.534 |          [NULL]
     38 |  43.071 |          [NULL]
     39 |  20.711 |          [NULL]
     41 |  35.776 | 136056716.250000
     42 |  20.594 |          [NULL]
     43 |  24.631 | 124583112.500000
     44 |  34.910 |  40788875.000000
(20 rows)
```

Note: The VARIANCE or VAR_SAMP function for an expression that consists of a single value returns `NULL`. The VAR_POP function for an expression that consists of a single value returns `0`).

CONVERT (SQL Server Migration Function)

Cast an expression to a specified data type using a specific style.

Note: The `CONVERT` function is rewritten to standard SQL and is intended only for use during migration from a Microsoft SQL Server application. See the Supported Conversions table for the exact rewrites. For all purposes other than migration, Yellowbrick strongly recommends that you use the standard SQL functions directly.

```
CONVERT(datatype, expr [, style])
```

datatype
: Target data type

expr
: Input expression to be converted to `datatype`

style
: Integer constant that identifies the type of input or output. See the Style Patterns table. If `style` is not provided, a default style is chosen based on the `datatype`.

Style Patterns

Style	Pattern	Notes
0, 100	Mon DD YYYY HH12:MIAM	
1	MM/DD/YY	US
101	MM/DD/YYYY	US
2	YY.MM.DD	ANSI
102	YYYY.MM.DD	ANSI
3	DD/MM/YY	British/French
103	DD/MM/YYYY	British/French
4	DD.MM.YY	German
104	DD.MM.YYYY	German
5	DD-MM-YY	Italian
105	DD-MM-YYYY	Italian
6	DD Mon YY	
106	DD Mon YYYY	
7	Mon DD, YY	

Style	Pattern	Notes
107	Mon DD, YYYY	
8, 24, 108	HH24:MI:SS	
9, 109	Mon DD YYYY HH12:MI:SS:MSAM	
10	MM-DD-YY	USA
110	MM-DD-YYYY	USA
11	YY/MM/DD	Japan
111	YYYY/MM/DD	Japan
12	YYMMDD	ISO
112	YYYYMMDD	ISO
13, 113	DD Mon YYYY HH24:MI:SS:MS	Europe default + milliseconds
14, 114	HH24:MI:SS:MS	
20, 120	YYYY-MM-DD HH24:MI:SS	ODBC canonical
21, 25, 121	YYYY-MM-DD HH24:MI:SS.MS	ODBC canonical + milliseconds, also default style in the two argument version
22	MM/DD/YY HH12:MI:SS AM	US
23	YYYY-MM-DD	ISO8601
126	YYYY-MM-DD"T"HH24:MI:SS.MS	ISO8601
127	Unsupported	`ERROR: convert: ISO8601 with time zone UTC unsupported`
130, 131	Unsupported	`ERROR: convert: Hijri calendar unsupported`

Supported Conversions

From	To	Rewrite
unknown, char, varchar, text	date	`to_date(expr::text, style_pattern)`
unknown, char, varchar, text	timestamp	`to_timestamp(expr::text, style_pattern)::timestamp`
unknown, char, varchar, text	timestamptz	`to_timestamp(expr::text, style_pattern)`
unknown, char, varchar, text	float4, float8	`cast(expr as datatype)` The two argument form of `CONVERT` is allowed, but the three argument form will result in an error.
unknown, char, varchar, text	time, interval	`error: convert: cannot convert from exprtype to datatype`
date, timestamp, timestamptz	char, varchar, text	`to_char(expr, style_pattern)::datatype`

From	To	Rewrite
float4, float8	char, varchar, text	`cast(expr as datatype)` The two argument form of `CONVERT` is allowed, but the three argument form will result in an error.
time, interval	char, varchar, text	`error: convert: cannot convert from exprtype to datatype`
numeric	float4, float8	`error: convert: cannot convert from exprtype to datatype`

Any `CONVERT` function that is not listed above devolves to `CAST(expr as datatype)`.

Usage Notes

- SQL Server `date` and `datetime2` types correlate well with Yellowbrick `date` and `timestamp` types respectively.
- However, SQL Server `datetimeoffset` type does not correlate well with Yellowbrick `timestamptz` type. In this case, use standard SQL functions directly rather than `CONVERT`.
- `CONVERT` does not support arbitrary whitespace or / as a delimiter. To work around this, use `REPLACE` to prepare the input. For example, the inputs `2000/01/01` and `2001 - 01 - 01` would have to be prepared in the following way:

```
SELECT CONVERT(date, '2001-01-01', 120) x, CONVERT(date, REPLACE('2001/01/01', '/', '-'), 120) y,
CONVERT(date, REPLACE('2001 - 01 - 01', ' ', ''), 120) z;
```

Examples

The following example converts varchar Mar 31, 22, which is in the style 7 pattern, to a character string date:

```
premdb=# SELECT CONVERT(date, 'Mar 31, 22', 7);
   date
------------
 2022-03-31
(1 row)
```

The opposite can be done by starting with a date and extracting a varchar in a specified format:

```
premdb=# SELECT CONVERT(varchar(100), CAST('2001-01-01' as date), 7);
  varchar
------------
 Jan 01, 01
(1 row)
```

Attempting to convert unrelated datatypes will result in an error:

```
premdb=# SELECT CONVERT(float8, '1s'::interval);
ERROR:  cannot cast type interval to double precision
LINE 1: SELECT CONVERT(float8, '1s'::interval);
                               ^
```

Attempting to use a random integer as a style will result in an error in cases where style matters. If style is not needed, the random integer will be ignored.

```
premdb=# SELECT CONVERT(date, '2001-01-01', 3000);
ERROR:  convert: invalid style 3000
```

```
premdb=# SELECT CONVERT(int8, 1::int2, 3000);
 int8
------
    1
(1 row)
```

GETBIT Function

Return the bit value (`0` or `1`) in a number at the specified position, according to the binary representation of that number.

```
getbit(target_number, bit_position)
```

target_number

A specific number or an expression that evaluates to a number. The number must be a `SMALLINT`, `INT`, or `BIGINT` data type.

bit_position

Bit position in the target number. The position must be an integer that is in range for the size of the target number. For example, if the target is a 16-bit data type, the position must be between `0` and `15`. For a 32-bit data type, the position must be between `0` and `31`.

For example:

```
premdb=# select htid, getbit(htid,2) from team;
 htid | getbit
------+--------
   2 |      0
   3 |      0
   4 |      1
   5 |      1
   6 |      1
   7 |      1
   8 |      0
   9 |      0
   10 |      0
...
```

```
premdb=# select getbit(987654321,31) from sys.const;
 getbit
--------
      0
(1 row)
```

JSON_ARRAY_STR

Compose a JSON array from the specified inputs and return the array as a `VARCHAR` column.

```
json_array_str(input [, ...])
```

The data types of the individual inputs may vary. In turn, the length of the returned `VARCHAR` column varies, depending on its inputs. If necessary, you can use an explicit cast to modify the return type.

Single-quoted input strings are double-quoted in the output. Double-quoted input strings are interpreted as column names.

Examples

For example, compose an array from two columns in the `match` table:

```
premdb=# select matchday, htid, atid, htscore, ftscore, json_array_str(htscore, ftscore) score_array
from match where seasonid=21 and htid=3
order by 1;
     matchday       | htid | atid | htscore | ftscore |  score_array
--------------------+------+------+---------+---------+---------------
 2012-08-25 00:00:00 |    3 |   67 | 0-3     | 1-3     | ["0-3", "1-3"]
 2012-09-15 00:00:00 |    3 |   89 | 1-0     | 2-0     | ["1-0", "2-0"]
 2012-09-30 00:00:00 |    3 |   93 | 0-0     | 1-1     | ["0-0", "1-1"]
 2012-10-27 00:00:00 |    3 |   78 | 1-0     | 1-1     | ["1-0", "1-1"]
 2012-11-10 00:00:00 |    3 |   75 | 1-0     | 2-3     | ["1-0", "2-3"]
 2012-11-24 00:00:00 |    3 |   51 | 0-0     | 0-0     | ["0-0", "0-0"]
 2012-11-27 00:00:00 |    3 |   83 | 0-0     | 1-0     | ["0-0", "1-0"]
 2012-12-08 00:00:00 |    3 |   87 | 0-0     | 0-0     | ["0-0", "0-0"]
 2012-12-26 00:00:00 |    3 |   91 | 0-0     | 0-4     | ["0-0", "0-4"]
 2012-12-29 00:00:00 |    3 |   95 | 0-1     | 0-3     | ["0-1", "0-3"]
...
```

Compose an array that consists of a column and the result of an aggregate function:

```
premdb=# select json_array_str(ftscore, count(*))
from match group by ftscore order by 1;
   ?column?
---------------
 ["0-0", 738]
 ["0-1", 633]
 ["0-2", 371]
 ["0-3", 162]
 ["0-4", 74]
 ["0-5", 19]
 ["0-6", 7]
 ["1-0", 943]
 ["1-1", 1024]
 ["1-2", 513]
```

In this example, the array consists of four inputs, based on a join of two tables:

```
premdb=# select json_array_str(season.seasonid,htid,atid,matchday::date) season22
from season, match where match.seasonid=season.seasonid and season.seasonid=22
order by season.seasonid;
         season22
---------------------------
 [22, 2, 52, "2013-08-17"]
 [22, 2, 61, "2014-01-01"]
 [22, 2, 63, "2013-12-23"]
 [22, 2, 65, "2014-02-02"]
 [22, 2, 67, "2013-12-08"]
 [22, 2, 68, "2014-01-18"]
 [22, 2, 69, "2013-12-04"]
...
```

JSON_INGEST

> This function is deprecated, use JSONB and JSON Accessor instead.

Convert a JSON input string into a Document Object Model (DOM) representation of the string that you can query with the JSON_LOOKUP function.

```sql
JSON_INGEST(json_input_string)
```

where `json_input_string` is a JSON document that Yellowbrick stores as a `VARCHAR` data type.

Usage Notes

- **Important:** Running `JSON_INGEST` as a first step is only required when you plan to run `JSON_LOOKUP` with the `jpath` option. If you run `JSON_LOOKUP` repeatedly on the same data, performance will improve considerably on repeated queries because the `jpath` option uses a hash index optimization. The other path types do not have this capability.

If you use the `jpointer` or `jpointer_simdjson` path type when you run queries with the `JSON_LOOKUP` function, you do not need to run `JSON_INGEST` on the JSON input first; you can use `JSON_LOOKUP` to query the data directly.

- When you use the `JSON_INGEST` function, Yellowbrick recommends that you store the original JSON data as well as the function results. A new version of the function in a later release may require the generation of new hash values.
- The `JSON_INGEST` function returns a `VARCHAR` data type.
- The size of the output DOM string may be 2 to 3 three times larger than the original JSON input string. The maximum ingested output cannot be greater than 64000 bytes (the maximum length of a `VARCHAR` column), so the input string must be significantly smaller.
- The `JSON_INGEST` function supports JSON strings with a maximum depth of 250 levels.
- The `ybload` client can load JSON documents in `text` or `csv` format.

Example

See also the examples for JSON_LOOKUP.

First, create a table that stores the JSON data:

```sql
create table json_movies (jsonstr varchar(1000));

insert into json_movies values('{"title":"After Dark in Central Park","year":1900,"cast":[],"genres":[]}');

select * from json_movies;
                              jsonstr
-----------------------------------------------------------------------
 {"title":"After Dark in Central Park","year":1900,"cast":[],"genres":[]}
```

Now create a table that can store the JSON data alongside the results of the `JSON_INGEST` function:

```sql
create table json_movies_ingested as select jsonstr, json_ingest(jsonstr) from json_movies;

\d json_movies_ingested
       Table "public.json_movies_ingested"
   Column    |           Type           | Modifiers
-------------+--------------------------+-----------
 jsonstr     | character varying(1000)  |
 json_ingest | character varying(64000) |

Distribution: Hash (jsonstr)
```

Now you can write queries with the `JSON_LOOKUP` function:

```sql
select json_lookup(json_ingest,'$.title','jpath') from json_movies_ingested;
       json_lookup
----------------------------
 After Dark in Central Park

select json_lookup(json_ingest,'$.year','jpath') from json_movies_ingested;
 json_lookup
-------------
 1900
```

JSON_LOOKUP

> This function is deprecated, use JSONB and JSON_ACCESSOR instead.

Return the key value for a JSON object.

```sql
JSON_LOOKUP(column_name, path_to_json_key, path_type)
```

Parameters

column_name

A table column that stores the DOM representation of the JSON input (as a result of running JSON_INGEST on the input).

path_to_json_key

String that represents the JSON key for which you want to return the corresponding value. Supported operators depend on the path type.

- `jpath` : `$, ., []` (root is `'$'`). For example:
  - `$.year`
  - `$.year.month`
- `jpointer` and `jpointer_simdjson` : `/, /[array index], /[object key]` (root is the empty string `''`, not `'/'`). For example:
  - `/year`
  - `/year/0`
  - `/year/month`

path_type

One of the following types must be specified:

- `jpath` : the RapidJSON library is used in the query.

  Important: Running `JSON_INGEST` as a first step is always required when you plan to run `JSON_LOOKUP` with the `jpath` option. If you run `JSON_LOOKUP` repeatedly on the same data, performance will improve considerably because the `jpath` option uses a hash index optimization. The other path types do not have this capability.

- `jpointer` : the RapidJSON library is used in the query.

  Important: If you use the `jpointer` or `jpointer_simdjson` path type, you do not need to run `JSON_INGEST` on the JSON input first; you can use `JSON_LOOKUP` directly on the original JSON data.

- `jpointer_simdjson` : the SIMDJSON library is used in the query.

Usage Notes

- The function returns a `VARCHAR` data type.
- Numeric data is supported as text, and its representation is preserved (for example, `1.02`, `102e-2`).
- The function returns `true` and `false` for Boolean values.
- The function returns `null` when it encounters empty results, empty JSON documents, empty strings, or an empty `jpath`.
- The function returns `"null"` as a string for cases like this: `{"a": null}` with path `$.a`
- The function returns the PostgreSQL `NULL` value for missing values: `{"b": 10}` with path `$.a`
- A single string or number is valid JSON. This function supports inputs where the root of a JSON document is a scalar value only, not a key-value object. For example, the whole document may be the value `2018-2019`, instead of, for example, `{"Season Name": 2018-2019}`.
- The function returns errors when any JSON input is malformed. Any error in any row will cause the query to fail. For example:

```
ERROR: json_lookup: malformed input, reason: Missing a closing quotation mark in string. pos: 14/14 input: "Hello, world!
```

- Errors for queries that use the `jpointer_simdjson` path type do not return the position of the failure in the message.

Examples

The following examples operate on simple JSON strings that look like this:

```sql
select "25 Seasons" from seasons_json;
                    25 Seasons
-------------------------------------------------------
 {"Season Number" : 1, "Season Years" : "1992-1993"}
 {"Season Number" : 2, "Season Years" : "1993-1994"}
 {"Season Number" : 3, "Season Years" : "1994-1995"}
 {"Season Number" : 4, "Season Years" : "1995-1996"}
 {"Season Number" : 5, "Season Years" : "1996-1997"}
...
```

First, create a table that applies the `JSON_INGEST` function to the original JSON strings:

```sql
create table seasons_ingest as select "25 Seasons", json_ingest("25 Seasons") from seasons_json;
SELECT 25

\d seasons_ingest
          Table "public.seasons_ingest"
   Column    |           Type           | Modifiers
-------------+--------------------------+-----------
 25 Seasons  | character varying(116)   |
 json_ingest | character varying(64000) |

Distribution: Replicated
```

Now query the ingested JSON column, using the `jpath` type:

```sql
select json_lookup(json_ingest,'$.Season Years','jpath') from seasons_ingest;
 json_lookup
-------------
 1992-1993
 1993-1994
 1994-1995
 1995-1996
 1996-1997
...
```

Now query with the `jpointer` type, which means you can query the original `seasons_json` table, not the table with the ingested strings:

```sql
select json_lookup("25 Seasons",'/Season Number','jpointer')::int
from seasons_json order by 1 desc limit 5;
 json_lookup
-------------
          25
          24
          23
          22
          21
(5 rows)
```

Note that this function always returns a `VARCHAR` string. To order by a string that represents a number, use an explicit cast as shown in the previous query. Without the cast, the same query orders by character strings and returns:

```sql
select json_lookup("25 Seasons",'/Season Number','jpointer')
from seasons_json order by 1 desc limit 5;
 json_lookup
-------------
 9
 8
 7
 6
 5
(5 rows)
```

The following example shows how to query JSON strings that have nested data. Create a table and insert one row of nested JSON data:

```sql
create table json_gloss(entry varchar(1000));

insert into json_gloss(entry) values('{"glossary": {"title": "example glossary","GlossDiv": {"title": "S","GlossList": {"GlossEntr
INSERT 0 1
```

Now create a table that holds both the original data and the output for the `JSON_INGEST` function:

```sql
create table json_gloss_ingested as select entry, json_ingest(entry) from json_gloss;
SELECT 1
```

Now run queries on the `json_gloss_ingested` table:

```sql
select json_lookup(json_ingest,'$.glossary.title','jpath') from json_gloss_ingested;
   json_lookup
------------------
 example glossary
(1 row)

select json_lookup(json_ingest,'$.glossary.GlossDiv.GlossList.GlossEntry.Acronym','jpath') from json_gloss_ingested;
 json_lookup
-------------
 SGML
(1 row)
```

The following example shows support for JSON strings that consist only of single scalar values. Note that the root for `jpointer` is specified with the empty string `''`:

```sql
select * from scalar_json;
    col1
---------------
 "2018-2019"
 "Twenty-five"
 "25"
(3 rows)

select json_lookup(col1,'','jpointer') from scalar_json;
 json_lookup
-------------
 2018-2019
 Twenty-five
 25
(3 rows)
```

JSON_OBJECT_STR

Compose a JSON object from one or more key-value pairs and return the object as a `VARCHAR` column.

```
JSON_OBJECT_STR(key, value [, ...])
```

Each input may be a string literal, a numeric constant, a scalar function, a scalar subquery, or a table column.

The data types of the individual inputs may vary. In turn, the length of the returned `VARCHAR` column varies, depending on its inputs. If necessary, you can use an explicit cast to modify the return type.

The input list must contain an even number of elements. All keys must specify a value. NULL is allowed as a value, but not as a key. Duplicate keys with the same name are allowed.

Single-quoted input strings are double-quoted in the output. Double-quoted input strings are interpreted as column names.

Examples

For example, the following query returns two key-value pairs from each `season` table row:

```
premdb=# select json_object_str('Season Number', seasonid, 'Season Years', season_name) "25 Seasons"
from season order by seasonid;
                    25 Seasons
-----------------------------------------------------
 {"Season Number" : 1, "Season Years" : "1992-1993"}
 {"Season Number" : 2, "Season Years" : "1993-1994"}
 {"Season Number" : 3, "Season Years" : "1994-1995"}
 {"Season Number" : 4, "Season Years" : "1995-1996"}
 {"Season Number" : 5, "Season Years" : "1996-1997"}
...
```

The following example shows how to return a whole table (all of its column names and their respective values) as a set of JSON objects, one per row:

```
premdb=# select json_object_str('seasonid',seasonid,'season_name', season_name, 'numteams', numteams, 'winners', winners) seasonta
                                        seasontable
-----------------------------------------------------------------------------------------------
 {"seasonid" : 1, "season_name" : "1992-1993", "numteams" : 22, "winners" : "Manchester United"}
 {"seasonid" : 2, "season_name" : "1993-1994", "numteams" : 22, "winners" : "Manchester United"}
 {"seasonid" : 3, "season_name" : "1994-1995", "numteams" : 22, "winners" : "Blackburn Rovers"}
 {"seasonid" : 4, "season_name" : "1995-1996", "numteams" : 20, "winners" : "Manchester United"}
 {"seasonid" : 5, "season_name" : "1996-1997", "numteams" : 20, "winners" : "Manchester United"}
 {"seasonid" : 6, "season_name" : "1997-1998", "numteams" : 20, "winners" : "Arsenal"}
 {"seasonid" : 7, "season_name" : "1998-1999", "numteams" : 20, "winners" : "Manchester United"}
 {"seasonid" : 8, "season_name" : "1999-2000", "numteams" : 20, "winners" : "Manchester United"}
 {"seasonid" : 9, "season_name" : "2000-2001", "numteams" : 20, "winners" : "Manchester United"}
 {"seasonid" : 10, "season_name" : "2001-2002", "numteams" : 20, "winners" : "Arsenal"}
 {"seasonid" : 11, "season_name" : "2002-2003", "numteams" : 20, "winners" : "Manchester United"}
 {"seasonid" : 12, "season_name" : "2003-2004", "numteams" : 20, "winners" : "Arsenal"}
 {"seasonid" : 13, "season_name" : "2004-2005", "numteams" : 20, "winners" : "Chelsea"}
 {"seasonid" : 14, "season_name" : "2005-2006", "numteams" : 20, "winners" : "Chelsea"}
```

```
 {"seasonid" : 15, "season_name" : "2006-2007", "numteams" : 20, "winners" : "Manchester United"}
 {"seasonid" : 16, "season_name" : "2007-2008", "numteams" : 20, "winners" : "Manchester United"}
 {"seasonid" : 17, "season_name" : "2008-2009", "numteams" : 20, "winners" : "Manchester United"}
 {"seasonid" : 18, "season_name" : "2009-2010", "numteams" : 20, "winners" : "Chelsea"}
 {"seasonid" : 19, "season_name" : "2010-2011", "numteams" : 20, "winners" : "Manchester United"}
 {"seasonid" : 20, "season_name" : "2011-2012", "numteams" : 20, "winners" : "Manchester City"}
 {"seasonid" : 21, "season_name" : "2012-2013", "numteams" : 20, "winners" : "Manchester United"}
 {"seasonid" : 22, "season_name" : "2013-2014", "numteams" : 20, "winners" : "Manchester City"}
 {"seasonid" : 23, "season_name" : "2014-2015", "numteams" : 20, "winners" : "Chelsea"}
 {"seasonid" : 24, "season_name" : "2015-2016", "numteams" : 20, "winners" : "Leicester City"}
 {"seasonid" : 25, "season_name" : "2016-2017", "numteams" : 20, "winners" : null}
(25 rows)
```

The following example returns an error because the 7th parameter (`param=>7` in the `DETAIL` message) contains a `NULL` value. The 7th parameter is the `winners` column in the `season` table.

```
premdb=# select json_object_str(seasonid,1,season_name,2,numteams,3,winners,4) from season;
ERROR:  The key for this json object is invalid / null / not a string
DETAIL:   [param=>7]
```

NEXTVAL Function

Return the next value for the specified sequence. The return type is `BIGINT`.

```
NEXTVAL('sequence_name')
```

You must specify the name of the sequence, using single quotes.

For example, the following ALTER SEQUENCE command restarts the `matchid` sequence. Then the SELECT statement returns the next value for the sequence:

```
premdb=# alter sequence matchid restart;
ALTER SEQUENCE
premdb=# select nextval('matchid');
 nextval
---------
   3071
(1 row)

premdb=# select nextval('matchid');
 nextval
---------
   4095
(1 row)
```

The system guarantees uniqueness of all generated sequence values.

You cannot use the `NEXTVAL` function or run an `ALTER SEQUENCE` command on a database that is in `HOT_STANDBY` mode.

Note: Yellowbrick does not support other sequence functions such as `CURRVAL`, `LASTVAL`, and `SETVAL`.

Set Returning Functions

Yellowbrick Documentation > Reference > SQL Reference > SQL Functions and Operators > Set Returning Functions

Platforms: All platforms

Parent topic: SQL Functions and Operators

> Prerequisite: while the feature is in beta, you must explicitly enable it by setting `enable_full_funcscan` and `enable_full_lateral_join` to `ON`.

Note: To use Set Returning Functions (SRFs), both configuration parameters must be enabled together.

Below is the list of Set Returning Functions (SRFs) that can be used:

Function	Description	Output
`generate_series(start,stop,step)`	Generates a sequence of numbers from `start` to `stop` with a step size of `step`	Row of integers
`FLATTEN(jsonb_expression)`	For details, see FLATTEN.	A JSONB object with the fields "index", "key" and "value" for each element contained in the input JSONB.

SRFs can return zero, one or multiple rows. For example:

```sql
SELECT * FROM generate_series(3, 100, 29);
```

```sql
 generate_series
-----------------
               3
              32
              61
              90
(4 rows)
```

Supported contexts

Set returning functions (SRFs) are not supported in the same contexts as normal functions. Using an SRF in a place where it is not allowed results in an error:

```sql
CREATE TABLE t (start INT, stop INT, step INT);

SELECT generate_series(start, stop, step) FROM t;
-- ERROR:  set-valued function called in context that cannot accept a set
```

SRFs in FROM clause

SRFs can be used in the `FROM` clause, as shown in the first example on this page

```sql
SELECT * FROM generate_series(3, 100, 29);
```

```sql
generate_series
----------------
              3
             32
             61
             90
(4 rows)
```

SRFs in Joins

SRFs can be used in joins. In the following examples, we refer to table t:

```sql
CREATE TABLE t (start INT, stop INT, step INT);

INSERT INTO t VALUES (3, 100, 29), (7, 8, 9), (3, 2, 1);
```

The arguments from table t can be forwarded to generate_series by performing a join:

```sql
SELECT * FROM t JOIN generate_series(start, stop, step) ON TRUE;
```

```sql
 start | stop | step | generate_series
-------+------+------+-----------------
     3 |  100 |   29 |               3
     3 |  100 |   29 |              32
     3 |  100 |   29 |              61
     3 |  100 |   29 |              90
     7 |    8 |    9 |               7
(5 rows)
```

When using an SRF in a `JOIN`, a condition must be provided, which is why there is an `ON TRUE` part at the end of the query.

There is another way to write queries with no join condition that is more concise but less powerful. The previous query can be rewritten as shown below, giving the same results:

```sql
SELECT * FROM t, generate_series(start, stop, step);
```

```sql
 start | stop | step | generate_series
-------+------+------+-----------------
     3 |  100 |   29 |               3
     3 |  100 |   29 |              32
     3 |  100 |   29 |              61
     3 |  100 |   29 |              90
     7 |    8 |    9 |               7
(5 rows)
```

In the previous examples, the generate_series call for the row with `(start = 3, stop = 2, step = 1)` did not generate any values and thus the join did not generate any values for that row.

Different behavior can be achieved by using a different kind of join, such as a left join:

```sql
SELECT * FROM t LEFT JOIN generate_series(start, stop, step) ON TRUE;
```

```sql
 start | stop | step | generate_series
-------+------+------+-----------------
     3 |  100 |   29 |               3
     3 |  100 |   29 |              32
     3 |  100 |   29 |              61
     3 |  100 |   29 |              90
     7 |    8 |    9 |               7
     3 |    2 |    1 |
(6 rows)
```

Supported Functions and Return Types

This topic contains an alphabetical list of supported functions, their return types, and function category. See the linked sections for more details and examples of each function.

Function	Return Type	Category
ABS	SMALLINT	Mathematical
ACOS	DOUBLE	Mathematical
ADD_MONTHS	Depends on input type	Datetime
AGE	INTERVAL	Datetime
ASCII	INTEGER	String
ASIN	DOUBLE	Mathematical
AVG	Depends on input type	Aggregate, Window
BIT_LENGTH	INTEGER	String
BTRIM	VARCHAR	String
CASE	Depends on input type	Conditional Expression
CBRT	DOUBLE	Mathematical
CEIL	DOUBLE	Mathematical
CHAR_LENGTH	INTEGER	String
CHR	VARCHAR	String
COALESCE	Depends on input type	Conditional Expression
CONCAT	VARCHAR	String
CONCAT_WS	VARCHAR	String
COS	DOUBLE	Mathematical
CONTAINS	BOOLEAN	Network Address
COUNT	BIGINT	Aggregate, Window
COUNT(DISTINCT)	BIGINT	Aggregate
CUME_DIST	DOUBLE	Window
CURRENT_DATABASE()	NAME	System
CURRENT_DATE	DATE	Datetime

Function	Return Type	Category
CURRENT_QUERY()	VARCHAR	System
CURRENT_SCHEMA	NAME	System
CURRENT_SCHEMAS()	NAME	System
CURRENT_SETTING	VARCHAR	System
CURRENT_TIMESTAMP	TIMESTAMPTZ	Datetime
CURRENT_USER	NAME	System
CURRENT_UTC_TIMESTAMP()	TIMESTAMP	Datetime
DATEADD	TIMESTAMP	Datetime
DATEDIFF	DOUBLE	Datetime
DATENAME	VARCHAR	Datetime
DATE_PART	DOUBLE	Datetime
DATE_TRUNC	TIMESTAMP	Datetime
DAY	DOUBLE	Datetime
DAYS_BETWEEN	DOUBLE	Datetime
DECODE	Depends on input type	Conditional Expression
DECRYPT	VARCHAR	String
DECRYPT_KS	VARCHAR	String
DEGREES	DOUBLE	Mathematical
DENSE_RANK	BIGINT	Window
DIV	DECIMAL	Mathematical
ENCODE	VARCHAR(64000)	String
ENCRYPT	VARCHAR	String
ENCRYPT_KS	VARCHAR	String
ERF	DOUBLE	Mathematical
EXP	DOUBLE	Mathematical
EXTRACT	DOUBLE	Datetime
FIRST_VALUE	Same as input type	Window
FLOAT4, FLOAT8	FLOAT4, FLOAT8	Formatting
FLOOR	DOUBLE	Mathematical
GETBIT	SMALLINT	Bit Manipulation
GETDATE	TIMESTAMP	Datetime
GETDATABASEENCODING()	NAME	System

Function	Return Type	Category
GREATEST and LEAST	Depends on input type	Conditional Expression
GROUP_CONCAT	VARCHAR	Aggregate
GROUPING	INTEGER	Aggregate
HAS_DATABASE_PRIVILEGE	BOOLEAN	System Information
HAS_FUNCTION_PRIVILEGE	BOOLEAN	System Information
HAS_KEY_PRIVILEGE	BOOLEAN	System Information
HAS_ROLE_PRIVILEGE	BOOLEAN	System Information
HAS_SCHEMA_PRIVILEGE	BOOLEAN	System Information
HAS_SYSTEM_PRIVILEGE	BOOLEAN	System Information
HAS_TABLE_PRIVILEGE	BOOLEAN	System Information
HASH	VARCHAR(128)	String
HASH4	INTEGER	String
HASH8	BIGINT	String
HMAC	VARCHAR(128)	String
HMAC_KS	VARCHAR(128)	String
IIF	Depends on input type	Conditional Expression
INITCAP	VARCHAR	String
INSTR	INTEGER	String
INT2, INT4, INT8	INT2, INT4, INT8	Formatting
INV_NORM	DOUBLE	Mathematical
IPV4_GET_INET, IPV6_GET_INET	IPV4, IPV6	Network Address
IPV4_GET_MASKBITS, IPV6_GET_MASKBITS	INTEGER	Network Address
IPV4_HOSTMASK, IPV6_HOSTMASK	IPV4, IPV6	Network Address
IPV4_INET_MERGE, IPV6_INET_MERGE	VARCHAR	Network Address
IPV4_NETMASK, IPV6_NETMASK	IPV4, IPV6	Network Address
ISNULL	Depends on input type	Conditional Expression
JSON_ARRAY_STR	VARCHAR	JSON
JSON_INGEST	VARCHAR	JSON
JSON_LOOKUP	VARCHAR	JSON
JSON_OBJECT_STR	VARCHAR	JSON
JUSTIFY_DAYS	INTERVAL	Datetime
JUSTIFY_HOURS	INTERVAL	Datetime

Function	Return Type	Category
JUSTIFY_INTERVAL	INTERVAL	Datetime
LAG	Same as input type	Window
LAST_DAY	DATE	Datetime
LAST_VALUE	Same as input type	Window
LEAD	Same as input type	Window
LEFT	VARCHAR	String
LENGTH	INTEGER	String
LISTAGG	VARCHAR	Aggregate
LN	DOUBLE	Mathematical
LOCALTIME	TIME	Datetime
LOCALTIMESTAMP	TIMESTAMP	Datetime
LOG	DOUBLE or DECIMAL	Mathematical
LOWER	VARCHAR	String
LPAD	VARCHAR	String
LTRIM	VARCHAR	String
MACADDR8_SET7BIT	MACADDR8	Network Address
MAKE_DATE	DATE	Datetime
MAKE_INTERVAL	INTERVAL	Datetime
MAKE_TIME	TIME	Datetime
MAKE_TIMESTAMP	TIMESTAMP	Datetime
MAKE_TIMESTAMPTZ	TIMESTAMPTZ	Datetime
MAX	Depends on input type	Aggregate, Window
MD5	VARCHAR	String
MEDIAN()	DOUBLE	Aggregate
MIN	Depends on input type	Aggregate, Window
MOD	Depends on input type	Mathematical
MONTH	DOUBLE	Datetime
MONTHS_BETWEEN	DOUBLE	Datetime
NEXT_DAY	DATE	Datetime
NEXTVAL	BIGINT	Sequence
NOW()	TIMESTAMPTZ	Datetime
NTILE	BIGINT	String

Function	Return Type	Category
NULLIF	Depends on input type	Conditional Expression
NVL	Depends on input type	Conditional Expression
NVL2	Depends on input type	Conditional Expression
NYSIIS	VARCHAR(6)	String
OCTET_LENGTH	INTEGER	String
OVERLAPS	BOOLEAN	Datetime
PERCENT_RANK	DOUBLE	Window
PERCENTILE_CONT	DOUBLE	Aggregate, Window
PERCENTILE_DISC	Depends on the ORDER BY expression	Aggregate, Window
POSITION	INTEGER	String
POWER	DOUBLE or DECIMAL	Mathematical
PROBNORM	DECIMAL	Mathematical
RANDOM()	DOUBLE	Mathematical
RANK	BIGINT	Window
REPEAT	CHAR or VARCHAR	String
REPLACE	VARCHAR	String
REVERSE	VARCHAR	String
RIGHT	VARCHAR	String
ROUND	DECIMAL	Mathematical
ROUND_VAR	DECIMAL	Mathematical
ROW_NUMBER	BIGINT	Window
RPAD	VARCHAR	String
RTRIM	VARCHAR	String
SAFE_TO_DATE	DATE	Type-Safe Compatibility
SAFE_TO_DOUBLE	DOUBLE	Type-Safe Compatibility
SAFE_TO_TIMESTAMP	TIMESTAMP	Type-Safe Compatibility
SESSION_USER	NAME	System Information
SIGN	DOUBLE	Mathematical
SIN	DOUBLE	Mathematical
SPLIT_PART	VARCHAR	String
SQRT	DOUBLE or DECIMAL	Mathematical
STDDEV_POP	DOUBLE or DECIMAL	Aggregate

Function	Return Type	Category
STDDEV_SAMP	DOUBLE or DECIMAL	Aggregate
STRING_AGG	VARCHAR	Aggregate
STRPOS	INTEGER	String
SUBSTR	The data type of the first argument to the function.	String
SUBSTRING	VARCHAR	String, Pattern Matching
SUM	Depends on input type	Aggregate, Window
SYS.GEN_RANDOM_UUID()	UUID	System
TEXT (IPV4 or IPV6)	VARCHAR	Network Address
TIMEOFDAY()	VARCHAR	Datetime
TIMEZONE	TIMESTAMPTZ	Datetime
TO_ASCII	VARCHAR	String
TO_CHAR	VARCHAR	Formatting
TO_DATE	DATE	Formatting
TO_HEX	VARCHAR	String
TO_NUMBER	DECIMAL	Formatting
TO_TIMESTAMP	TIMESTAMPTZ	Formatting
TRANSLATE	VARCHAR	String
TRIM	VARCHAR	String
TRUNC	DECIMAL	Mathematical
TRUNC	MACADDR, MACADDR8	Network Address
UPPER	VARCHAR	String
VAR_POP	DECIMAL or DOUBLE	Aggregate
VAR_SAMP	DECIMAL or DOUBLE	Aggregate
VERSION()	VARCHAR	System
YB_TERMINATE_SESSION	BOOLEAN	System
YEAR	DOUBLE	Datetime
YEARS_BETWEEN	DOUBLE	Datetime

Notes

The following data types are not supported as table column types; you may need to use explicit casts if functions or expressions return these types:

- TEXT
- NAME

- INTERVAL
- TIMETZ

SQL Reserved Words

Platforms: All platforms

Parent topic: SQL Reference

This section lists Yellowbrick reserved words. You can use these reserved words as identifiers in SQL statements only if they are quoted. See also SQL Identifiers.

```
ADVANCED
ALL
ANALYSE
ANALYZE
AND
ANTI
ANY
ARRAY
AS
ASC
ASYMMETRIC
AUTHORIZATION
AUTOANALYZE
AWS
AZURE
BINARY
BOTH
BUCKETS
CASE
CAST
CHECK
CLUSTERING
COLLATE
COLLATION
COLUMN
CONCURRENTLY
CONSTRAINT
CREATE
CREDENTIAL
CROSS
CURRENT_CATALOG
CURRENT_DATE
CURRENT_ROLE
CURRENT_SCHEMA
CURRENT_TIME
CURRENT_TIMESTAMP
CURRENT_USER
DEFAULT
DEFERRABLE
DESC
DISTINCT
DISTRIBUTION
DO
ELSE
END
ENDPOINT
EXCEPT
FALSE
FETCH
FIELDS
FOR
```

```
FOREIGN
FORMAT
FREEZE
FROM
FULL
GCP
GRANT
GROUP
HAVING
HLL
IGNORE
ILIKE
IN
INITIALLY
INNER
INTERSECT
INTO
IS
ISNULL
ISOLATED
JOIN
LATERAL
LEADING
LEFT
LIKE
LIMIT
LIST
LOCALTIME
LOCALTIMESTAMP
MINUS
NATURAL
NFS
NOT
NOTNULL
NULL
OBJECTS
OFFSET
ON
ONLY
OR
ORDER
OUTER
OVERLAPS
PARTITION
PARTITIONING
PATH
PLACING
PREFIX
PREM
PRIMARY
REFERENCES
REGION
RESPECT
RETENTION_AGE
RETENTION_SIZE
RETURNING
RIGHT
ROWID
SAMPLED
SELECT
SEMI
SESSION_USER
SHARED
SIMILAR
SOME
SORTING
SOURCE
SUFFIX
```

```
SYMMETRIC
TABLE
TABLESAMPLE
THEN
TO
TRAILING
TRUE
UNION
UNIQUE
UNEXPANDED
USAGE
USER
USING
VARIADIC
VERBOSE
WHEN
WHERE
WINDOW
WITH
```

Error Codes

Yellowbrick Documentation > Reference > Error Codes

Platforms: All platforms

Parent topic: Reference

This section is a reference for all of the error messages and codes returned by the system. Each class of messages is listed in its own subsection. See also Recoverable Error Codes.

Common Expected Exceptions

```
CO001    E    BufferManager
CO002    E    Metadata
YB001    E    QueryNotReady
YB002    X    OutOfMemoryInRowPacket    Not enough space reserved in RowPacket
YB003    X    MissingQueryParameterData Missing data for query parameter
YB004    X    OutOfHashMemory           Hash table too big
YB005    X    SpillingSort              Attempting to spill a sort to disc
YB006    X    TempTableOutOfMemory      Temp table exceeded max size
YB007    X    MissingTempTable          Temp table cannot be found or is not readable
YB008    X    NoSuchExecutionID         ExecutionID not found
YB009    X    NoSuchNodeID              NodeID not found
YB010    X    InvalidRegistration       Invalid worker registration.
YB011    XS   InvalidTreeTopology       The plan tree is incorrectly built
YB012    X    Parse                     Invalid input string
YB013    X    ExternalDataSocketError   External data socket error
YB014    X    InvalidBulkData           The bulk load data stream was invalid
YB015    X    InvalidBulkDataCompression The bulk load data stream was invalid (bad compression)
YB016    X    InvalidBulkDataLength     The bulk load data stream was invalid (bad length)
YB017    X    InvalidBulkDataCrc        The bulk load data stream was invalid (bad CRC)
YB018    X    ExternalConnectTimeout    External client establishing connection timed out
YB019    X    JSONInvalidKey            The key for this json object is invalid / null / not a string
YB020    E    NestingTooDeep            Nesting of statement too deep
YB021    E    SequenceError             Invalid sequence values or overflow
YB022    E    BulkClientError           Ybload Error
YB023    E    UnloadClientError         Ybunload Error
YB024    X    MaxSeqnoExceeded          Maximum number of rows per transaction exceeded
YB042    X    InjectedError             Injected Error #
YB999    X    ERRCODE_CONFIGURATION_ERROR general configuration error

# WL Error codes from the work load manager
WL001    E    ERRCODE_WORKLOAD_ADMIN_CANCELLED                          admin cancelled
WL002    E    ERRCODE_WORKLOAD_WAIT_TIME_EXCEEDED                       wait time exceeded
WL003    E    ERRCODE_WORKLOAD_EXECUTION_TIME_EXCEEDED                  execution time exceeded
WL004    E    ERRCODE_WORKLOAD_ROWCOUNT_EXCEEDED                        row count exceeded
WL005    E    ERRCODE_WORKLOAD_MEMORY_EXCEEDED                          memory exceeded
WL006    E    ERRCODE_WORKLOAD_OTEHR_ERROR                              WLM other error
WL007    E    ERRCODE_WORKLOAD_SHUTTING_DOWN                            WLM shutting down
WL008    E    ERRCODE_WORKLOAD_COMPILE_ERROR                            compile error
WL009    E    ERRCODE_WORKLOAD_CONFIG_ERROR                             WLM configuration error
WL009    E    ERRCODE_WORKLOAD_CANNOT_MOVE_QUERY                        cannot move query
WL010    E    ERRCODE_WORKLOAD_CANNOT_RESTART_QUERY                     cannot restart query
WL011    E    ERRCODE_WORKLOAD_PRIOR_QUERY_NOT_COMPLETED                prior query for session not complete
WL012    E    ERRCODE_WORKLOAD_QUERY_CANNOT_THROTTLE                    query would deadlock session on throttle
WL013    E    ERRCODE_WORKLOAD_COMPILE_SHUTDOWN                         query compilation was cancelled by shutdown signal
WL014    E    ERRCODE_WORKLOAD_RULE_ABORT                               WLM rule aborted query
WL015    E    ERRCODE_WORKLOAD_RESOURCE_POOL_FULL                       WLM resource pool queue full
WL016    E    ERRCODE_WORKLOAD_RESTART_CLIENT_WAIT                      query cancelled during restart due to client wait
```

```
WL017    E    ERRCODE_WORKLOAD_CANCEL_CLIENT_WAIT                          query cancelling and so client wait should not procee

# WM Error codes from the worker manager
WM001    E   ERRCODE_WORKER_OFFLINE                                        worker offline
WM002    E   ERRCODE_SYSTEM_NOT_READY                                      system not ready
WM003    E   ERRCODE_CANCEL_FOR_RESTART                                    Query restarting
WM004    E   ERRCODE_MISMATCHED_WORKER_ID                                  mismatched worker id
WM005    E   ERRCODE_WORKER_CONFIG_ERROR                                   worker misconfigured

# Error codes during compilation
CC001   E   ERRCODE_EXPRESSION_TOO_COMPLEX   Expression too complex
CC002   E   ERRCODE_INVALID_DATETIME_TOKEN   Invalid date or timestamp unit
CC003   E   ERRCODE_INVALID_ARGUMENT_TYPE    Invalid function argument type

# Datetime related
22008   XS  DateTimeInvalidChar Invalid character for date time pattern

# Errors thrown by the EE
EEOOM     E   ERRCODE_QUERY_OOM               Query out of memory
```

Common Unexpected Exceptions

```
Base: UnexpectedException

YU___    E    TempUnknownCode          unknown error (temporary)
YU001    X    InvalidWriteSeqno        invalid write sequence number
YU002    X    NoSuchWorkerID           invalid worker id
YU003    X    NoSuchAuthKey            invalid authorization key
YU004    X    InvalidWriteGid          invalid write gid
YU005    X    CompilerError            compiler error in Lime
YU006    X    UnexpectTopology         compiler error in PG
YU007    XS   UDFError                 Error evaluating user defined function

# Error codes for the cluster manager
CM001   E   ERRCODE_PARITY_REBUILD_FAILED    Parity rebuild failed
CM002   E   ERRCODE_CLUSTER_CONFIG_ERROR     unable to update cluster state
CM003   E   ERRCODE_FSCK_ERROR               failed to FSCK

# Error where Lime has supplied bad input to the execution engine that we can't handle
EEX01   E   InvalidTypeSystemInput    Invalid compiler input to type system

# Error codes for backup
BK001   E   ERRCODE_

# BAR2
BR001   E   ERRCODE_BAR2_NO_VALID_BACKUP                    No valid backup in a backup set
BR003   E   ERRCODE_BAR2_NO_VALID_SNAPSHOT_IN_TARGET        Target does not contain required snapshot
BR004   E   ERRCODE_BAR2_TARGET_MODIFIED                    Target has been modified
BR005   E   ERRCODE_BAR2_NOTHING_TO_RESTORE                 Target does not contain snapshot to be restored
BR006   E   ERRCODE_BAR2_CROSS_DB_VIEW                      Cannot restore cross-DB view
BR007   E   ERRCODE_BAR2_RESTORE_TO_NON_EMPTY_DB            Cannot restore to target DB
BR008   E   ERRCODE_BAR2_NO_VALID_SNAPSHOT_IN_BACKUP_SET    Cannot locate snapshot in backup set
BR011   E   ERRCODE_BAR2_INVALID_LOCATION                   Cannot find DB metadata for incremental / cumulative backup

#
# Format:
#  <error code> <flags> <name> <optional description>
#
# where flags are:
# W - warning# E - error
# X - generate exception class
# S - add string parameter exception constructors
```

Kernel Unexpected Exceptions

```
Base: UnexpectedException
KU000    XS  NotImplemented                  Feature not implemented
KUSEG    E   Segv                            Segfault
KU001    XS  InvalidConfigAttributeIndex     Invalid config attribute index
KU002    XS  InvalidConfigTypeName           Invalid config type name
KU003    XS  IOValidation                    Malformed IORequest
KU004    XS  InvalidIORequest                Malformed IORequest
KU005    XS  InvalidNVMeIORequest            Malformed NVMe IORequest
KU006    XS  FBDDriverFailure                FakeBlockDevice driver failure
KU007    XS  NVMeDriverFailure               NMVe driver failure
KU008    XS  UnexpectedTCPSocketError        Unexpected TCP socket error
KU009    X   ScheduleFailure                 Failed to schedule task
KU010    XS  IBDriverFailure                 IB driver failure
KU011    X   FileSystemMissingPrefix         Could not find file with prefix for Parity
KU012    X   FileSystemNonUniquePrefix       Too many files with prefix for Parity
KU013        FILESYSTEMOUTOFINODE            Out of inode on file system
KU016        FPGADRIVERFAILURE               FPGA driver failure
KU017        BOUNCEBUFFERDRIVERFAILURE       Bounce buffer driver failure
KUSEG        SEGV                            Segfault
KUUUU    E   UnexpectedException             Generic UnexpectedException
KUXXX    XS  FileSystemOutOfINode            Out of inode on file system
```

Kernel Expected Exceptions

```
Base: ExpectedException
KE001    E   MissingFile                      File not found
KE002    E   OutOfMemory                     Out of memory
KE003    X   ShuttingDown                    Shutting down
KE004    X   QueueFull                       Queue full
KE005    X   InvalidTopology                 Invalid system topology
KE006    XS  InvalidConfig                   Invalid config
KE007    E   MissingID                       Missing id
KE008    E   UnknownConfigAttribute          Unkown config attribute
KE009    E   ConfigParsingError              Config parsing error
KE010    X   FileRead                        File read exception
KE011    X   FileWrite                       File write exception
KE012    X   OutOfRange                      Out of range
KE013    E   ArithmeticError                 Arithmetic error
KE014    X   SystemBusy                      System busy
KE015    X   FileSystemSpaceFull             File system full
KE016    X   FileSystemCorrupt               File system corrupt
KE017    X   FileSystemMissingFile           Missing file
KE018    X   FileSystemMetadataFull          Metadata storage full
KE019    X   FileSystemOutOfBounds           File system out of bounds
KE020    X   FileSystemBusy                  File system busy
KE021    X   FileSystemDuplicateName         File system duplicate name
KE022    XS  TCPSocketError                  TCP socket error
KE023    XS  RPCProtocolError                RPC protocol error
KE024    X   BadArguments                    Bad arguments
KE025    X   FileSystemUnsupportedDeviceSize Unsupported device size
KE026    X   MissingShard                    Missing one or more shards from FSCK
KE027    X   MissingKRI                      Missing one or more KRI from FSCK
KE028    X   UnreconstructableFile           File is unreconstructable
KE029    X   SpillLimitsExceeded            WLM assigned limits exceeded during spill
KE030    X   FileNotVisibleInGC             Given file is not visible to GC transaction
KE031    X   FileSystemSpillFull            File system space reserved for spilling full
KE032    X   SpillLimitExceeded             Attempt to use spill space above limit
KE033    x   BadTimeZone                     Bad Time Zone passed to query
KE036        TCPSOCKETTIMEOUT                TCP socket timeout
KE038        RPCCHANNELCLOSED                RPC channel closed
KE039        RPCCHANNELBROKEN                RPC channel broken
```

Class 00 - Successful Completion

```
00000    S   ERRCODE_SUCCESSFUL_COMPLETION                          successful_completion
```

Class 01 - Warning

```
# do not use this class for failure conditions
01000    W   ERRCODE_WARNING                                        warning
0100C    W   ERRCODE_WARNING_DYNAMIC_RESULT_SETS_RETURNED           dynamic_result_sets_returned
01008    W   ERRCODE_WARNING_IMPLICIT_ZERO_BIT_PADDING              implicit_zero_bit_padding
01003    W   ERRCODE_WARNING_NULL_VALUE_ELIMINATED_IN_SET_FUNCTION  null_value_eliminated_in_set_function
01007    W   ERRCODE_WARNING_PRIVILEGE_NOT_GRANTED                  privilege_not_granted
01006    W   ERRCODE_WARNING_PRIVILEGE_NOT_REVOKED                  privilege_not_revoked
01004    W   ERRCODE_WARNING_STRING_DATA_RIGHT_TRUNCATION           string_data_right_truncation
01P01    W   ERRCODE_WARNING_DEPRECATED_FEATURE                     deprecated_feature
```

Class 02 - No Data (also a warning class per the SQL standard)

```
# do not use this class for failure conditions
02000    W   ERRCODE_NO_DATA                                        no_data
02001    W   ERRCODE_NO_ADDITIONAL_DYNAMIC_RESULT_SETS_RETURNED     no_additional_dynamic_result_sets_returned
```

Class 03 - SQL Statement Not Yet Complete

```
03000    E   ERRCODE_SQL_STATEMENT_NOT_YET_COMPLETE                sql_statement_not_yet_complete
```

Class 08 - Connection Exception

```
08000    E   ERRCODE_CONNECTION_EXCEPTION                          connection_exception
08003    E   ERRCODE_CONNECTION_DOES_NOT_EXIST                     connection_does_not_exist
08006    E   ERRCODE_CONNECTION_FAILURE                            connection_failure
08001    E   ERRCODE_SQLCLIENT_UNABLE_TO_ESTABLISH_SQLCONNECTION   sqlclient_unable_to_establish_sqlconnection
08004    E   ERRCODE_SQLSERVER_REJECTED_ESTABLISHMENT_OF_SQLCONNECTION  sqlserver_rejected_establishment_of_sqlconnection
08007    E   ERRCODE_TRANSACTION_RESOLUTION_UNKNOWN                transaction_resolution_unknown
08P01    E   ERRCODE_PROTOCOL_VIOLATION                            protocol_violation
08P02    E   ERRCODE_INVALID_EXECID                                invalid_execution_id
```

Class 09 - Triggered Action Exception

```
09000    E   ERRCODE_TRIGGERED_ACTION_EXCEPTION                    triggered_action_exception
```

Class 0A - Feature Not Supported

```
0A000    E   ERRCODE_FEATURE_NOT_SUPPORTED                          feature_not_supported
```

Class 0B - Invalid Transaction Initiation

```
0B000    E   ERRCODE_INVALID_TRANSACTION_INITIATION                 invalid_transaction_initiation
```

Class 0F - Locator Exception

```
0F000    E   ERRCODE_LOCATOR_EXCEPTION                             locator_exception
0F001    E   ERRCODE_L_E_INVALID_SPECIFICATION                     invalid_locator_specification
```

Class 0L - Invalid Grantor

```
0L000    E   ERRCODE_INVALID_GRANTOR                               invalid_grantor
0LP01    E   ERRCODE_INVALID_GRANT_OPERATION                       invalid_grant_operation
```

Class 0P - Invalid Role Specification

```
0P000    E   ERRCODE_INVALID_ROLE_SPECIFICATION                    invalid_role_specification
```

Class 0Z - Diagnostics Exception

```
0Z000    E   ERRCODE_DIAGNOSTICS_EXCEPTION                         diagnostics_exception
0Z002    E   ERRCODE_STACKED_DIAGNOSTICS_ACCESSED_WITHOUT_ACTIVE_HANDLER   stacked_diagnostics_accessed_without_active_handler
```

Class 20 - Case Not Found

```
20000    E   ERRCODE_CASE_NOT_FOUND                                case_not_found
```

Class 21 - Cardinality Violation

```
# this means something returned the wrong number of rows
21000    E   ERRCODE_CARDINALITY_VIOLATION                         more_than_one_row_returned_by_a_sub-select_used_as_an_e
21001    E   ERRCODE_CARDINALITY_DUPLICATE                         data_contains_duplicate_values
```

Class 22 - Data Exception

```
22000    E   ERRCODE_DATA_EXCEPTION                                data_exception
2202E    E   ERRCODE_ARRAY_ELEMENT_ERROR
```

```
2202E    E   ERRCODE_ARRAY_SUBSCRIPT_ERROR                          array_subscript_error
22021    E   ERRCODE_CHARACTER_NOT_IN_REPERTOIRE                    character_not_in_repertoire
22008    E   ERRCODE_DATETIME_FIELD_OVERFLOW                        datetime_field_overflow
22031    E   ERRCODE_DATETIME_VALUE_OUT_OF_RANGE                    datetime value out of range
22012    E   ERRCODE_DIVISION_BY_ZERO                               division_by_zero
22005    E   ERRCODE_ERROR_IN_ASSIGNMENT                           error_in_assignment
2200B    E   ERRCODE_ESCAPE_CHARACTER_CONFLICT                      escape_character_conflict
22022    E   ERRCODE_INDICATOR_OVERFLOW                            indicator_overflow
22015    E   ERRCODE_INTERVAL_FIELD_OVERFLOW                       interval_field_overflow
2201E    E   ERRCODE_INVALID_ARGUMENT_FOR_LOG                       invalid_argument_for_logarithm
22014    E   ERRCODE_INVALID_ARGUMENT_FOR_NTILE                    invalid_argument_for_ntile_function
22016    E   ERRCODE_INVALID_ARGUMENT_FOR_NTH_VALUE                invalid_argument_for_nth_value_function
2201F    E   ERRCODE_INVALID_ARGUMENT_FOR_POWER_FUNCTION            invalid_argument_for_power_function
2201G    E   ERRCODE_INVALID_ARGUMENT_FOR_WIDTH_BUCKET_FUNCTION    invalid_argument_for_width_bucket_function
22018    E   ERRCODE_INVALID_CHARACTER_VALUE_FOR_CAST              invalid_character_value_for_cast
22007    E   ERRCODE_INVALID_DATETIME_FORMAT                       invalid_datetime_format
22019    E   ERRCODE_INVALID_ESCAPE_CHARACTER                      invalid_escape_character
2200D    E   ERRCODE_INVALID_ESCAPE_OCTET                          invalid_escape_octet
22025    E   ERRCODE_INVALID_ESCAPE_SEQUENCE                       invalid_escape_sequence
22P06    E   ERRCODE_NONSTANDARD_USE_OF_ESCAPE_CHARACTER           nonstandard_use_of_escape_character
22010    E   ERRCODE_INVALID_INDICATOR_PARAMETER_VALUE             invalid_indicator_parameter_value
22023    E   ERRCODE_INVALID_PARAMETER_VALUE                      invalid_parameter_value
2201B    E   ERRCODE_INVALID_REGULAR_EXPRESSION                    invalid_regular_expression
2201W    E   ERRCODE_INVALID_ROW_COUNT_IN_LIMIT_CLAUSE             invalid_row_count_in_limit_clause
2201X    E   ERRCODE_INVALID_ROW_COUNT_IN_RESULT_OFFSET_CLAUSE     invalid_row_count_in_result_offset_clause
2202H    E   ERRCODE_INVALID_TABLESAMPLE_ARGUMENT                  invalid_tablesample_argument
2202G    E   ERRCODE_INVALID_TABLESAMPLE_REPEAT                    invalid_tablesample_repeat
22009    E   ERRCODE_INVALID_TIME_ZONE_DISPLACEMENT_VALUE          invalid_time_zone_displacement_value
2200C    E   ERRCODE_INVALID_USE_OF_ESCAPE_CHARACTER              invalid_use_of_escape_character
2200G    E   ERRCODE_MOST_SPECIFIC_TYPE_MISMATCH                   most_specific_type_mismatch
22004    E   ERRCODE_NULL_VALUE_NOT_ALLOWED                        null_value_not_allowed
22002    E   ERRCODE_NULL_VALUE_NO_INDICATOR_PARAMETER             null_value_no_indicator_parameter
22003    E   ERRCODE_NUMERIC_VALUE_OUT_OF_RANGE                    numeric_value_out_of_range
22026    E   ERRCODE_STRING_DATA_LENGTH_MISMATCH                  string_data_length_mismatch
22001    E   ERRCODE_STRING_DATA_RIGHT_TRUNCATION                 string_data_right_truncation
22011    E   ERRCODE_SUBSTRING_ERROR                              substring_error
22027    E   ERRCODE_TRIM_ERROR                                    trim_error
22024    E   ERRCODE_UNTERMINATED_C_STRING                        unterminated_c_string
2200F    E   ERRCODE_ZERO_LENGTH_CHARACTER_STRING                 zero_length_character_string
22P01    E   ERRCODE_FLOATING_POINT_EXCEPTION                     floating_point_exception
22P02    E   ERRCODE_INVALID_TEXT_REPRESENTATION                 invalid_text_representation
22P03    E   ERRCODE_INVALID_BINARY_REPRESENTATION               invalid_binary_representation
22P04    E   ERRCODE_BAD_COPY_FILE_FORMAT                        bad_copy_file_format
22P05    E   ERRCODE_UNTRANSLATABLE_CHARACTER                    untranslatable_character
2200L    E   ERRCODE_NOT_AN_XML_DOCUMENT                         not_an_xml_document
2200M    E   ERRCODE_INVALID_XML_DOCUMENT                        invalid_xml_document
2200N    E   ERRCODE_INVALID_XML_CONTENT                         invalid_xml_content
2200S    E   ERRCODE_INVALID_XML_COMMENT                         invalid_xml_comment
2200T    E   ERRCODE_INVALID_XML_PROCESSING_INSTRUCTION          invalid_xml_processing_instruction
22030    E   ERRCODE_RANGE_PARTITION_VALUE_ERROR                 inserted_value_is_out_of_bounds_of_a_range-partition
```

Class 23 - Integrity Constraint Violation

```
23000    E   ERRCODE_INTEGRITY_CONSTRAINT_VIOLATION    integrity_constraint_violation
23001    E   ERRCODE_RESTRICT_VIOLATION                restrict_violation
23502    E   ERRCODE_NOT_NULL_VIOLATION                not_null_violation
23503    E   ERRCODE_FOREIGN_KEY_VIOLATION             foreign_key_violation
23505    E   ERRCODE_UNIQUE_VIOLATION                  unique_violation
23514    E   ERRCODE_CHECK_VIOLATION                   check_violation
23P01    E   ERRCODE_EXCLUSION_VIOLATION               exclusion_violation
```

Class 24 - Invalid Cursor State

```
24000    E   ERRCODE_INVALID_CURSOR_STATE                                invalid_cursor_state
```

Class 25 - Invalid Transaction State

```
25000    E   ERRCODE_INVALID_TRANSACTION_STATE                           invalid_transaction_state
25001    E   ERRCODE_ACTIVE_SQL_TRANSACTION                              active_sql_transaction
25002    E   ERRCODE_BRANCH_TRANSACTION_ALREADY_ACTIVE                   branch_transaction_already_active
25008    E   ERRCODE_HELD_CURSOR_REQUIRES_SAME_ISOLATION_LEVEL          held_cursor_requires_same_isolation_level
25003    E   ERRCODE_INAPPROPRIATE_ACCESS_MODE_FOR_BRANCH_TRANSACTION    inappropriate_access_mode_for_branch_transaction
25004    E   ERRCODE_INAPPROPRIATE_ISOLATION_LEVEL_FOR_BRANCH_TRANSACTION inappropriate_isolation_level_for_branch_transaction
25005    E   ERRCODE_NO_ACTIVE_SQL_TRANSACTION_FOR_BRANCH_TRANSACTION    no_active_sql_transaction_for_branch_transaction
25006    E   ERRCODE_READ_ONLY_SQL_TRANSACTION                           read_only_sql_transaction
25007    E   ERRCODE_SCHEMA_AND_DATA_STATEMENT_MIXING_NOT_SUPPORTED     schema_and_data_statement_mixing_not_supported
25P01    E   ERRCODE_NO_ACTIVE_SQL_TRANSACTION                          no_active_sql_transaction
25P02    E   ERRCODE_IN_FAILED_SQL_TRANSACTION                         in_failed_sql_transaction
```

Class 26 - Invalid SQL Statement Name

```
# (prepared statements)
26000    E   ERRCODE_INVALID_SQL_STATEMENT_NAME                        invalid_sql_statement_name
```

Class 27 - Triggered Data Change Violation

```
27000    E   ERRCODE_TRIGGERED_DATA_CHANGE_VIOLATION                   triggered_data_change_violation
```

Class 28 - Invalid Authorization Specification

```
28000    E   ERRCODE_INVALID_AUTHORIZATION_SPECIFICATION               invalid_authorization_specification
28P01    E   ERRCODE_INVALID_PASSWORD                                   invalid_password
```

Class 2B - Dependent Privilege Descriptors Still Exist

```
2B000    E   ERRCODE_DEPENDENT_PRIVILEGE_DESCRIPTORS_STILL_EXIST       dependent_privilege_descriptors_still_exist
2BP01    E   ERRCODE_DEPENDENT_OBJECTS_STILL_EXIST                     dependent_objects_still_exist
```

Class 2D - Invalid Transaction Termination

```
2D000    E   ERRCODE_INVALID_TRANSACTION_TERMINATION                  invalid_transaction_termination
```

Class 2F - SQL Routine Exception

```
2F000    E   ERRCODE_SQL_ROUTINE_EXCEPTION                           sql_routine_exception
2F005    E   ERRCODE_S_R_E_FUNCTION_EXECUTED_NO_RETURN_STATEMENT     function_executed_no_return_statement
```

```
2F002    E   ERRCODE_S_R_E_MODIFYING_SQL_DATA_NOT_PERMITTED              modifying_sql_data_not_permitted
2F003    E   ERRCODE_S_R_E_PROHIBITED_SQL_STATEMENT_ATTEMPTED           prohibited_sql_statement_attempted
2F004    E   ERRCODE_S_R_E_READING_SQL_DATA_NOT_PERMITTED               reading_sql_data_not_permitted
```

Class 34 - Invalid Cursor Name

```
34000    E   ERRCODE_INVALID_CURSOR_NAME                               invalid_cursor_name
```

Class 38 - External Routine Exception

```
38000    E    ERRCODE_EXTERNAL_ROUTINE_EXCEPTION                        external_routine_exception
38001    E    ERRCODE_E_R_E_CONTAINING_SQL_NOT_PERMITTED                containing_sql_not_permitted
38002    E    ERRCODE_E_R_E_MODIFYING_SQL_DATA_NOT_PERMITTED            modifying_sql_data_not_permitted
38003    E    ERRCODE_E_R_E_PROHIBITED_SQL_STATEMENT_ATTEMPTED          prohibited_sql_statement_attempted
38004    E    ERRCODE_E_R_E_READING_SQL_DATA_NOT_PERMITTED              reading_sql_data_not_permitted
```

Class 39 - External Routine Invocation Exception

```
39000   E    ERRCODE_EXTERNAL_ROUTINE_INVOCATION_EXCEPTION             external_routine_invocation_exception
39001    E   ERRCODE_E_R_I_E_INVALID_SQLSTATE_RETURNED                 invalid_sqlstate_returned
39004    E   ERRCODE_E_R_I_E_NULL_VALUE_NOT_ALLOWED                   null_value_not_allowed
39P01    E   ERRCODE_E_R_I_E_TRIGGER_PROTOCOL_VIOLATED                 trigger_protocol_violated
39P02    E   ERRCODE_E_R_I_E_SRF_PROTOCOL_VIOLATED                     srf_protocol_violated
39P03    E   ERRCODE_E_R_I_E_EVENT_TRIGGER_PROTOCOL_VIOLATED          event_trigger_protocol_violated
```

Class 3B - Savepoint Exception

```
3B000    E   ERRCODE_SAVEPOINT_EXCEPTION                              savepoint_exception
3B001    E   ERRCODE_S_E_INVALID_SPECIFICATION                        invalid_savepoint_specification Section:
```

Class 3D - Invalid Catalog Name

```
3D000    E   ERRCODE_INVALID_CATALOG_NAME                             invalid_catalog_name
```

Class 3F - Invalid Schema Name

```
3F000    E   ERRCODE_INVALID_SCHEMA_NAME                              invalid_schema_name
```

Class 40 - Transaction Rollback

```
40000    E   ERRCODE_TRANSACTION_ROLLBACK                             transaction_rollback
40002    E   ERRCODE_T_R_INTEGRITY_CONSTRAINT_VIOLATION               transaction_integrity_constraint_violation
40001    E   ERRCODE_T_R_SERIALIZATION_FAILURE                        serialization_failure
```

```
40003    E   ERRCODE_T_R_STATEMENT_COMPLETION_UNKNOWN                statement_completion_unknown
40P01    E   ERRCODE_T_R_DEADLOCK_DETECTED                           deadlock_detected
```

Class 42 - Syntax Error or Access Rule Violation

```
42000    E   ERRCODE_SYNTAX_ERROR_OR_ACCESS_RULE_VIOLATION           syntax_error_or_access_rule_violation
42601    E   ERRCODE_SYNTAX_ERROR                                    syntax_error
42501    E   ERRCODE_INSUFFICIENT_PRIVILEGE                         insufficient_privilege
42846    E   ERRCODE_CANNOT_COERCE                                   cannot_coerce
42803    E   ERRCODE_GROUPING_ERROR                                 grouping_error
42P20    E   ERRCODE_WINDOWING_ERROR                                 windowing_error
42P19    E   ERRCODE_INVALID_RECURSION                               invalid_recursion
42830    E   ERRCODE_INVALID_FOREIGN_KEY                            invalid_foreign_key
42602    E   ERRCODE_INVALID_NAME                                    invalid_name
42622    E   ERRCODE_NAME_TOO_LONG                                   name_too_long
42939    E   ERRCODE_RESERVED_NAME                                   reserved_name
42804    E   ERRCODE_DATATYPE_MISMATCH                               datatype_mismatch
42P18    E   ERRCODE_INDETERMINATE_DATATYPE                          indeterminate_datatype
42P21    E   ERRCODE_COLLATION_MISMATCH                              collation_mismatch
42P22    E   ERRCODE_INDETERMINATE_COLLATION                        indeterminate_collation
42809    E   ERRCODE_WRONG_OBJECT_TYPE                               wrong_object_type

# Note: for ERRCODE purposes, we divide nameable objects into these categories:
# databases, schemas, prepared statements, cursors, tables, columns,
# functions (including operators), and all else ("objects").
# (The first four categories are mandated by the existence of separate
# SQLSTATE classes; in this file, however, we group
# the ERRCODE names with all the rest under class 42.) Parameters are
# sort-of-named objects and get their own ERRCODE.
#
# The same breakdown is used for "duplicate" and "ambiguous" complaints,
# as well as complaints associated with incorrect declarations.

42703    E   ERRCODE_UNDEFINED_COLUMN                               undefined_column
34000    E   ERRCODE_UNDEFINED_CURSOR
3D000    E   ERRCODE_UNDEFINED_DATABASE
42883    E   ERRCODE_UNDEFINED_FUNCTION                             undefined_function
26000    E   ERRCODE_UNDEFINED_PSTATEMENT
3F000    E   ERRCODE_UNDEFINED_SCHEMA
42P01    E   ERRCODE_UNDEFINED_TABLE                                undefined_table
42P02    E   ERRCODE_UNDEFINED_PARAMETER                            undefined_parameter
42P23    E   ERRCODE_RELATION_NOT_FOUND                            relation_not_found
42704    E   ERRCODE_UNDEFINED_OBJECT                               undefined_object
42701    E   ERRCODE_DUPLICATE_COLUMN                               duplicate_column
42P03    E   ERRCODE_DUPLICATE_CURSOR                               duplicate_cursor
42P04    E   ERRCODE_DUPLICATE_DATABASE                             duplicate_database
42723    E   ERRCODE_DUPLICATE_FUNCTION                             duplicate_function
42P05    E   ERRCODE_DUPLICATE_PSTATEMENT                          duplicate_prepared_statement
42P06    E   ERRCODE_DUPLICATE_SCHEMA                              duplicate_schema
42P07    E   ERRCODE_DUPLICATE_TABLE                               duplicate_table
42712    E   ERRCODE_DUPLICATE_ALIAS                                duplicate_alias
42710    E   ERRCODE_DUPLICATE_OBJECT                               duplicate_object
42711    E   ERRCODE_DUPLICATE_TARGET_ROW                          duplicate_target_row
42702    E   ERRCODE_AMBIGUOUS_COLUMN                               ambiguous_column
42725    E   ERRCODE_AMBIGUOUS_FUNCTION                             ambiguous_function
42P08    E   ERRCODE_AMBIGUOUS_PARAMETER                            ambiguous_parameter
42P09    E   ERRCODE_AMBIGUOUS_ALIAS                                ambiguous_alias
42P10    E   ERRCODE_INVALID_COLUMN_REFERENCE                      invalid_column_reference
42611    E   ERRCODE_INVALID_COLUMN_DEFINITION                     invalid_column_definition
42P11    E   ERRCODE_INVALID_CURSOR_DEFINITION                     invalid_cursor_definition
42P12    E   ERRCODE_INVALID_DATABASE_DEFINITION                   invalid_database_definition
42P13    E   ERRCODE_INVALID_FUNCTION_DEFINITION                   invalid_function_definition
42P14    E   ERRCODE_INVALID_PSTATEMENT_DEFINITION                invalid_prepared_statement_definition
42P15    E   ERRCODE_INVALID_SCHEMA_DEFINITION                     invalid_schema_definition
```

```
42P16    E    ERRCODE_INVALID_TABLE_DEFINITION                    invalid_table_definition
42P17    E    ERRCODE_INVALID_OBJECT_DEFINITION                   invalid_object_definition
42P42    E    ERRCODE_BACKEND_ONLY                                backend_only
```

Class 44 - WITH CHECK OPTION Violation

```
44000    E    ERRCODE_WITH_CHECK_OPTION_VIOLATION                 with_check_option_violation
```

Class 53 - Insufficient Resources

```
# (PostgreSQL-specific error class)
53000    E    ERRCODE_INSUFFICIENT_RESOURCES                      insufficient_resources
53100    E    ERRCODE_DISK_FULL                                   disk_full
53200    E    ERRCODE_OUT_OF_MEMORY                               out_of_memory
53300    E    ERRCODE_TOO_MANY_CONNECTIONS                        too_many_connections
53400    E    ERRCODE_CONFIGURATION_LIMIT_EXCEEDED                configuration_limit_exceeded
```

Class 54 - Program Limit Exceeded

```
# this is for wired-in limits, not resource exhaustion problems
54000    E    ERRCODE_PROGRAM_LIMIT_EXCEEDED                      program_limit_exceeded
54001    E    ERRCODE_STATEMENT_TOO_COMPLEX                       statement_too_complex
54011    E    ERRCODE_TOO_MANY_COLUMNS                            too_many_columns
54023    E    ERRCODE_TOO_MANY_ARGUMENTS                          too_many_arguments
```

Class 55 - Object Not In Prerequisite State

```
55000    E    ERRCODE_OBJECT_NOT_IN_PREREQUISITE_STATE            object_not_in_prerequisite_state
55006    E    ERRCODE_OBJECT_IN_USE                               object_in_use
55P02    E    ERRCODE_CANT_CHANGE_RUNTIME_PARAM                   cant_change_runtime_param
55P03    E    ERRCODE_LOCK_NOT_AVAILABLE                          lock_not_available
```

Class 57 - Operator Intervention

```
57000    E    ERRCODE_OPERATOR_INTERVENTION                       operator_intervention
57014    E    ERRCODE_QUERY_CANCELED                              query_canceled
57P01    E    ERRCODE_ADMIN_SHUTDOWN                              admin_shutdown
57P02    E    ERRCODE_CRASH_SHUTDOWN                              crash_shutdown
57P03    E    ERRCODE_CANNOT_CONNECT_NOW                          cannot_connect_now
57P04    E    ERRCODE_DATABASE_DROPPED                            database_dropped
```

Class 58 - System Error

```
58000    E    ERRCODE_SYSTEM_ERROR                                system_error
58030    E    ERRCODE_IO_ERROR                                    io_error
58P01    E    ERRCODE_UNDEFINED_FILE                              undefined_file
58P02    E    ERRCODE_DUPLICATE_FILE                              duplicate_file
```

Class F0 - Configuration File Error

```
F0000    E   ERRCODE_CONFIG_FILE_ERROR                          config_file_error
F0001    E   ERRCODE_LOCK_FILE_EXISTS                           lock_file_exists
```

Class HV - Foreign Data Wrapper Error (SQL/MED)

```
# (SQL/MED-specific error class)
HV000    E   ERRCODE_FDW_ERROR                                  fdw_error
HV005    E   ERRCODE_FDW_COLUMN_NAME_NOT_FOUND                  fdw_column_name_not_found
HV002    E   ERRCODE_FDW_DYNAMIC_PARAMETER_VALUE_NEEDED         fdw_dynamic_parameter_value_needed
HV010    E   ERRCODE_FDW_FUNCTION_SEQUENCE_ERROR               fdw_function_sequence_error
HV021    E   ERRCODE_FDW_INCONSISTENT_DESCRIPTOR_INFORMATION   fdw_inconsistent_descriptor_information
HV024    E   ERRCODE_FDW_INVALID_ATTRIBUTE_VALUE               fdw_invalid_attribute_value
HV007    E   ERRCODE_FDW_INVALID_COLUMN_NAME                   fdw_invalid_column_name
HV008    E   ERRCODE_FDW_INVALID_COLUMN_NUMBER                 fdw_invalid_column_number
HV004    E   ERRCODE_FDW_INVALID_DATA_TYPE                     fdw_invalid_data_type
HV006    E   ERRCODE_FDW_INVALID_DATA_TYPE_DESCRIPTORS        fdw_invalid_data_type_descriptors
HV091    E   ERRCODE_FDW_INVALID_DESCRIPTOR_FIELD_IDENTIFIER  fdw_invalid_descriptor_field_identifier
HV00B    E   ERRCODE_FDW_INVALID_HANDLE                       fdw_invalid_handle
HV00C    E   ERRCODE_FDW_INVALID_OPTION_INDEX                 fdw_invalid_option_index
HV00D    E   ERRCODE_FDW_INVALID_OPTION_NAME                  fdw_invalid_option_name
HV090    E   ERRCODE_FDW_INVALID_STRING_LENGTH_OR_BUFFER_LENGTH  fdw_invalid_string_length_or_buffer_length
HV00A    E   ERRCODE_FDW_INVALID_STRING_FORMAT                fdw_invalid_string_format
HV009    E   ERRCODE_FDW_INVALID_USE_OF_NULL_POINTER          fdw_invalid_use_of_null_pointer
HV014    E   ERRCODE_FDW_TOO_MANY_HANDLES                     fdw_too_many_handles
HV001    E   ERRCODE_FDW_OUT_OF_MEMORY                        fdw_out_of_memory
HV00P    E   ERRCODE_FDW_NO_SCHEMAS                           fdw_no_schemas
HV00J    E   ERRCODE_FDW_OPTION_NAME_NOT_FOUND                fdw_option_name_not_found
HV00K    E   ERRCODE_FDW_REPLY_HANDLE                         fdw_reply_handle
HV00Q    E   ERRCODE_FDW_SCHEMA_NOT_FOUND                     fdw_schema_not_found
HV00R    E   ERRCODE_FDW_TABLE_NOT_FOUND                      fdw_table_not_found
HV00L    E   ERRCODE_FDW_UNABLE_TO_CREATE_EXECUTION           fdw_unable_to_create_execution
HV00M    E   ERRCODE_FDW_UNABLE_TO_CREATE_REPLY               fdw_unable_to_create_reply
HV00N    E   ERRCODE_FDW_UNABLE_TO_ESTABLISH_CONNECTION       fdw_unable_to_establish_connection
```

Class P0: PL/pgSQL Error

```
P0000    E   ERRCODE_PLPGSQL_ERROR                              plpgsql_error
P0001    E   ERRCODE_RAISE_EXCEPTION                            raise_exception
P0002    E   ERRCODE_NO_DATA_FOUND                              no_data_found
P0003    E   ERRCODE_TOO_MANY_ROWS                              too_many_rows
P0004    E   ERRCODE_ASSERT_FAILURE                             assert_failure
```

Class XX: Internal Error

```
# (PostgreSQL-specific error class)
XX000    E   ERRCODE_INTERNAL_ERROR                             internal_error
XX001    E   ERRCODE_DATA_CORRUPTED                             data_corrupted
XX002    E   ERRCODE_INDEX_CORRUPTED                            index_corrupted
```

Yellowbrick End User License Agreement for Software

Please read this document carefully before proceeding. By signing below or accessing or using the Yellowbrick software (the first to occur being the "**Effective Date**"), you are agreeing to the terms of this end user license agreement ("**Agreement**").

This is a legal agreement between you ("**Customer**") and Yellowbrick Data, Inc., a Delaware corporation with offices at 660 W. Dana Street, Mountain View, CA 94041 ("**Yellowbrick**"). As used herein, each of Yellowbrick and Customer may be referred to as a "**Party**" and collectively as the "**Parties**."

This Agreement governs your use of Yellowbrick's various software products, including any updates and upgrades Yellowbrick implements thereto ("**Licensed Software**").

1. License Grant and Restrictions

1.1 License Grant. Subject to the terms and conditions of this Agreement, Yellowbrick hereby grants to Customer a non-exclusive, non-sublicensable, non-assignable license to use the License Software, in object code form, provided to Customer by subscription.

1.2 Additional Restrictions. Customer shall not, and shall not permit any third party, including any parent, subsidiary, affiliate, or agent of Customer, to:

(a) assign, sell, lease, distribute, license, sublicense or otherwise transfer or attempt to transfer rights to the Licensed Software;
(b) extract, reverse engineer, decompile, disassemble or otherwise attempt to derive source code or algorithms from the Licensed Software, except to the extent expressly permitted by applicable law notwithstanding this restriction;
(c) modify, translate, or create derivative works, adaptations or compilations of, or based on, any part of the Licensed Software;
(d) remove or otherwise interfere with any part of the Licensed Software designed to monitor Customer's compliance with this Agreement;
(e) copy the Licensed Software, in whole or in part, except as specifically authorized by this Agreement;
(f) remove any proprietary notices or labels on or in any of the Licensed Software; or
(g) use the Licensed Software for operations of any type on aircraft, ships, nuclear plants, life support machines, communications systems or any other equipment in which the malfunctioning of the Licensed Software could lead to personal injury, death or environmental damage.

1.3 Reservation of Rights. Except for the limited rights granted in Section 1.1, Yellowbrick retains all right, title and interest in and to the Licensed Software, and all intellectual property rights therein and thereto. Nothing in this Agreement shall constitute a transfer of any ownership rights by Yellowbrick to Customer in the Licensed Software or otherwise. All rights in the Licensed Software not expressly granted hereunder are reserved by Yellowbrick and its licensors.

1.4 Processing of any Personal Data. To the extent that Yellowbrick receives, or is processing personal data of Customer or on Customer's behalf in a capacity as data processor, Customer will ensure that it has secured all necessary consents, registrations and notifications as may be required to enable the lawful transfer of the personal data to Yellowbrick and in order for Yellowbrick to process such personal data to the extent required for, and for the duration of, Yellowbrick's provision of the Licensed Software to the Customer. Yellowbrick's use of any such data shall be subject Yellowbrick's Privacy Policy which can be found at Yellowbrick.com.

1.5 Updates. During the term of this Agreement, Yellowbrick may update the Licensed Software to reflect changes in, for instance, laws, regulations, technology, industry practices and patterns of use. Any updates will not materially reduce the level of performance, functionality, security or availability of the Licensed Software.

2. Disclaimers and Limitations of Liability

2.1 Disclaimer. YELLOWBRICK HEREBY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE LICENSED SOFTWARE, SUBSCRIPTION SERVICES OR ANY OTHER MATERIALS AND SERVICES FURNISHED OR PROVIDED UNDER OR IN CONNECTION WITH THIS AGREEMENT, AND WITH RESPECT TO THE USE OF ANY OF THE FOREGOING. YELLOWBRICK DOES NOT WARRANT THAT EACH WILL BE ERROR-FREE OR SECURE, OR WILL WORK WITHOUT INTERRUPTIONS.

2.2 Limitation of Liability. IN NO EVENT WILL YELLOWBRICK BE LIABLE FOR ANY LOSS OF PROFITS, LOSS OF USE, BUSINESS INTERRUPTION, LOSS OF DATA, COST OF COVER, OR INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE OR CONSEQUENTIAL DAMAGES OF ANY KIND IN CONNECTION WITH OR ARISING OUT OF THE

FURNISHING, PERFORMANCE, OR USE OF THE LICENSED SOFTWARE, SUBSCRIPTION SERVICES OR ANY OTHER SERVICES, WHETHER ALLEGED AS A BREACH OF CONTRACT, TORT (INCLUDING NEGLIGENCE), OR ANY OTHER CLAIM OR CAUSE OF ACTION, EVEN IF YELLOWBRICK HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. YELLOWBRICK'S LIABILITY UNDER THIS AGREEMENT FOR DAMAGES WILL NOT, IN ANY EVENT, EXCEED THE AMOUNTS PAID BY CUSTOMER TO YELLOWBRICK FOR THE SOFTWARE DURING THE TWELVE (12) MONTHS PRECEDING THE EVENTS WHICH GAVE RISE TO THE DAMAGES.

3. Term and Termination

3.1 Term. Unless terminated pursuant to Section 3.2, this Agreement shall commence on the Effective Date and continue for the term of the Licensed Software purchased by Customer and any renewal thereof ("**Subscription Term**").

3.2 Termination for Cause. Either Party may, by providing written notice to the other Party, terminate this Agreement if the other Party is in material breach of any term, condition or provision of this Agreement, which breach, if capable of being cured, is not cured within thirty (30) days after the non-defaulting Party provides the other Party with written notice of such breach.

3.3 Effect of Termination. Upon termination of this Agreement pursuant to Section 3.2, or upon expiration of the Subscription Term, the license granted to Customer in Section 1.1 will immediately cease and Customer will have no further rights to use the Licensed Software. Subject to the following sentence, upon the effective date of expiration or termination of this Agreement any current or future payment obligations of Customer to Yellowbrick under this Agreement or order form will become due immediately. In addition, if Customer terminates this Agreement pursuant to Section 3.2, Customer shall receive a refund of any pre-paid Licensed Software fees paid to Yellowbrick for the period from the date of termination to the expiry of the Subscription Term, had this Agreement not been so terminated by Customer.

3.4 Survival. The following provisions will survive any termination of this Agreement: 1.2, 1.4, 2.2, 3 and 5.

4. Indemnity

Yellowbrick shall defend and pay damages finally awarded for, or at its option settle, any third-party claim brought against Customer to the extent it alleges that any of the Licensed Software used as authorized in this Agreement infringes any intellectual property rights of any third party; provided that Customer provides Yellowbrick with (i) prompt written notice of such claim; (ii) sole control over the defense and settlement of such claim; (iii) all information and assistance reasonably requested by Yellowbrick in connection with the defense or settlement of such claim; and (iv) not making any admission of liability, agreement or compromise in relation to any claim without the prior written consent of Yellowbrick. In the event any such claim is brought or threatened, Yellowbrick may, at its sole option and expense: (a) procure for Customer the right to continue to use the Licensed Software; (b) modify or amend the Licensed Software to make it non-infringing; (c) replace the Licensed Software with a non-infringing Licensed Software having substantially similar capabilities; or (d) if (a) – (c) are not commercially feasible, terminate this agreement immediately by notice in writing to Customer, and refund any pre-paid Licensed Software fees paid by Customer to Yellowbrick for the period from the date of termination to the expiry of the Subscription Term, had this Agreement not been so terminated by Yellowbrick. Notwithstanding the foregoing, Yellowbrick will have no liability to Customer for any claim of infringement to the extent such claim arises out of or is based upon (i) any modification of the Licensed Software not made or authorized in writing by Yellowbrick; (ii) Customer's failure to use the Licensed Software in accordance with this Agreement, or instructions provided by Yellowbrick, or otherwise using the Licensed Software for purposes for which it was not designed or intended; (iii) Licensed Software updates provided by Yellowbrick to comply with the designs, requirements or specifications requested by Customer; (iv) use of any specified release of the Licensed Software after Yellowbrick notifies Customer that continued use of such release may subject Customer to a claim of infringement, if Yellowbrick provides Customer with a replacement release; or (v) combination of the Licensed Software with other software or products not provided by Yellowbrick. THE FOREGOING STATES THE ENTIRE LIABILITY AND OBLIGATIONS OF YELLOWBRICK AND THE EXCLUSIVE REMEDY OF CUSTOMER WITH RESPECT TO ANY ALLEGED OR ACTUAL INFRINGEMENT BY THE LICENSED SOFTWARE OF A THIRD PARTY'S INTELLECTUAL PROPERTY RIGHTSPATENTS OR COPYRIGHTS.

5. Miscellaneous

5.1 Assignment. Neither this Agreement nor any rights under this Agreement may be assigned or otherwise transferred by Customer, in whole or in part without the prior written consent of Yellowbrick, which shall not be unreasonably withheld. Yellowbrick may assign or transfer this Agreement, in whole or in part. A change of control of a Party shall not be considered an assignment. Subject to the foregoing, this Agreement will be binding upon and will inure to the benefit of the Parties and their respective successors and assigns.

5.2 Relationship of Parties. Nothing contained in this Agreement shall be construed as creating any agency, partnership or other form of joint enterprise between the Parties. The relationship between the Parties shall at all times be that of independent contractors. Neither Party shall have the authority to contract for or bind the other in any manner whatsoever. This Agreement confers no rights upon either Party except those expressly granted herein.

5.3 Backup Copies. Yellowbrick is not responsible for the backup storage of Customer's data. Accordingly, Customer shall be responsible for the backup of its data and Customer assumes responsibility for any loss or damage from its failure to so maintain backup copies.

5.4 Notices. Any notice required or permitted under the terms of this Agreement or required by law must be in writing and must be (a) delivered in person, (b) sent by first class registered mail, or air mail, as appropriate or (c) sent by overnight air courier, in each case properly posted and fully prepaid to the address of the Party concerned.

5.5 Export Control. Customer agrees to comply with all applicable export control laws and regulations. Customer shall not sell, export, reexport, transfer, divert or otherwise dispose of, whether directly or indirectly, any regulated item or information to anyone outside the U.S. in connection with this Agreement without first complying with all export control laws and regulations which may be imposed by the U.S. Government and any country or organization of nations within whose jurisdiction Customer operates or does business.

5.6 Waiver. Any waiver of the provisions of this Agreement or of a Party's rights or remedies under this Agreement must be in writing to be effective. Failure, neglect or delay by a Party to enforce the provisions of this Agreement or its rights or remedies at any time will not be construed as, and will not be deemed to be, a waiver of such Party's rights under this Agreement, and will not in any way affect the validity of the whole or any part of this Agreement or prejudice such Party's right to take subsequent action. No exercise or enforcement by either Party of any right or remedy under this Agreement will preclude the enforcement by such Party of any other right or remedy under this Agreement or any other right or remedy that such Party is entitled by law to enforce.

5.7 Severability. If any term, condition or provision in this Agreement is found to be invalid, unlawful or unenforceable to any extent, such invalid term, condition or provision will be severed from the remaining terms, conditions and provisions, which will continue to be valid and enforceable to the fullest extent permitted by law.

5.8 Integration. This Agreement contains the entire agreement of the Parties with respect to the subject matter of this Agreement and supersedes all previous communications, representations, understandings and agreements, either oral or written, between the Parties with respect to said subject matter. This Agreement may not be amended, except by a writing signed by both Parties.

5.9 Governing Law. This Agreement will be interpreted and construed in accordance with the laws of the State of California and the United States of America, without regard to conflict of law principles. The Parties agree that the United Nations Convention on Contracts for the International Sale of Goods is specifically excluded from application to this Agreement. Any judicial action or proceeding arising hereunder or relating hereto shall be brought in, and the Parties hereby consent to the exclusive personal jurisdiction of, the state and federal courts located in Santa Clara County, California.

5.10 Federal Government Customers. Licensed Software and the associated rights under this Agreement to use the Licensed Software include only those rights customarily provided to the public. This customary commercial license is provided in accordance with FAR 12.211 (Technical Data) and FAR 12.212 (Software) and, for Department of Defense transactions, DFAR 252.227-7015 (Technical Data – Commercial Items) and DFAR 227.7202-3 (Rights in Commercial Computer Software or Computer Software Documentation). If a government agency wishes to receive additional rights, Yellowbrick will enter into good faith negotiations with the agency to determine whether it can accommodate the agency's request. Any such accommodation must be included in an addendum to this Agreement.

Glossary

Atomicity, Consistency, Isolation, and Durability (ACID)
The four key properties that ensure reliable database transactions.

availability zone (AZ)
Instances are hosted in multiple locations worldwide. An AZ is a set of isolated locations within a region. A *region* is a separate geographic area.

Azure Blob Storage (ABS)
Azure service used to allocate permanent storage volumes for Yellowbrick instances.

Azure Kubernetes Service (AKS) cluster
An AKS control plane and a set of nodes that run Kubernetes software. The cluster runs in an account managed by Azure on its own set of Azure virtual machine instances.

bulk loader (ybload)
The client program used for running bulk loads on Yellowbrick tables.

capacity expansion
The process of adding nodes to a cluster.

CloudFormation
AWS service that deploys software stacks for cloud applications, using standard templates and web-based forms.

cluster
Virtual compute cluster, which runs queries and other operations. See compute clusters.

column-level encryption
Automatic encryption of sensitive data loaded into specific columns, as defined in the CREATE TABLE statement.

column store
Persistent columnar storage of table rows in shards on the compute node file systems.

compute cluster
See compute clusters.

compute node
A node in a virtual compute cluster, backed by an appliance blade or cloud-based hardware of a specific instance type.

compute processes
Processes launched on the compute nodes. In general, Yellowbrick queries use massively parallel processing, with all of the nodes sharing the work. One process on a given node is elected to run the final phase of a given query.

cross-database queries
Queries that reference tables and views in different databases.

data warehouse instance
A deployed and provisioned data warehouse that runs in a cloud environment, typically with limits imposed on its hardware resources.

database
A physical SQL database created in a Yellowbrick Data Warehouse instance.

distribution key

A column in a table that is declared in the CREATE TABLE statement as the key for distributing data evenly among the analytic blades.

DNS zone
Domain Name System (DNS) web service that connects user requests to internet applications running on Azure. A DNS zone assigns a custom domain name in your VPC, and provides access to that domain, making use of internal Azure resources and servers.

Elastic Block Storage (EBS)
AWS service used to allocate permanent storage volumes for Yellowbrick instances.

EC2 instance
A virtual server in Amazon's Elastic Compute Cloud (EC2) for running applications on the AWS infrastructure.

EKS cluster
An Elastic Kubernetes Service (EKS) control plane and a set of nodes that run Kubernetes software. The cluster runs in an account managed by AWS on its own set of Amazon EC2 instances.

flex pool
A WLM resource pool with flexible concurrency settings.

flushing
Background operation that periodically moves table rows from the row store to the column store.

object storage
Remote source files supported by `ybload` and `ybunload`, such as Azure Blob and AWS S3.

parquet format
Apache Parquet is a binary structured data format that you can load into Yellowbrick tables. Unloads in `parquet` format are also supported.

PG
PostgreSQL front-end database.

Helm chart
A collection of files that describe a related set of Kubernetes resources, laid out in a specific directory tree and packaged for cloud deployment.

hosted zone
See Route 53 hosted zone and DNS zone.

Kubernetes
Open-source software that automates deployment, scaling, and management of containerized applications.

observability stack
The stack that supports Grafana monitoring and logging to an observability bucket for remote diagnostics.

pod
A deployable group of Kubernetes containers, with shared storage and network resources.

profile
In AWS, an account configuration with specific settings for region, role, start URL, and so on.

replicated table
A table that is copied rather than distributed across all of the nodes so that all compute node processes have immediate local access to all of its data.

Resource Manager
An Azure service that deploys software stacks for cloud applications, using standard templates and web-based forms.

Route 53 hosted zone
Route 53 is a Domain Name System (DNS) web service, connecting user requests to internet applications running on AWS. A Route 53 zone assigns a custom domain name in your VPC, and provides access to that domain, making use of internal AWS resources and servers.

row store

Temporary storage for results of `INSERT INTO...VALUES` statements and `ybsql \copy` operations. Rows are periodically flushed to the column store.

savepoint
A marker within a transaction. Transactions may be rolled back to savepoints.

sort key
A column in a table that is declared in the CREATE TABLE statement as the key for sorting data in the storage system on the analytic blades.

SSL
Secure Sockets Layer, a communications protocol for authenticated and encrypted connections over a network. See TLS.

stored procedure
Executable PL/pgSQL code that combines application logic and SQL constructs to perform actions in the database.

system table
A table in the system catalog that contains persistently stored data (data from log files). System tables describe database objects, storage data, and other metadata.

system view
A view built over one or more system tables or virtual tables. System views are available from the `sys` schema.

TLS
Transport Layer Security, a communications protocol for authenticated and encrypted connections over a network. The TLS and SSL terms tend to be used interchangeably. SSL/TLS is also used.

virtual compute cluster
See cluster.

virtual table
A table in the system catalog that is dynamically generated based on current activity on the system. These tables capture statistics for queries, CPU and memory utilization, and file system information.

ybdumproles
Yellowbrick client tool that generates DDL for users and roles.

ybdumpschema
Yellowbrick client tool for generating DDL.

ybload
Yellowbrick bulk load client tool.

ybrelay
Client tool that works with Apache Spark to load data from source files that `ybload` cannot read directly.

ybsql
Yellowbrick SQL client tool.

ybtools
Yellowbrick client tools package, available on several platforms.

ybunload
Yellowbrick client tool for unloading data.

Third-Party Licenses

Yellowbrick Documentation > Reference > Third-Party Licenses

Platforms: All platforms

Parent topic: Reference

The following third-party components are included in binary form in Yellowbrick Data Warehouse, and redistributed under the following licenses permitted by the original author/organization:

```
--------------------------------------------------------------------------------
oro:oro(2.0.8)

Apache-1.1
URL: https://app.snyk.io/vuln/snyk:lic:Apache-1.1

--------------------------------------------------------------------------------
biz.aQute:bndlib(1.50.0)
com.amazonaws:aws-java-sdk-core(1.11.613)
com.amazonaws:aws-java-sdk-kms(1.11.613)
com.amazonaws:aws-java-sdk-s3(1.11.613)
com.amazonaws:jmespath-java(1.11.613)
com.carrotsearch:hppc(0.7.2)
com.clearspring.analytics:stream(2.7.0)
com.fasterxml.jackson.core:jackson-annotations(2.12.5,2.7.3)
com.fasterxml.jackson.core:jackson-core(2.12.5,2.7.3)
com.fasterxml.jackson.core:jackson-databind(2.12.5,2.7.3)
com.fasterxml.jackson.dataformat:jackson-dataformat-cbor(2.6.7)
com.fasterxml.jackson.dataformat:jackson-dataformat-xml(2.12.4)
com.fasterxml.jackson.datatype:jackson-datatype-jsr310(2.12.4)
com.fasterxml.jackson.module:jackson-module-jaxb-annotations(2.12.4)
com.fasterxml.jackson.module:jackson-module-paranamer(2.6.5,2.7.9)
com.fasterxml.jackson.module:jackson-module-scala_2.11(2.6.5,2.6.7.1)
com.fasterxml.woodstox:woodstox-core(5.0.3,6.1.1)
com.github.stephenc.jcip:jcip-annotations(1.0-1)
com.google.code.findbugs:jsr305(3.0.1)
com.google.code.gson:gson(2.4)
com.google.guava:guava(19.0)
com.google.inject:guice(3.0)
com.googlecode.json-simple:json-simple(1.1.1)
com.ibm.async:asyncutil(unknown)
com.impossibl.pgjdbc-ng:pgjdbc-ng(0.8.8)
com.impossibl.pgjdbc-ng:spy(0.8.8)
com.jamesmurty.utils:java-xmlbuilder(0.4)
com.nimbusds:lang-tag(1.6)
com.nimbusds:nimbus-jose-jwt(4.41.1,7.9,9.23)
com.nimbusds:oauth2-oidc-sdk(6.14)
com.ning:compress-lzf(1.0.3)
com.opencsv:opencsv(4.5)
com.squareup.okhttp:okhttp(2.7.5)
com.squareup.okio:okio(1.6.0)
com.twitter:chill-java(0.8.0,0.9.3)
com.twitter:chill_2.11(0.8.0,0.9.3)
com.typesafe.netty:netty-reactive-streams(2.0.4)
com.typesafe.netty:netty-reactive-streams-http(2.0.4)
com.univocity:univocity-parsers(2.1.1,2.7.3)
com.vlkan:flatbuffers(1.2.0-3f79e055)
com.zaxxer:HikariCP(2.4.1)
com.zaxxer:HikariCP-java7(2.4.12)
commons-beanutils:commons-beanutils(1.9.3,1.9.4)
```

```
commons-beanutils:commons-beanutils-core(1.8.0)
commons-cli:commons-cli(1.2,1.4)
commons-codec:commons-codec(1.10,1.11,1.14,1.4,1.9)
commons-collections:commons-collections(3.2,3.2.1,3.2.2)
commons-configuration:commons-configuration(1.6)
commons-daemon:commons-daemon(1.0.15)
commons-digester:commons-digester(1.8)
commons-fileupload:commons-fileupload(1.3.1)
commons-httpclient:commons-httpclient(3.1)
commons-io:commons-io(1.3.2,2.1,2.2,2.4,2.5,2.8.0)
commons-lang:commons-lang(2.6)
commons-logging:commons-logging(1.1,1.1.3,1.2)
commons-net:commons-net(2.2,3.1,3.5)
commons-pool:commons-pool(1.6)
io.airlift:aircompressor(0.10)
io.dropwizard.metrics:metrics-core(3.1.0,3.1.2,4.0.6)
io.dropwizard.metrics:metrics-graphite(3.1.2,3.1.5)
io.dropwizard.metrics:metrics-jmx(4.0.6)
io.dropwizard.metrics:metrics-json(3.1.2,3.1.5)
io.dropwizard.metrics:metrics-jvm(3.1.0,3.1.2,3.1.5)
io.hawt:hawtio-core(1.4.66)
io.hawt:hawtio-karaf(1.4.66)
io.hawt:hawtio-osgi-jmx(1.4.66)
io.hawt:hawtio-system(1.4.66)
io.hawt:hawtio-util(1.4.66)
io.netty:netty(3.10.6.Final,3.8.0.Final,3.9.9.Final)
io.netty:netty-all(4.1.45.Final,4.1.49.Final)
io.netty:netty-buffer(4.1.24.Final,4.1.49.Final,4.1.53.Final,4.1.63.Final,4.1.67.Final)
io.netty:netty-codec(4.1.24.Final,4.1.49.Final,4.1.53.Final,4.1.63.Final,4.1.67.Final)
io.netty:netty-codec-dns(4.1.49.Final,4.1.66.Final)
io.netty:netty-codec-http(4.1.49.Final,4.1.53.Final,4.1.67.Final)
io.netty:netty-codec-http2(4.1.49.Final,4.1.53.Final,4.1.67.Final)
io.netty:netty-codec-socks(4.1.49.Final,4.1.67.Final)
io.netty:netty-common(4.1.24.Final,4.1.49.Final,4.1.53.Final,4.1.63.Final,4.1.67.Final)
io.netty:netty-handler(4.1.24.Final,4.1.49.Final,4.1.53.Final,4.1.63.Final,4.1.67.Final)
io.netty:netty-handler-proxy(4.1.49.Final,4.1.67.Final)
io.netty:netty-resolver(4.1.24.Final,4.1.49.Final,4.1.53.Final,4.1.63.Final,4.1.67.Final)
io.netty:netty-resolver-dns(4.1.49.Final,4.1.66.Final)
io.netty:netty-resolver-dns-native-macos(4.1.66.Final)
io.netty:netty-tcnative-boringssl-static(2.0.24.Final,2.0.40.Final)
io.netty:netty-transport(4.1.24.Final,4.1.49.Final,4.1.53.Final,4.1.63.Final,4.1.67.Final)
io.netty:netty-transport-native-epoll(4.1.49.Final,4.1.53.Final,4.1.63.Final,4.1.67.Final)
io.netty:netty-transport-native-kqueue(4.1.49.Final,4.1.63.Final,4.1.67.Final)
io.netty:netty-transport-native-unix-common(4.1.49.Final,4.1.53.Final,4.1.63.Final,4.1.67.Final)
io.projectreactor.netty:reactor-netty(1.0.10)
io.projectreactor.netty:reactor-netty-core(1.0.10)
io.projectreactor.netty:reactor-netty-http(1.0.10)
io.projectreactor.netty:reactor-netty-http-brave(1.0.10)
io.projectreactor:reactor-core(3.4.9)
io.prometheus:simpleclient(0.6.0)
io.prometheus:simpleclient_common(0.6.0)
io.prometheus:simpleclient_hotspot(0.6.0)
io.prometheus:simpleclient_pushgateway(0.6.0)
io.quarkus.arc:arc(1.9.2.Final)
io.quarkus.security:quarkus-security(1.1.3.Final)
io.quarkus:quarkus-arc(1.9.2.Final)
io.quarkus:quarkus-bootstrap-runner(1.9.2.Final)
io.quarkus:quarkus-core(1.9.2.Final)
io.quarkus:quarkus-development-mode-spi(1.9.2.Final)
io.quarkus:quarkus-ide-launcher(1.9.2.Final)
io.quarkus:quarkus-jsonb(1.9.2.Final)
io.quarkus:quarkus-jsonp(1.9.2.Final)
io.quarkus:quarkus-netty(1.9.2.Final)
io.quarkus:quarkus-resteasy(1.9.2.Final)
io.quarkus:quarkus-resteasy-common(1.9.2.Final)
io.quarkus:quarkus-resteasy-jsonb(1.9.2.Final)
io.quarkus:quarkus-resteasy-server-common(1.9.2.Final)
io.quarkus:quarkus-security-runtime-spi(1.9.2.Final)
```

```
io.quarkus:quarkus-vertx-core(1.9.2.Final)
io.quarkus:quarkus-vertx-http(1.9.2.Final)
io.smallrye.common:smallrye-common-annotation(1.4.0)
io.smallrye.common:smallrye-common-classloader(1.4.0)
io.smallrye.common:smallrye-common-constraint(1.4.0)
io.smallrye.common:smallrye-common-expression(1.4.0)
io.smallrye.common:smallrye-common-function(1.4.0)
io.smallrye.config:smallrye-config(1.9.3)
io.smallrye.config:smallrye-config-common(1.9.3)
io.smallrye.reactive:mutiny(0.9.0)
io.zipkin.brave:brave(5.13.3)
io.zipkin.brave:brave-instrumentation-http(5.13.3)
io.zipkin.reporter2:zipkin-reporter(2.16.3)
io.zipkin.reporter2:zipkin-reporter-brave(2.16.3)
io.zipkin.zipkin2:zipkin(2.23.2)
jakarta.enterprise:jakarta.enterprise.cdi-api(2.0.2)
jakarta.inject:jakarta.inject-api(1.0)
jakarta.validation:jakarta.validation-api(2.0.2)
javax.enterprise:cdi-api(1.0-SP4)
javax.inject:javax.inject(1)
javax.validation:validation-api(1.1.0.Final)
joda-time:joda-time(2.8.1,2.9,2.9.9)
log4j:log4j(1.2.12,1.2.16,1.2.17)
net.java.dev.jets3t:jets3t(0.7.1,0.9.0)
net.jpountz.lz4:lz4(1.3.0)
net.minidev:accessors-smart(1.2)
net.minidev:json-smart(2.3)
net.openhft:zero-allocation-hashing(0.9)
net.sf.ehcache:ehcache(2.10.4)
net.shibboleth.utilities:java-support(7.3.0)
no.fiken.oss.junixsocket:junixsocket-common(1.0.2)
no.fiken.oss.junixsocket:junixsocket-native-common(1.0.2)
org.apache.activemq.protobuf:activemq-protobuf(1.1)
org.apache.activemq:activeio-core(3.1.4)
org.apache.activemq:activemq-broker(5.14.5)
org.apache.activemq:activemq-client(5.14.5)
org.apache.activemq:activemq-console(5.14.5)
org.apache.activemq:activemq-http(5.14.5)
org.apache.activemq:activemq-jms-pool(5.14.5)
org.apache.activemq:activemq-kahadb-store(5.14.5)
org.apache.activemq:activemq-karaf(5.14.5)
org.apache.activemq:activemq-log4j-appender(5.14.5)
org.apache.activemq:activemq-mqtt(5.14.5)
org.apache.activemq:activemq-openwire-legacy(5.14.5)
org.apache.activemq:activemq-pool(5.14.5)
org.apache.activemq:activemq-spring(5.14.5)
org.apache.activemq:activemq-stomp(5.14.5)
org.apache.aries.blueprint:blueprint-parser(1.4.0)
org.apache.aries.blueprint:org.apache.aries.blueprint.api(1.0.0,1.0.1)
org.apache.aries.blueprint:org.apache.aries.blueprint.core(1.4.4,1.6.1)
org.apache.aries.proxy:org.apache.aries.proxy.api(1.0.0,1.0.1)
org.apache.aries.quiesce:org.apache.aries.quiesce.api(1.0.0)
org.apache.aries:org.apache.aries.util(1.0.0)
org.apache.arrow:arrow-format(0.10.0)
org.apache.arrow:arrow-memory(0.10.0)
org.apache.arrow:arrow-vector(0.10.0)
org.apache.avro:avro(1.10.0,1.7.7,1.8.2)
org.apache.avro:avro-ipc(1.7.7,1.8.2)
org.apache.avro:avro-mapred(1.7.7,1.8.2)
org.apache.camel.karaf:apache-camel(2.16.2)
org.apache.camel:apt(2.16.2)
org.apache.camel:camel-core(2.16.2)
org.apache.camel:camel-core-osgi(2.16.2)
org.apache.camel:camel-jaxb(2.16.2)
org.apache.camel:spi-annotations(2.16.2)
org.apache.commons:commons-collections4(4.0,4.1,4.2)
org.apache.commons:commons-compress(1.5)
org.apache.commons:commons-crypto(1.0.0)
```

```
org.apache.commons:commons-lang3(3.3.2,3.4,3.5,3.6,3.8.1,3.9)
org.apache.commons:commons-math(2.1)
org.apache.commons:commons-math3(3.1.1,3.4.1,3.6.1)
org.apache.commons:commons-pool2(2.4.2)
org.apache.commons:commons-text(1.3)
org.apache.curator:curator-client(2.13.0,2.4.0,2.6.0,4.3.0)
org.apache.curator:curator-framework(2.13.0,2.4.0,2.6.0,4.3.0)
org.apache.curator:curator-recipes(2.13.0,2.4.0,2.6.0,4.3.0)
org.apache.cxf.karaf:apache-cxf(3.2.4)
org.apache.cxf.karaf:cxf-karaf-commands(3.2.4)
org.apache.cxf.services.sts:cxf-services-sts-core(3.2.4)
org.apache.cxf.services.ws-discovery:cxf-services-ws-discovery-api(3.2.4)
org.apache.cxf.services.ws-discovery:cxf-services-ws-discovery-service(3.2.4)
org.apache.cxf.services.wsn:cxf-services-wsn-api(3.2.4)
org.apache.cxf.services.wsn:cxf-services-wsn-core(3.2.4)
org.apache.cxf.services.wsn:cxf-services-wsn-osgi(3.2.4)
org.apache.cxf:cxf-core(3.2.4)
org.apache.cxf:cxf-rt-bindings-soap(3.2.4)
org.apache.cxf:cxf-rt-bindings-xml(3.2.4)
org.apache.cxf:cxf-rt-databinding-jaxb(3.2.4)
org.apache.cxf:cxf-rt-frontend-jaxrs(3.2.4)
org.apache.cxf:cxf-rt-frontend-jaxws(3.2.4)
org.apache.cxf:cxf-rt-frontend-simple(3.2.4)
org.apache.cxf:cxf-rt-management(3.2.4)
org.apache.cxf:cxf-rt-rs-client(3.2.4)
org.apache.cxf:cxf-rt-rs-extension-providers(3.2.4)
org.apache.cxf:cxf-rt-rs-json-basic(3.2.4)
org.apache.cxf:cxf-rt-rs-security-jose(3.2.4)
org.apache.cxf:cxf-rt-security(3.2.4)
org.apache.cxf:cxf-rt-security-saml(3.2.4)
org.apache.cxf:cxf-rt-transports-http(3.2.4)
org.apache.cxf:cxf-rt-transports-http-jetty(3.2.4)
org.apache.cxf:cxf-rt-transports-udp(3.2.4)
org.apache.cxf:cxf-rt-ws-addr(3.2.4)
org.apache.cxf:cxf-rt-ws-mex(3.2.4)
org.apache.cxf:cxf-rt-ws-policy(3.2.4)
org.apache.cxf:cxf-rt-ws-rm(3.2.4)
org.apache.cxf:cxf-rt-ws-security(3.2.4)
org.apache.cxf:cxf-rt-wsdl(3.2.4)
org.apache.directory.api:api-asn1-api(1.0.0-M20)
org.apache.directory.api:api-util(1.0.0-M20)
org.apache.directory.server:apacheds-i18n(2.0.0-M15)
org.apache.directory.server:apacheds-kerberos-codec(2.0.0-M15)
org.apache.felix:org.apache.felix.bundlerepository(2.0.4)
org.apache.felix:org.apache.felix.configadmin(1.8.8)
org.apache.felix:org.apache.felix.fileinstall(3.1.4,3.5.4)
org.apache.felix:org.apache.felix.framework(5.4.0)
org.apache.felix:org.apache.felix.gogo.runtime(0.16.2,0.6.1)
org.apache.felix:org.apache.felix.resolver(1.8.0)
org.apache.felix:org.apache.felix.scr(2.0.2)
org.apache.felix:org.apache.felix.shell(1.0.0)
org.apache.felix:org.apache.felix.utils(1.10.0,1.4.2,1.8.2)
org.apache.geronimo.javamail:geronimo-javamail_1.4_mail(1.8.4)
org.apache.geronimo.specs:geronimo-j2ee-management_1.1_spec(1.0.1)
org.apache.geronimo.specs:geronimo-jcache_1.0_spec(1.0-alpha-1)
org.apache.geronimo.specs:geronimo-jms_1.1_spec(1.1.1)
org.apache.geronimo.specs:geronimo-jta_1.0.1B_spec(1.0.1)
org.apache.geronimo.specs:geronimo-servlet_2.5_spec(1.2)
org.apache.hadoop:hadoop-annotations(2.10.1,2.2.0,2.6.5)
org.apache.hadoop:hadoop-auth(2.10.1,2.2.0,2.6.5)
org.apache.hadoop:hadoop-client(2.10.1,2.2.0,2.6.5)
org.apache.hadoop:hadoop-common(2.10.1,2.2.0,2.6.5)
org.apache.hadoop:hadoop-hdfs(2.2.0,2.6.5)
org.apache.hadoop:hadoop-hdfs-client(2.10.1)
org.apache.hadoop:hadoop-mapreduce-client-app(2.10.1,2.2.0,2.6.5)
org.apache.hadoop:hadoop-mapreduce-client-common(2.10.1,2.2.0,2.6.5)
org.apache.hadoop:hadoop-mapreduce-client-core(2.10.1,2.2.0,2.6.5)
org.apache.hadoop:hadoop-mapreduce-client-jobclient(2.10.1,2.2.0,2.6.5)
```

```
org.apache.hadoop:hadoop-mapreduce-client-shuffle(2.10.1,2.2.0,2.6.5)
org.apache.hadoop:hadoop-yarn-api(2.10.1,2.2.0,2.6.5)
org.apache.hadoop:hadoop-yarn-client(2.10.1,2.2.0,2.6.5)
org.apache.hadoop:hadoop-yarn-common(2.10.1,2.2.0,2.6.5)
org.apache.hadoop:hadoop-yarn-registry(2.10.1)
org.apache.hadoop:hadoop-yarn-server-common(2.10.1,2.2.0,2.6.5)
org.apache.htrace:htrace-core4(4.1.0-incubating,4.2.0-incubating)
org.apache.httpcomponents:httpclient(4.5,4.5.9)
org.apache.httpcomponents:httpcore(4.4.11)
org.apache.ivy:ivy(2.4.0)
org.apache.kafka:connect-api(1.1.1)
org.apache.kafka:kafka-clients(1.1.1)
org.apache.karaf.decanter:apache-karaf-decanter(1.0.0)
org.apache.karaf.diagnostic:org.apache.karaf.diagnostic.boot(4.0.5)
org.apache.karaf.features:base(4.0.5)
org.apache.karaf.features:enterprise(4.0.5)
org.apache.karaf.features:framework(4.0.5)
org.apache.karaf.features:org.apache.karaf.features.core(4.0.5)
org.apache.karaf.features:spring(4.0.5)
org.apache.karaf.features:standard(4.0.5)
org.apache.karaf.instance:org.apache.karaf.instance.core(4.0.5)
org.apache.karaf.jaas:org.apache.karaf.jaas.boot(4.0.5,4.1.2)
org.apache.karaf.jaas:org.apache.karaf.jaas.config(4.0.3,4.0.5)
org.apache.karaf.jaas:org.apache.karaf.jaas.modules(4.0.3,4.0.5)
org.apache.karaf.jdbc:org.apache.karaf.jdbc.core(4.0.5)
org.apache.karaf.log:org.apache.karaf.log.core(4.0.5)
org.apache.karaf.management:org.apache.karaf.management.server(4.0.5)
org.apache.karaf.service:org.apache.karaf.service.guard(4.0.5)
org.apache.karaf.shell:org.apache.karaf.shell.console(4.0.3,4.0.5)
org.apache.karaf.shell:org.apache.karaf.shell.core(4.0.5,4.1.2)
org.apache.karaf:org.apache.karaf.client(4.0.5)
org.apache.karaf:org.apache.karaf.exception(4.0.5)
org.apache.karaf:org.apache.karaf.main(4.0.5)
org.apache.karaf:org.apache.karaf.util(4.0.5,4.1.2)
org.apache.logging.log4j:log4j-1.2-api(2.17.2)
org.apache.logging.log4j:log4j-api(2.11.2,2.17.2)
org.apache.logging.log4j:log4j-core(2.17.2)
org.apache.logging.log4j:log4j-slf4j-impl(2.17.2)
org.apache.mesos:mesos(0.21.1)
org.apache.mina:mina-core(2.0.15)
org.apache.neethi:neethi(3.1.1)
org.apache.orc:orc-core(1.5.2)
org.apache.orc:orc-mapreduce(1.5.2)
org.apache.orc:orc-shims(1.5.2)
org.apache.parquet:parquet-column(1.10.0,1.12.0,1.7.0)
org.apache.parquet:parquet-common(1.10.0,1.12.0,1.7.0)
org.apache.parquet:parquet-encoding(1.10.0,1.12.0,1.7.0)
org.apache.parquet:parquet-format(2.3.0-incubating,2.4.0)
org.apache.parquet:parquet-format-structures(1.12.0)
org.apache.parquet:parquet-generator(1.7.0)
org.apache.parquet:parquet-hadoop(1.10.0,1.12.0,1.7.0)
org.apache.parquet:parquet-jackson(1.10.0,1.12.0,1.7.0)
org.apache.poi:poi(3.17)
org.apache.poi:poi-ooxml(3.17)
org.apache.poi:poi-ooxml-schemas(3.17)
org.apache.santuario:xmlsec(2.1.1)
org.apache.servicemix.bundles:org.apache.servicemix.bundles.freemarker(2.3.23_1)
org.apache.servicemix.bundles:org.apache.servicemix.bundles.josql(1.5_5)
org.apache.spark:spark-catalyst_2.11(2.0.1,2.4.0)
org.apache.spark:spark-core_2.11(2.0.1,2.4.0)
org.apache.spark:spark-kvstore_2.11(2.4.0)
org.apache.spark:spark-launcher_2.11(2.0.1,2.4.0)
org.apache.spark:spark-network-common_2.11(2.0.1,2.4.0)
org.apache.spark:spark-network-shuffle_2.11(2.0.1,2.4.0)
org.apache.spark:spark-sketch_2.11(2.0.1,2.4.0)
org.apache.spark:spark-sql_2.11(2.0.1,2.4.0)
org.apache.spark:spark-tags_2.11(2.0.1,2.4.0)
org.apache.spark:spark-unsafe_2.11(2.0.1,2.4.0)
```

```
org.apache.sshd:sshd-core(0.14.0,1.6.0)
org.apache.velocity:velocity(1.7)
org.apache.ws.xmlschema:xmlschema-core(2.2.3)
org.apache.wss4j:wss4j-bindings(2.2.1)
org.apache.wss4j:wss4j-policy(2.2.1)
org.apache.wss4j:wss4j-ws-security-common(2.2.1)
org.apache.wss4j:wss4j-ws-security-dom(2.2.1)
org.apache.wss4j:wss4j-ws-security-policy-stax(2.2.1)
org.apache.wss4j:wss4j-ws-security-stax(2.2.1)
org.apache.xbean:xbean-asm-util(4.1)
org.apache.xbean:xbean-bundleutils(4.1)
org.apache.xbean:xbean-finder(4.1)
org.apache.xbean:xbean-spring(4.2)
org.apache.xmlbeans:xmlbeans(2.6.0)
org.apache.yetus:audience-annotations(0.12.0)
org.apache.zookeeper:zookeeper(3.4.14,3.4.5,3.4.6,3.5.7)
org.apache.zookeeper:zookeeper-jute(3.5.7)
org.codehaus.jackson:jackson-core-asl(1.9.11,1.9.13)
org.codehaus.jackson:jackson-mapper-asl(1.9.11,1.9.13)
org.codehaus.jettison:jettison(1.1)
org.cyclopsgroup:caff(0.4.0)
org.cyclopsgroup:jcli(1.0.0-beta-2)
org.cyclopsgroup:jmxterm(1.0.0)
org.eclipse.microprofile.config:microprofile-config-api(1.4)
org.eclipse.microprofile.context-propagation:microprofile-context-propagation-api(1.0.1)
org.ehcache:ehcache(3.3.1)
org.fusesource.hawtbuf:hawtbuf(1.11)
org.fusesource.hawtbuf:hawtbuf-proto(1.11)
org.fusesource.hawtdispatch:hawtdispatch(1.22)
org.fusesource.hawtdispatch:hawtdispatch-transport(1.22)
org.fusesource.mqtt-client:mqtt-client(1.14)
org.htrace:htrace-core(3.0.4)
org.ini4j:ini4j(0.5.4)
org.jasypt:jasypt(1.9.2)
org.jboss.logging:jboss-logging(3.4.1.Final)
org.jboss.logging:jboss-logging-annotations(2.1.0.Final)
org.jboss.logmanager:jboss-logmanager(2.1.9.Final)
org.jboss.logmanager:jboss-logmanager-embedded(1.0.6)
org.jboss.netty:netty(3.2.8.Final)
org.jboss.resteasy:resteasy-core(4.5.8.Final)
org.jboss.resteasy:resteasy-core-spi(4.5.8.Final)
org.jboss.resteasy:resteasy-json-binding-provider(4.5.8.Final)
org.jboss.resteasy:resteasy-json-p-provider(4.5.8.Final)
org.jboss.threads:jboss-threads(3.1.1.Final)
org.jetbrains:annotations(15.0,23.0.0)
org.jledit:core(0.2.1)
org.jolokia:jolokia-core(1.3.4)
org.jolokia:jolokia-osgi(1.3.4)
org.jooq:jooq(3.7.3,${jooq.version})
org.jooq:jooq-codegen(${jooq.version})
org.jooq:jooq-meta(${jooq.version})
org.json4s:json4s-ast_2.11(3.2.11,3.5.3)
org.json4s:json4s-core_2.11(3.2.11,3.5.3)
org.json4s:json4s-jackson_2.11(3.2.11,3.5.3)
org.json4s:json4s-scalap_2.11(3.5.3)
org.keycloak:keycloak-adapter-core(10.0.1)
org.keycloak:keycloak-adapter-spi(10.0.1)
org.keycloak:keycloak-common(10.0.1)
org.keycloak:keycloak-core(10.0.1)
org.keycloak:keycloak-osgi-features(10.0.1)
org.keycloak:keycloak-servlet-adapter-spi(10.0.1)
org.keycloak:keycloak-servlet-filter-adapter(10.0.1)
org.lz4:lz4-java(1.4.1)
org.mortbay.jetty:jetty-sslengine(6.1.26)
org.mvel:mvel2(2.2.4.Final)
org.objenesis:objenesis(2.1,2.5.1)
org.opensaml:opensaml-core(3.3.0)
org.opensaml:opensaml-profile-api(3.3.0)
```

```
org.opensaml:opensaml-saml-api(3.3.0)
org.opensaml:opensaml-saml-impl(3.3.0)
org.opensaml:opensaml-security-api(3.3.0)
org.opensaml:opensaml-security-impl(3.3.0)
org.opensaml:opensaml-soap-api(3.3.0)
org.opensaml:opensaml-xacml-api(3.3.0)
org.opensaml:opensaml-xacml-impl(3.3.0)
org.opensaml:opensaml-xacml-saml-api(3.3.0)
org.opensaml:opensaml-xacml-saml-impl(3.3.0)
org.opensaml:opensaml-xmlsec-api(3.3.0)
org.opensaml:opensaml-xmlsec-impl(3.3.0)
org.ops4j.base:ops4j-base-lang(1.2.3,1.5.0)
org.ops4j.base:ops4j-base-monitors(1.2.3)
org.ops4j.base:ops4j-base-net(1.2.3)
org.ops4j.base:ops4j-base-spi(1.5.0)
org.ops4j.base:ops4j-base-util-collections(1.5.0)
org.ops4j.base:ops4j-base-util-xml(1.5.0)
org.ops4j.pax.logging:pax-logging-api(1.8.5)
org.ops4j.pax.logging:pax-logging-service(1.8.5)
org.ops4j.pax.swissbox:pax-swissbox-core(1.8.2)
org.ops4j.pax.swissbox:pax-swissbox-extender(1.8.2)
org.ops4j.pax.swissbox:pax-swissbox-lifecycle(1.8.2)
org.ops4j.pax.swissbox:pax-swissbox-optional-jcl(1.8.2)
org.ops4j.pax.swissbox:pax-swissbox-tracker(1.8.2)
org.ops4j.pax.url:pax-url-aether(2.4.7)
org.ops4j.pax.web:pax-web-api(4.2.6)
org.ops4j.pax.web:pax-web-deployer(4.2.6)
org.ops4j.pax.web:pax-web-extender-war(4.2.6)
org.ops4j.pax.web:pax-web-jetty(4.2.6)
org.ops4j.pax.web:pax-web-spi(4.2.6)
org.osgi:org.osgi.compendium(5.0.0)
org.osgi:org.osgi.core(6.0.0)
org.osgi:org.osgi.enterprise(5.0.0)
org.osgi:osgi.annotation(6.0.1)
org.osgi:osgi.cmpn(6.0.0)
org.osgi:osgi.core(6.0.0)
org.roaringbitmap:RoaringBitmap(0.5.11)
org.scalatest:scalatest_2.11(2.2.6)
org.sonatype.sisu.inject:cglib(2.2.1-v20090111)
org.spark-project.spark:unused(1.0.0)
org.springframework.osgi:spring-osgi-core(1.2.1)
org.springframework.osgi:spring-osgi-io(1.2.1)
org.springframework:spring-aop(4.3.14.RELEASE)
org.springframework:spring-beans(4.2.5.RELEASE,4.3.14.RELEASE)
org.springframework:spring-context(4.3.14.RELEASE)
org.springframework:spring-core(4.2.5.RELEASE,4.3.14.RELEASE)
org.springframework:spring-expression(4.3.14.RELEASE)
org.springframework:spring-jdbc(4.2.5.RELEASE)
org.springframework:spring-jms(4.1.9.RELEASE)
org.springframework:spring-messaging(4.1.9.RELEASE)
org.springframework:spring-tx(4.1.9.RELEASE,4.2.5.RELEASE)
org.wildfly.common:wildfly-common(1.5.4.Final-format-001)
org.xerial.snappy:snappy-java(1.1.2.6,1.1.7.1,1.1.8)
software.amazon.awssdk:annotations(2.15.6,2.15.71)
software.amazon.awssdk:apache-client(2.15.6,2.15.71)
software.amazon.awssdk:arns(2.15.6,2.15.71)
software.amazon.awssdk:auth(2.15.6,2.15.71)
software.amazon.awssdk:aws-core(2.15.6,2.15.71)
software.amazon.awssdk:aws-query-protocol(2.15.6,2.15.71)
software.amazon.awssdk:aws-xml-protocol(2.15.6,2.15.71)
software.amazon.awssdk:http-client-spi(2.15.6,2.15.71)
software.amazon.awssdk:metrics-spi(2.15.71)
software.amazon.awssdk:netty-nio-client(2.15.6,2.15.71)
software.amazon.awssdk:profiles(2.15.6,2.15.71)
software.amazon.awssdk:protocol-core(2.15.6,2.15.71)
software.amazon.awssdk:regions(2.15.6,2.15.71)
software.amazon.awssdk:s3(2.15.71)
software.amazon.awssdk:sdk-core(2.15.6,2.15.71)
```

```
software.amazon.awssdk:sts(2.15.71)
software.amazon.awssdk:utils(2.15.6,2.15.71)
software.amazon.eventstream:eventstream(1.0.1)
software.amazon.ion:ion-java(1.0.2)
stax:stax-api(1.0.1)
xerces:xercesImpl(2.9.1)
xml-resolver:xml-resolver(1.2)

Apache-2.0
URL: https://app.snyk.io/vuln/snyk:lic:Apache-2.0

--------------------------------------------------------------------------------
xml-apis:xml-apis(1.3.04)

Apache-2.0 AND W3C
URL: https://app.snyk.io/vuln/snyk:lic:Apache-2.0
URL: https://app.snyk.io/vuln/snyk:lic:W3C

--------------------------------------------------------------------------------
org.apache.xbean:xbean-asm5-shaded(4.4)
org.apache.xbean:xbean-asm6-shaded(4.8)

Apache-2.0 OR BSD-3-Clause
URL: https://app.snyk.io/vuln/snyk:lic:Apache-2.0
URL: https://app.snyk.io/vuln/snyk:lic:BSD-3-Clause

--------------------------------------------------------------------------------
io.vertx:vertx-auth-common(3.9.4)
io.vertx:vertx-bridge-common(3.9.4)
io.vertx:vertx-core(3.9.4)
io.vertx:vertx-web(3.9.4)
io.vertx:vertx-web-common(3.9.4)
org.eclipse.jetty.aggregate:jetty-all(9.2.13.v20150730,9.2.15.v20160210)
org.eclipse.jetty.alpn:alpn-api(1.1.2.v20150522)
org.eclipse.jetty.orbit:javax.mail.glassfish(1.4.1.v201005082020)
org.eclipse.jetty.orbit:javax.security.auth.message(1.0.0.v201108011116)
org.eclipse.jetty.osgi:jetty-osgi-alpn(9.2.15.v20160210)
org.eclipse.jetty.spdy:spdy-client(9.2.15.v20160210)
org.eclipse.jetty.spdy:spdy-core(9.2.15.v20160210)
org.eclipse.jetty.spdy:spdy-http-common(9.2.15.v20160210)
org.eclipse.jetty.spdy:spdy-http-server(9.2.15.v20160210)
org.eclipse.jetty.spdy:spdy-server(9.2.15.v20160210)
org.eclipse.jetty.websocket:javax-websocket-client-impl(9.2.15.v20160210)
org.eclipse.jetty.websocket:javax-websocket-server-impl(9.2.15.v20160210)
org.eclipse.jetty.websocket:websocket-api(9.2.15.v20160210)
org.eclipse.jetty.websocket:websocket-client(9.2.15.v20160210)
org.eclipse.jetty.websocket:websocket-common(9.2.15.v20160210)
org.eclipse.jetty.websocket:websocket-server(9.2.15.v20160210)
org.eclipse.jetty.websocket:websocket-servlet(9.2.15.v20160210)
org.eclipse.jetty:jetty-alpn-client(9.2.15.v20160210)
org.eclipse.jetty:jetty-alpn-server(9.2.15.v20160210)
org.eclipse.jetty:jetty-annotations(9.2.15.v20160210)
org.eclipse.jetty:jetty-client(9.2.15.v20160210)
org.eclipse.jetty:jetty-continuation(9.2.15.v20160210,9.4.8.v20171121)
org.eclipse.jetty:jetty-deploy(9.2.15.v20160210)
org.eclipse.jetty:jetty-http(9.2.15.v20160210,9.4.8.v20171121)
org.eclipse.jetty:jetty-io(9.2.15.v20160210,9.4.8.v20171121)
org.eclipse.jetty:jetty-jaas(9.2.15.v20160210)
org.eclipse.jetty:jetty-jaspi(9.2.15.v20160210)
org.eclipse.jetty:jetty-jmx(9.2.15.v20160210)
org.eclipse.jetty:jetty-jndi(9.2.15.v20160210)
org.eclipse.jetty:jetty-plus(9.2.15.v20160210)
org.eclipse.jetty:jetty-quickstart(9.2.15.v20160210)
org.eclipse.jetty:jetty-rewrite(9.2.15.v20160210)
org.eclipse.jetty:jetty-security(9.2.15.v20160210,9.4.8.v20171121)
org.eclipse.jetty:jetty-server(9.2.15.v20160210,9.4.8.v20171121)
org.eclipse.jetty:jetty-servlet(9.2.15.v20160210,9.2.16.v20160414,9.3.24.v20180605)
org.eclipse.jetty:jetty-servlets(9.2.15.v20160210)
```

```
org.eclipse.jetty:jetty-util(9.2.15.v20160210,9.4.8.v20171121)
org.eclipse.jetty:jetty-webapp(9.2.15.v20160210)
org.eclipse.jetty:jetty-xml(9.2.15.v20160210)
org.mortbay.jetty:jetty(6.1.26)
org.mortbay.jetty:jetty-util(6.1.26)

Apache-2.0 OR EPL-1.0
URL: https://app.snyk.io/vuln/snyk:lic:Apache-2.0
URL: https://app.snyk.io/vuln/snyk:lic:EPL-1.0

--------------------------------------------------------------------------------
org.mortbay.jetty.alpn:alpn-boot(8.1.2.v20141202)

Apache-2.0 OR GPL-3.0
URL: https://app.snyk.io/vuln/snyk:lic:Apache-2.0
URL: https://app.snyk.io/vuln/snyk:lic:GPL-3.0

--------------------------------------------------------------------------------
net.java.dev.jna:jna(5.4.0)
net.java.dev.jna:jna-platform(5.4.0)
org.codehaus.jackson:jackson-jaxrs(1.8.3,1.9.13)
org.codehaus.jackson:jackson-xc(1.8.3,1.9.13)

Apache-2.0 OR LGPL-2.1
URL: https://app.snyk.io/vuln/snyk:lic:Apache-2.0
URL: https://app.snyk.io/vuln/snyk:lic:LGPL-2.1

--------------------------------------------------------------------------------
org.javassist:javassist(3.18.1-GA)

Apache-2.0 OR LGPL-2.1 OR MPL-1.1
URL: https://app.snyk.io/vuln/snyk:lic:Apache-2.0
URL: https://app.snyk.io/vuln/snyk:lic:LGPL-2.1
URL: https://app.snyk.io/vuln/snyk:lic:MPL-1.1

--------------------------------------------------------------------------------
org.cryptacular:cryptacular(1.1.1)

Apache-2.0 OR LGPL-3.0
URL: https://app.snyk.io/vuln/snyk:lic:Apache-2.0
URL: https://app.snyk.io/vuln/snyk:lic:LGPL-3.0

--------------------------------------------------------------------------------
asm:asm(3.1)
com.github.luben:zstd-jni(1.3.2-2,1.4.9-1)
com.github.virtuald:curvesapi(1.04)
com.thoughtworks.paranamer:paranamer(2.3,2.6,2.7)
com.thoughtworks.xstream:xstream(1.4.9)
jline:jline(2.12.1,2.14.1)
net.sf.kxml:kxml2(2.2.2)
net.sf.py4j:py4j(0.10.3,0.10.7)
org.antlr:ST4(4.3.3)
org.antlr:antlr-runtime(3.5.2)
org.antlr:antlr4-runtime(4.5.3,4.7)
org.codehaus.woodstox:stax2-api(3.1.4,4.2)
org.jline:jline(3.4.0)
org.ow2.asm:asm(5.0.3,5.0.4,5.2)
org.ow2.asm:asm-commons(5.0.3)
org.ow2.asm:asm-tree(5.0.3)
org.postgresql:postgresql(42.2.16,42.2.18)
xmlenc:xmlenc(0.52)

BSD-2-Clause
URL: https://app.snyk.io/vuln/snyk:lic:BSD-2-Clause

--------------------------------------------------------------------------------
com.esotericsoftware:kryo-shaded(3.0.3,4.0.2)
com.esotericsoftware:minlog(1.3.0)
```

```
com.google.protobuf:protobuf-java(2.5.0,3.13.0,3.5.1)
com.jcraft:jsch(0.1.55)
com.sun.activation:jakarta.activation(1.2.1)
org.codehaus.janino:commons-compiler(2.7.8,3.0.9)
org.codehaus.janino:janino(2.7.8,3.0.9)
org.fusesource.leveldbjni:leveldbjni-all(1.8)
org.hamcrest:hamcrest-core(1.3)
org.hamcrest:hamcrest-library(1.3)
org.nanohttpd:nanohttpd(2.3.1)
org.ow2.asm:asm(8.0.1)
org.scala-lang.modules:scala-parser-combinators_2.11(1.0.1,1.1.0)
org.scala-lang.modules:scala-xml_2.11(1.0.2,1.0.6)
org.scala-lang:scala-compiler(2.11.0)
org.scala-lang:scala-library(2.11.12,2.11.8)
org.scala-lang:scala-reflect(2.11.12,2.11.8)
org.scala-lang:scalap(2.11.0)

BSD-3-Clause
URL: https://app.snyk.io/vuln/snyk:lic:BSD-3-Clause

--------------------------------------------------------------------------------
org.reactivestreams:reactive-streams(1.0.2,1.0.3)
org.scijava:native-lib-loader(2.0.2)

CC0-1.0
URL: https://app.snyk.io/vuln/snyk:lic:CC0-1.0

--------------------------------------------------------------------------------
javax.activation:activation(1.1,1.1.1)
javax.annotation:jsr250-api(1.0)
javax.websocket:javax.websocket-api(1.1)

CDDL-1.1
URL: https://app.snyk.io/vuln/snyk:lic:CDDL-1.1

--------------------------------------------------------------------------------
javax.servlet.jsp:jsp-api(2.1)
javax.servlet:servlet-api(2.3,2.5)
javax.transaction:javax.transaction-api(1.2)
org.glassfish.hk2.external:aopalliance-repackaged(2.4.0-b34)
org.glassfish.hk2.external:javax.inject(2.4.0-b34)
org.glassfish.hk2:hk2-api(2.4.0-b34)
org.glassfish.hk2:hk2-locator(2.4.0-b34)
org.glassfish.hk2:hk2-utils(2.4.0-b34)
org.glassfish.hk2:osgi-resource-locator(1.0.1)
org.glassfish.main.external:jmxremote_optional-repackaged(4.1)

CDDL-1.1 OR GPL-2.0
URL: https://app.snyk.io/vuln/snyk:lic:CDDL-1.1
URL: https://app.snyk.io/vuln/snyk:lic:GPL-2.0

--------------------------------------------------------------------------------
com.sun.jersey:jersey-client(1.9)
com.sun.jersey:jersey-core(1.9)
com.sun.jersey:jersey-json(1.9)
com.sun.jersey:jersey-server(1.9)
com.sun.mail:javax.mail(1.6.1)
com.sun.xml.bind:jaxb-core(2.2.11)
com.sun.xml.bind:jaxb-impl(2.2.3-1)
javax.annotation:javax.annotation-api(1.2,1.3,1.3.2)
javax.servlet:javax.servlet-api(3.1.0)
javax.ws.rs:javax.ws.rs-api(2.0.1,2.1)
javax.xml.bind:jaxb-api(2.2.2)

CDDL-1.1 OR GPL-2.0-with-classpath-exception
URL: https://app.snyk.io/vuln/snyk:lic:CDDL-1.1
URL: https://app.snyk.io/vuln/snyk:lic:GPL-2.0-with-classpath-exception
```

```
--------------------------------------------------------------------------------
com.betfair.net.java.opendmk:core(1.0-b02)
com.sun.xml.bind:jaxb-impl(2.2.11)
javax.xml.stream:stax-api(1.0-2)
org.glassfish.jersey.bundles.repackaged:jersey-guava(2.22.2)
org.glassfish.jersey.containers:jersey-container-servlet(2.22.2)
org.glassfish.jersey.containers:jersey-container-servlet-core(2.22.2)
org.glassfish.jersey.core:jersey-client(2.22.2)
org.glassfish.jersey.core:jersey-common(2.22.2)
org.glassfish.jersey.core:jersey-server(2.22.2)
org.glassfish.jersey.media:jersey-media-jaxb(2.22.2)
org.jboss.spec.javax.servlet:jboss-servlet-api_3.0_spec(1.0.2.Final)

CDDL-1.1 OR GPL-3.0
URL: https://app.snyk.io/vuln/snyk:lic:CDDL-1.1
URL: https://app.snyk.io/vuln/snyk:lic:GPL-3.0

--------------------------------------------------------------------------------
wsdl4j:wsdl4j(1.6.3)

CPL-1.0
URL: https://app.snyk.io/vuln/snyk:lic:CPL-1.0

--------------------------------------------------------------------------------
jakarta.activation:jakarta.activation-api(1.2.1)
jakarta.xml.bind:jakarta.xml.bind-api(2.3.2)
org.jboss.spec.javax.xml.bind:jboss-jaxb-api_2.3_spec(2.0.0.Final)

EDL-1.0
URL: https://app.snyk.io/vuln/snyk:lic:EDL-1.0

--------------------------------------------------------------------------------
org.eclipse:yasson(1.0.8)

EDL-1.0 OR EPL-2.0
URL: https://app.snyk.io/vuln/snyk:lic:EDL-1.0
URL: https://app.snyk.io/vuln/snyk:lic:EPL-2.0

--------------------------------------------------------------------------------
junit:junit(4.12)
org.eclipse.birt.runtime:org.eclipse.osgi(3.10.2.v20150203-1939)
org.eclipse.equinox:org.eclipse.equinox.region(1.1.0.v20120522-1841)
org.eclipse.jdt:org.eclipse.jdt.annotation(2.0.0)

EPL-1.0
URL: https://app.snyk.io/vuln/snyk:lic:EPL-1.0

--------------------------------------------------------------------------------
jakarta.annotation:jakarta.annotation-api(1.3.5)
jakarta.ejb:jakarta.ejb-api(3.2.6)
jakarta.el:jakarta.el-api(3.0.3)
jakarta.interceptor:jakarta.interceptor-api(1.2.5)
jakarta.json.bind:jakarta.json.bind-api(1.0.2)
jakarta.json:jakarta.json-api(1.1.6)
jakarta.transaction:jakarta.transaction-api(1.3.3)
org.glassfish:jakarta.json(1.1.6)
org.jboss.spec.javax.servlet:jboss-servlet-api_4.0_spec(unknown)
org.jboss.spec.javax.ws.rs:jboss-jaxrs-api_2.1_spec(2.0.1.Final)

EPL-2.0 OR GPL-2.0-with-classpath-exception
URL: https://app.snyk.io/vuln/snyk:lic:EPL-2.0
URL: https://app.snyk.io/vuln/snyk:lic:GPL-2.0-with-classpath-exception

--------------------------------------------------------------------------------
com.github.spotbugs:spotbugs-annotations(3.1.9)

LGPL-2.1
URL: https://app.snyk.io/vuln/snyk:lic:LGPL-2.1
```

```
--------------------------------------------------------------------------------
org.jboss.logging:jboss-logging-processor(2.1.0.Final)
org.jboss.spec.javax.interceptor:jboss-interceptors-api_1.1_spec(1.0.0.Beta1)

LGPL-3.0
URL: https://app.snyk.io/vuln/snyk:lic:LGPL-3.0

--------------------------------------------------------------------------------
com.auth0:java-jwt(3.3.0)
com.azure:azure-core(1.20.0)
com.azure:azure-core-http-netty(1.11.0)
com.azure:azure-identity(1.1.0-beta.3)
com.azure:azure-storage-blob(12.14.0)
com.azure:azure-storage-common(12.13.0)
com.azure:azure-storage-internal-avro(12.1.0)
com.microsoft.azure:msal4j(1.3.0)
com.microsoft.sqlserver:mssql-jdbc(6.2.1.jre7)
de.bytefish:pgbulkinsert(2.2)
net.razorvine:pyrolite(4.13,4.9)
org.bouncycastle:bcpkix-jdk15on(1.62)
org.bouncycastle:bcprov-jdk15on(1.54,1.62)
org.checkerframework:checker-qual(2.5.2,3.5.0)
org.codehaus.mojo:animal-sniffer-annotations(1.9)
org.easymock:easymock(2.4)
org.glyptodon.guacamole:guacamole-common(0.9.9)
org.slf4j:jcl-over-slf4j(1.7.12,1.7.16)
org.slf4j:jul-to-slf4j(1.7.16)
org.slf4j:slf4j-api(1.7.12)
org.slf4j:slf4j-log4j12(1.7.12,1.7.16)

MIT
URL: https://app.snyk.io/vuln/snyk:lic:MIT

--------------------------------------------------------------------------------
com.h2database:h2(1.4.200)

MPL-2.0 OR EPL-1.0
URL: https://app.snyk.io/vuln/snyk:lic:MPL-2.0
URL: https://app.snyk.io/vuln/snyk:lic:EPL-1.0

--------------------------------------------------------------------------------
classworlds:classworlds(1.1)

Plexus
URL: https://app.snyk.io/vuln/snyk:lic:Plexus

--------------------------------------------------------------------------------
aopalliance:aopalliance(1.0)
org.jboss.jdeparser:jdeparser(2.0.2.Final)
org.jboss.slf4j:slf4j-jboss-logging(1.2.0.Final)
org.tukaani:xz(1.2,1.5)
xmlpull:xmlpull(1.1.3.1)
xpp3:xpp3_min(1.1.4c)

Public-Domain
URL: https://app.snyk.io/vuln/snyk:lic:Public-Domain

--------------------------------------------------------------------------------
org.graalvm.sdk:graal-sdk(20.2.0)

UPL-1.0
URL: https://app.snyk.io/vuln/snyk:lic:UPL-1.0

--------------------------------------------------------------------------------
avalon-framework:avalon-framework(4.1.3)
avalon-framework:avalon-framework-api(4.3)
classworlds:classworlds-boot(1.0)
```

```
commons-beanutils:commons-beanutils(1.7.0)
logkit:logkit(1.0.1)
org.osgi:osgi_R4_compendium(1.0)

Unknown
URL: https://app.snyk.io/vuln/snyk:lic:Unknown

-------------------------------------------------------------------------------
Blake2
================================================================================
Creative Commons Legal Code

CC0 1.0 Universal

    CREATIVE COMMONS CORPORATION IS NOT A LAW FIRM AND DOES NOT PROVIDE
    LEGAL SERVICES. DISTRIBUTION OF THIS DOCUMENT DOES NOT CREATE AN
    ATTORNEY-CLIENT RELATIONSHIP. CREATIVE COMMONS PROVIDES THIS
    INFORMATION ON AN "AS-IS" BASIS. CREATIVE COMMONS MAKES NO WARRANTIES
    REGARDING THE USE OF THIS DOCUMENT OR THE INFORMATION OR WORKS
    PROVIDED HEREUNDER, AND DISCLAIMS LIABILITY FOR DAMAGES RESULTING FROM
    THE USE OF THIS DOCUMENT OR THE INFORMATION OR WORKS PROVIDED
    HEREUNDER.

Statement of Purpose

The laws of most jurisdictions throughout the world automatically confer
exclusive Copyright and Related Rights (defined below) upon the creator
and subsequent owner(s) (each and all, an "owner") of an original work of
authorship and/or a database (each, a "Work").

Certain owners wish to permanently relinquish those rights to a Work for
the purpose of contributing to a commons of creative, cultural and
scientific works ("Commons") that the public can reliably and without fear
of later claims of infringement build upon, modify, incorporate in other
works, reuse and redistribute as freely as possible in any form whatsoever
and for any purposes, including without limitation commercial purposes.
These owners may contribute to the Commons to promote the ideal of a free
culture and the further production of creative, cultural and scientific
works, or to gain reputation or greater distribution for their Work in
part through the use and efforts of others.

For these and/or other purposes and motivations, and without any
expectation of additional consideration or compensation, the person
associating CC0 with a Work (the "Affirmer"), to the extent that he or she
is an owner of Copyright and Related Rights in the Work, voluntarily
elects to apply CC0 to the Work and publicly distribute the Work under its
terms, with knowledge of his or her Copyright and Related Rights in the
Work and the meaning and intended legal effect of CC0 on those rights.

1. Copyright and Related Rights. A Work made available under CC0 may be
protected by copyright and related or neighboring rights ("Copyright and
Related Rights"). Copyright and Related Rights include, but are not
limited to, the following:

  i. the right to reproduce, adapt, distribute, perform, display,
     communicate, and translate a Work;
 ii. moral rights retained by the original author(s) and/or performer(s);
iii. publicity and privacy rights pertaining to a person's image or
     likeness depicted in a Work;
 iv. rights protecting against unfair competition in regards to a Work,
     subject to the limitations in paragraph 4(a), below;
  v. rights protecting the extraction, dissemination, use and reuse of data
     in a Work;
 vi. database rights (such as those arising under Directive 96/9/EC of the
     European Parliament and of the Council of 11 March 1996 on the legal
     protection of databases, and under any national implementation
     thereof, including any amended or successor version of such
     directive); and
```

```
vii. other similar, equivalent or corresponding rights throughout the
    world based on applicable law or treaty, and any national
    implementations thereof.

2. Waiver. To the greatest extent permitted by, but not in contravention
of, applicable law, Affirmer hereby overtly, fully, permanently,
irrevocably and unconditionally waives, abandons, and surrenders all of
Affirmer's Copyright and Related Rights and associated claims and causes
of action, whether now known or unknown (including existing as well as
future claims and causes of action), in the Work (i) in all territories
worldwide, (ii) for the maximum duration provided by applicable law or
treaty (including future time extensions), (iii) in any current or future
medium and for any number of copies, and (iv) for any purpose whatsoever,
including without limitation commercial, advertising or promotional
purposes (the "Waiver"). Affirmer makes the Waiver for the benefit of each
member of the public at large and to the detriment of Affirmer's heirs and
successors, fully intending that such Waiver shall not be subject to
revocation, rescission, cancellation, termination, or any other legal or
equitable action to disrupt the quiet enjoyment of the Work by the public
as contemplated by Affirmer's express Statement of Purpose.

3. Public License Fallback. Should any part of the Waiver for any reason
be judged legally invalid or ineffective under applicable law, then the
Waiver shall be preserved to the maximum extent permitted taking into
account Affirmer's express Statement of Purpose. In addition, to the
extent the Waiver is so judged Affirmer hereby grants to each affected
person a royalty-free, non transferable, non sublicensable, non exclusive,
irrevocable and unconditional license to exercise Affirmer's Copyright and
Related Rights in the Work (i) in all territories worldwide, (ii) for the
maximum duration provided by applicable law or treaty (including future
time extensions), (iii) in any current or future medium and for any number
of copies, and (iv) for any purpose whatsoever, including without
limitation commercial, advertising or promotional purposes (the
"License"). The License shall be deemed effective as of the date CC0 was
applied by Affirmer to the Work. Should any part of the License for any
reason be judged legally invalid or ineffective under applicable law, such
partial invalidity or ineffectiveness shall not invalidate the remainder
of the License, and in such case Affirmer hereby affirms that he or she
will not (i) exercise any of his or her remaining Copyright and Related
Rights in the Work or (ii) assert any associated claims and causes of
action with respect to the Work, in either case contrary to Affirmer's
express Statement of Purpose.

4. Limitations and Disclaimers.

 a. No trademark or patent rights held by Affirmer are waived, abandoned,
    surrendered, licensed or otherwise affected by this document.
 b. Affirmer offers the Work as-is and makes no representations or
    warranties of any kind concerning the Work, express, implied,
    statutory or otherwise, including without limitation warranties of
    title, merchantability, fitness for a particular purpose, non
    infringement, or the absence of latent or other defects, accuracy, or
    the present or absence of errors, whether or not discoverable, all to
    the greatest extent permissible under applicable law.
 c. Affirmer disclaims responsibility for clearing rights of other persons
    that may apply to the Work or any use thereof, including without
    limitation any person's Copyright and Related Rights in the Work.
    Further, Affirmer disclaims responsibility for obtaining any necessary
    consents, permissions or other rights required for any use of the
    Work.
 d. Affirmer understands and acknowledges that Creative Commons is not a
    party to this document and has no duty or obligation with respect to
    this CC0 or use of the Work.


CLhash
================================================================================
                                 Apache License
```

```
                                 Version 2.0, January 2004
                              http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

   "License" shall mean the terms and conditions for use, reproduction,
   and distribution as defined by Sections 1 through 9 of this document.

   "Licensor" shall mean the copyright owner or entity authorized by
   the copyright owner that is granting the License.

   "Legal Entity" shall mean the union of the acting entity and all
   other entities that control, are controlled by, or are under common
   control with that entity. For the purposes of this definition,
   "control" means (i) the power, direct or indirect, to cause the
   direction or management of such entity, whether by contract or
   otherwise, or (ii) ownership of fifty percent (50%) or more of the
   outstanding shares, or (iii) beneficial ownership of such entity.

   "You" (or "Your") shall mean an individual or Legal Entity
   exercising permissions granted by this License.

   "Source" form shall mean the preferred form for making modifications,
   including but not limited to software source code, documentation
   source, and configuration files.

   "Object" form shall mean any form resulting from mechanical
   transformation or translation of a Source form, including but
   not limited to compiled object code, generated documentation,
   and conversions to other media types.

   "Work" shall mean the work of authorship, whether in Source or
   Object form, made available under the License, as indicated by a
   copyright notice that is included in or attached to the work
   (an example is provided in the Appendix below).

   "Derivative Works" shall mean any work, whether in Source or Object
   form, that is based on (or derived from) the Work and for which the
   editorial revisions, annotations, elaborations, or other modifications
   represent, as a whole, an original work of authorship. For the purposes
   of this License, Derivative Works shall not include works that remain
   separable from, or merely link (or bind by name) to the interfaces of,
   the Work and Derivative Works thereof.

   "Contribution" shall mean any work of authorship, including
   the original version of the Work and any modifications or additions
   to that Work or Derivative Works thereof, that is intentionally
   submitted to Licensor for inclusion in the Work by the copyright owner
   or by an individual or Legal Entity authorized to submit on behalf of
   the copyright owner. For the purposes of this definition, "submitted"
   means any form of electronic, verbal, or written communication sent
   to the Licensor or its representatives, including but not limited to
   communication on electronic mailing lists, source code control systems,
   and issue tracking systems that are managed by, or on behalf of, the
   Licensor for the purpose of discussing and improving the Work, but
   excluding communication that is conspicuously marked or otherwise
   designated in writing by the copyright owner as "Not a Contribution."

   "Contributor" shall mean Licensor and any individual or Legal Entity
   on behalf of whom a Contribution has been received by Licensor and
   subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of
   this License, each Contributor hereby grants to You a perpetual,
   worldwide, non-exclusive, no-charge, royalty-free, irrevocable
   copyright license to reproduce, prepare Derivative Works of,
```

```
   publicly display, publicly perform, sublicense, and distribute the
   Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of
   this License, each Contributor hereby grants to You a perpetual,
   worldwide, non-exclusive, no-charge, royalty-free, irrevocable
   (except as stated in this section) patent license to make, have made,
   use, offer to sell, sell, import, and otherwise transfer the Work,
   where such license applies only to those patent claims licensable
   by such Contributor that are necessarily infringed by their
   Contribution(s) alone or by combination of their Contribution(s)
   with the Work to which such Contribution(s) was submitted. If You
   institute patent litigation against any entity (including a
   cross-claim or counterclaim in a lawsuit) alleging that the Work
   or a Contribution incorporated within the Work constitutes direct
   or contributory patent infringement, then any patent licenses
   granted to You under this License for that Work shall terminate
   as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the
   Work or Derivative Works thereof in any medium, with or without
   modifications, and in Source or Object form, provided that You
   meet the following conditions:

   (a) You must give any other recipients of the Work or
       Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices
       stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works
       that You distribute, all copyright, patent, trademark, and
       attribution notices from the Source form of the Work,
       excluding those notices that do not pertain to any part of
       the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its
       distribution, then any Derivative Works that You distribute must
       include a readable copy of the attribution notices contained
       within such NOTICE file, excluding those notices that do not
       pertain to any part of the Derivative Works, in at least one
       of the following places: within a NOTICE text file distributed
       as part of the Derivative Works; within the Source form or
       documentation, if provided along with the Derivative Works; or,
       within a display generated by the Derivative Works, if and
       wherever such third-party notices normally appear. The contents
       of the NOTICE file are for informational purposes only and
       do not modify the License. You may add Your own attribution
       notices within Derivative Works that You distribute, alongside
       or as an addendum to the NOTICE text from the Work, provided
       that such additional attribution notices cannot be construed
       as modifying the License.

   You may add Your own copyright statement to Your modifications and
   may provide additional or different license terms and conditions
   for use, reproduction, or distribution of Your modifications, or
   for any such Derivative Works as a whole, provided Your use,
   reproduction, and distribution of the Work otherwise complies with
   the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
   any Contribution intentionally submitted for inclusion in the Work
   by You to the Licensor shall be under the terms and conditions of
   this License, without any additional terms or conditions.
   Notwithstanding the above, nothing herein shall supersede or modify
   the terms of any separate license agreement you may have executed
   with Licensor regarding such Contributions.
```

```
   6. Trademarks. This License does not grant permission to use the trade
      names, trademarks, service marks, or product names of the Licensor,
      except as required for reasonable and customary use in describing the
      origin of the Work and reproducing the content of the NOTICE file.

   7. Disclaimer of Warranty. Unless required by applicable law or
      agreed to in writing, Licensor provides the Work (and each
      Contributor provides its Contributions) on an "AS IS" BASIS,
      WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
      implied, including, without limitation, any warranties or conditions
      of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
      PARTICULAR PURPOSE. You are solely responsible for determining the
      appropriateness of using or redistributing the Work and assume any
      risks associated with Your exercise of permissions under this License.

   8. Limitation of Liability. In no event and under no legal theory,
      whether in tort (including negligence), contract, or otherwise,
      unless required by applicable law (such as deliberate and grossly
      negligent acts) or agreed to in writing, shall any Contributor be
      liable to You for damages, including any direct, indirect, special,
      incidental, or consequential damages of any character arising as a
      result of this License or out of the use or inability to use the
      Work (including but not limited to damages for loss of goodwill,
      work stoppage, computer failure or malfunction, or any and all
      other commercial damages or losses), even if such Contributor
      has been advised of the possibility of such damages.

   9. Accepting Warranty or Additional Liability. While redistributing
      the Work or Derivative Works thereof, You may choose to offer,
      and charge a fee for, acceptance of support, warranty, indemnity,
      or other liability obligations and/or rights consistent with this
      License. However, in accepting such obligations, You may act only
      on Your own behalf and on Your sole responsibility, not on behalf
      of any other Contributor, and only if You agree to indemnify,
      defend, and hold each Contributor harmless for any liability
      incurred by, or claims asserted against, such Contributor by reason
      of your accepting any such warranty or additional liability.

   END OF TERMS AND CONDITIONS

   APPENDIX: How to apply the Apache License to your work.

      To apply the Apache License to your work, attach the following
      boilerplate notice, with the fields enclosed by brackets "{}"
      replaced with your own identifying information. (Don't include
      the brackets!)  The text should be enclosed in the appropriate
      comment syntax for the file format. We also recommend that a
      file or class name and description of purpose be included on the
      same "printed page" as the copyright notice for easier
      identification within third-party archives.

   Copyright {yyyy} {name of copyright owner}

   Licensed under the Apache License, Version 2.0 (the "License");
   you may not use this file except in compliance with the License.
   You may obtain a copy of the License at

       http://www.apache.org/licenses/LICENSE-2.0

   Unless required by applicable law or agreed to in writing, software
   distributed under the License is distributed on an "AS IS" BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
   See the License for the specific language governing permissions and
   limitations under the License.


Colm
================================================================================
```

```
Copyright (c) 2006-2018 Adrian Thurston <thurston@colm.net>

Permission is hereby granted, free of charge, to any person obtaining a copy of
this software and associated documentation files (the "Software"), to deal in
the Software without restriction, including without limitation the rights to
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies
of the Software, and to permit persons to whom the Software is furnished to do
so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.


Decnumber
================================================================================
COPYRIGHT AND PERMISSION NOTICE

Copyright (c) 1995-2015 International Business Machines Corporation and others

All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of
this software and associated documentation files (the "Software"), to deal in
the Software without restriction, including without limitation the rights to
use, copy, modify, merge, publish, distribute, and/or sell copies of the
Software, and to permit persons to whom the Software is furnished to do so,
provided that the above copyright notice(s) and this permission notice appear in
all copies of the Software and that both the above copyright notice(s) and this
permission notice appear in supporting documentation.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS
FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT
SHALL THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE BE LIABLE FOR ANY
CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES
WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF
CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION
WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Except as contained in this notice, the name of a copyright holder shall not be
used in advertising or otherwise to promote the sale, use or other dealings in
this Software without prior written authorization of the copyright holder.

All trademarks and registered trademarks mentioned herein are the property of
their respective owners.

Duktape
================================================================================
===============
Duktape license
===============

(http://opensource.org/licenses/MIT)

Copyright (c) 2013-2019 by Duktape authors (see AUTHORS.rst)

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
```

```
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.


Farmhash
================================================================================
Copyright (c) 2014 Google, Inc.

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.

Fastfunc
================================================================================
This is free and unencumbered software released into the public domain.

  Anyone is free to copy, modify, publish, use, compile, sell, or
  distribute this software, either in source code form or as a compiled
  binary, for any purpose, commercial or non-commercial, and by any
  means.

  In jurisdictions that recognize copyright laws, the author or authors
  of this software dedicate any and all copyright interest in the
  software to the public domain. We make this dedication for the benefit
  of the public at large and to the detriment of our heirs and
  successors. We intend this dedication to be an overt act of
  relinquishment in perpetuity of all present and future rights to this
  software under copyright law.

  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
  EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
  MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
  IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR
  OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
  ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
  OTHER DEALINGS IN THE SOFTWARE.

  For more information, please refer to <http://unlicense.org/>

FastUTF8
================================================================================
Copyright (c) 2008-2009 Bjoern Hoehrmann <bjoern@hoehrmann.de>
```

```
Permission is hereby granted, free of charge, to any person obtaining a copy of
this software and associated documentation files (the "Software"), to deal in
the Software without restriction, including without limitation the rights to
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of
the Software, and to permit persons to whom the Software is furnished to do so,
subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS
 FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER
IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.


Google Breakpad
================================================================================
Copyright (c) 2006, Google Inc.
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

   * Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.
   * Redistributions in binary form must reproduce the above
copyright notice, this list of conditions and the following disclaimer
in the documentation and/or other materials provided with the
distribution.
   * Neither the name of Google Inc. nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

---------------------------------------------------------------------

Copyright 2001-2004 Unicode, Inc.

Disclaimer

This source code is provided as is by Unicode, Inc. No claims are
made as to fitness for any particular purpose. No warranties of any
kind are expressed or implied. The recipient agrees to determine
applicability of information provided. If this file has been
purchased on magnetic or optical media from Unicode, Inc., the
sole remedy for any claim will be exchange of defective media
within 90 days of receipt.

Limitations on Rights to Redistribute This Code

Unicode, Inc. hereby grants the right to freely use the information
supplied in this file in the creation of products supporting the
Unicode Standard, and to make copies of this file in any form
```

```
for internal or external distribution as long as this notice
remains attached.



Google Test
================================================================================
Copyright 2008, Google Inc.
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

   * Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.
   * Redistributions in binary form must reproduce the above
copyright notice, this list of conditions and the following disclaimer
in the documentation and/or other materials provided with the
distribution.
   * Neither the name of Google Inc. nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.


ICU License - ICU 1.8.1 and later
================================================================================
COPYRIGHT AND PERMISSION NOTICE

Copyright (c) 1995-2015 International Business Machines Corporation and others

All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF

Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sa

All trademarks and registered trademarks mentioned herein are the property of their respective owners.

Third-Party Software Licenses
This section contains third-party software notices and/or additional terms for licensed third-party software components included w
1. Unicode Data Files and Software
COPYRIGHT AND PERMISSION NOTICE

Copyright © 1991-2015 Unicode, Inc. All rights reserved.
Distributed under the Terms of Use in
http://www.unicode.org/copyright.html.

Permission is hereby granted, free of charge, to any person obtaining
a copy of the Unicode data files and any associated documentation
(the "Data Files") or Unicode software and any associated documentation
(the "Software") to deal in the Data Files or Software
without restriction, including without limitation the rights to use,
copy, modify, merge, publish, distribute, and/or sell copies of
```

```
the Data Files or Software, and to permit persons to whom the Data Files
or Software are furnished to do so, provided that
(a) this copyright and permission notice appear with all copies
of the Data Files or Software,
(b) this copyright and permission notice appear in associated
documentation, and
(c) there is clear notice in each modified Data File or in the Software
as well as in the documentation associated with the Data File(s) or
Software that the data or software has been modified.

THE DATA FILES AND SOFTWARE ARE PROVIDED "AS IS", WITHOUT WARRANTY OF
ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE
WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
NONINFRINGEMENT OF THIRD PARTY RIGHTS.
IN NO EVENT SHALL THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS
NOTICE BE LIABLE FOR ANY CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL
DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE,
DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER
TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
PERFORMANCE OF THE DATA FILES OR SOFTWARE.

Except as contained in this notice, the name of a copyright holder
shall not be used in advertising or otherwise to promote the sale,
use or other dealings in these Data Files or Software without prior
written authorization of the copyright holder.
2. Chinese/Japanese Word Break Dictionary Data (cjdict.txt)
   The Google Chrome software developed by Google is licensed under the BSD license. Other software included in this distribution

    The BSD License
    http://opensource.org/licenses/bsd-license.php
    Copyright (C) 2006-2008, Google Inc.

    All rights reserved.

    Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following con

    Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
    Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer
    Neither the name of  Google Inc. nor the names of its contributors may be used to endorse or promote products derived from thi


    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,


    The word list in cjdict.txt are generated by combining three word lists listed
    below with further processing for compound word breaking. The frequency is generated
    with an iterative training against Google web corpora.

    * Libtabe (Chinese)
      - https://sourceforge.net/project/?group_id=1519
      - Its license terms and conditions are shown below.

    * IPADIC (Japanese)
      - http://chasen.aist-nara.ac.jp/chasen/distribution.html
      - Its license terms and conditions are shown below.

    ---------COPYING.libtabe ---- BEGIN--------------------

    /*
     * Copyrighy (c) 1999 TaBE Project.
     * Copyright (c) 1999 Pai-Hsiang Hsiao.
     * All rights reserved.
     *
     * Redistribution and use in source and binary forms, with or without
     * modification, are permitted provided that the following conditions
     * are met:
     *
     * . Redistributions of source code must retain the above copyright
```

```
 *   notice, this list of conditions and the following disclaimer.
 * . Redistributions in binary form must reproduce the above copyright
 *   notice, this list of conditions and the following disclaimer in
 *   the documentation and/or other materials provided with the
 *   distribution.
 * . Neither the name of the TaBE Project nor the names of its
 *   contributors may be used to endorse or promote products derived
 *   from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
 * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
 * REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
 * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
 * OF THE POSSIBILITY OF SUCH DAMAGE.
 */

/*
 * Copyright (c) 1999 Computer Systems and Communication Lab,
 *                    Institute of Information Science, Academia Sinica.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * . Redistributions of source code must retain the above copyright
 *   notice, this list of conditions and the following disclaimer.
 * . Redistributions in binary form must reproduce the above copyright
 *   notice, this list of conditions and the following disclaimer in
 *   the documentation and/or other materials provided with the
 *   distribution.
 * . Neither the name of the Computer Systems and Communication Lab
 *   nor the names of its contributors may be used to endorse or
 *   promote products derived from this software without specific
 *   prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
 * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
 * REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
 * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
 * OF THE POSSIBILITY OF SUCH DAMAGE.
 */

Copyright 1996 Chih-Hao Tsai @ Beckman Institute, University of Illinois
c-tsai4@uiuc.edu  http://casper.beckman.uiuc.edu/~c-tsai4

---------------COPYING.libtabe-----END-----------------------------------


---------------COPYING.ipadic-----BEGIN----------------------------------

Copyright 2000, 2001, 2002, 2003 Nara Institute of Science
and Technology.  All Rights Reserved.
```

```
Use, reproduction, and distribution of this software is permitted.
Any copy of this software, whether in its original form or modified,
must include both the above copyright notice and the following
paragraphs.

Nara Institute of Science and Technology (NAIST),
the copyright holders, disclaims all warranties with regard to this
software, including all implied warranties of merchantability and
fitness, in no event shall NAIST be liable for
any special, indirect or consequential damages or any damages
whatsoever resulting from loss of use, data or profits, whether in an
action of contract, negligence or other tortuous action, arising out
of or in connection with the use or performance of this software.

A large portion of the dictionary entries
originate from ICOT Free Software.  The following conditions for ICOT
Free Software applies to the current dictionary as well.

Each User may also freely distribute the Program, whether in its
original form or modified, to any third party or parties, PROVIDED
that the provisions of Section 3 ("NO WARRANTY") will ALWAYS appear
on, or be attached to, the Program, which is distributed substantially
in the same form as set out herein and that such intended
distribution, if actually made, will neither violate or otherwise
contravene any of the laws and regulations of the countries having
jurisdiction over the User or the intended distribution itself.

NO WARRANTY

The program was produced on an experimental basis in the course of the
research and development conducted during the project and is provided
to users as so produced on an experimental basis.  Accordingly, the
program is provided without any warranty whatsoever, whether express,
implied, statutory or otherwise.  The term "warranty" used herein
includes, but is not limited to, any warranty of the quality,
performance, merchantability and fitness for a particular purpose of
the program and the nonexistence of any infringement or violation of
any right of any third party.

Each user of the program will agree and understand, and be deemed to
have agreed and understood, that there is no warranty whatsoever for
the program and, accordingly, the entire risk arising from or
otherwise connected with the program is assumed by the user.

Therefore, neither ICOT, the copyright holder, or any other
organization that participated in or was otherwise related to the
development of the program and their respective officials, directors,
officers and other employees shall be held liable for any and all
damages, including, without limitation, general, special, incidental
and consequential damages, arising out of or otherwise in connection
with the use or inability to use the program or any product, material
or result produced or otherwise obtained by using the program,
regardless of whether they have been advised of, or otherwise had
knowledge of, the possibility of such damages at any time during the
project or thereafter.  Each user will be deemed to have agreed to the
foregoing by his or her commencement of use of the program.  The term
"use" as used herein includes, but is not limited to, the use,
modification, copying and distribution of the program and the
production of secondary products from the program.

In the case where the program, whether in its original form or
modified, was distributed or delivered to or received by a user from
any person, organization or entity other than ICOT, unless it makes or
grants independently of ICOT any specific warranty to the user in
writing, such person, organization or entity, will also be exempted
from and not be held liable to the user for any such damages as noted
above as far as the program is concerned.
```

```
    --------------COPYING.ipadic-----END----------------------------------
3. Lao Word Break Dictionary Data (laodict.txt)
    Copyright (c) 2013 International Business Machines Corporation
    and others. All Rights Reserved.

    Project:    http://code.google.com/p/lao-dictionary/
    Dictionary: http://lao-dictionary.googlecode.com/git/Lao-Dictionary.txt
    License:    http://lao-dictionary.googlecode.com/git/Lao-Dictionary-LICENSE.txt
                (copied below)

    This file is derived from the above dictionary, with slight modifications.
    --------------------------------------------------------------------------------
    Copyright (C) 2013 Brian Eugene Wilson, Robert Martin Campbell.
    All rights reserved.

    Redistribution and use in source and binary forms, with or without modification,
    are permitted provided that the following conditions are met:

        Redistributions of source code must retain the above copyright notice, this
        list of conditions and the following disclaimer. Redistributions in binary
        form must reproduce the above copyright notice, this list of conditions and
        the following disclaimer in the documentation and/or other materials
        provided with the distribution.

    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
    ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
    WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
    DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
    ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
    (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
    LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
    ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
    (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
    SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
    --------------------------------------------------------------------------------
4. Burmese Word Break Dictionary Data (burmesedict.txt)
    Copyright (c) 2014 International Business Machines Corporation
    and others. All Rights Reserved.

    This list is part of a project hosted at:
      github.com/kanyawtech/myanmar-karen-word-lists

    --------------------------------------------------------------------------------
    Copyright (c) 2013, LeRoy Benjamin Sharon
    All rights reserved.

    Redistribution and use in source and binary forms, with or without modification,
    are permitted provided that the following conditions are met:

      Redistributions of source code must retain the above copyright notice, this
      list of conditions and the following disclaimer.

      Redistributions in binary form must reproduce the above copyright notice, this
      list of conditions and the following disclaimer in the documentation and/or
      other materials provided with the distribution.

      Neither the name Myanmar Karen Word Lists, nor the names of its
      contributors may be used to endorse or promote products derived from
      this software without specific prior written permission.

    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
    ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
    WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
    DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
    ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
    (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
    LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
    ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
```

```
    (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
    SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
    -----------------------------------------------------------------------------
5. Time Zone Database
ICU uses the public domain data and code derived from Time Zone Database for its time zone support. The ownership of the TZ databa

7.  Database Ownership

   The TZ database itself is not an IETF Contribution or an IETF
   document.  Rather it is a pre-existing and regularly updated work
   that is in the public domain, and is intended to remain in the public
   domain.  Therefore, BCPs 78 [RFC5378] and 79 [RFC3979] do not apply
   to the TZ Database or contributions that individuals make to it.
   Should any claims be made and substantiated against the TZ Database,
   the organization that is providing the IANA Considerations defined in
   this RFC, under the memorandum of understanding with the IETF,
   currently ICANN, may act in accordance with all competent court
   orders.  No ownership claims will be made by ICANN or the IETF Trust
   on the database or the code.  Any person making a contribution to the
   database or code waives all rights to future claims in that
   contribution or in the TZ Database.


intel math
============================================================================
Intel Simplified Software License (Version August 2021)

Use and Redistribution. You may use and redistribute the software (the
"Software"), without modification, provided the following conditions are met:

* Redistributions must reproduce the above copyright notice and the following
  terms of use in the Software and in the documentation and/or other materials
  provided with the distribution.
* Neither the name of Intel nor the names of its suppliers may be used to
  endorse or promote products derived from this Software without specific prior
  written permission.
* No reverse engineering, decompilation, or disassembly of this Software is
  permitted.

No other licenses.  Except as provided in the preceding section, Intel grants
no licenses or other rights by implication, estoppel or otherwise to, patent, copyright,
trademark, trade name, service mark or other intellectual property licenses or rights of Intel.

Third party software.  The Software may contain Third Party Software. "Third
Party Software" is open source software, third party software, or other Intel software
that may be identified in the Software itself or in the files (if any) listed in the
"third-party-software.txt" or similarly named text file included with the Software.
Third Party Software, even if included with the distribution of the Software, may be governed
by separate license terms, including without limitation, open source software license terms, third
party software license terms, and other Intel software license terms.
Those separate license terms solely govern your use of the Third Party Software, and nothing
in this license limits any rights under, or grants rights that supersede, the terms of the applicable
license terms.

DISCLAIMER. THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT ARE
DISCLAIMED. THIS SOFTWARE IS NOT INTENDED FOR USE IN SYSTEMS OR APPLICATIONS
WHERE FAILURE OF THE SOFTWARE MAY CAUSE PERSONAL INJURY OR DEATH AND YOU AGREE
THAT YOU ARE FULLY RESPONSIBLE FOR ANY CLAIMS, COSTS, DAMAGES, EXPENSES, AND
ATTORNEYS' FEES ARISING OUT OF ANY SUCH USE, EVEN IF ANY CLAIM ALLEGES THAT
INTEL WAS NEGLIGENT REGARDING THE DESIGN OR MANUFACTURE OF THE MATERIALS.

LIMITATION OF LIABILITY. IN NO EVENT WILL INTEL BE LIABLE FOR ANY DIRECT,
INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
```

```
OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

No support. Intel may make changes to the Software, at any time without notice,
and is not obligated to support, update or provide training for the Software.

Termination. Your right to use the Software is terminated in the event of your breach
of this license.

Feedback. Should you provide Intel with comments, modifications, corrections,
enhancements or other input ("Feedback") related to the Software, Intel will be
free to use, disclose, reproduce, license or otherwise distribute or exploit the
Feedback in its sole discretion without any obligations or restrictions of any
kind, including without limitation, intellectual property rights or licensing
obligations.

Compliance with laws. You agree to comply with all relevant laws and regulations
governing your use, transfer, import or export (or prohibition thereof) of the
Software.

Governing law. All disputes will be governed by the laws of the United States of
America and the State of Delaware without reference to conflict of law
principles and subject to the exclusive jurisdiction of the state or federal
courts sitting in the State of Delaware, and each party agrees that it submits
to the personal jurisdiction and venue of those courts and waives any
objections. The United Nations Convention on Contracts for the International
Sale of Goods (1980) is specifically excluded and will not apply to the
Software.

invert_normal
================================================================================

Copyright (c) 2002
Mark Joshi

Permission to use, copy, modify, distribute and sell this
software for any purpose is hereby
granted without fee, provided that the above copyright notice
appear in all copies and that both that copyright notice and
this permission notice appear in supporting documentation.
Mark Joshi makes no representations about the
suitability of this software for any purpose. It is provided
"as is" without express or implied warranty.


jemalloc
================================================================================

Unless otherwise specified, files in the jemalloc source distribution are
subject to the following license:
--------------------------------------------------------------------------------
Copyright (C) 2002-2014 Jason Evans <jasone@canonware.com>.
All rights reserved.
Copyright (C) 2007-2012 Mozilla Foundation.  All rights reserved.
Copyright (C) 2009-2014 Facebook, Inc.  All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:
1. Redistributions of source code must retain the above copyright notice(s),
   this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice(s),
   this list of conditions and the following disclaimer in the documentation
   and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDER(S) ``AS IS'' AND ANY EXPRESS
OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO
EVENT SHALL THE COPYRIGHT HOLDER(S) BE LIABLE FOR ANY DIRECT, INDIRECT,
```

```
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
----------------------------------------------------------------------------


libPCIAccess
============================================================================

(C) Copyright IBM Corporation 2006, 2007
(C) Copyright Eric Anholt 2006
(C) Copyright Mark Kettenis 2011
(C) Copyright Robert Millan 2012
Copyright (c) 2007, 2008, 2009, 2011, 2012, 2013 Oracle and/or its affiliates.
Copyright 2009, 2012 Red Hat, Inc.
All Rights Reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to
deal in the Software without restriction, including without limitation on
the rights to use, copy, modify, merge, publish, distribute, sub license,
and/or sell copies of the Software, and to permit persons to whom the
Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice (including the next
paragraph) shall be included in all copies or substantial portions of the
Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT.  IN NO EVENT SHALL
IBM AND/OR THEIR SUPPLIERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
IN THE SOFTWARE.

----------------------------------------------------------------------------

Copyright (c) 2008 Juan Romero Pardines
Copyright (c) 2008, 2011 Mark Kettenis
Copyright (c) 2009 Michael Lorenz
Copyright (c) 2009, 2012 Samuel Thibault

Permission to use, copy, modify, and distribute this software for any
purpose with or without fee is hereby granted, provided that the above
copyright notice and this permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

----------------------------------------------------------------------------

Copyright (C) 2000 The XFree86 Project, Inc.  All Rights Reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to
deal in the Software without restriction, including without limitation the
rights to use, copy, modify, merge, publish, distribute, sublicense, and/or
sell copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:
```

```
The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.  IN NO EVENT SHALL THE
XFREE86 PROJECT BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER
IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Except as contained in this notice, the name of the XFree86 Project shall
not be used in advertising or otherwise to promote the sale, use or other
dealings in this Software without prior written authorization from the
XFree86 Project.

------------------------------------------------------------------------------

Copyright (c) 2007 Paulo R. Zanoni, Tiago Vignatti
Copyright (c) 2009 Tiago Vignatti

Permission is hereby granted, free of charge, to any person
obtaining a copy of this software and associated documentation
files (the "Software"), to deal in the Software without
restriction, including without limitation the rights to use,
copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the
Software is furnished to do so, subject to the following
conditions:

The above copyright notice and this permission notice shall be
included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
OTHER DEALINGS IN THE SOFTWARE.


LZ4 Library
================================================================================
Copyright (c) 2011-2016, Yann Collet
All rights reserved.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this
  list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice, this
  list of conditions and the following disclaimer in the documentation and/or
  other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

```
MurmurHash3
================================================================================
MurmurHash3 was written by Austin Appleby, and is placed in the public
domain. The author hereby disclaims copyright to this source code.

Project Nayuki
================================================================================
Copyright © 2018 Project Nayuki. (MIT License)

Permission is hereby granted, free of charge, to any person obtaining a copy of
this software and associated documentation files (the "Software"), to deal in
the Software without restriction, including without limitation the rights to
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of
the Software, and to permit persons to whom the Software is furnished to do so,
subject to the following conditions:

* The above copyright notice and this permission notice shall be included in
  all copies or substantial portions of the Software.

* The Software is provided "as is", without warranty of any kind, express or
  implied, including but not limited to the warranties of merchantability,
  fitness for a particular purpose and noninfringement. In no event shall the
  authors or copyright holders be liable for any claim, damages or other
  liability, whether in an action of contract, tort or otherwise, arising from,
  out of or in connection with the Software or the use or other dealings in the
  Software.

openwall
================================================================================

This is an OpenSSL-compatible implementation of the RSA Data Security, Inc.
MD5 Message-Digest Algorithm (RFC 1321).

Homepage:
http://openwall.info/wiki/people/solar/software/public-domain-source-code/md5

Author:
Alexander Peslyak, better known as Solar Designer <solar at openwall.com>

This software was written by Alexander Peslyak in 2001.  No copyright is
claimed, and the software is hereby placed in the public domain.
In case this attempt to disclaim copyright and place the software in the
public domain is deemed null and void, then the software is
Copyright (c) 2001 Alexander Peslyak and it is hereby released to the
general public under the following terms:

Redistribution and use in source and binary forms, with or without
modification, are permitted.

There's ABSOLUTELY NO WARRANTY, express or implied.

(This is a heavily cut-down "BSD license".)

This differs from Colin Plumb's older public domain implementation in that
no exactly 32-bit integer data type is required (any 32-bit or wider
unsigned integer data type will do), there's no compile-time endianness
configuration, and the function prototypes match OpenSSL's.  No code from
Colin Plumb's implementation has been reused; this comment merely compares
the properties of the two independent implementations.

The primary goals of this implementation are portability and ease of use.
It is meant to be fast, but not as fast as possible.  Some known
optimizations are not included to reduce source code size and avoid
compile-time configuration.
```

```
Protegrity Application Protector API definitions
================================================================================
(c) 2007-2013 Protegrity Corporation. All Rights Reserved.

ragel
================================================================================
Copyright (c) 2001-2018 Adrian Thurston <thurston@colm.net> et al.

Permission is hereby granted, free of charge, to any person obtaining a copy of
this software and associated documentation files (the "Software"), to deal in
the Software without restriction, including without limitation the rights to
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies
of the Software, and to permit persons to whom the Software is furnished to do
so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.


Use of this software is granted under one of the following two licenses,
to be chosen freely by the user.

1. Boost Software License - Version 1.0 - August 17th, 2003
================================================================================

Copyright (c) 2006, 2007 Marcin Kalicinski

Permission is hereby granted, free of charge, to any person or organization
obtaining a copy of the software and accompanying documentation covered by
this license (the "Software") to use, reproduce, display, distribute,
execute, and transmit the Software, and to prepare derivative works of the
Software, and to permit third-parties to whom the Software is furnished to
do so, all subject to the following:

The copyright notices in the Software and this entire statement, including
the above license grant, this restriction and the following disclaimer,
must be included in all copies of the Software, in whole or in part, and
all derivative works of the Software, unless such copies or derivative
works are solely in the form of machine-executable object code generated by
a source language processor.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT
SHALL THE COPYRIGHT HOLDERS OR ANYONE DISTRIBUTING THE SOFTWARE BE LIABLE
FOR ANY DAMAGES OR OTHER LIABILITY, WHETHER IN CONTRACT, TORT OR OTHERWISE,
ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN THE SOFTWARE.

2. The MIT License
================================================================================

Copyright (c) 2006, 2007 Marcin Kalicinski

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies
of the Software, and to permit persons to whom the Software is furnished to do so,
subject to the following conditions:
```

```
The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
IN THE SOFTWARE.


rsocket
===============================================================================

Apache License
                           Version 2.0, January 2004
                        http://www.apache.org/licenses/

   TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

   1. Definitions.

      "License" shall mean the terms and conditions for use, reproduction,
      and distribution as defined by Sections 1 through 9 of this document.

      "Licensor" shall mean the copyright owner or entity authorized by
      the copyright owner that is granting the License.

      "Legal Entity" shall mean the union of the acting entity and all
      other entities that control, are controlled by, or are under common
      control with that entity. For the purposes of this definition,
      "control" means (i) the power, direct or indirect, to cause the
      direction or management of such entity, whether by contract or
      otherwise, or (ii) ownership of fifty percent (50%) or more of the
      outstanding shares, or (iii) beneficial ownership of such entity.

      "You" (or "Your") shall mean an individual or Legal Entity
      exercising permissions granted by this License.

      "Source" form shall mean the preferred form for making modifications,
      including but not limited to software source code, documentation
      source, and configuration files.

      "Object" form shall mean any form resulting from mechanical
      transformation or translation of a Source form, including but
      not limited to compiled object code, generated documentation,
      and conversions to other media types.

      "Work" shall mean the work of authorship, whether in Source or
      Object form, made available under the License, as indicated by a
      copyright notice that is included in or attached to the work
      (an example is provided in the Appendix below).

      "Derivative Works" shall mean any work, whether in Source or Object
      form, that is based on (or derived from) the Work and for which the
      editorial revisions, annotations, elaborations, or other modifications
      represent, as a whole, an original work of authorship. For the purposes
      of this License, Derivative Works shall not include works that remain
      separable from, or merely link (or bind by name) to the interfaces of,
      the Work and Derivative Works thereof.

      "Contribution" shall mean any work of authorship, including
      the original version of the Work and any modifications or additions
      to that Work or Derivative Works thereof, that is intentionally
      submitted to Licensor for inclusion in the Work by the copyright owner
      or by an individual or Legal Entity authorized to submit on behalf of
```

```
      the copyright owner. For the purposes of this definition, "submitted"
      means any form of electronic, verbal, or written communication sent
      to the Licensor or its representatives, including but not limited to
      communication on electronic mailing lists, source code control systems,
      and issue tracking systems that are managed by, or on behalf of, the
      Licensor for the purpose of discussing and improving the Work, but
      excluding communication that is conspicuously marked or otherwise
      designated in writing by the copyright owner as "Not a Contribution."

      "Contributor" shall mean Licensor and any individual or Legal Entity
      on behalf of whom a Contribution has been received by Licensor and
      subsequently incorporated within the Work.

   2. Grant of Copyright License. Subject to the terms and conditions of
      this License, each Contributor hereby grants to You a perpetual,
      worldwide, non-exclusive, no-charge, royalty-free, irrevocable
      copyright license to reproduce, prepare Derivative Works of,
      publicly display, publicly perform, sublicense, and distribute the
      Work and such Derivative Works in Source or Object form.

   3. Grant of Patent License. Subject to the terms and conditions of
      this License, each Contributor hereby grants to You a perpetual,
      worldwide, non-exclusive, no-charge, royalty-free, irrevocable
      (except as stated in this section) patent license to make, have made,
      use, offer to sell, sell, import, and otherwise transfer the Work,
      where such license applies only to those patent claims licensable
      by such Contributor that are necessarily infringed by their
      Contribution(s) alone or by combination of their Contribution(s)
      with the Work to which such Contribution(s) was submitted. If You
      institute patent litigation against any entity (including a
      cross-claim or counterclaim in a lawsuit) alleging that the Work
      or a Contribution incorporated within the Work constitutes direct
      or contributory patent infringement, then any patent licenses
      granted to You under this License for that Work shall terminate
      as of the date such litigation is filed.

   4. Redistribution. You may reproduce and distribute copies of the
      Work or Derivative Works thereof in any medium, with or without
      modifications, and in Source or Object form, provided that You
      meet the following conditions:

      (a) You must give any other recipients of the Work or
          Derivative Works a copy of this License; and

      (b) You must cause any modified files to carry prominent notices
          stating that You changed the files; and

      (c) You must retain, in the Source form of any Derivative Works
          that You distribute, all copyright, patent, trademark, and
          attribution notices from the Source form of the Work,
          excluding those notices that do not pertain to any part of
          the Derivative Works; and

      (d) If the Work includes a "NOTICE" text file as part of its
          distribution, then any Derivative Works that You distribute must
          include a readable copy of the attribution notices contained
          within such NOTICE file, excluding those notices that do not
          pertain to any part of the Derivative Works, in at least one
          of the following places: within a NOTICE text file distributed
          as part of the Derivative Works; within the Source form or
          documentation, if provided along with the Derivative Works; or,
          within a display generated by the Derivative Works, if and
          wherever such third-party notices normally appear. The contents
          of the NOTICE file are for informational purposes only and
          do not modify the License. You may add Your own attribution
          notices within Derivative Works that You distribute, alongside
          or as an addendum to the NOTICE text from the Work, provided
          that such additional attribution notices cannot be construed
```

```
      as modifying the License.

   You may add Your own copyright statement to Your modifications and
   may provide additional or different license terms and conditions
   for use, reproduction, or distribution of Your modifications, or
   for any such Derivative Works as a whole, provided Your use,
   reproduction, and distribution of the Work otherwise complies with
   the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
   any Contribution intentionally submitted for inclusion in the Work
   by You to the Licensor shall be under the terms and conditions of
   this License, without any additional terms or conditions.
   Notwithstanding the above, nothing herein shall supersede or modify
   the terms of any separate license agreement you may have executed
   with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
   names, trademarks, service marks, or product names of the Licensor,
   except as required for reasonable and customary use in describing the
   origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
   agreed to in writing, Licensor provides the Work (and each
   Contributor provides its Contributions) on an "AS IS" BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
   implied, including, without limitation, any warranties or conditions
   of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
   PARTICULAR PURPOSE. You are solely responsible for determining the
   appropriateness of using or redistributing the Work and assume any
   risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
   whether in tort (including negligence), contract, or otherwise,
   unless required by applicable law (such as deliberate and grossly
   negligent acts) or agreed to in writing, shall any Contributor be
   liable to You for damages, including any direct, indirect, special,
   incidental, or consequential damages of any character arising as a
   result of this License or out of the use or inability to use the
   Work (including but not limited to damages for loss of goodwill,
   work stoppage, computer failure or malfunction, or any and all
   other commercial damages or losses), even if such Contributor
   has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
   the Work or Derivative Works thereof, You may choose to offer,
   and charge a fee for, acceptance of support, warranty, indemnity,
   or other liability obligations and/or rights consistent with this
   License. However, in accepting such obligations, You may act only
   on Your own behalf and on Your sole responsibility, not on behalf
   of any other Contributor, and only if You agree to indemnify,
   defend, and hold each Contributor harmless for any liability
   incurred by, or claims asserted against, such Contributor by reason
   of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

   To apply the Apache License to your work, attach the following
   boilerplate notice, with the fields enclosed by brackets "[]"
   replaced with your own identifying information. (Don't include
   the brackets!)  The text should be enclosed in the appropriate
   comment syntax for the file format. We also recommend that a
   file or class name and description of purpose be included on the
   same "printed page" as the copyright notice for easier
   identification within third-party archives.
```

```
   Copyright [yyyy] [name of copyright owner]

   Licensed under the Apache License, Version 2.0 (the "License");
   you may not use this file except in compliance with the License.
   You may obtain a copy of the License at

      http://www.apache.org/licenses/LICENSE-2.0

   Unless required by applicable law or agreed to in writing, software
   distributed under the License is distributed on an "AS IS" BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
   See the License for the specific language governing permissions and
   limitations under the License.

sparsehash
================================================================================
Copyright (c) 2005, Google Inc.
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

   * Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.
   * Redistributions in binary form must reproduce the above
copyright notice, this list of conditions and the following disclaimer
in the documentation and/or other materials provided with the
distribution.
   * Neither the name of Google Inc. nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.


TommyDS
================================================================================
Tommy is released with a 2-clause BSD license.
Copyright (c) 2010, Andrea Mazzoleni. All rights reserved.

 Redistribution and use in source and binary forms, with or without
 modification, are permitted provided that the following conditions
 are met:

 1. Redistributions of source code must retain the above copyright
    notice, this list of conditions and the following disclaimer.

 2. Redistributions in binary form must reproduce the above copyright
    notice, this list of conditions and the following disclaimer in the
    documentation and/or other materials provided with the distribution.

 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
```

```
 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 POSSIBILITY OF SUCH DAMAGE.

libunwind
================================================================================
Copyright (c) 2002 Hewlett-Packard Co.

Permission is hereby granted, free of charge, to any person obtaining
a copy of this software and associated documentation files (the
"Software"), to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish,
distribute, sublicense, and/or sell copies of the Software, and to
permit persons to whom the Software is furnished to do so, subject to
the following conditions:

The above copyright notice and this permission notice shall be
included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE
LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

vectorclass
================================================================================
(c) Copyright 2012-2019 Agner Fog.
Apache License version 2.0 or later.

xxHash Library
================================================================================
Copyright (c) 2012-2020 Yann Collet
All rights reserved.

BSD 2-Clause License (https://www.opensource.org/licenses/bsd-license.php)

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this
  list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice, this
  list of conditions and the following disclaimer in the documentation and/or
  other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

    ----------------------------------------------------------------------------------------
    angular-animate:1.5.5
    MIT
```

```
URL:

--------------------------------------------------------------------------------
angular-clipboard:1.3.0
MIT

    The MIT License (MIT)

    Copyright (c) 2015

    Permission is hereby granted, free of charge, to any person obtaining a copy
    of this software and associated documentation files (the "Software"), to deal
    in the Software without restriction, including without limitation the rights
    to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
    copies of the Software, and to permit persons to whom the Software is
    furnished to do so, subject to the following conditions:

    The above copyright notice and this permission notice shall be included in all
    copies or substantial portions of the Software.

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
    FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
    AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
    LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
    OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
    SOFTWARE.


URL: LICENSE

--------------------------------------------------------------------------------
angular-daterangepicker:0.2.2
MIT

    The MIT License (MIT)

    Copyright (c) 2015 Fragaria, s.r.o.

    Permission is hereby granted, free of charge, to any person obtaining a copy
    of this software and associated documentation files (the "Software"), to deal
    in the Software without restriction, including without limitation the rights
    to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
    copies of the Software, and to permit persons to whom the Software is
    furnished to do so, subject to the following conditions:

    The above copyright notice and this permission notice shall be included in
    all copies or substantial portions of the Software.

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
    FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
    AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
    LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
    OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
    THE SOFTWARE.


URL: LICENSE.md

--------------------------------------------------------------------------------
angular-debounce:1.0.3
MIT

    Copyright (C) 2014 by Ruben Vermeersch <ruben@rocketeer.be>

    Permission is hereby granted, free of charge, to any person obtaining a copy
    of this software and associated documentation files (the "Software"), to deal
```

```
    in the Software without restriction, including without limitation the rights
    to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
    copies of the Software, and to permit persons to whom the Software is
    furnished to do so, subject to the following conditions:

    The above copyright notice and this permission notice shall be included in
    all copies or substantial portions of the Software.

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
    FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
    AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
    LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
    OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
    THE SOFTWARE.


URL: LICENSE

--------------------------------------------------------------------------------
angular-files-drop:1.0.0
MIT



URL:

--------------------------------------------------------------------------------
angular-gettext:2.2.1
MIT

    Copyright (C) 2013-2015 by Ruben Vermeersch <ruben@rocketeer.be>

    Permission is hereby granted, free of charge, to any person obtaining a copy
    of this software and associated documentation files (the "Software"), to deal
    in the Software without restriction, including without limitation the rights
    to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
    copies of the Software, and to permit persons to whom the Software is
    furnished to do so, subject to the following conditions:

    The above copyright notice and this permission notice shall be included in
    all copies or substantial portions of the Software.

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
    FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
    AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
    LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
    OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
    THE SOFTWARE.


URL: LICENSE

--------------------------------------------------------------------------------
angular-hotkeys:1.6.0
MIT

    The MIT License (MIT)

    Copyright (c) 2014 Wes Cruver

    Permission is hereby granted, free of charge, to any person obtaining a copy of
    this software and associated documentation files (the "Software"), to deal in
    the Software without restriction, including without limitation the rights to
    use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of
    the Software, and to permit persons to whom the Software is furnished to do so,
    subject to the following conditions:
```

```
    The above copyright notice and this permission notice shall be included in all
    copies or substantial portions of the Software.

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS
    FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
    COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER
    IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
    CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.


URL: LICENSE

--------------------------------------------------------------------------------
angular-location-update:0.0.2
MIT

    The MIT License (MIT)

    Copyright (c) 2015 anglibs

    Permission is hereby granted, free of charge, to any person obtaining a copy
    of this software and associated documentation files (the "Software"), to deal
    in the Software without restriction, including without limitation the rights
    to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
    copies of the Software, and to permit persons to whom the Software is
    furnished to do so, subject to the following conditions:

    The above copyright notice and this permission notice shall be included in all
    copies or substantial portions of the Software.

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
    FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
    AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
    LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
    OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
    SOFTWARE.



URL: LICENSE

--------------------------------------------------------------------------------
angular-messages:1.5.5
MIT



URL:

--------------------------------------------------------------------------------
angular-mocks:1.5.0
MIT



URL:

--------------------------------------------------------------------------------
angular-percent-circle-directive:^1.1.1
MIT

    The MIT License (MIT)

    Copyright (c) 2015 Walter Staeblein
```

```
    Permission is hereby granted, free of charge, to any person obtaining a copy
    of this software and associated documentation files (the "Software"), to deal
    in the Software without restriction, including without limitation the rights
    to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
    copies of the Software, and to permit persons to whom the Software is
    furnished to do so, subject to the following conditions:

    The above copyright notice and this permission notice shall be included in all
    copies or substantial portions of the Software.

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
    FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
    AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
    LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
    OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
    SOFTWARE.



URL: LICENSE

--------------------------------------------------------------------------------
angular-promise-extras:0.1.3
ISC



URL:

--------------------------------------------------------------------------------
angular-resize:1.0.0
MIT

    Copyright (C) 2015 by Ruben Vermeersch <ruben@rocketeer.be>

    Permission is hereby granted, free of charge, to any person obtaining a copy
    of this software and associated documentation files (the "Software"), to deal
    in the Software without restriction, including without limitation the rights
    to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
    copies of the Software, and to permit persons to whom the Software is
    furnished to do so, subject to the following conditions:

    The above copyright notice and this permission notice shall be included in
    all copies or substantial portions of the Software.

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
    FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
    AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
    LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
    OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
    THE SOFTWARE.


URL: LICENSE

--------------------------------------------------------------------------------
angular-route:1.5.0
MIT



URL:

--------------------------------------------------------------------------------
angular-sanitize:1.5.5
MIT
```

```
URL:

--------------------------------------------------------------------------------
angular-screenfull:0.1.0
MIT

    The MIT License (MIT)

    Copyright (c) <year> <copyright holders>

    Permission is hereby granted, free of charge, to any person obtaining a copy
    of this software and associated documentation files (the "Software"), to deal
    in the Software without restriction, including without limitation the rights
    to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
    copies of the Software, and to permit persons to whom the Software is
    furnished to do so, subject to the following conditions:

    The above copyright notice and this permission notice shall be included in
    all copies or substantial portions of the Software.

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
    FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
    AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
    LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
    OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
    THE SOFTWARE.


URL: LICENSE

--------------------------------------------------------------------------------
angular-steps:0.2.3
MIT

    The MIT License (MIT)

    Copyright (c) 2015

    Permission is hereby granted, free of charge, to any person obtaining a copy
    of this software and associated documentation files (the "Software"), to deal
    in the Software without restriction, including without limitation the rights
    to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
    copies of the Software, and to permit persons to whom the Software is
    furnished to do so, subject to the following conditions:

    The above copyright notice and this permission notice shall be included in all
    copies or substantial portions of the Software.

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
    FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
    AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
    LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
    OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
    SOFTWARE.


URL: LICENSE

--------------------------------------------------------------------------------
angular-tree-repeat:0.0.1


    The MIT License
```

```
    Copyright (c) 2014 Paul Thomas <paul@stackfull.com>

    Permission is hereby granted, free of charge, to any person obtaining a copy
    of this software and associated documentation files (the "Software"), to deal
    in the Software without restriction, including without limitation the rights
    to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
    copies of the Software, and to permit persons to whom the Software is
    furnished to do so, subject to the following conditions:

    The above copyright notice and this permission notice shall be included in
    all copies or substantial portions of the Software.

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
    FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
    AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
    LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
    OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
    THE SOFTWARE.



URL: LICENCE

--------------------------------------------------------------------------------
angular-ui:1.3.2
MIT



URL:

--------------------------------------------------------------------------------
angular-vs-repeat:2.0.13
MIT

    The MIT License (MIT)

    Copyright (c) 2014 kamilkp

    Permission is hereby granted, free of charge, to any person obtaining a copy
    of this software and associated documentation files (the "Software"), to deal
    in the Software without restriction, including without limitation the rights
    to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
    copies of the Software, and to permit persons to whom the Software is
    furnished to do so, subject to the following conditions:

    The above copyright notice and this permission notice shall be included in all
    copies or substantial portions of the Software.

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
    FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
    AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
    LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
    OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
    SOFTWARE.

URL: LICENSE

--------------------------------------------------------------------------------
angularjs-filters:0.0.2
MIT



URL:
```

```
--------------------------------------------------------------------------------
angularjs-slider:2.10.2
MIT

    The MIT License (MIT)

    Copyright (c) 2013 Rafal Zajac <rzajac@gmail.com>

    Permission is hereby granted, free of charge, to any person obtaining a copy
    of this software and associated documentation files (the "Software"), to deal
    in the Software without restriction, including without limitation the rights
    to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
    copies of the Software, and to permit persons to whom the Software is
    furnished to do so, subject to the following conditions:

    The above copyright notice and this permission notice shall be included in
    all copies or substantial portions of the Software.

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
    FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
    AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
    LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
    OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
    THE SOFTWARE.


URL: LICENSE

--------------------------------------------------------------------------------
backstretch:2.1.2
MIT

    Copyright (c) 2012 Scott Robbin

    Permission is hereby granted, free of charge, to any person
    obtaining a copy of this software and associated documentation
    files (the "Software"), to deal in the Software without
    restriction, including without limitation the rights to use,
    copy, modify, merge, publish, distribute, sublicense, and/or sell
    copies of the Software, and to permit persons to whom the
    Software is furnished to do so, subject to the following
    conditions:

    The above copyright notice and this permission notice shall be
    included in all copies or substantial portions of the Software.

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
    EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
    OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
    NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
    HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
    WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
    FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
    OTHER DEALINGS IN THE SOFTWARE.


URL: LICENSE-MIT

--------------------------------------------------------------------------------
bootbox:4.4.0
MIT

    # License

    (The MIT License)
```

```
    Copyright (C) 2011-2014 by Nick Payne <nick@kurai.co.uk>

    Permission is hereby granted, free of charge, to any person obtaining a copy
    of this software and associated documentation files (the "Software"), to deal
    in the Software without restriction, including without limitation the rights
    to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
    copies of the Software, and to permit persons to whom the Software is
    furnished to do so, subject to the following conditions:

    The above copyright notice and this permission notice shall be included in
    all copies or substantial portions of the Software.

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
    FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
    AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
    LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
    OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
    THE SOFTWARE


URL: LICENSE.md

----------------------------------------------------------------------------------------
bootstrap:3.3.6
MIT



URL:

----------------------------------------------------------------------------------------
bootstrap-daterangepicker:2.1.24
MIT



URL:

----------------------------------------------------------------------------------------
bootstrap-tour:0.10.3
MIT

    Copyright (c) 2013-2015 The Bootstrap Tour community

    Permission is hereby granted, free of charge, to any person obtaining a copy
    of this software and associated documentation files (the "Software"), to deal
    in the Software without restriction, including without limitation the rights
    to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
    copies of the Software, and to permit persons to whom the Software is
    furnished to do so, subject to the following conditions:

    The above copyright notice and this permission notice shall be included in
    all copies or substantial portions of the Software.

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
    FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.  IN NO EVENT SHALL THE
    AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
    LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
    OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
    THE SOFTWARE.


URL: LICENSE

----------------------------------------------------------------------------------------
browser-update:3.2.2
```

```
MIT

    Copyright (c) 2007-2017 http://browser-update.org/

    Permission is hereby granted, free of charge, to any person obtaining
    a copy of this software and associated documentation files (the
    "Software"), to deal in the Software without restriction, including
    without limitation the rights to use, copy, modify, merge, publish,
    distribute, sublicense, and/or sell copies of the Software, and to
    permit persons to whom the Software is furnished to do so, subject to
    the following conditions:

    The above copyright notice and this permission notice shall be
    included in all copies or substantial portions of the Software.

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
    EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
    MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
    NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE
    LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
    OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
    WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

URL: LICENSE.txt

--------------------------------------------------------------------------------
c3:0.4.11
MIT

    The MIT License (MIT)

    Copyright (c) 2013 Masayuki Tanaka

    Permission is hereby granted, free of charge, to any person obtaining a copy of
    this software and associated documentation files (the "Software"), to deal in
    the Software without restriction, including without limitation the rights to
    use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of
    the Software, and to permit persons to whom the Software is furnished to do so,
    subject to the following conditions:

    The above copyright notice and this permission notice shall be included in all
    copies or substantial portions of the Software.

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS
    FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
    COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER
    IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
    CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.


URL: LICENSE

--------------------------------------------------------------------------------
caching-plugin-babel:^0.0.3
MIT

    MIT License

    Copyright (c) 2017 Kurt Westerfeld

    Permission is hereby granted, free of charge, to any person obtaining a copy
    of this software and associated documentation files (the "Software"), to deal
    in the Software without restriction, including without limitation the rights
    to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
    copies of the Software, and to permit persons to whom the Software is
    furnished to do so, subject to the following conditions:
```

```
    The above copyright notice and this permission notice shall be included in all
    copies or substantial portions of the Software.

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
    FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
    AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
    LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
    OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
    SOFTWARE.


URL: LICENSE

--------------------------------------------------------------------------------
codeflask:1.4.1
MIT

    The MIT License (MIT)

    Copyright (c) 2016 Claudio Holanda

    Permission is hereby granted, free of charge, to any person obtaining a copy
    of this software and associated documentation files (the "Software"), to deal
    in the Software without restriction, including without limitation the rights
    to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
    copies of the Software, and to permit persons to whom the Software is
    furnished to do so, subject to the following conditions:

    The above copyright notice and this permission notice shall be included in all
    copies or substantial portions of the Software.

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
    FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
    AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
    LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
    OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
    SOFTWARE.


URL: LICENSE

--------------------------------------------------------------------------------
colorbrewer:1.0.0
Apache License

    Apache Software License, Version 2.0

URL: http://www.apache.org/licenses/LICENSE-2.0.html

--------------------------------------------------------------------------------
configurable.js:0.1.0
MIT



URL:

--------------------------------------------------------------------------------
css:0.1.34


    The MIT License (MIT)

    Copyright (c) 2014-2015 Guy Bedford

    Permission is hereby granted, free of charge, to any person obtaining a copy of
```

```
    this software and associated documentation files (the "Software"), to deal in
    the Software without restriction, including without limitation the rights to
    use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of
    the Software, and to permit persons to whom the Software is furnished to do so,
    subject to the following conditions:

    The above copyright notice and this permission notice shall be included in all
    copies or substantial portions of the Software.

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS
    FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
    COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER
    IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
    CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.


URL: LICENSE

--------------------------------------------------------------------------------
cubism:1.6.0
Apache License

    Apache Software License, Version 2.0

URL: http://www.apache.org/licenses/LICENSE-2.0.html

--------------------------------------------------------------------------------
d3:3.5.17
BSD-3-Clause



URL:

--------------------------------------------------------------------------------
d3-flame-graph:0.4.2
Apache License

    Apache Software License, Version 2.0

URL: http://www.apache.org/licenses/LICENSE-2.0.html

--------------------------------------------------------------------------------
d3-scale:0.6.3
BSD-3-Clause

    Copyright 2010-2015 Mike Bostock
    All rights reserved.

    Redistribution and use in source and binary forms, with or without modification,
    are permitted provided that the following conditions are met:

    * Redistributions of source code must retain the above copyright notice, this
      list of conditions and the following disclaimer.

    * Redistributions in binary form must reproduce the above copyright notice,
      this list of conditions and the following disclaimer in the documentation
      and/or other materials provided with the distribution.

    * Neither the name of the author nor the names of contributors may be used to
      endorse or promote products derived from this software without specific prior
      written permission.

    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
    ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
    WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
    DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR
```

```
    ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
    (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
    LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
    ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
    (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
    SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.


URL: LICENSE

--------------------------------------------------------------------------------
d3-tip:0.6.7
MIT

    The MIT License (MIT)
    Copyright (c) 2013 Justin Palmer

    Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation

    The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Softw

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRA

URL: LICENSE

--------------------------------------------------------------------------------
dagre:0.7.4
MIT

    Copyright (c) 2012-2014 Chris Pettitt

    Permission is hereby granted, free of charge, to any person obtaining a copy
    of this software and associated documentation files (the "Software"), to deal
    in the Software without restriction, including without limitation the rights
    to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
    copies of the Software, and to permit persons to whom the Software is
    furnished to do so, subject to the following conditions:

    The above copyright notice and this permission notice shall be included in
    all copies or substantial portions of the Software.

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
    FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
    AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
    LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
    OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
    THE SOFTWARE.


URL: LICENSE

--------------------------------------------------------------------------------
dagre-d3:0.4.17
MIT

    Copyright (c) 2013 Chris Pettitt

    Permission is hereby granted, free of charge, to any person obtaining a copy
    of this software and associated documentation files (the "Software"), to deal
    in the Software without restriction, including without limitation the rights
    to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
    copies of the Software, and to permit persons to whom the Software is
    furnished to do so, subject to the following conditions:

    The above copyright notice and this permission notice shall be included in
    all copies or substantial portions of the Software.
```

```
    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
    FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
    AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
    LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
    OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
    THE SOFTWARE.


URL: LICENSE

--------------------------------------------------------------------------------
debounce-promise:1.1.0
MIT



URL:

--------------------------------------------------------------------------------
decamelize:1.1.1
MIT



URL:

--------------------------------------------------------------------------------
dexie:1.5.1
Apache License

    Apache Software License, Version 2.0

URL: http://www.apache.org/licenses/LICENSE-2.0.html

--------------------------------------------------------------------------------
escape-string-regexp:1.0.5
MIT



URL:

--------------------------------------------------------------------------------
event-drops:0.3.0-alpha1-fork2
MIT



URL:

--------------------------------------------------------------------------------
file-saver:1.3.3
MIT

    The MIT License

    Copyright © 2016 [Eli Grey][1].

    Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation

    The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Softw

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRA

      [1]: http://eligrey.com
```

```
URL: LICENSE.md

--------------------------------------------------------------------------------
font-awesome:4.5.0
(OFL-1.1 AND MIT)



URL:

--------------------------------------------------------------------------------
formattor:0.0.2
MIT



URL:

--------------------------------------------------------------------------------
humanize:0.0.9


    Copyright (c) 2012 Tai-Jin Lee http://www.taijinlee.com

    Permission is hereby granted, free of charge, to any person obtaining
    a copy of this software and associated documentation files (the
    "Software"), to deal in the Software without restriction, including
    without limitation the rights to use, copy, modify, merge, publish,
    distribute, sublicense, and/or sell copies of the Software, and to
    permit persons to whom the Software is furnished to do so, subject to
    the following conditions:

    The above copyright notice and this permission notice shall be
    included in all copies or substantial portions of the Software.

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
    EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
    MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
    NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE
    LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
    OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
    WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.


URL: LICENSE

--------------------------------------------------------------------------------
jquery:2.1.4
MIT

    Copyright 2014 jQuery Foundation and other contributors
    http://jquery.com/

    Permission is hereby granted, free of charge, to any person obtaining
    a copy of this software and associated documentation files (the
    "Software"), to deal in the Software without restriction, including
    without limitation the rights to use, copy, modify, merge, publish,
    distribute, sublicense, and/or sell copies of the Software, and to
    permit persons to whom the Software is furnished to do so, subject to
    the following conditions:

    The above copyright notice and this permission notice shall be
    included in all copies or substantial portions of the Software.

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
    EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
    MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
    NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE
```

```
    LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
    OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
    WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.


URL: MIT-LICENSE.txt

--------------------------------------------------------------------------------
jquery-resizable-dom:^0.26.0
MIT



URL:

--------------------------------------------------------------------------------
jsonpath-plus:0.13.0
MIT



URL:

--------------------------------------------------------------------------------
jspm-nodelibs-process:0.2.1


    (The MIT License)

    Copyright (c) 2013 Roman Shtylman <shtylman@gmail.com>

    Permission is hereby granted, free of charge, to any person obtaining
    a copy of this software and associated documentation files (the
    'Software'), to deal in the Software without restriction, including
    without limitation the rights to use, copy, modify, merge, publish,
    distribute, sublicense, and/or sell copies of the Software, and to
    permit persons to whom the Software is furnished to do so, subject to
    the following conditions:

    The above copyright notice and this permission notice shall be
    included in all copies or substantial portions of the Software.

    THE SOFTWARE IS PROVIDED 'AS IS', WITHOUT WARRANTY OF ANY KIND,
    EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
    MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
    IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
    CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
    TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
    SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.


URL: LICENSE

--------------------------------------------------------------------------------
netmask:^1.0.6
MIT



URL:

--------------------------------------------------------------------------------
ng-file-model:0.0.5
MIT

    The MIT License (MIT)

    Copyright (c) 2015 Mistralworks
```

```
    Permission is hereby granted, free of charge, to any person obtaining a copy
    of this software and associated documentation files (the "Software"), to deal
    in the Software without restriction, including without limitation the rights
    to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
    copies of the Software, and to permit persons to whom the Software is
    furnished to do so, subject to the following conditions:

    The above copyright notice and this permission notice shall be included in all
    copies or substantial portions of the Software.

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
    FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
    AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
    LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
    OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
    SOFTWARE.



URL: LICENSE

--------------------------------------------------------------------------------
lodash:3.10.0
MIT

    Copyright 2012-2015 The Dojo Foundation <http://dojofoundation.org/>
    Based on Underscore.js, copyright 2009-2015 Jeremy Ashkenas,
    DocumentCloud and Investigative Reporters & Editors <http://underscorejs.org/>

    Permission is hereby granted, free of charge, to any person obtaining
    a copy of this software and associated documentation files (the
    "Software"), to deal in the Software without restriction, including
    without limitation the rights to use, copy, modify, merge, publish,
    distribute, sublicense, and/or sell copies of the Software, and to
    permit persons to whom the Software is furnished to do so, subject to
    the following conditions:

    The above copyright notice and this permission notice shall be
    included in all copies or substantial portions of the Software.

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
    EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
    MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
    NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE
    LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
    OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
    WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.


URL: LICENSE

--------------------------------------------------------------------------------
moment:2.11.1
MIT



URL:

--------------------------------------------------------------------------------
moment-duration-format:1.3.0
MIT

    The MIT License (MIT)

    Copyright (c) 2013 John Madhavan-Reese
```

```
    Permission is hereby granted, free of charge, to any person obtaining a copy of
    this software and associated documentation files (the "Software"), to deal in
    the Software without restriction, including without limitation the rights to
    use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of
    the Software, and to permit persons to whom the Software is furnished to do so,
    subject to the following conditions:

    The above copyright notice and this permission notice shall be included in all
    copies or substantial portions of the Software.

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS
    FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
    COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER
    IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
    CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.


URL: LICENSE

--------------------------------------------------------------------------------
ng-table:1.0.0
BSD

    Copyright (c) 2013, esvit.
    All rights reserved.

    Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following

    Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaime
    Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following discla
    Neither the name of the esvit nor the names of its contributors may be used to endorse or promote products derived from th
    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDI

URL: LICENSE

--------------------------------------------------------------------------------
ngBootbox:0.1.2
MIT



URL:

--------------------------------------------------------------------------------
ngdraggable:0.1.10
MIT

    The MIT License (MIT)

    Copyright (c) 2014 philippd

    Permission is hereby granted, free of charge, to any person obtaining a copy
    of this software and associated documentation files (the "Software"), to deal
    in the Software without restriction, including without limitation the rights
    to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
    copies of the Software, and to permit persons to whom the Software is
    furnished to do so, subject to the following conditions:

    The above copyright notice and this permission notice shall be included in all
    copies or substantial portions of the Software.

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
    FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
    AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
    LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
```

```
    OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
    SOFTWARE.


URL: LICENSE.txt

--------------------------------------------------------------------------------
pako:1.0.0
MIT



URL:

--------------------------------------------------------------------------------
prismjs:1.17.1
MIT



URL:

--------------------------------------------------------------------------------
screenfull:3.3.3
MIT

    MIT License

    Copyright (c) Sindre Sorhus <sindresorhus@gmail.com> (sindresorhus.com)

    Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation

    The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Softw

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRA


URL: license

--------------------------------------------------------------------------------
sprintf:0.1.5
BSD-3-Clause



URL:

--------------------------------------------------------------------------------
stacktrace-js:^2.0.0
MIT

    Copyright (c) 2017 Eric Wendelin and other contributors

    Permission is hereby granted, free of charge, to any person obtaining a copy of
    this software and associated documentation files (the "Software"), to deal in
    the Software without restriction, including without limitation the rights to
    use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies
    of the Software, and to permit persons to whom the Software is furnished to do
    so, subject to the following conditions:

    The above copyright notice and this permission notice shall be included in all
    copies or substantial portions of the Software.

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
    FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
    AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
    LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
```

```
    OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
    SOFTWARE.


URL: LICENSE

--------------------------------------------------------------------------------
systemjs-hmr:^2.0.9
MIT

    Copyright (c) 2017 Alexis Vincent

    Permission is hereby granted, free of charge, to any person obtaining a copy
    of this software and associated documentation files (the "Software"), to deal
    in the Software without restriction, including without limitation the rights
    to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
    copies of the Software, and to permit persons to whom the Software is
    furnished to do so, subject to the following conditions:

    The above copyright notice and this permission notice shall be included in all
    copies or substantial portions of the Software.

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
    FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
    AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
    LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
    OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
    SOFTWARE.

URL: LICENCE

--------------------------------------------------------------------------------
systemjs-hot-reloader:1.0.0
MIT

    The MIT License (MIT)

    Copyright (c) 2015 Jiri Spac

    Permission is hereby granted, free of charge, to any person obtaining a copy
    of this software and associated documentation files (the "Software"), to deal
    in the Software without restriction, including without limitation the rights
    to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
    copies of the Software, and to permit persons to whom the Software is
    furnished to do so, subject to the following conditions:

    The above copyright notice and this permission notice shall be included in all
    copies or substantial portions of the Software.

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
    FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
    AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
    LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
    OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
    SOFTWARE.



URL: LICENSE

--------------------------------------------------------------------------------
text:0.0.9
MIT

    The MIT License (MIT)
```

```
    Copyright (c) 2013 jspm

    Permission is hereby granted, free of charge, to any person obtaining a copy of
    this software and associated documentation files (the "Software"), to deal in
    the Software without restriction, including without limitation the rights to
    use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of
    the Software, and to permit persons to whom the Software is furnished to do so,
    subject to the following conditions:

    The above copyright notice and this permission notice shall be included in all
    copies or substantial portions of the Software.

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS
    FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
    COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER
    IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
    CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.


URL: LICENSE

-----------------------------------------------------------------------------------------
toastr:2.1.2
MIT



URL:

-----------------------------------------------------------------------------------------
typicons:2.0.7
Artwork: CCA: https://creativecommons.org/licenses/by-sa/3.0/legalcode,FONT: SIL: https://scripts.sil.org/OFL



URL:

-----------------------------------------------------------------------------------------
urijs:1.17.0
MIT



URL:

-----------------------------------------------------------------------------------------
uuid:2.0.1


    Copyright (c) 2010-2012 Robert Kieffer
    MIT License - http://opensource.org/licenses/mit-license.php


URL: LICENSE.md

-----------------------------------------------------------------------------------------
visibilityjs:2.0.2
MIT

    The MIT License (MIT)

    Copyright 2011 Andrey Sitnik <andrey@sitnik.ru>

    Permission is hereby granted, free of charge, to any person obtaining a copy of
    this software and associated documentation files (the "Software"), to deal in
    the Software without restriction, including without limitation the rights to
    use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of
```

```
    the Software, and to permit persons to whom the Software is furnished to do so,
    subject to the following conditions:

    The above copyright notice and this permission notice shall be included in all
    copies or substantial portions of the Software.

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS
    FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
    COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER
    IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
    CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.


URL: LICENSE

-------------------------------------------------------------------------------
yamm3:1.1.0


    The MIT License (MIT)

    Copyright (c) 2015 Geedmo

    Permission is hereby granted, free of charge, to any person obtaining a copy
    of this software and associated documentation files (the "Software"), to deal
    in the Software without restriction, including without limitation the rights
    to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
    copies of the Software, and to permit persons to whom the Software is
    furnished to do so, subject to the following conditions:

    The above copyright notice and this permission notice shall be included in
    all copies or substantial portions of the Software.

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
    FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
    AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
    LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
    OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
    THE SOFTWARE.

URL: LICENSE.txt
```

Integrations

Yellowbrick Documentation > Integrations

Platforms: All platforms

Parent topic: Yellowbrick Documentation

Yellowbrick integrates with the ecosystem by virtue of its PostgreSQL compatibility and native connectors. Using the PostgreSQL ODBC and JDBC support that third-party tools provide is the quickest way to connect to Yellowbrick. If you encounter issues when connecting using the PostgreSQL drivers try overriding the version.

Here are some guides and tutorials on how to connect tools using the PostgreSQL drivers and Yellowbrick-specific connectors:

- Airbyte
- Airflow
- DBeaver
- Informatica
- Jupyter Notebook
- Kafka
- LangChain
- MicroStrategy
- PostgreSQL ODBC/JDBC drivers
- Spark / Databricks
- Tableau

7.3.0 Yellowbrick Release Notes

Release Information	
Cloud Platform	7.3.0-74895.181ea900
Appliance Platform	7.3.0-74895
Release Date	7/25/2025, 5:05:58 PM

This release of Yellowbrick adds new features for observability, infrastructure management and developer capabilities.

New features

- **Telemetry and Monitoring**:
  - Yellowbrick deployments on AWS, Azure, and GCP now include a new observability framework powered by Prometheus for improved monitoring, alerting and troubleshooting.
  - Standardized metrics and alert rules are exported from core components via Prometheus endpoints.
  - Metrics and alerts are grouped by source for easier dashboarding, alert routing, and triage.
- **Infrastructure and Platform Upgrades**:
  - Yellowbrick Deployer support for fully managed EKS, AKS and GKE cluster upgrades to control plane version 1.31.
  - A new dry-run and upgrade plan capability allows users to preview the full scope of changes before initiating an upgrade. Upgrades may include:
    - Kubernetes control plane and node version updates.
    - IAM roles, policies, and permissions.
    - Components such as Yellowbrick Operator, Manager, and Monitoring.
    - Yellowbrick Data Warehouse software version.
- **Enhanced ybtools support**
  - Support for Java 17 (in addition to Java 11), allowing compatibility with modern environments and updated security requirements.
- Developer SQL capabilities:
  - **SQL Unload**: Support for unloading data directly to external object storage using the UNLOAD TABLE SQL command.
  - **Sample SQL**: Support for the TABLESAMPLE() query function to return randomized or deterministic, representative subsets of rows from a table with options for percentage or count-based sampling.
- **AWS Marketplace**: Yellowbrick Enterprise Edition is now available on the AWS Marketplace, supporting fully integrated deployment and subscription management.

Upgrade Requirements

For appliance upgrades, contact Customer Support to review the process and schedule the upgrade.

Detailed changelog

See the detailed changelog for itemized lists of behavioral and compatibility changes, bug fixes, known issues and CVE remediations.

Changes in Behavior in This Version

7.3.0 Queries issued by SMC and DCS services are now visible by default in sys.query

Superusers or roles with `TRACE QUERY` can now observe these queries in `sys.query` or `SMC` active queries page. These queries can be excluded by enabling the “Exclude System Work” feature.

7.3.0 External formats now support two modes

`CREATE EXTERNAL FORMAT` now supports two modes: `load` and `unload`. If no mode is specified, `load` is assumed by default. For UNLOAD TABLE, the external format must be defined with the `unload` option; for `LOAD TABLE`, `load` mode is required.

7.2.0 New method of unloading data to Object Stores

A new unloading mechanism has been implemented for exporting data to object stores. This new method breaks the earlier access of s3 uri and requires `--object-store-<option>` for `ybunload`. See Object Storage Options. The maximum file size for any single output file is limited to 50GB in this release. In some cases multiple output path locations were supported by `ybunload`. This is no longer supported.

7.2.0 New alert to track blade disk usage

A new system alert (`Compute Blade Filesystem Usage`) has been added. It tracks disk space usage per individual physical drive (SSD). This differs from `Compute Blade Disk Used` as the latter tracks overall disk usage and doesn't account for data skewness. This new alert is enabled by default and will trigger once any drive usage is above `95%`. More information can be found in the alert section.

7.2.0 Appliance alert for node disk wear has been updated

Manager Node Disk Wear alert has been updated from a user-defined to a System Alert enabled by default, ensuring consistency with the Compute Blade Disk Wear system alert. Additionally, the critical wear threshold has been standardized at 90% to enhance proactive monitoring.

7.1.0 Empty string is no longer a valid JSONB

Version 7.1.0 does not permit casting an empty string into a JSONB, preventing the creation of an "empty JSONB". Many operations empty JSONB values are no longer supported. See https://docs.yellowbrick.com/latest/ybd_sqlref/jsonb_handle_empty.html

7.1.0 Removed support for index lists and slicing

Support for slicing (eg. $[1:3]) and index lists (eg. $[1,2]) in JSONPath expressions has been removed.

7.1.0 Removed support for JSONPaths not starting with $

In 7.0.1, it was possible to omit the $ character when using bracket notation or omit $. when using dot notation. This is not supported in 7.1.0.

7.1.0 Changes to FLATTEN behavior

The values in the index, key, value JSONB returned by FLATTEN are set differently in 7.1.0. "index" now contains null when flattening primitives or objects, whereas before it was 0 for primitives and contained integers for objects. "key" now contains null for arrays, whereas before "key" contained the same value as "index".

7.1.0 Changes to JSONPath name selector grammar

In 7.1.0, the syntax $[name], $["name"] or $."name" is no longer valid for JSONPath name selectors. $.name, $.'name' and $['name'] are supported.

7.0.1 Stricter checks for casts to numeric types

Version 7.0.1 introduces Postgres-compatible checks for coercing text into numeric types. Specifically, it will no longer silently return zero for empty strings, nor will it parse strings starting with "x" as hexadecimal numbers (the full "0x" prefix is now required). Instead, an error will be raised in these cases.

Changes in Compatibility in This Version

7.0.1 Client tools tzdata compatibility

When executing ybload and load table SQL, the outcome of loading a timestamp with timezone data to a timestamp column may vary based on the tzdata version. Differences between the JRE's tzdata version and the Yellowbrick database's tzdata version can lead to inconsistencies between timestamp directly inserted into the database and those loaded using ybload or load table SQL. For example, in the case of JRE with tzdata version 2022b, Chile's Daylight Saving Time (DST) transition is delayed by a week in September 2022. As a result, a timestamp such as 04-09-2022 12:00:00 Chile/Continental might be directly inserted into the database as 2022-04-09 15:00:00+00 while being loaded as 2022-04-09 16:00:00+00 due to differences in timezone conversions. It is recommended to check tzdata version used JRE in the environment, using the tzupdater tool provided by Oracle.

7.0.1 ybtools Compatibility

Yellowbrick recommends that you always upgrade ybtools to match the Yellowbrick server version you are running (for example, upgrade to ybtools Version 7.0.1 before connecting to a Version 7.0.1 data warehouse). In most cases, using an older version of ybtools, such as 7.0.0, to connect to a Version 7.0.1 Data Warehouse will work but is not recommended. Using Version 7.0.1 of ybtools with an older server version (such as Version 7.0.0) may result in error messages for some commands or a mismatch between client and server features.

Important: On CentOS and Red Hat client platforms, you must first remove any existing 5.x version of ybtools. Then you can proceed with the installation of the Version 6.0 ybtools. You cannot upgrade directly from an earlier 5.x version of ybtools to Version 6.0.

7.0.1 BAR2 and Replication Compatibility

Important: The source and target systems used for database backup and restore (BAR2) and database replication must be compatible. To support full functionality, the source and target systems should both be running a 7.0.1 version of Yellowbrick software. Nonetheless, you can back up and restore a database, or replicate a database, from:

- 5.2.x to 6.5.x
- 5.4.3 or later to 6.5.x
- 6.1.x to 6.5.x
- 6.2.x to 6.5.x
- 6.3.x to 6.5.x
- 6.4.x to 6.5.x
- 6.5.x to 6.6.x Although you can start replication from a 5.2, 5.4, 6.1, 6.2... or 7.0.0 source to a 7.0.1 target, and you can fail over to the 7.0.1 target, *you cannot fail back to the 5.2, 5.4, 6.1, 6.2, 6.3, 6.4 or 7.0.0 source*. In general, you cannot fail back to a Yellowbrick version with a catalog version that is earlier than the current catalog version.

What's New in This Version

7.3.0 Added support for geospatial data type in ybload and ybunload

This enhancement allows customers to bulk load and unload spatial datasets for all supported object stores and file formats.

- Input files loaded using `ybload` must use Well-Known Text (WKT) representation for GEOGRAPHY columns.
- Output files unloaded using `ybunload` will also have WKT strings for GEOGRAPHY columns.
- Support for GEOGRAPHY columns in `ybload` and `ybunload` is not backward compatible and may produce unexpected results when used with server versions earlier than 7.3.0.

7.3.0 New Prometheus-based observability framework for Cloud platform

Yellowbrick on Cloud now includes a new observability framework powered by Prometheus for improved monitoring, alerting, and troubleshooting:

- Standardized metrics and alert rules are exported from core components via Prometheus endpoints.
- Built-in alerts use Prometheus rule files.
- Metrics and alerts are grouped by source for easier dashboarding, alert routing, and triage.
- Learn more on the Observability Overview.

7.3.0 New SQL command to bulk unload to cloud storage

A new SQL-based `UNLOAD TABLE` command is now available (beta), enabling users to export table data directly to external storage without relying on the ybunload client. This feature supports both table names and `SELECT` queries as input, and integrates with `EXTERNAL LOCATION` and `EXTERNAL FORMAT` definitions. The feature is in beta mode and can be enabled by the configuration parameter enable_sql_unload.

Only S3-compatible external storage locations are currently supported.

7.3.0 Added columns memory_usage_bytes & peak_memory_usage_bytes to pg_stat_activity view

View `pg_stat_activity` now has two more columns (`memory_usage_bytes` & `peak_memory_usage_bytes`) to help with monitoring of active queries virtual memory consumption:

- `memory_usage_bytes` : Current virtual memory being allocated for the query at this point of time
- `peak_memory_usage_bytes` : Maximum virtual memory ever allocated for the query, as recorded at each `state_change` times.

7.2.0 Improved support for ybunload to object stores

`ybunload` now supports the same `--object-store-<option>` as `ybload` for all formats (`csv` , `text` , `parquet`).

7.1.0 Major Platform and Developer Experience Updates

This release delivers several major enhancements focused on accessibility, scalability, and deployment flexibility:

- Community Edition (CE): A freely available version of Yellowbrick for development and smaller datasets, deployable on x86 platforms via Docker and the AWS Marketplace.
- Compute Cluster Load Balancer: Dynamic workload distribution across multiple compute clusters, providing zero-downtime scaling and improved concurrency.
- Bring Your Own AMI: Support for customer-managed AMIs in AWS deployments, enabling use of hardened machine images.
- Enhanced Kubernetes Support: Quality of life improvements for deployments in multi-tenant Kubernetes environments.

7.0.1 New functionality

In addition to wider platform support, 7.0.1 release has a great deal of new functionality.

- Self-service cloud installation and upgrade, through a new installation tool, the Yellowbrick Deployer.
- Support for software licensing with free trials in cloud deployments.
- A new Kubernetes Operator for Kubernetes administration.
- Support for Google Cloud Platform, GCP.
- Support for fully offline, dark installations.
- Large reduction in usage of third party software and associated CVEs.
- JSON semi-structured data support, including optimized binary storage and flattening.
- Improvements to needle-in-haystack table scans, especially those with large in lists.
- Configurable password policies for local database users.
- Support for out-of-database user, role and grant auditing. GovCloud deployments will be supported in the upcoming 7.1.x release. Kubernetes Version 1.30 is used across all cloud platforms.

7.0.0 Extended SE pushdown (SEFilters) feature

Improved filters can be pushed down to the Storage Engine so that less data will be read and decompressed

- Controlled by the configuration parameter `enable_se_filter_expression_generation`
- Visible in `EXPLAIN` with the `se_filters` modifier
- Adds skipping support for IN lists up to about 100,000 items
- Adds skipping support for more complex expressions
- Adds skipping support for one OR term

Known Issues in This Version

You may encounter the following issues in Version 7.3.0. Use the workarounds provided in the description and contact Customer Support for additional information.

Issue #	Description
38832	The `sys.user` view's `create_role` and `create_database` fields may not accurately reflect a user's ability to perform these operations when the privileges are granted through role membership.
39953	There are two known (and related) issues with respect to read-only transactions and temporary tables. Temporary tables can be created during read-only transactions. Creating a temporary table modifies the catalog, which should not occur during a read-only transaction. This may produce an error about the database being unable to insert into `sys.yrs_flush`. Inserting/updating/deleting rows in a temporary table during a read-only transaction may also produce errors about the database being unable to to insert into `sys.yrs_flush`. This error should not be produced and users should be able to manipulate the rows in a temporary table during a read-only transaction. In newer versions of the product this behavior will be addressed. It will not be possible to create or drop temporary tables inside a read-only transaction, and `INSERT` / `UPDATE` / `DELETE` commands issued against a temporary table during a read-only transaction will not produce an error.

Issues Fixed in This Version

The following issues are addressed in this version:

Release	Issue #	Description
7.3.0	45124	Updates cloud region and availability zone definitions for the installer UI.
7.3.0	45037	Adds resource requirements to diagnostics tasks.

Release	Issue #	Description
7.3.0	44940	Fixes YB Manager startup with special characters in banner messages.
7.3.0	44922	Improved cancellation of incremental backup jobs to prevent disparity between backup bundle data and the backup chain stored on the YB server.
7.3.0	44607	Fixed an issue where a query executed by the Parity reconstruction service could exceed WLM memory limits, leading to system hang and reconfiguration.
7.3.0	44472	Reduced memory usage for metering queries to prevent spilling and out-of-memory errors such as KE002 and KE032.
7.3.0	43722	Updates Monaco Editor web worker loading to use ES modules and strengthens cross-origin policies.
7.3.0	43611	Fixed `ybload` duplicate-handler documentation. The table was not correctly rendered.
7.3.0	42982	Fixed an issue where casting a `JSONPath` expression containing a null byte triggered a compute cluster reconfiguration. Such casts now fail by default with the error: `null byte casting is disabled`. Alternatively, a configuration parameter allows replacing null bytes with the Unicode replacement character &#xFFFD; (U+FFFD).
7.3.0	42982	Fixed a backward compatibility issue in ybdumpschema when connecting to earlier YB server releases.
7.3.0	42701	Validate that workload identity is enabled when installing into an existing private AKS cluster.
7.3.0	42699	Fixed casting to JSONB; operations no longer fail with `jsonb reached maximum nested depth` under typical nesting levels (refer JSONB limits for more details).
7.3.0	42678	Fixed an issue where view creation could fail with the error: `Unable to locate var references of rnode in inner or outer`.
7.3.0	42266	Enhances installer resilience with network error handling and API timeouts.
7.3.0	41918	Fixed an issue where renaming a WLM profile could result in loss of its profile definitions.
7.3.0	35693, 43579	Resolved an issue where restarting a DB instance could cause queries on temporary tables to fail with: `assertThrow failure: TableLocationMap.h:169 nullptr == Attempt to add a TableLocRecord which is different from existing`.
7.3.0	35656	Fixed an issue where querying a query statistics virtual table could trigger a system reconfiguration.
7.3.0	35501	`LOAD TABLE` statements are now supported over JDBC connections.
7.3.0	40737	Fix an issue with premature rollbacks when bulk loading using Informatica. You will need to download the new version (Informatica 1.2.86), to take advantage of this fix.
7.2.3	41918	Addressed a WLM profile rename bug that, in many cases, caused associated rules and pools to be deleted.
7.2.2	44157	Fix an issue with error reporting during Helm apply.
7.2.2	44113	Fix an issue with the upgrade control script caused by a missing companion file.
7.2.2	44083	Fix an issue with ybsql in CentOS 10 and RHEL 9 casused by an incompatible libssl version.
7.2.2	42160	Clear a large WAL directory before upgrade to prevent an OOM condition on the Prometheus pod.

Release	Issue #	Description
7.2.2	43341, 43478	Prevent any INSERT/UPDATE that violates the maximum row size limit. This could lead to errors running GC such as `failed with error code 54000 (program limit exceeded)`.
7.2.2	43738	Set a limit on the recursion depth in JSON.
7.2.2	43849	Add retry logic to placement group deletion to handle eventual consistency.
7.2.2	43723	Preserve phone home and proxy settings after upgrade to 7.x.
7.2.1	43027	Improve resiliency of FSCK operations during compute cluster resume. This could lead to delays resuming compute clusters due to SIGSEGV.
7.3.0, 7.2.1, 7.1.3	43230	Fixed a performance issue with `NULL` outputs from `safe_to_date()`, `safe_to_timestamp()` and `safe_to_double()`.
7.2.1	42873	Session Timeout for Yellowbrick Manager is ignored. Idle detection determined more aggressively.
7.2.1	42884	Disable CORS headers for Yellowbrick Manager APIs.
7.2.1	42886	Disable Yellowbrick Manager graphql development console.
7.2.1	42888	Add option to Yellowbrick Manager to reduce detail supplied in error messages, to assist in more locked-down security controls.
7.2.1	42741	Installer flow for network access configuration is clearer and more user-friendly.
7.2.0	43235	Allow Yellowbrick Manager to work with google oauth2 provider.
7.2.0	30343	Fixed ybload issue when using Kerberos authentication.
7.2.0	43185	Fixed a restore database problem caused by a single backup chain containing backups from 5.4 and 7.1. The upgrade process now respects the legacy `sys.cluster_id` and uses it to set `sys.instance_id` for on-prem deployments.
7.2.0	40070	Fixed an alerting issue that caused system hardware alerts to not be sent to SMC nor configured alert endpoints. Alerts will now be generated for chassis power, fans, and SBBs in error conditions.
7.2.0	29699	OS and process interactions would interrupt LDAP, causing LDAP login to fail with the following error: `Can't contact LDAP server.`
7.2.0	41128	Fixed an issue that could cause certain `GEOGRAPHY` functions to crash when processing invalid geographies.
7.2.0	41130	Fixed an issue that prevented `GEOGRAPHY` to `GEOMETRY` projection from supporting NaN values.
7.1.1, 7.2.0	15598	Fixed an issue where encrypted columns from underlying tables would not get automatically decrypted when referenced in a view.
7.1.0	40480	Fixed an issue where orphaned temporary objects, left behind after a PostgreSQL crash, could prevent the autovacuum process from functioning correctly. In extreme cases, this could cause databases to enter READONLY mode.
7.1.0	40210	Fixed an issue with `enable_implied_pushdown` behavior: When `enable_implied_pushdown` was set to `on` (default is `off` in 7.0 releases), an `IS NULL` clause could incorrectly push below a `LEFT JOIN`, causing premature row removal. The pushdown logic has been updated to prevent this issue.

Release	Issue #	Description
7.1.0	40063	During the Azure platform installation, the Automatic Node OS Upgrade was disabled to prevent unexpected operations on the system that could lead to the database going down.
7.0.0, 7.0.1	37405	Improved upgrade performance by optimizing the manipulation of the `astCache`. This applies only to appliances. From this release, a much faster operation during the backup phase of the installer is performed. This can speed up upgrades considerably, especially with systems having large astCaches (greater than 100GB).
7.0.0, 7.0.1	38129	Improved algorithm for processing duplicated entries in large `IN` operations. The algorithm to parse and plan these types of queries was improved and we should expect reductions in plan time by up to 90%. The plan time now grows linearly with the number of items in large lists.
7.0.0, 7.0.1	38278	Increased visibility of Garbage Collection operations in `sys.log_query`.
7.0.1	38140	Fixed a race in the row unique manager cache that caused an assert and forced the database to restart.

7.0.1	37989	Executing a parameterized query via `npgsql` versions `7.x` and `8.x` using `IN LIST` operations with more than 10 elements may fail with:
		`Severity: ERROR`
		`SqlState: 42704`
		`MessageText: no value found for parameter 1`

7.0.1	37056	Fixed a rare race condition that could lead to a system restart for operations such as bulk `unload` and `yrestore`.
7.0.0	36374	Fixed the WriteKRI RPC call to support KRI lists greater than 2 MB. Prior to this fix, sending such a large list would cause the worker to terminate, killing all running queries. This issue typically arose from deleting or updating a significant number of shards in a single statement.
7.0.0	36259, 36799	Fixed a DB frontend crash related to the usage of an incorrect memory context after aborting a query execution.
7.0.0	36863	Fixed an issue with CREATE REMOTE SERVER that ignore NOHOSTNAMECHECK option.
7.0.0	34140	Improved support for decorrelating operators mixing correlated and uncorrelated expressions on both sides. For instance: `select 0 = (select down.b from down where down.b = up.b + down.a) from up;`
7.0.0	35859	Stop reporting `invalid command \n` when executing queries directly from a manager node with `ybsql`.
7.0.0	36262	Fixed an issue with native memory leak held by ssl connection callbacks. This affects databases that establish many ssl connections for replication, backup or restore and load or unload.
7.0.0	36511	Fixed an issue when running a single table restore for a table DDL with the cluster on.
7.0.0	36584,36585	Adjust a fail-safe mechanism to prevent it from causing overly aggressive restarts of the compute blades in the presence of `SCHED_YIELD_PROFILE` warnings.
7.0.0	36691	Upgraded ybtoken's logging from log4j-1.2.17 to log4j2, which is used by ybtoken, to address several high priority CVEs.
7.0.0	36820	Fixed a rare occurence of an assertion during bulk deletion of files in the Big Block File System (BBFS).
7.0.0	36863	Fixed an issue with CREATE REMOTE SERVER that ignore NOHOSTNAMECHECK option.
7.0.0	36880	Improved the performance of a DB upgrade by skipping the regeneration of statistics in the fast upgrade path.

Release	Issue #	Description
7.0.0	37025	In rare conditions, rows present in the rowstore are discarded during a `JOIN`` operation where the build side is replicated and there are no rows processed by another worker when constructing its build node. This issue has been resolved.
7.0.0	37285	Fixed an issue when performing large `delete` or `update` statements that touch more than 1M shards (writeKRI). This could lead to an `Out of Memory` error and, in rare cases, could lead to an outage and system restart.The usage of a segmented vector in the memory allocation algorithm is now used to handle these large operations.

CVE Fixes in This Version

The following CVEs are addressed in this version:

Release	CVE Fix	Description
7.3.0	CVE-2024-0450	An issue was found in the CPython `zipfile` module affecting versions 3.12.1, 3.11.7, 3.10.13, 3.9.18, and 3.8.18 and prior.
7.3.0	CVE-2025-24928	libxml2 before 2.12.10 and 2.13.x before 2.13.6 has a stack-based buffer overflow in xmlSnprintfElements in valid.c. To exploit this, DTD validation must occur for an untrusted document or untrusted DTD. NOTE: this is similar to CVE-2017-9047.
7.3.0	CVE-2024-2004	When a protocol selection parameter option disables all protocols without adding any then the default set of protocols would remain in the allowed set due to an error in the logic for removing protocols. The below command would perform a request to curl.se with a plaintext protocol which has been explicitly disabled. curl --proto -all,-http http://curl.se The flaw is only present if the set of selected protocols disables the entire set of available protocols, in itself a command with no practical use and therefore unlikely to be encountered in real situations. The curl security team has thus assessed this to be low severity bug.
7.3.0	CVE-2024-55549	xsltGetInheritedNsList in libxslt before 1.1.43 has a use-after-free issue related to exclusion of result prefixes.
7.3.0	CVE-2023-26604	systemd before 247 does not adequately block local privilege escalation for some Sudo configurations, e.g., plausible sudoers files in which the "systemctl status" command may be executed. Specifically, systemd does not set LESSSECURE to 1, and thus other programs may be launched from the less program. This presents a substantial security risk when running systemctl from Sudo, because less executes as root when the terminal size is too small to show the complete systemctl output.
7.3.0	CVE-2025-24855	numbers.c in libxslt before 1.1.43 has a use-after-free because, in nested XPath evaluations, an XPath context node can be modified but never restored. This is related to xsltNumberFormatGetValue, xsltEvalXPathPredicate, xsltEvalXPathStringNs, and xsltComputeSortResultInternal.
7.3.0	CVE-2024-56171	libxml2 before 2.12.10 and 2.13.x before 2.13.6 has a use-after-free in xmlSchemaIDCFillNodeTables and xmlSchemaBubbleIDCNodeTables in xmlschemas.c. To exploit this, a crafted XML document must be validated against an XML schema with certain identity constraints, or a crafted XML schema must be used.
7.3.0	CVE-2024-2379	libcurl skips the certificate verification for a QUIC connection under certain conditions, when built to use wolfSSL. If told to use an unknown/bad cipher or curve, the error path accidentally skips the verification and returns OK, thus ignoring any certificate problems.
7.3.0	CVE-2024-7264	libcurl's ASN1 parser code has the `GTime2str()` function, used for parsing an ASN.1 Generalized Time field. If given an syntactically incorrect field, the parser might end up using -1 for the length of the *time fraction*, leading to a `strlen()` getting performed on a pointer to a heap buffer area that is not (purposely) null terminated.
7.3.0	CVE-2021-33574	The mq_notify function in the GNU C Library (aka glibc) versions 2.32 and 2.33 has a use-after-free. It may use the notification thread attributes object (passed through its struct sigevent parameter) after it has been freed by the caller, leading to a denial of service (application crash) or possibly unspecified other impact.
7.3.0	CVE-2021-45261	An Invalid Pointer vulnerability exists in GNU patch 2.7 via the another_hunk function, which causes a Denial of Service.

Release	CVE Fix	Description
7.3.0	CVE-2024-6197	libcurl's ASN1 parser has this utf8asn1str() function used for parsing an ASN.1 UTF-8 string. Itcan detect an invalid field and return error. Unfortunately, when doing so it also invokes `free()` on a 4 byte localstack buffer. Most modern malloc implementations detect this error and immediately abort. Some however accept the input pointer and add that memory to its list of available chunks. This leads to the overwriting of nearby stack memory. The content of the overwrite is decided by the `free()` implementation; likely to be memory pointers and a set of flags. The most likely outcome of exploting this flaw is a crash, although it cannot be ruled out that more serious results can be had in special circumstances.
7.3.0	CVE-2024-0853	curl inadvertently kept the SSL session ID for connections in its cache even when the verify status (*OCSP stapling*) test failed. A subsequent transfer to the same hostname could then succeed if the session ID cache was still fresh, which then skipped the verify status check.
7.3.0	CVE-2025-4373	A flaw was found in GLib, which is vulnerable to an integer overflow in the g_string_insert_unichar() function. When the position at which to insert the character is large, the position will overflow, leading to a buffer underwrite.
7.3.0	CVE-2023-34969	D-Bus before 1.15.6 sometimes allows unprivileged users to crash dbus-daemon. If a privileged user with control over the dbus-daemon is using the org.freedesktop.DBus.Monitoring interface to monitor message bus traffic, then an unprivileged user with the ability to connect to the same dbus-daemon can cause a dbus-daemon crash under some circumstances via an unreplyable message. When done on the well-known system bus, this is a denial-of-service vulnerability. The fixed versions are 1.12.28, 1.14.8, and 1.15.6.
7.3.0	CVE-2024-6874	libcurl's URL API function curl_url_get() offers punycode conversions, to and from IDN. Asking to convert a name that is exactly 256 bytes, libcurl ends up reading outside of a stack based buffer when built to use the *macidn* IDN backend. The conversion function then fills up the provided buffer exactly - but does not null terminate the string.
7.3.0	CVE-2025-3576	A vulnerability in the MIT Kerberos implementation allows GSSAPI-protected messages using RC4-HMAC-MD5 to be spoofed due to weaknesses in the MD5 checksum design. If RC4 is preferred over stronger encryption types, an attacker could exploit MD5 collisions to forge message integrity codes. This may lead to unauthorized message tampering.
7.3.0	CVE-2021-4189	A flaw was found in Python, specifically in the FTP (File Transfer Protocol) client library in PASV (passive) mode. The issue is how the FTP client trusts the host from the PASV response by default. This flaw allows an attacker to set up a malicious FTP server that can trick FTP clients into connecting back to a given IP address and port. This vulnerability could lead to FTP client scanning ports, which otherwise would not have been possible.
7.3.0	CVE-2024-8096	When curl is told to use the Certificate Status Request TLS extension, often referred to as OCSP stapling, to verify that the server certificate is valid, it might fail to detect some OCSP problems and instead wrongly consider the response as fine. If the returned status reports another error than 'revoked' (like for example 'unauthorized') it is not treated as a bad certficate.
7.3.0	CVE-2024-11053	When asked to both use a `.netrc` file for credentials and to follow HTTP redirects, curl could leak the password used for the first host to the followed-to host under certain circumstances.
7.3.0	CVE-2024-2398	When an application tells libcurl it wants to allow HTTP/2 server push, and the amount of received headers for the push surpasses the maximum allowed limit (1000), libcurl aborts the server push. When aborting, libcurl inadvertently does not free all the previously allocated headers and instead leaks the memory. Further, this error condition fails silently and is therefore not easily detected by an application.
7.3.0	CVE-2025-27113	libxml2 before 2.12.10 and 2.13.x before 2.13.6 has a NULL pointer dereference in xmlPatMatch in pattern.c.
7.3.0	CVE-2019-1010204	GNU binutils gold gold v1.11-v1.16 (GNU binutils v2.21-v2.31.1) is affected by: Improper Input Validation, Signed/Unsigned Comparison, Out-of-bounds Read. The impact is: Denial of service. The component is: gold/fileread.cc:497, elfcpp/elfcpp_file.h:644. The attack vector is: An ELF file with an invalid e_shoff header field must be opened.
7.3.0	CVE-2025-0725	When libcurl is asked to perform automatic gzip decompression of content-encoded HTTP responses with the `CURLOPT_ACCEPT_ENCODING` option, **using zlib 1.2.0.3 or older**, an attacker-controlled integer overflow would make libcurl perform a buffer overflow.

Release	CVE Fix	Description
7.3.0	CVE-2024-9681	When curl is asked to use HSTS, the expiry time for a subdomain might overwrite a parent domain's cache entry, making it end sooner or later than otherwise intended.
7.3.0	CVE-2024-25629	c-ares is a C library for asynchronous DNS requests. `ares__read_line()` is used to parse local configuration files such as `/etc/resolv.conf`, `/etc/nsswitch.conf`, the `HOSTALIASES` file, and if using a c-ares version prior to 1.27.0, the `/etc/hosts` file. If any of these configuration files has an embedded `NULL` character as the first character in a new line, it can lead to attempting to read memory prior to the start of the given buffer which may result in a crash. This issue is fixed in c-ares 1.27.0. No known workarounds exist.
7.3.0	CVE-2024-2466	libcurl did not check the server certificate of TLS connections done to a host specified as an IP address, when built to use mbedTLS. libcurl would wrongly avoid using the set hostname function when the specified hostname was given as an IP address, therefore completely skipping the certificate check. This affects all uses of TLS protocols (HTTPS, FTPS, IMAPS, POPS3, SMTPS, etc).
7.3.0	CVE-2024-11168	The urllib.parse.urlsplit() and urlparse() functions improperly validated bracketed hosts (`[]`), allowing hosts that weren't IPv6 or IPvFuture. This behavior was not conformant to RFC 3986 and potentially enabled SSRF if a URL is processed by more than one URL parser.
7.3.0	CVE-2025-0938	The Python standard library functions `urllib.parse.urlsplit` and `urlparse` accepted domain names that included square brackets which isn't valid according to RFC 3986. Square brackets are only meant to be used as delimiters for specifying IPv6 and IPvFuture hosts in URLs. This could result in differential parsing across the Python URL parser and other specification-compliant URL parsers.
7.3.0	CVE-2023-27043	The email module of Python through 3.11.3 incorrectly parses e-mail addresses that contain a special character. The wrong portion of an RFC2822 header is identified as the value of the addr-spec. In some applications, an attacker can bypass a protection mechanism in which application access is granted only after verifying receipt of e-mail to a specific domain (e.g., only @company.example.com addresses may be used for signup). This occurs in email/_parseaddr.py in recent versions of Python.
7.3.0	CVE-2022-48064	GNU Binutils before 2.40 was discovered to contain an excessive memory consumption vulnerability via the function bfd_dwarf2_find_nearest_line_with_alt at dwarf2.c. The attacker could supply a crafted ELF file and cause a DNS attack.
7.3.0	CVE-2025-0665	libcurl would wrongly close the same eventfd file descriptor twice when taking down a connection channel after having completed a threaded name resolve.
7.3.0	CVE-2022-45061	An issue was discovered in Python before 3.11.1. An unnecessary quadratic algorithm exists in one path when processing some inputs to the IDNA (RFC 3490) decoder, such that a crafted, unreasonably long name being presented to the decoder could lead to a CPU denial of service. Hostnames are often supplied by remote servers that could be controlled by a malicious actor; in such a scenario, they could trigger excessive CPU consumption on the client attempting to make use of an attacker-supplied supposed hostname. For example, the attack payload could be placed in the Location header of an HTTP response with status code 302. A fix is planned in 3.11.1, 3.10.9, 3.9.16, 3.8.16, and 3.7.16.
7.3.0	CVE-2016-2568	pkexec, when used with --user nonpriv, allows local users to escape to the parent session via a crafted TIOCSTI ioctl call, which pushes characters to the terminal's input buffer.
7.3.0	CVE-2018-6952	A double free exists in the another_hunk function in pch.c in GNU patch through 2.7.6.
7.3.0	CVE-2013-4235	shadow: TOCTOU (time-of-check time-of-use) race condition when copying and removing directory trees
7.3.0	CVE-2025-29088	In SQLite 3.49.0 before 3.49.1, certain argument values to sqlite3_db_config (in the C-language API) can cause a denial of service (application crash). An sz*nBig multiplication is not cast to a 64-bit integer, and consequently some memory allocations may be incorrect.
7.2.2	CVE-2023-38408	The PKCS#11 feature in ssh-agent in OpenSSH before 9.3p2 has an insufficiently trustworthy search path, leading to remote code execution if an agent is forwarded to an attacker-controlled system.

Release	CVE Fix	Description
7.1.3	CVE-2025-1094	Improper neutralization of quoting syntax in PostgreSQL libpq functions PQescapeLiteral(), PQescapeIdentifier(), PQescapeString(), and PQescapeStringConn() allows a database input provider to achieve SQL injection in certain usage patterns. Specifically, SQL injection requires the application to use the function result to construct input to psql, the PostgreSQL interactive terminal. Similarly, improper neutralization of quoting syntax in PostgreSQL command line utility programs allows a source of command line arguments to achieve SQL injection when client_encoding is BIG5 and server_encoding is one of EUC_TW or MULE_INTERNAL.
7.1.0	CVE-2021-45261	An Invalid Pointer vulnerability exists in GNU patch 2.7 via the another_hunk function, which causes a Denial of Service.
7.1.0	CVE-2022-3857	'Rejected reason: Maintainer contacted. This is a false-positive. The flaw does not actually exist and was erroneously tested.'
7.1.0	CVE-2024-7592	There is a LOW severity vulnerability affecting CPython, specifically the 'http.cookies' standard library module.
7.1.0	CVE-2021-33574	The mq_notify function in the GNU C Library (aka glibc) versions 2.32 and 2.33 has a use-after-free. It may use the notification thread attributes object (passed through its struct sigevent parameter) after it has been freed by the caller, leading to a denial of service (application crash) or possibly unspecified other impact.
7.1.0	CVE-2024-4603	Checking excessively long DSA keys or parameters may be very slow.
7.1.0	CVE-2024-6119	Applications performing certificate name checks (e.g., TLS clients checking server certificates) may attempt to read an invalid memory address resulting in abnormal termination of the application process.
7.1.0	CVE-2023-45918	'Rejected reason: DO NOT USE THIS CANDIDATE NUMBER. ConsultIDs: none. Reason: This candidate was withdrawn by its CNA. Further investigation showed that it was not a security issue. Notes: none.'
7.1.0	CVE-2024-0727	Processing a maliciously formatted PKCS12 file may lead OpenSSL to crash leading to a potential Denial of Service attack
7.1.0	CVE-2022-27943	libiberty/rust-demangle.c in GNU GCC 11.2 allows stack consumption in demangle_const, as demonstrated by nm-new.
7.1.0	CVE-2022-48064	GNU Binutils before 2.40 was discovered to contain an excessive memory consumption vulnerability via the function bfd_dwarf2_find_nearest_line_with_alt at dwarf2.c. The attacker could supply a crafted ELF file and cause a DNS attack.
7.1.0	CVE-2019-1010024	'GNU Libc current is affected by: Mitigation bypass. The impact is: Attacker may bypass ASLR using cache of thread stack and heap. The component is: glibc. NOTE: Upstream comments indicate "this is being treated as a non-security bug and no real threat.'
7.1.0	CVE-2024-36137	'## Overview node is a JavaScript runtime built on Chrome"s V8 JavaScript engine. Affected versions of this package are vulnerable to Authorization Bypass via `fs.fchown` or `fs.fchmod` operations which can use a "read-only" file descriptor to change the owner and permissions of a file. **Note:** This is only exploitable for users using the experimental permission when the `--allow-fs-write` flag is used ## Remediation Upgrade `node` to version 20.15.1, 22.4.1 or higher. ## References - GitHub New Releases - GitHub New Releases - Nodejs Blog'
7.1.0	CVE-2013-4235	'shadow: TOCTOU (time-of-check time-of-use) race condition when copying and removing directory trees'
7.1.0	CVE-2019-1010022	'GNU Libc current is affected by: Mitigation bypass. The impact is: Attacker may bypass stack guard protection. The component is: nptl. The attack vector is: Exploit stack buffer overflow vulnerability and use this bypass vulnerability to bypass stack guard. NOTE: Upstream comments indicate "this is being treated as a non-security bug and no real threat.'
7.1.0	CVE-2023-4039	""
7.1.0	CVE-2016-20013	sha256crypt and sha512crypt through 0.6 allow attackers to cause a denial of service (CPU consumption) because the algorithm's runtime is proportional to the square of the length of the password.
7.1.0	CVE-2016-2781	chroot in GNU coreutils, when used with --userspec, allows local users to escape to the parent session via a crafted TIOCSTI ioctl call, which pushes characters to the terminal's input buffer.

Release	CVE Fix	Description
7.1.0	CVE-2023-29383	'In Shadow 4.13, it is possible to inject control characters into fields provided to the SUID program chfn (change finger). Although it is not possible to exploit this directly (e.g., adding a new user fails because \n is in the block list), it is possible to misrepresent the /etc/passwd file when viewed. Use of \r manipulations and Unicode characters to work around blocking of the : character make it possible to give the impression that a new user has been added. In other words, an adversary may be able to convince a system administrator to take the system offline (an indirect, social-engineered denial of service) by demonstrating that "cat /etc/passwd" shows a rogue user account.'
7.1.0	CVE-2023-42364	A use-after-free vulnerability in BusyBox v.1.36.1 allows attackers to cause a denial of service via a crafted awk pattern in the awk.c evaluate function.
7.1.0	CVE-2019-1010204	'GNU binutils gold gold v1.11-v1.16 (GNU binutils v2.21-v2.31.1) is affected by: Improper Input Validation, Signed/Unsigned Comparison, Out-of-bounds Read. The impact is: Denial of service. The component is: gold/fileread.cc:497, elfcpp/elfcpp_file.h:644. The attack vector is: An ELF file with an invalid e_shoff header field must be opened.'
7.1.0	CVE-2024-25629	c-ares is a C library for asynchronous DNS requests. `ares__read_line()` is used to parse local configuration files such as `/etc/resolv.conf`, `/etc/nsswitch.conf`, the `HOSTALIASES` file, and if using a c-ares version prior to 1.27.0, the `/etc/hosts` file. If any of these configuration files has an embedded `NULL` character as the first character in a new line, it can lead to attempting to read memory prior to the start of the given buffer which may result in a crash. This issue is fixed in c-ares 1.27.0. No known workarounds exist.
7.1.0	CVE-2018-20796	In the GNU C Library (aka glibc or libc6) through 2.29, check_dst_limits_calc_pos_1 in posix/regexec.c has Uncontrolled Recursion, as demonstrated by '(\227|)(\1\1|t1|\\2537)+' in grep.
7.1.0	CVE-2023-25193	hb-ot-layout-gsubgpos.hh in HarfBuzz through 6.0.0 allows attackers to trigger O(n^2) growth via consecutive marks during the process of looking back for base glyphs when attaching marks.
7.1.0	CVE-2024-34459	An issue was discovered in xmllint (from libxml2) before 2.11.8 and 2.12.x before 2.12.7. Formatting error messages with xmllint --htmlout can result in a buffer over-read in xmlHTMLPrintFileContext in xmllint.c.
7.1.0	CVE-2023-26604	systemd before 247 does not adequately block local privilege escalation for some Sudo configurations, e.g., plausible sudoers files in which the "systemctl status" command may be executed. Specifically, systemd does not set LESSSECURE to 1, and thus other programs may be launched from the less program. This presents a substantial security risk when running systemctl from Sudo, because less executes as root when the terminal size is too small to show the complete systemctl output.
7.1.0	CVE-2024-26461	Kerberos 5 (aka krb5) 1.21.2 contains a memory leak vulnerability in /krb5/src/lib/gssapi/krb5/k5sealv3.c.
7.1.0	CVE-2024-39689	Certifi is a curated collection of Root Certificates for validating the trustworthiness of SSL certificates while verifying the identity of TLS hosts. Certifi starting in 2021.05.30 and prior to 2024.07.4 recognized root certificates from `GLOBALTRUST`. Certifi 2024.07.04 removes root certificates from `GLOBALTRUST` from the root store. These are in the process of being removed from Mozilla's trust store. `GLOBALTRUST`'s root certificates are being removed pursuant to an investigation which identified "long-running and unresolved compliance issues."
7.1.0	CVE-2023-6129	The POLY1305 MAC (message authentication code) implementation contains a bug that might corrupt the internal state of applications running on PowerPC CPU based platforms if the CPU provides vector instructions.
7.1.0	CVE-2024-45490	An issue was discovered in libexpat before 2.6.3. xmlparse.c does not reject a negative length for XML_ParseBuffer.
7.1.0	CVE-2022-41409	Integer overflow vulnerability in pcre2test before 10.41 allows attackers to cause a denial of service or other unspecified impacts via negative input.
7.1.0	CVE-2023-33460	There's a memory leak in yajl 2.1.0 with use of yajl_tree_parse function. which will cause out-of-memory in server and cause crash.
7.1.0	CVE-2024-0853	curl inadvertently kept the SSL session ID for connections in its cache even when the verify status (*OCSP stapling*) test failed. A subsequent transfer to the same hostname could then succeed if the session ID cache was still fresh, which then skipped the verify status check.

Release	CVE Fix	Description
7.1.0	CVE-2024-6197	libcurl's ASN1 parser has this utf8asn1str() function used for parsing an ASN.1 UTF-8 string. Itcan detect an invalid field and return error. Unfortunately, when doing so it also invokes `free()` on a 4 byte localstack buffer. Most modern malloc implementations detect this error and immediately abort. Some however accept the input pointer and add that memory to its list of available chunks. This leads to the overwriting of nearby stack memory. The content of the overwrite is decided by the `free()` implementation; likely to be memory pointers and a set of flags. The most likely outcome of exploting this flaw is a crash, although it cannot be ruled out that more serious results can be had in special circumstances.
7.1.0	CVE-2024-4741	Calling the OpenSSL API function SSL_free_buffers may cause memory to be accessed that was previously freed in some situations
7.1.0	CVE-2024-41996	Validating the order of the public keys in the Diffie-Hellman Key Agreement Protocol, when an approved safe prime is used, allows remote attackers (from the client side) to trigger unnecessarily expensive server-side DHE modular-exponentiation calculations. The client may cause asymmetric resource consumption. The basic attack scenario is that the client must claim that it can only communicate with DHE, and the server must be configured to allow DHE and validate the order of the public key.
7.1.0	CVE-2024-28757	libexpat through 2.6.1 allows an XML Entity Expansion attack when there is isolated use of external parsers (created via XML_ExternalEntityParserCreate).
7.1.0	CVE-2024-2466	libcurl did not check the server certificate of TLS connections done to a host specified as an IP address, when built to use mbedTLS. libcurl would wrongly avoid using the set hostname function when the specified hostname was given as an IP address, therefore completely skipping the certificate check. This affects all uses of TLS protocols (HTTPS, FTPS, IMAPS, POPS3, SMTPS, etc).
7.1.0	CVE-2023-27043	The email module of Python through 3.11.3 incorrectly parses e-mail addresses that contain a special character. The wrong portion of an RFC2822 header is identified as the value of the addr-spec. In some applications, an attacker can bypass a protection mechanism in which application access is granted only after verifying receipt of e-mail to a specific domain (e.g., only @company.example.com addresses may be used for signup). This occurs in email/_parseaddr.py in recent versions of Python.
7.1.0	CVE-2023-7008	A vulnerability was found in systemd-resolved. This issue may allow systemd-resolved to accept records of DNSSEC-signed domains even when they have no signature, allowing man-in-the-middles (or the upstream DNS resolver) to manipulate records.
7.1.0	CVE-2010-4756	The glob implementation in the GNU C Library (aka glibc or libc6) allows remote authenticated users to cause a denial of service (CPU and memory consumption) via crafted glob expressions that do not match any pathnames, as demonstrated by glob expressions in STAT commands to an FTP daemon, a different vulnerability than CVE-2010-2632.
7.1.0	CVE-2023-48161	Buffer Overflow vulnerability in GifLib Project GifLib v.5.2.1 allows a local attacker to obtain sensitive information via the DumpSCreen2RGB function in gif2rgb.c
7.1.0	CVE-2024-43802	Vim is an improved version of the unix vi text editor. When flushing the typeahead buffer, Vim moves the current position in the typeahead buffer but does not check whether there is enough space left in the buffer to handle the next characters. So this may lead to the tb_off position within the typebuf variable to point outside of the valid buffer size, which can then later lead to a heap-buffer overflow in e.g. ins_typebuf(). Therefore, when flushing the typeahead buffer, check if there is enough space left before advancing the off position. If not, fall back to flush current typebuf contents. It's not quite clear yet, what can lead to this situation. It seems to happen when error messages occur (which will cause Vim to flush the typeahead buffer) in comnination with several long mappgins and so it may eventually move the off position out of a valid buffer size. Impact is low since it is not easily reproducible and requires to have several mappings active and run into some error condition. But when this happens, this will cause a crash. The issue has been fixed as of Vim patch v9.1.0697. Users are advised to upgrade. There are no known workarounds for this issue.
7.1.0	CVE-2021-4189	A flaw was found in Python, specifically in the FTP (File Transfer Protocol) client library in PASV (passive) mode. The issue is how the FTP client trusts the host from the PASV response by default. This flaw allows an attacker to set up a malicious FTP server that can trick FTP clients into connecting back to a given IP address and port. This vulnerability could lead to FTP client scanning ports, which otherwise would not have been possible.
7.1.0	CVE-2024-2398	When an application tells libcurl it wants to allow HTTP/2 server push, and the amount of received headers for the push surpasses the maximum allowed limit (1000), libcurl aborts the server push. When aborting, libcurl inadvertently does not free all the previously allocated headers and instead leaks the memory. Further, this error condition fails silently and is therefore not easily detected by an application.

Release	CVE Fix	Description
7.1.0	CVE-2024-2511	Some non-default TLS server configurations can cause unbounded memory growth when processing TLSv1.3 sessions
7.1.0	CVE-2024-8096	When curl is told to use the Certificate Status Request TLS extension, often referred to as OCSP stapling, to verify that the server certificate is valid, it might fail to detect some OCSP problems and instead wrongly consider the response as fine. If the returned status reports another error than 'revoked' (like for example 'unauthorized') it is not treated as a bad certficate.
7.1.0	CVE-2024-25062	An issue was discovered in libxml2 before 2.11.7 and 2.12.x before 2.12.5. When using the XML Reader interface with DTD validation and XInclude expansion enabled, processing crafted XML documents can lead to an xmlValidatePopElement use-after-free.
7.1.0	CVE-2018-6952	A double free exists in the another_hunk function in pch.c in GNU patch through 2.7.6.
7.1.0	CVE-2016-2568	pkexec, when used with --user nonpriv, allows local users to escape to the parent session via a crafted TIOCSTI ioctl call, which pushes characters to the terminal's input buffer.
7.1.0	CVE-2024-45492	An issue was discovered in libexpat before 2.6.3. nextScaffoldPart in xmlparse.c can have an integer overflow for m_groupSize on 32-bit platforms (where UINT_MAX equals SIZE_MAX).
7.1.0	CVE-2023-50495	NCurse v6.4-20230418 was discovered to contain a segmentation fault via the component _nc_wrap_entry().
7.1.0	CVE-2022-4899	A vulnerability was found in zstd v1.4.10, where an attacker can supply empty string as an argument to the command line tool to cause buffer overrun.
7.1.0	CVE-2017-13716	The C++ symbol demangler routine in cplus-dem.c in libiberty, as distributed in GNU Binutils 2.29, allows remote attackers to cause a denial of service (excessive memory allocation and application crash) via a crafted file, as demonstrated by a call from the Binary File Descriptor (BFD) library (aka libbfd).
7.1.0	CVE-2024-5535	Calling the OpenSSL API function SSL_select_next_proto with an empty supported client protocols buffer may cause a crash or memory contents to be sent to the peer.
7.1.0	CVE-2023-24329	An issue in the urllib.parse component of Python before 3.11.4 allows attackers to bypass blocklisting methods by supplying a URL that starts with blank characters.
7.1.0	CVE-2024-22018	'## Overview node is a JavaScript runtime built on Chrome"s V8 JavaScript engine. Affected versions of this package are vulnerable to Authorization Bypass due to a failure to restrict file stats through the `fs.lstat` API that allows attackers to retrieve stats from files to which they do not have explicit read access. **Note:** This is exploitable only for users of the experimental permission model when the `--allow-fs-read` flag is used. ## Remediation Upgrade `node` to version 20.15.1, 22.4.1 or higher. ## References - GitHub New Releases - GitHub New Releases - Nodejs Blog'
7.1.0	CVE-2024-47175	CUPS is a standards-based, open-source printing system, and `libppd` can be used for legacy PPD file support. The `libppd` function `ppdCreatePPDFromIPP2` does not sanitize IPP attributes when creating the PPD buffer. When used in combination with other functions such as `cfGetPrinterAttributes5`, can result in user controlled input and ultimately code execution via Foomatic. This vulnerability can be part of an exploit chain leading to remote code execution (RCE), as described in CVE-2024-47176.
7.1.0	CVE-2018-20657	The demangle_template function in cplus-dem.c in GNU libiberty, as distributed in GNU Binutils 2.31.1, has a memory leak via a crafted string, leading to a denial of service (memory consumption), as demonstrated by cxxfilt, a related issue to CVE-2018-12698.
7.1.0	CVE-2017-11164	In PCRE 8.41, the OP_KETRMAX feature in the match function in pcre_exec.c allows stack exhaustion (uncontrolled recursion) when processing a crafted regular expression.
7.1.0	CVE-2024-2379	libcurl skips the certificate verification for a QUIC connection under certain conditions, when built to use wolfSSL. If told to use an unknown/bad cipher or curve, the error path accidentally skips the verification and returns OK, thus ignoring any certificate problems.
7.1.0	CVE-2024-45491	An issue was discovered in libexpat before 2.6.3. dtdCopy in xmlparse.c can have an integer overflow for nDefaultAtts on 32-bit platforms (where UINT_MAX equals SIZE_MAX).

Release	CVE Fix	Description
7.1.0	CVE-2024-2004	When a protocol selection parameter option disables all protocols without adding any then the default set of protocols would remain in the allowed set due to an error in the logic for removing protocols. The below command would perform a request to curl.se with a plaintext protocol which has been explicitly disabled. curl --proto -all,-http http://curl.se The flaw is only present if the set of selected protocols disables the entire set of available protocols, in itself a command with no practical use and therefore unlikely to be encountered in real situations. The curl security team has thus assessed this to be low severity bug.
7.1.0	CVE-2019-9192	'In the GNU C Library (aka glibc or libc6) through 2.29, check_dst_limits_calc_pos_1 in posix/regexec.c has Uncontrolled Recursion, as demonstrated by '(|)(\1\1)*' in grep, a different issue than CVE-2018-20796. NOTE: the software maintainer disputes that this is a vulnerability because the behavior occurs only with a crafted pattern'
7.1.0	CVE-2020-36325	'An issue was discovered in Jansson through 2.13.1. Due to a parsing error in json_loads, there's an out-of-bounds read-access bug. NOTE: the vendor reports that this only occurs when a programmer fails to follow the API specification'
7.1.0	CVE-2023-34969	D-Bus before 1.15.6 sometimes allows unprivileged users to crash dbus-daemon. If a privileged user with control over the dbus-daemon is using the org.freedesktop.DBus.Monitoring interface to monitor message bus traffic, then an unprivileged user with the ability to connect to the same dbus-daemon can cause a dbus-daemon crash under some circumstances via an unreplyable message. When done on the well-known system bus, this is a denial-of-service vulnerability. The fixed versions are 1.12.28, 1.14.8, and 1.15.6.
7.1.0	CVE-2024-2236	A timing-based side-channel flaw was found in libgcrypt's RSA implementation. This issue may allow a remote attacker to initiate a Bleichenbacher-style attack, which can lead to the decryption of RSA ciphertexts.
7.1.0	CVE-2024-6874	libcurl's URL API function curl_url_get() offers punycode conversions, to and from IDN. Asking to convert a name that is exactly 256 bytes, libcurl ends up reading outside of a stack based buffer when built to use the *macidn* IDN backend. The conversion function then fills up the provided buffer exactly - but does not null terminate the string.
7.1.0	CVE-2023-42363	A use-after-free vulnerability was discovered in xasprintf function in xfuncs_printf.c:344 in BusyBox v.1.36.1.
7.1.0	CVE-2023-42365	A use-after-free vulnerability was discovered in BusyBox v.1.36.1 via a crafted awk pattern in the awk.c copyvar function.
7.1.0	CVE-2022-3219	GnuPG can be made to spin on a relatively small input by (for example) crafting a public key with thousands of signatures attached, compressed down to just a few KB.
7.1.0	CVE-2024-26458	Kerberos 5 (aka krb5) 1.21.2 contains a memory leak in /krb5/src/lib/rpc/pmap_rmt.c.
7.1.0	CVE-2018-5709	An issue was discovered in MIT Kerberos 5 (aka krb5) through 1.16. There is a variable "dbentry->n_key_data" in kadmin/dbutil/dump.c that can store 16-bit data but unknowingly the developer has assigned a "u4" variable to it, which is for 32-bit data. An attacker can use this vulnerability to affect other artifacts of the database as we know that a Kerberos database dump file contains trusted data.
7.1.0	CVE-2019-1010025	'GNU Libc current is affected by: Mitigation bypass. The impact is: Attacker may guess the heap addresses of pthread_created thread. The component is: glibc. NOTE: the vendor's position is "ASLR bypass itself is not a vulnerability.'
7.1.0	CVE-2023-42366	A heap-buffer-overflow was discovered in BusyBox v.1.36.1 in the next_token function at awk.c:1159.
7.1.0	CVE-2023-6237	Checking excessively long invalid RSA public keys may take a long time.
7.1.0	CVE-2022-0391	A flaw was found in Python, specifically within the urllib.parse module. This module helps break Uniform Resource Locator (URL) strings into components. The issue involves how the urlparse method does not sanitize input and allows characters like '\r' and '\n' in the URL path. This flaw allows an attacker to input a crafted URL, leading to injection attacks. This flaw affects Python versions prior to 3.10.0b1, 3.9.5, 3.8.11, 3.7.11 and 3.6.14.

Release	CVE Fix	Description
7.1.0	CVE-2024-37372	'## Overview node is a JavaScript runtime built on Chrome"s V8 JavaScript engine. Affected versions of this package are vulnerable to Improper Handling of Values. This is because the Permission Model assumes wrongly that any path starting with two backslashes `\` has a four-character prefix that can be ignored. **Note:** This vulnerability affects only Windows users of the Node.js Permission Model ## Remediation Upgrade `node` to version 20.15.1, 22.4.1 or higher. ## References - GitHub New Releases - GitHub New Releases - Nodejs Blog'
7.1.0	CVE-2017-18018	In GNU Coreutils through 8.29, chown-core.c in chown and chgrp does not prevent replacement of a plain file with a symlink during use of the POSIX "-R -L" options, which allows local users to modify the ownership of arbitrary files by leveraging a race condition.
7.1.0	CVE-2019-1010023	'GNU Libc current is affected by: Re-mapping current loaded library with malicious ELF file. The impact is: In worst case attacker may evaluate privileges. The component is: libld. The attack vector is: Attacker sends 2 ELF files to victim and asks to run ldd on it. ldd execute code. NOTE: Upstream comments indicate "this is being treated as a non-security bug and no real threat.'
7.1.0	CVE-2024-26462	Kerberos 5 (aka krb5) 1.21.2 contains a memory leak vulnerability in /krb5/src/kdc/ndr.c.
7.1.0	CVE-2023-5678	Generating excessively long X9.42 DH keys or checking excessively long X9.42 DH keys or parameters may be very slow.
7.1.0	CVE-2012-2663	'extensions/libxt_tcp.c in iptables through 1.4.21 does not match TCP SYN+FIN packets in --syn rules, which might allow remote attackers to bypass intended firewall restrictions via crafted packets. NOTE: the CVE-2012-6638 fix makes this issue less relevant.'
7.0.1	CVE-2023-25433	libtiff 4.5.0 is vulnerable to Buffer Overflow via /libtiff/tools/tiffcrop.c:8499. Incorrect updating of buffer size after rotateImage() in tiffcrop cause heap-buffer-overflow and SEGV.
7.0.1	CVE-2023-25652	Git is a revision control system. Prior to versions 2.30.9, 2.31.8, 2.32.7, 2.33.8, 2.34.8, 2.35.8, 2.36.6, 2.37.7, 2.38.5, 2.39.3, and 2.40.1, by feeding specially crafted input to `git apply --reject`, a path outside the working tree can be overwritten with partially controlled contents (corresponding to the rejected hunk(s) from the given patch). A fix is available in versions 2.30.9, 2.31.8, 2.32.7, 2.33.8, 2.34.8, 2.35.8, 2.36.6, 2.37.7, 2.38.5, 2.39.3, and 2.40.1. As a workaround, avoid using `git apply` with `--reject` when applying patches from an untrusted source. Use `git apply --stat` to inspect a patch before applying; avoid applying one that create a conflict where a link corresponding to the `*.rej` file exists.
7.0.1	CVE-2024-32465	'Git is a revision control system. The Git project recommends to avoid working in untrusted repositories, and instead to clone it first with `git clone --no-local` to obtain a clean copy. Git has specific protections to make that a safe operation even with an untrusted source repository, but vulnerabilities allow those protections to be bypassed. In the context of cloning local repositories owned by other users, this vulnerability has been covered in CVE-2024-32004. But there are circumstances where the fixes for CVE-2024-32004 are not enough: For example, when obtaining a `.zip` file containing a full copy of a Git repository, it should not be trusted by default to be safe, as e.g. hooks could be configured to run within the context of that repository. The problem has been patched in versions 2.45.1, 2.44.1, 2.43.4, 2.42.2, 2.41.1, 2.40.2, and 2.39.4. As a workaround, avoid using Git in repositories that have been obtained via archives from untrusted sources.'
7.0.1	CVE-2023-36191	sqlite3 v3.40.1 was discovered to contain a segmentation violation at /sqlite3_aflpp/shell.c.
7.0.1	CVE-2023-5363	A bug has been identified in the processing of key and initialisation vector (IV) lengths. This can lead to potential truncation or overruns during the initialisation of some symmetric ciphers.
7.0.1	CVE-2019-19882	shadow 4.8, in certain circumstances affecting at least Gentoo, Arch Linux, and Void Linux, allows local users to obtain root access because setuid programs are misconfigured. Specifically, this affects shadow 4.8 when compiled using --with-libpam but without explicitly passing --disable-account-tools-setuid, and without a PAM configuration suitable for use with setuid account management tools. This combination leads to account management tools (groupadd, groupdel, groupmod, useradd, userdel, usermod) that can easily be used by unprivileged local users to escalate privileges to root in multiple ways. This issue became much more relevant in approximately December 2019 when an unrelated bug was fixed (i.e., the chmod calls to suidusbins were fixed in the upstream Makefile which is now included in the release version 4.8).
7.0.1	CVE-2013-4392	systemd, when updating file permissions, allows local users to change the permissions and SELinux security contexts for arbitrary files via a symlink attack on unspecified files.

Release	CVE Fix	Description
7.0.1	CVE-2022-1210	A vulnerability classified as problematic was found in LibTIFF 4.3.0. Affected by this vulnerability is the TIFF File Handler of tiff2ps. Opening a malicious file leads to a denial of service. The attack can be launched remotely but requires user interaction. The exploit has been disclosed to the public and may be used.
7.0.1	CVE-2017-5563	LibTIFF version 4.0.7 is vulnerable to a heap-based buffer over-read in tif_lzw.c resulting in DoS or code execution via a crafted bmp image to tools/bmp2tiff.
7.0.1	CVE-2023-32570	VideoLAN dav1d before 1.2.0 has a thread_task.c race condition that can lead to an application crash, related to dav1d_decode_frame_exit.
7.0.1	CVE-2022-0563	A flaw was found in the util-linux chfn and chsh utilities when compiled with Readline support. The Readline library uses an "INPUTRC" environment variable to get a path to the library config file. When the library cannot parse the specified file, it prints an error message containing data from the file. This flaw allows an unprivileged user to read root-owned files, potentially leading to privilege escalation. This flaw affects util-linux versions prior to 2.37.4.
7.0.1	CVE-2017-17973	'** DISPUTED ** In LibTIFF 4.0.8, there is a heap-based use-after-free in the t2p_writeproc function in tiff2pdf.c. NOTE: there is a third-party report of inability to reproduce this issue.'
7.0.1	CVE-2023-6246	A heap-based buffer overflow was found in the __vsyslog_internal function of the glibc library. This function is called by the syslog and vsyslog functions. This issue occurs when the openlog function was not called, or called with the ident argument set to NULL, and the program name (the basename of argv[0]) is bigger than 1024 bytes, resulting in an application crash or local privilege escalation. This issue affects glibc 2.36 and newer.
7.0.1	CVE-2016-1585	In all versions of AppArmor mount rules are accidentally widened when compiled.
7.0.1	CVE-2023-6228	*This vulnerability has not been analyzed by NVD yet.*
7.0.1	CVE-2018-1000021	GIT version 2.15.1 and earlier contains a Input Validation Error vulnerability in Client that can result in problems including messing up terminal configuration to RCE. This attack appear to be exploitable via The user must interact with a malicious git server, (or have their traffic modified in a MITM attack).
7.0.1	CVE-2023-25815	In Git for Windows, the Windows port of Git, no localized messages are shipped with the installer. As a consequence, Git is expected not to localize messages at all, and skips the gettext initialization. However, due to a change in MINGW-packages, the `gettext()` function's implicit initialization no longer uses the runtime prefix but uses the hard-coded path `C:\mingw64\share\locale` to look for localized messages. And since any authenticated user has the permission to create folders in `C:\` (and since `C:\mingw64` does not typically exist), it is possible for low-privilege users to place fake messages in that location where `git.exe` will pick them up in version 2.40.1.
7.0.1	CVE-2023-49464	libheif v1.17.5 was discovered to contain a segmentation violation via the function UncompressedImageCodec::get_luma_bits_per_pixel_from_configuration_unci.
7.0.1	CVE-2023-31438	'** DISPUTED ** An issue was discovered in systemd 253. An attacker can truncate a sealed log file and then resume log sealing such that checking the integrity shows no error, despite modifications. NOTE: the vendor reportedly sent "a reply denying that any of the finding was a security vulnerability."'
7.0.1	CVE-2023-6135	Multiple NSS NIST curves were susceptible to a side-channel attack known as "Minerva". This attack could potentially allow an attacker to recover the private key. This vulnerability affects Firefox < 121.
7.0.1	CVE-2022-24975	The --mirror documentation for Git through 2.35.1 does not mention the availability of deleted content, aka the "GitBleed" issue. This could present a security risk if information-disclosure auditing processes rely on a clone operation without the --mirror option.
7.0.1	CVE-2023-49460	libheif v1.17.5 was discovered to contain a segmentation violation via the function UncompressedImageCodec::decode_uncompressed_image.

Release	CVE Fix	Description
7.0.1	CVE-2024-32004	Git is a revision control system. Prior to versions 2.45.1, 2.44.1, 2.43.4, 2.42.2, 2.41.1, 2.40.2, and 2.39.4, an attacker can prepare a local repository in such a way that, when cloned, will execute arbitrary code during the operation. The problem has been patched in versions 2.45.1, 2.44.1, 2.43.4, 2.42.2, 2.41.1, 2.40.2, and 2.39.4. As a workaround, avoid cloning repositories from untrusted sources.
7.0.1	CVE-2023-52425	libexpat through 2.5.0 allows a denial of service (resource consumption) because many full reparsings are required in the case of a large token for which multiple buffer fills are needed.
7.0.1	CVE-2024-21011	'Vulnerability in the Oracle Java SE, Oracle GraalVM for JDK, Oracle GraalVM Enterprise Edition product of Oracle Java SE (component: Hotspot). Supported versions that are affected are Oracle Java SE: 8u401, 8u401-perf, 11.0.22, 17.0.10, 21.0.2, 22; Oracle GraalVM for JDK: 17.0.10, 21.0.2, 22; Oracle GraalVM Enterprise Edition: 20.3.13 and 21.3.9. Difficult to exploit vulnerability allows unauthenticated attacker with network access via multiple protocols to compromise Oracle Java SE, Oracle GraalVM for JDK, Oracle GraalVM Enterprise Edition. Successful attacks of this vulnerability can result in unauthorized ability to cause a partial denial of service (partial DOS) of Oracle Java SE, Oracle GraalVM for JDK, Oracle GraalVM Enterprise Edition. Note: This vulnerability can be exploited by using APIs in the specified Component, e.g., through a web service which supplies data to the APIs. This vulnerability also applies to Java deployments, typically in clients running sandboxed Java Web Start applications or sandboxed Java applets, that load and run untrusted code (e.g., code that comes from the internet) and rely on the Java sandbox for security. CVSS 3.1 Base Score 3.7 (Availability impacts). CVSS Vector: (CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:N/A:L).'
7.0.1	CVE-2022-28506	There is a heap-buffer-overflow in GIFLIB 5.2.1 function DumpScreen2RGB() in gif2rgb.c:298:45.
7.0.1	CVE-2007-2768	OpenSSH, when using OPIE (One-Time Passwords in Everything) for PAM, allows remote attackers to determine the existence of certain user accounts, which displays a different response if the user account exists and is configured to use one-time passwords (OTP), a similar issue to CVE-2007-2243.
7.0.1	CVE-2024-1580	An integer overflow in dav1d AV1 decoder that can occur when decoding videos with large frame size. This can lead to memory corruption within the AV1 decoder. We recommend upgrading past version 1.4.0 of dav1d.
7.0.1	CVE-2023-31437	'** DISPUTED ** An issue was discovered in systemd 253. An attacker can modify a sealed log file such that, in some views, not all existing and sealed log messages are displayed. NOTE: the vendor reportedly sent "a reply denying that any of the finding was a security vulnerability."'
7.0.1	CVE-2023-31439	'** DISPUTED ** An issue was discovered in systemd 253. An attacker can modify the contents of past events in a sealed log file and then adjust the file such that checking the integrity shows no error, despite modifications. NOTE: the vendor reportedly sent "a reply denying that any of the finding was a security vulnerability."'
7.0.1	CVE-2024-0232	A heap use-after-free issue has been identified in SQLite in the jsonParseAddNodeArray() function in sqlite3.c. This flaw allows a local attacker to leverage a victim to pass specially crafted malicious input to the application, potentially causing a crash and leading to a denial of service.
7.0.1	CVE-2023-6780	An integer overflow was found in the __vsyslog_internal function of the glibc library. This function is called by the syslog and vsyslog functions. This issue occurs when these functions are called with a very long message, leading to an incorrect calculation of the buffer size to store the message, resulting in undefined behavior. This issue affects glibc 2.37 and newer.
7.0.1	CVE-2021-31879	GNU Wget through 1.21.1 does not omit the Authorization header upon a redirect to a different origin, a related issue to CVE-2018-1000007.
7.0.1	CVE-2023-52356	A segment fault (SEGV) flaw was found in libtiff that could be triggered by passing a crafted tiff file to the TIFFReadRGBATileExt() API. This flaw allows a remote attacker to cause a heap-buffer overflow, leading to a denial of service.
7.0.1	CVE-2023-3164	A heap-buffer-overflow vulnerability was found in LibTIFF, in extractImageSection() at tools/tiffcrop.c:7916 and tools/tiffcrop.c:7801. This flaw allows attackers to cause a denial of service via a crafted tiff file.
7.0.1	CVE-2023-3618	A flaw was found in libtiff. A specially crafted tiff file can lead to a segmentation fault due to a buffer overflow in the Fax3Encode function in libtiff/tif_fax3.c, resulting in a denial of service.

Release	CVE Fix	Description
7.0.1	CVE-2017-17740	contrib/slapd-modules/nops/nops.c in OpenLDAP through 2.4.45, when both the nops module and the memberof overlay are enabled, attempts to free a buffer that was allocated on the stack, which allows remote attackers to cause a denial of service (slapd crash) via a member MODDN operation.
7.0.1	CVE-2024-22365	linux-pam (aka Linux PAM) before 1.6.0 allows attackers to cause a denial of service (blocked login process) via mkfifo because the openat call (for protect_dir) lacks O_DIRECTORY.
7.0.1	CVE-2023-39130	GNU gdb (GDB) 13.0.50.20220805-git was discovered to contain a heap buffer overflow via the function pe_as16() at /gdb/coff-pe-read.c.
7.0.1	CVE-2024-28834	*This vulnerability has not been analyzed by NVD yet.*
7.0.1	CVE-2023-50387	Certain DNSSEC aspects of the DNS protocol (in RFC 4033, 4034, 4035, 6840, and related RFCs) allow remote attackers to cause a denial of service (CPU consumption) via one or more DNSSEC responses, aka the "KeyTrap" issue. One of the concerns is that, when there is a zone with many DNSKEY and RRSIG records, the protocol specification implies that an algorithm must evaluate all combinations of DNSKEY and RRSIG records.
7.0.1	CVE-2024-21094	'Vulnerability in the Oracle Java SE, Oracle GraalVM for JDK, Oracle GraalVM Enterprise Edition product of Oracle Java SE (component: Hotspot). Supported versions that are affected are Oracle Java SE: 8u401, 8u401-perf, 11.0.22, 17.0.10, 21.0.2, 22; Oracle GraalVM for JDK: 17.0.10, 21.0.2, 22; Oracle GraalVM Enterprise Edition: 20.3.13 and 21.3.9. Difficult to exploit vulnerability allows unauthenticated attacker with network access via multiple protocols to compromise Oracle Java SE, Oracle GraalVM for JDK, Oracle GraalVM Enterprise Edition. Successful attacks of this vulnerability can result in unauthorized update, insert or delete access to some of Oracle Java SE, Oracle GraalVM for JDK, Oracle GraalVM Enterprise Edition accessible data. Note: This vulnerability can be exploited by using APIs in the specified Component, e.g., through a web service which supplies data to the APIs. This vulnerability also applies to Java deployments, typically in clients running sandboxed Java Web Start applications or sandboxed Java applets, that load and run untrusted code (e.g., code that comes from the internet) and rely on the Java sandbox for security. CVSS 3.1 Base Score 3.7 (Integrity impacts). CVSS Vector: (CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:L/A:N).'
7.0.1	CVE-2024-33655	*This vulnerability has not been analyzed by NVD yet.*
7.0.1	CVE-2007-6755	'The NIST SP 800-90A default statement of the Dual Elliptic Curve Deterministic Random Bit Generation (Dual_EC_DRBG) algorithm contains point Q constants with a possible relationship to certain "skeleton key" values, which might allow context-dependent attackers to defeat cryptographic protection mechanisms by leveraging knowledge of those values. NOTE: this is a preliminary CVE for Dual_EC_DRBG; future research may provide additional details about point Q and associated attacks, and could potentially lead to a RECAST or REJECT of this CVE.'
7.0.1	CVE-2023-26965	loadImage() in tools/tiffcrop.c in LibTIFF through 4.5.0 has a heap-based use after free via a crafted TIFF image.
7.0.1	CVE-2021-32256	An issue was discovered in GNU libiberty, as distributed in GNU Binutils 2.36. It is a stack-overflow issue in demangle_type in rust-demangle.c.
7.0.1	CVE-2024-28085	wall in util-linux through 2.40, often installed with setgid tty permissions, allows escape sequences to be sent to other users' terminals through argv. (Specifically, escape sequences received from stdin are blocked, but escape sequences received from argv are not blocked.) There may be plausible scenarios where this leads to account takeover.
7.0.1	CVE-2024-21012	'Vulnerability in the Oracle Java SE, Oracle GraalVM for JDK, Oracle GraalVM Enterprise Edition product of Oracle Java SE (component: Networking). Supported versions that are affected are Oracle Java SE: 11.0.22, 17.0.10, 21.0.2, 22; Oracle GraalVM for JDK: 17.0.10, 21.0.2, 22; Oracle GraalVM Enterprise Edition: 20.3.13 and 21.3.9. Difficult to exploit vulnerability allows unauthenticated attacker with network access via multiple protocols to compromise Oracle Java SE, Oracle GraalVM for JDK, Oracle GraalVM Enterprise Edition. Successful attacks of this vulnerability can result in unauthorized update, insert or delete access to some of Oracle Java SE, Oracle GraalVM for JDK, Oracle GraalVM Enterprise Edition accessible data. Note: This vulnerability applies to Java deployments, typically in clients running sandboxed Java Web Start applications or sandboxed Java applets, that load and run untrusted code (e.g., code that comes from the internet) and rely on the Java sandbox for security. This vulnerability does not apply to Java deployments, typically in servers, that load and run only trusted code (e.g., code installed by an administrator). CVSS 3.1 Base Score 3.7 (Integrity impacts). CVSS Vector: (CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:L/A:N).'

Release	CVE Fix	Description
7.0.1	CVE-2011-3374	It was found that apt-key in apt, all versions, do not correctly validate gpg keys with the master keyring, leading to a potential man-in-the-middle attack.
7.0.1	CVE-2023-39616	AOMedia v3.0.0 to v3.5.0 was discovered to contain an invalid read memory access via the component assign_frame_buffer_p in av1/common/av1_common_int.h.
7.0.1	CVE-2023-6277	An out-of-memory flaw was found in libtiff. Passing a crafted tiff file to TIFFOpen() API may allow a remote attacker to cause a denial of service via a craft input with size smaller than 379 KB.
7.0.1	CVE-2018-9996	'An issue was discovered in cplus-dem.c in GNU libiberty, as distributed in GNU Binutils 2.30. Stack Exhaustion occurs in the C++ demangling functions provided by libiberty, and there are recursive stack frames: demangle_template_value_parm, demangle_integral_value, and demangle_expression.'
7.0.1	CVE-2020-15778	'scp in OpenSSH through 8.3p1 allows command injection in the scp.c toremote function, as demonstrated by backtick characters in the destination argument. NOTE: the vendor reportedly has stated that they intentionally omit validation of "anomalous argument transfers" because that could "stand a great chance of breaking existing workflows."'
7.0.1	CVE-2023-50868	The Closest Encloser Proof aspect of the DNS protocol (in RFC 5155 when RFC 9276 guidance is skipped) allows remote attackers to cause a denial of service (CPU consumption for SHA-1 computations) via DNSSEC responses in a random subdomain attack, aka the "NSEC3" issue. The RFC 5155 specification implies that an algorithm must perform thousands of iterations of a hash function in certain situations.
7.0.1	CVE-2023-31484	CPAN.pm before 2.35 does not verify TLS certificates when downloading distributions over HTTPS.
7.0.1	CVE-2017-16232	'** DISPUTED ** LibTIFF 4.0.8 has multiple memory leak vulnerabilities, which allow attackers to cause a denial of service (memory consumption), as demonstrated by tif_open.c, tif_lzw.c, and tif_aux.c. NOTE: Third parties were unable to reproduce the issue.'
7.0.1	CVE-2023-49462	libheif v1.17.5 was discovered to contain a segmentation violation via the component /libheif/exif.cc.
7.0.1	CVE-2023-52426	libexpat through 2.5.0 allows recursive XML Entity Expansion if XML_DTD is undefined at compile time.
7.0.1	CVE-2023-6879	Increasing the resolution of video frames, while performing a multi-threaded encode, can result in a heap overflow in av1_loop_restoration_dealloc().
7.0.1	CVE-2023-2908	A null pointer dereference issue was found in Libtiff's tif_dir.c file. This issue may allow an attacker to pass a crafted TIFF image file to the tiffcp utility which triggers a runtime error that causes undefined behavior. This will result in an application crash, eventually leading to a denial of service.
7.0.1	CVE-2008-3234	sshd in OpenSSH 4 on Debian GNU/Linux, and the 20070303 OpenSSH snapshot, allows remote authenticated users to obtain access to arbitrary SELinux roles by appending a :/ (colon slash) sequence, followed by the role name, to the username.
7.0.1	CVE-2016-20012	'** DISPUTED ** OpenSSH through 8.7 allows remote attackers, who have a suspicion that a certain combination of username and public key is known to an SSH server, to test whether this suspicion is correct. This occurs because a challenge is sent only when that combination could be valid for a login session. NOTE: the vendor does not recognize user enumeration as a vulnerability for this product.'
7.0.1	CVE-2024-32020	Git is a revision control system. Prior to versions 2.45.1, 2.44.1, 2.43.4, 2.42.2, 2.41.1, 2.40.2, and 2.39.4, local clones may end up hardlinking files into the target repository's object database when source and target repository reside on the same disk. If the source repository is owned by a different user, then those hardlinked files may be rewritten at any point in time by the untrusted user. Cloning local repositories will cause Git to either copy or hardlink files of the source repository into the target repository. This significantly speeds up such local clones compared to doing a "proper" clone and saves both disk space and compute time. When cloning a repository located on the same disk that is owned by a different user than the current user we also end up creating such hardlinks. These files will continue to be owned and controlled by the potentially-untrusted user and can be rewritten by them at will in the future. The problem has been patched in versions 2.45.1, 2.44.1, 2.43.4, 2.42.2, 2.41.1, 2.40.2, and 2.39.4.
7.0.1	CVE-2024-33602	'nscd: netgroup cache assumes NSS callback uses in-buffer strings'

Release	CVE Fix	Description
7.0.1	CVE-2023-1972	A potential heap based buffer overflow was found in _bfd_elf_slurp_version_tables() in bfd/elf.c. This may lead to loss of availability.
7.0.1	CVE-2020-15719	libldap in certain third-party OpenLDAP packages has a certificate-validation flaw when the third-party package is asserting RFC6125 support. It considers CN even when there is a non-matching subjectAltName (SAN). This is fixed in, for example, openldap-2.4.46-10.el8 in Red Hat Enterprise Linux.
7.0.1	CVE-2023-2953	A vulnerability was found in openldap. This security flaw causes a null pointer dereference in ber_memalloc_x() function.
7.0.1	CVE-2023-45853	'MiniZip in zlib through 1.3 has an integer overflow and resultant heap-based buffer overflow in zipOpenNewFileInZip4_64 via a long filename, comment, or extra field. NOTE: MiniZip is not a supported part of the zlib product.'
7.0.1	CVE-2024-32002	Git is a revision control system. Prior to versions 2.45.1, 2.44.1, 2.43.4, 2.42.2, 2.41.1, 2.40.2, and 2.39.4, repositories with submodules can be crafted in a way that exploits a bug in Git whereby it can be fooled into writing files not into the submodule's worktree but into a `.git/` directory. This allows writing a hook that will be executed while the clone operation is still running, giving the user no opportunity to inspect the code that is being executed. The problem has been patched in versions 2.45.1, 2.44.1, 2.43.4, 2.42.2, 2.41.1, 2.40.2, and 2.39.4. If symbolic link support is disabled in Git (e.g. via `git config --global core.symlinks false`), the described attack won't work. As always, it is best to avoid cloning repositories from untrusted sources.
7.0.1	CVE-2023-51792	Buffer Overflow vulnerability in libde265 v1.0.12 allows a local attacker to cause a denial of service via the allocation size exceeding the maximum supported size of 0x10000000000.
7.0.1	CVE-2018-6829	cipher/elgamal.c in Libgcrypt through 1.8.2, when used to encrypt messages directly, improperly encodes plaintexts, which allows attackers to obtain sensitive information by reading ciphertext data (i.e., it does not have semantic security in face of a ciphertext-only attack). The Decisional Diffie-Hellman (DDH) assumption does not hold for Libgcrypt's ElGamal implementation.
7.0.1	CVE-2010-0928	OpenSSL 0.9.8i on the Gaisler Research LEON3 SoC on the Xilinx Virtex-II Pro FPGA uses a Fixed Width Exponentiation (FWE) algorithm for certain signature calculations, and does not verify the signature before providing it to a caller, which makes it easier for physically proximate attackers to determine the private key via a modified supply voltage for the microprocessor, related to a "fault-based attack."
7.0.1	CVE-2020-12413	The Raccoon attack is a timing attack on DHE ciphersuites inherit in the TLS specification. To mitigate this vulnerability, Firefox disabled support for DHE ciphersuites.
7.0.1	CVE-2023-49463	libheif v1.17.5 was discovered to contain a segmentation violation via the function find_exif_tag at /libheif/exif.cc.
7.0.1	CVE-2021-45346	'A Memory Leak vulnerability exists in SQLite Project SQLite3 3.35.1 and 3.37.0 via maliciously crafted SQL Queries (made via editing the Database File), it is possible to query a record, and leak subsequent bytes of memory that extend beyond the record, which could let a malicious user obtain sensitive information. NOTE: The developer disputes this as a vulnerability stating that If you give SQLite a corrupted database file and submit a query against the database, it might read parts of the database that you did not intend or expect.'
7.0.1	CVE-2024-2961	The iconv() function in the GNU C Library versions 2.39 and older may overflow the output buffer passed to it by up to 4 bytes when converting strings to the ISO-2022-CN-EXT character set, which may be used to crash an application or overwrite a neighbouring variable.
7.0.1	CVE-2023-51767	'OpenSSH through 9.6, when common types of DRAM are used, might allow row hammer attacks (for authentication bypass) because the integer value of authenticated in mm_answer_authpassword does not resist flips of a single bit. NOTE: this is applicable to a certain threat model of attacker-victim co-location in which the attacker has user privileges.'
7.0.1	CVE-2022-29458	ncurses 6.3 before patch 20220416 has an out-of-bounds read and segmentation violation in convert_strings in tinfo/read_entry.c in the terminfo library.
7.0.1	CVE-2023-39742	giflib v5.2.1 was discovered to contain a segmentation fault via the component getarg.c.

Release	CVE Fix	Description
7.0.1	CVE-2020-22916	'An issue discovered in XZ 5.2.5 allows attackers to cause a denial of service via decompression of a crafted file. NOTE: the vendor disputes the claims of "endless output" and "denial of service" because decompression of the 17,486 bytes always results in 114,881,179 bytes, which is often a reasonable size increase.'
7.0.1	CVE-2023-3316	A NULL pointer dereference in TIFFClose() is caused by a failure to open an output file (non-existent path or a path that requires permissions like /dev/null) while specifying zones.
7.0.1	CVE-2011-3389	The SSL protocol, as used in certain configurations in Microsoft Windows and Microsoft Internet Explorer, Mozilla Firefox, Google Chrome, Opera, and other products, encrypts data by using CBC mode with chained initialization vectors, which allows man-in-the-middle attackers to obtain plaintext HTTP headers via a blockwise chosen-boundary attack (BCBA) on an HTTPS session, in conjunction with JavaScript code that uses (1) the HTML5 WebSocket API, (2) the Java URLConnection API, or (3) the Silverlight WebClient API, aka a "BEAST" attack.
7.0.1	CVE-2021-4217	A flaw was found in unzip. The vulnerability occurs due to improper handling of Unicode strings, which can lead to a null pointer dereference. This flaw allows an attacker to input a specially crafted zip file, leading to a crash or code execution.
7.0.1	CVE-2018-10126	LibTIFF 4.0.9 has a NULL pointer dereference in the jpeg_fdct_16x16 function in jfdctint.c.
7.0.1	CVE-2024-21068	'Vulnerability in the Oracle Java SE, Oracle GraalVM for JDK, Oracle GraalVM Enterprise Edition product of Oracle Java SE (component: Hotspot). Supported versions that are affected are Oracle Java SE: 8u401-perf, 11.0.22, 17.0.10, 21.0.2, 22; Oracle GraalVM for JDK: 17.0.10, 21.0.2 and 22; Oracle GraalVM Enterprise Edition: 21.3.9. Difficult to exploit vulnerability allows unauthenticated attacker with network access via multiple protocols to compromise Oracle Java SE, Oracle GraalVM for JDK, Oracle GraalVM Enterprise Edition. Successful attacks of this vulnerability can result in unauthorized update, insert or delete access to some of Oracle Java SE, Oracle GraalVM for JDK, Oracle GraalVM Enterprise Edition accessible data. Note: This vulnerability can be exploited by using APIs in the specified Component, e.g., through a web service which supplies data to the APIs. This vulnerability also applies to Java deployments, typically in clients running sandboxed Java Web Start applications or sandboxed Java applets, that load and run untrusted code (e.g., code that comes from the internet) and rely on the Java sandbox for security. CVSS 3.1 Base Score 3.7 (Integrity impacts). CVSS Vector: (CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:L/A:N).'
7.0.1	CVE-2023-31486	HTTP::Tiny before 0.083, a Perl core module since 5.13.9 and available standalone on CPAN, has an insecure default TLS configuration where users must opt in to verify certificates.
7.0.1	CVE-2021-40633	A memory leak (out-of-memory) in gif2rgb in util/gif2rgb.c in giflib 5.1.4 allows remote attackers trigger an out of memory exception or denial of service via a gif format file.
7.0.1	CVE-2023-39129	GNU gdb (GDB) 13.0.50.20220805-git was discovered to contain a heap use after free via the function add_pe_exported_sym() at /gdb/coff-pe-read.c.
7.0.1	CVE-2017-14159	slapd in OpenLDAP 2.4.45 and earlier creates a PID file after dropping privileges to a non-root account, which might allow local users to kill arbitrary processes by leveraging access to this non-root account for PID file modification before a root script executes a "kill `cat /pathname`" command, as demonstrated by openldap-initscript.
7.0.1	CVE-2007-5686	'initscripts in rPath Linux 1 sets insecure permissions for the /var/log/btmp file, which allows local users to obtain sensitive information regarding authentication attempts. NOTE: because sshd detects the insecure permissions and does not log certain events, this also prevents sshd from logging failed authentication attempts by remote attackers.'
7.0.1	CVE-2020-13844	Arm Armv8-A core implementations utilizing speculative execution past unconditional changes in control flow may allow unauthorized disclosure of information to an attacker with local user access via a side-channel analysis, aka "straight-line speculation."
7.0.1	CVE-2018-20712	A heap-based buffer over-read exists in the function d_expression_1 in cp-demangle.c in GNU libiberty, as distributed in GNU Binutils 2.31.1. A crafted input can cause segmentation faults, leading to denial-of-service, as demonstrated by c++filt.
7.0.1	CVE-2024-33600	'nscd: Null pointer crashes after notfound response'

Release	CVE Fix	Description
7.0.1	CVE-2020-14145	'The client side in OpenSSH 5.7 through 8.4 has an Observable Discrepancy leading to an information leak in the algorithm negotiation. This allows man-in-the-middle attackers to target initial connection attempts (where no host key for the server has been cached by the client). NOTE: some reports state that 8.5 and 8.6 are also affected.'
7.0.1	CVE-2023-4016	Under some circumstances, this weakness allows a user who has access to run the “ps” utility on a machine, the ability to write almost unlimited amounts of unfiltered data into the process heap.
7.0.1	CVE-2021-4214	A heap overflow flaw was found in libpngs' pngimage.c program. This flaw allows an attacker with local network access to pass a specially crafted PNG file to the pngimage utility, causing an application to crash, leading to a denial of service.
7.0.1	CVE-2024-33599	'nscd: Stack-based buffer overflow in netgroup cache'
7.0.1	CVE-2018-20673	The demangle_template function in cplus-dem.c in GNU libiberty, as distributed in GNU Binutils 2.31.1, contains an integer overflow vulnerability (for "Create an array for saving the template argument values") that can trigger a heap-based buffer overflow, as demonstrated by nm.
7.0.1	CVE-2015-3276	The nss_parse_ciphers function in libraries/libldap/tls_m.c in OpenLDAP does not properly parse OpenSSL-style multi-keyword mode cipher strings, which might cause a weaker than intended cipher to be used and allow remote attackers to have unspecified impact via unknown vectors.
7.0.1	CVE-2024-35195	Requests is a HTTP library. Prior to 2.32.0, when making requests through a Requests `Session`, if the first request is made with `verify=False` to disable cert verification, all subsequent requests to the same host will continue to ignore cert verification regardless of changes to the value of `verify`. This behavior will continue for the lifecycle of the connection in the connection pool. This vulnerability is fixed in 2.32.0.
7.0.1	CVE-2023-52355	An out-of-memory flaw was found in libtiff that could be triggered by passing a crafted tiff file to the TIFFRasterScanlineSize64() API. This flaw allows a remote attacker to cause a denial of service via a crafted input with a size smaller than 379 KB.
7.0.1	CVE-2023-29007	Git is a revision control system. Prior to versions 2.30.9, 2.31.8, 2.32.7, 2.33.8, 2.34.8, 2.35.8, 2.36.6, 2.37.7, 2.38.5, 2.39.3, and 2.40.1, a specially crafted `.gitmodules` file with submodule URLs that are longer than 1024 characters can used to exploit a bug in `config.c::git_config_copy_or_rename_section_in_file()`. This bug can be used to inject arbitrary configuration into a user's `$GIT_DIR/config` when attempting to remove the configuration section associated with that submodule. When the attacker injects configuration values which specify executables to run (such as `core.pager`, `core.editor`, `core.sshCommand`, etc.) this can lead to a remote code execution. A fix A fix is available in versions 2.30.9, 2.31.8, 2.32.7, 2.33.8, 2.34.8, 2.35.8, 2.36.6, 2.37.7, 2.38.5, 2.39.3, and 2.40.1. As a workaround, avoid running `git submodule deinit` on untrusted repositories or without prior inspection of any submodule sections in `$GIT_DIR/config`.
7.0.1	CVE-2024-32021	Git is a revision control system. Prior to versions 2.45.1, 2.44.1, 2.43.4, 2.42.2, 2.41.1, 2.40.2, and 2.39.4, when cloning a local source repository that contains symlinks via the filesystem, Git may create hardlinks to arbitrary user-readable files on the same filesystem as the target repository in the `objects/` directory. Cloning a local repository over the filesystem may creating hardlinks to arbitrary user-owned files on the same filesystem in the target Git repository's `objects/` directory. When cloning a repository over the filesystem (without explicitly specifying the `file://` protocol or `--no-local`), the optimizations for local cloning will be used, which include attempting to hard link the object files instead of copying them. While the code includes checks against symbolic links in the source repository, which were added during the fix for CVE-2022-39253, these checks can still be raced because the hard link operation ultimately follows symlinks. If the object on the filesystem appears as a file during the check, and then a symlink during the operation, this will allow the adversary to bypass the check and create hardlinks in the destination objects directory to arbitrary, user-readable files. The problem has been patched in versions 2.45.1, 2.44.1, 2.43.4, 2.42.2, 2.41.1, 2.40.2, and 2.39.4.
7.0.1	CVE-2024-27980	'## Overview node is a JavaScript runtime built on Chrome"s V8 JavaScript engine. Affected versions of this package are vulnerable to Improper Control of Generation of Code ("Code Injection") due to the improper handling of batch files in child_process.spawn or child_process.spawnSync. An attacker can inject arbitrary commands and achieve code execution even if the shell option is not enabled. **Note:** This vulnerability only affects Windows machines. ## Remediation Upgrade `node` to version 18.20.2, 20.12.2, 21.7.3 or higher. ## References - GitHub Commit - Node.js'

Release	CVE Fix	Description
7.0.1	CVE-2023-6779	An off-by-one heap-based buffer overflow was found in the __vsyslog_internal function of the glibc library. This function is called by the syslog and vsyslog functions. This issue occurs when these functions are called with a message bigger than INT_MAX bytes, leading to an incorrect calculation of the buffer size to store the message, resulting in an application crash. This issue affects glibc 2.37 and newer.
7.0.1	CVE-2023-4641	*This vulnerability has not been analyzed by NVD yet.*
7.0.1	CVE-2024-28182	nghttp2 is an implementation of the Hypertext Transfer Protocol version 2 in C. The nghttp2 library prior to version 1.61.0 keeps reading the unbounded number of HTTP/2 CONTINUATION frames even after a stream is reset to keep HPACK context in sync. This causes excessive CPU usage to decode HPACK stream. nghttp2 v1.61.0 mitigates this vulnerability by limiting the number of CONTINUATION frames it accepts per stream. There is no workaround for this vulnerability.
7.0.1	CVE-2020-23922	An issue was discovered in giflib through 5.1.4. DumpScreen2RGB in gif2rgb.c has a heap-based buffer over-read.
7.0.1	CVE-2024-25269	libheif <= 1.17.6 contains a memory leak in the function JpegEncoder::Encode. This flaw allows an attacker to cause a denial of service attack.
7.0.1	CVE-2005-2541	Tar 1.15.1 does not properly warn the user when extracting setuid or setgid files, which may allow local users or remote attackers to gain privileges.
7.0.1	CVE-2017-9937	In LibTIFF 4.0.8, there is a memory malloc failure in tif_jbig.c. A crafted TIFF document can lead to an abort resulting in a remote denial of service attack.
7.0.1	CVE-2017-9117	In LibTIFF 4.0.7, the program processes BMP images without verifying that biWidth and biHeight in the bitmap-information header match the actual input, leading to a heap-based buffer over-read in bmp2tiff.
7.0.1	CVE-2023-1916	A flaw was found in tiffcrop, a program distributed by the libtiff package. A specially crafted tiff file can lead to an out-of-bounds read in the extractImageSection function in tools/tiffcrop.c, resulting in a denial of service and limited information disclosure. This issue affects libtiff versions 4.x.
7.0.1	CVE-2019-6110	In OpenSSH 7.9, due to accepting and displaying arbitrary stderr output from the server, a malicious server (or Man-in-The-Middle attacker) can manipulate the client output, for example to use ANSI control codes to hide additional files being transferred.
7.0.1	CVE-2018-18483	The get_count function in cplus-dem.c in GNU libiberty, as distributed in GNU Binutils 2.31, allows remote attackers to cause a denial of service (malloc called with the result of an integer-overflowing calculation) or possibly have unspecified other impact via a crafted string, as demonstrated by c++filt.
7.0.1	CVE-2023-29659	A Segmentation fault caused by a floating point exception exists in libheif 1.15.1 using crafted heif images via the heif::Fraction::round() function in box.cc, which causes a denial of service.
7.0.1	CVE-2007-2243	OpenSSH 4.6 and earlier, when ChallengeResponseAuthentication is enabled, allows remote attackers to determine the existence of user accounts by attempting to authenticate via S/KEY, which displays a different response if the user account exists, a similar issue to CVE-2001-1483.
7.0.1	CVE-2023-37920	Certifi is a curated collection of Root Certificates for validating the trustworthiness of SSL certificates while verifying the identity of TLS hosts. Certifi prior to version 2023.07.22 recognizes "e-Tugra" root certificates. e-Tugra's root certificates were subject to an investigation prompted by reporting of security issues in their systems. Certifi 2023.07.22 removes root certificates from "e-Tugra" from the root store.
7.0.1	CVE-2018-15919	'Remotely observable behaviour in auth-gss2.c in OpenSSH through 7.8 could be used by remote attackers to detect existence of users on a target system when GSS2 is in use. NOTE: the discoverer states 'We understand that the OpenSSH developers do not want to treat such a username enumeration (or "oracle") as a vulnerability."
7.0.1	CVE-2024-28835	A flaw has been discovered in GnuTLS where an application crash can be induced when attempting to verify a specially crafted .pem bundle using the "certtool --verify-chain" command.

Release	CVE Fix	Description
7.0.1	CVE-2024-33601	'nscd: netgroup cache may terminate daemon on memory allocation failure'
7.0.1	CVE-2011-4116	_is_safe in the File::Temp module for Perl does not properly handle symlinks.
7.0.1	CVE-2023-7104	A vulnerability was found in SQLite SQLite3 up to 3.43.0 and classified as critical. This issue affects the function sessionReadRecord of the file ext/session/sqlite3session.c of the component make alltest Handler. The manipulation leads to heap-based buffer overflow. It is recommended to apply a patch to fix this issue. The associated identifier of this vulnerability is VDB-248999.
7.0.1	CVE-2023-39128	GNU gdb (GDB) 13.0.50.20220805-git was discovered to contain a stack overflow via the function ada_decode at /gdb/ada-lang.c.
7.0.1	CVE-2023-26966	libtiff 4.5.0 is vulnerable to Buffer Overflow in uv_encode() when libtiff reads a corrupted little-endian TIFF file and specifies the output to be big-endian.
7.0.1	CVE-2024-34397	An issue was discovered in GNOME GLib before 2.78.5, and 2.79.x and 2.80.x before 2.80.1. When a GDBus-based client subscribes to signals from a trusted system service such as NetworkManager on a shared computer, other users of the same computer can send spoofed D-Bus signals that the GDBus-based client will wrongly interpret as having been sent by the trusted system service. This could lead to the GDBus-based client behaving incorrectly, with an application-dependent impact.
6.9.3	CVE-2024-27983	**Overview** node **is a JavaScript runtime built on Chrome's V8 JavaScript engine. Affected versions of this package are vulnerable to Allocation of Resources Without Limits or Throttling due to a race condition in** `Http2Session` **when nghttp2 data is left in memory after a connection is reset while processing HTTP/2** `CONTINUATION` **frames. An attacker can cause denial of service by sending such frames then triggering the** `Http2Session` **destructor. # Details Denial of Service (DoS) describes a family of attacks, all aimed at making a system inaccessible to its intended and legitimate users. Unlike other vulnerabilities, DoS attacks usually do not aim at breaching security. Rather, they are focused on making websites and services unavailable to genuine users resulting in downtime. One popular Denial of Service vulnerability is DDoS (a Distributed Denial of Service), an attack that attempts to clog network pipes to the system by generating a large volume of traffic from many machines. When it comes to open source libraries, DoS vulnerabilities allow attackers to trigger such a crash or crippling of the service by using a flaw either in the application code or from the use of open source libraries. Two common types of DoS vulnerabilities: * High CPU/Memory Consumption- An attacker sending crafted requests that could cause the system to take a disproportionate amount of time to process. For example,** commons-fileupload:commons-fileupload**. * Crash - An attacker sending crafted requests that could cause the system to crash. For Example,** npm `ws` package **## Remediation Upgrade** `node` **to version 18.20.1, 20.12.1, 21.7.2 or higher. ## References -** CERT/CC Advisory **-** GitHub Commit **-** GitHub Commit **-** GitHub Commit **-** Node.js Release Notes **-** RedHat Bugzilla Bug **-** Vulnerability Report
6.9.3	CVE-2018-5709	An issue was discovered in MIT Kerberos 5 (aka krb5) through 1.16. There is a variable "dbentry->n_key_data" in kadmin/dbutil/dump.c that can store 16-bit data but unknowingly the developer has assigned a "u4" variable to it, which is for 32-bit data. An attacker can use this vulnerability to affect other artifacts of the database as we know that a Kerberos database dump file contains trusted data.
6.9.3	CVE-2022-47007	An issue was discovered function stab_demangle_v3_arg in stabs.c in Binutils 2.34 thru 2.38, allows attackers to cause a denial of service due to memory leaks.

Release	CVE Fix	Description
6.9.3	CVE-2018-20657	The demangle_template function in cplus-dem.c in GNU libiberty, as distributed in GNU Binutils 2.31.1, has a memory leak via a crafted string, leading to a denial of service (memory consumption), as demonstrated by cxxfilt, a related issue to CVE-2018-12698.
6.9.3	CVE-2023-52356	A segment fault (SEGV) flaw was found in libtiff that could be triggered by passing a crafted tiff file to the TIFFReadRGBATileExt() API. This flaw allows a remote attacker to cause a heap-buffer overflow, leading to a denial of service.
6.9.3	CVE-2007-2243	OpenSSH 4.6 and earlier, when ChallengeResponseAuthentication is enabled, allows remote attackers to determine the existence of user accounts by attempting to authenticate via S/KEY, which displays a different response if the user account exists, a similar issue to CVE-2001-1483
6.9.3	CVE-2020-14145	The client side in OpenSSH 5.7 through 8.4 has an Observable Discrepancy leading to an information leak in the algorithm negotiation. This allows man-in-the-middle attackers to target initial connection attempts (where no host key for the server has been cached by the client). NOTE: some reports state that 8.5 and 8.6 are also affected.
6.9.3	CVE-2020-22916	An issue discovered in XZ 5.2.5 allows attackers to cause a denial of service via decompression of a crafted file. NOTE: the vendor disputes the claims of "endless output" and "denial of service" because decompression of the 17,486 bytes always results in 114,881,179 bytes, which is often a reasonable size increase.
6.9.3	CVE-2022-47011	An issue was discovered function parse_stab_struct_fields in stabs.c in Binutils 2.34 thru 2.38, allows attackers to cause a denial of service due to memory leaks.
6.9.3	CVE-2024-24758	**Overview** node **is a JavaScript runtime built on Chrome's V8 JavaScript engine. Affected versions of this package are vulnerable to Permissive Cross-domain Policy with Untrusted Domains due to not clearing** `Proxy-Authentication` **headers on cross-origin redirects. An attacker can intercept the improperly cleared headers. ## Remediation Upgrade** `node` **to version 18.19.1, 20.11.1, 21.6.2 or higher. ## References** - GitHub Commit - GitHub Commit - GitHub Commit - GitHub Commit - GitHub Commit
6.9.3	CVE-2023-29383	In Shadow 4.13, it is possible to inject control characters into fields provided to the SUID program chfn (change finger). Although it is not possible to exploit this directly (e.g., adding a new user fails because \n is in the block list), it is possible to misrepresent the /etc/passwd file when viewed. Use of \r manipulations and Unicode characters to work around blocking of the : character make it possible to give the impression that a new user has been added. In other words, an adversary may be able to convince a system administrator to take the system offline (an indirect, social-engineered denial of service) by demonstrating that "cat /etc/passwd" shows a rogue user account.
6.9.3	CVE-2021-4189	A flaw was found in Python, specifically in the FTP (File Transfer Protocol) client library in PASV (passive) mode. The issue is how the FTP client trusts the host from the PASV response by default. This flaw allows an attacker to set up a malicious FTP server that can trick FTP clients into connecting back to a given IP address and port. This vulnerability could lead to FTP client scanning ports, which otherwise would not have been possible.
6.9.3	CVE-2024-20926	Vulnerability in the Oracle Java SE, Oracle GraalVM for JDK, Oracle GraalVM Enterprise Edition product of Oracle Java SE (component: Scripting). Supported versions that are affected are Oracle Java SE: 8u391, 8u391-perf, 11.0.21; Oracle GraalVM for JDK: 17.0.9; Oracle GraalVM Enterprise Edition: 20.3.12, 21.3.8 and 22.3.4. Difficult to exploit vulnerability allows unauthenticated attacker with network access via multiple protocols to compromise Oracle Java SE, Oracle GraalVM for JDK, Oracle GraalVM Enterprise Edition. Successful attacks of this vulnerability can result in unauthorized access to critical data or complete access to all Oracle Java SE, Oracle GraalVM for JDK, Oracle GraalVM Enterprise Edition accessible data. Note: This vulnerability can be exploited by using APIs in the specified Component, e.g., through a web service which supplies data to the APIs. This vulnerability also applies to Java deployments, typically in clients running sandboxed Java Web Start applications or sandboxed Java applets, that load and run untrusted code (e.g., code that comes from the internet) and rely on the Java sandbox for security. CVSS 3.1 Base Score 5.9 (Confidentiality impacts). CVSS Vector: (CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:N/A:N).

Release	CVE Fix	Description
6.9.3	CVE-2020-36325	An issue was discovered in Jansson through 2.13.1. Due to a parsing error in json_loads, there's an out-of-bounds read-access bug. NOTE: the vendor reports that this only occurs when a programmer fails to follow the API specification
6.9.3	CVE-2024-2961	The iconv() function in the GNU C Library versions 2.39 and older may overflow the output buffer passed to it by up to 4 bytes when converting strings to the ISO-2022-CN-EXT character set, which may be used to crash an application or overwrite a neighbouring variable.
6.9.3	CVE-2023-6237	*This vulnerability has not been analyzed by NVD yet.*
6.9.3	CVE-2024-26461	Kerberos 5 (aka krb5) 1.21.2 contains a memory leak vulnerability in /krb5/src/lib/gssapi/krb5/k5sealv3.c.
6.9.3	CVE-2023-46809	**Overview** node **is a JavaScript runtime built on Chrome's V8 JavaScript engine. Affected versions of this package are vulnerable to Observable Timing Discrepancy due to the implementation of PKCS#1 v1.5 padding. An attacker can infer the private key used in the cryptographic operation by observing the time taken to execute cryptographic operations (Marvin). ## Remediation Upgrade** `node` **to version 18.19.1, 20.11.1, 21.6.2 or higher. ## References -** GitHub Commit **-** GitHub Commit **-** GitHub Commit **-** RedHat Bugzilla Bug **-** Vulnerability Report
6.9.3	CVE-2023-6129	The POLY1305 MAC (message authentication code) implementation contains a bug that might corrupt the internal state of applications running on PowerPC CPU based platforms if the CPU provides vector instructions.
6.9.3	CVE-2023-4421	The NSS code used for checking PKCS#1 v1.5 was leaking information useful in mounting Bleichenbacher-like attacks. Both the overall correctness of the padding as well as the length of the encrypted message was leaking through timing side-channel. By sending large number of attacker-selected ciphertexts, the attacker would be able to decrypt a previously intercepted PKCS#1 v1.5 ciphertext (for example, to decrypt a TLS session that used RSA key exchange), or forge a signature using the victim's key. The issue was fixed by implementing the implicit rejection algorithm, in which the NSS returns a deterministic random message in case invalid padding is detected, as proposed in the Marvin Attack paper. This vulnerability affects NSS < 3.61.
6.9.3	CVE-2018-20673	The demangle_template function in cplus-dem.c in GNU libiberty, as distributed in GNU Binutils 2.31.1, contains an integer overflow vulnerability (for "Create an array for saving the template argument values") that can trigger a heap-based buffer overflow, as demonstrated by nm.
6.9.3	CVE-2023-6879	Increasing the resolution of video frames, while performing a multi-threaded encode, can result in a heap overflow in av1_loop_restoration_dealloc().

Release	CVE Fix	Description
6.9.3	CVE-2024-22019	**Overview** node **is a JavaScript runtime built on Chrome's V8 JavaScript engine. Affected versions of this package are vulnerable to Allocation of Resources Without Limits or Throttling due to a lack of safeguards on chunk extension bytes. The server may read an unbounded number of bytes from a single connection, which allows an attacker to cause denial of service via CPU and network bandwidth exhaustion. ## Details Denial of Service (DoS) describes a family of attacks, all aimed at making a system inaccessible to its intended and legitimate users. Unlike other vulnerabilities, DoS attacks usually do not aim at breaching security. Rather, they are focused on making websites and services unavailable to genuine users resulting in downtime. One popular Denial of Service vulnerability is DDoS (a Distributed Denial of Service), an attack that attempts to clog network pipes to the system by generating a large volume of traffic from many machines. When it comes to open source libraries, DoS vulnerabilities allow attackers to trigger such a crash or crippling of the service by using a flaw either in the application code or from the use of open source libraries. Two common types of DoS vulnerabilities: * High CPU/Memory Consumption- An attacker sending crafted requests that could cause the system to take a disproportionate amount of time to process. For example,** commons-fileupload:commons-fileupload**. * Crash - An attacker sending crafted requests that could cause the system to crash. For Example,** npm `ws` package **## Remediation Upgrade** `node` **to version 18.19.1, 20.11.1, 21.6.2 or higher. ## References -** GitHub Commit **-** GitHub Commit **-** GitHub Commit **-** RedHat Bugzilla Bug
6.9.3	CVE-2013-4235	shadow: TOCTOU (time-of-check time-of-use) race condition when copying and removing directory trees.
6.9.3	CVE-2023-1916	A flaw was found in tiffcrop, a program distributed by the libtiff package. A specially crafted tiff file can lead to an out-of-bounds read in the extractImageSection function in tools/tiffcrop.c, resulting in a denial of service and limited information disclosure. This issue affects libtiff versions 4.x.
6.9.3	CVE-2023-45918	ncurses 6.4-20230610 has a NULL pointer dereference in tgetstr in tinfo/lib_termcap.c.
6.9.3	CVE-2023-48161	Buffer Overflow vulnerability in GifLib Project GifLib v.5.2.1 allows a local attacker to obtain sensitive information via the DumpSCreen2RGB function in gif2rgb.c
6.9.3	CVE-2023-45853	MiniZip in zlib through 1.3 has an integer overflow and resultant heap-based buffer overflow in zipOpenNewFileInZip4_64 via a long filename, comment, or extra field. NOTE: MiniZip is not a supported part of the zlib product. NOTE: pyminizip through 0.2.6 is also vulnerable because it bundles an affected zlib version, and exposes the applicable MiniZip code through its compress API.
6.9.3	CVE-2023-6918	A flaw was found in the libssh implements abstract layer for message digest (MD) operations implemented by different supported crypto backends. The return values from these were not properly checked, which could cause low-memory situations failures, NULL dereferences, crashes, or usage of the uninitialized memory as an input for the KDF. In this case, non-matching keys will result in decryption/integrity failures, terminating the connection.
6.9.3	CVE-2024-2511	Some non-default TLS server configurations can cause unbounded memory growth when processing TLSv1.3 sessions.

Release	CVE Fix	Description
6.9.3	CVE-2024-20945	Vulnerability in the Oracle Java SE, Oracle GraalVM for JDK, Oracle GraalVM Enterprise Edition product of Oracle Java SE (component: Security). Supported versions that are affected are Oracle Java SE: 8u391, 8u391-perf, 11.0.21, 17.0.9, 21.0.1; Oracle GraalVM for JDK: 17.0.9, 21.0.1; Oracle GraalVM Enterprise Edition: 20.3.12, 21.3.8 and 22.3.4. Difficult to exploit vulnerability allows low privileged attacker with logon to the infrastructure where Oracle Java SE, Oracle GraalVM for JDK, Oracle GraalVM Enterprise Edition executes to compromise Oracle Java SE, Oracle GraalVM for JDK, Oracle GraalVM Enterprise Edition. Successful attacks of this vulnerability can result in unauthorized access to critical data or complete access to all Oracle Java SE, Oracle GraalVM for JDK, Oracle GraalVM Enterprise Edition accessible data. Note: This vulnerability can be exploited by using APIs in the specified Component, e.g., through a web service which supplies data to the APIs. This vulnerability also applies to Java deployments, typically in clients running sandboxed Java Web Start applications or sandboxed Java applets, that load and run untrusted code (e.g., code that comes from the internet) and rely on the Java sandbox for security. CVSS 3.1 Base Score 4.7 (Confidentiality impacts). CVSS Vector: (CVSS:3.1/AV:L/AC:H/PR:L/UI:N/S:U/C:H/I:N/A:N).
6.9.3	CVE-2023-32570	VideoLAN dav1d before 1.2.0 has a thread_task.c race condition that can lead to an application crash, related to dav1d_decode_frame_exit.
6.9.3	CVE-2023-3164	A heap-buffer-overflow vulnerability was found in LibTIFF, in extractImageSection() at tools/tiffcrop.c:7916 and tools/tiffcrop.c:7801. This flaw allows attackers to cause a denial of service via a crafted tiff file.
6.9.3	CVE-2024-22667	Vim before 9.0.2142 has a stack-based buffer overflow because did_set_langmap in map.c calls sprintf to write to the error buffer that is passed down to the option callback functions.
6.9.3	CVE-2024-21892	**Overview** node **is a JavaScript runtime built on Chrome's V8 JavaScript engine. Affected versions of this package are vulnerable to Code Injection due to the incorrect handling of environment variables on Linux when the process is running with elevated privileges that the current user lacks (does not apply to** `CAP_NET_BIND_SERVICE`**). ## Remediation Upgrade** `node` **to version 18.19.1, 20.11.1, 21.6.2 or higher. ## References -** GitHub Commit **-** GitHub Commit **-** GitHub Commit **-** RedHat Bugzilla Bug
6.9.3	CVE-2024-26462	Kerberos 5 (aka krb5) 1.21.2 contains a memory leak vulnerability in /krb5/src/kdc/ndr.c.
6.9.3	CVE-2023-26965	loadImage() in tools/tiffcrop.c in LibTIFF through 4.5.0 has a heap-based use after free via a crafted TIFF image.
6.9.3	CVE-2024-0727	Processing a maliciously formatted PKCS12 file may lead OpenSSL to crash leading to a potential Denial of Service attack
6.9.3	CVE-2021-45261	An Invalid Pointer vulnerability exists in GNU patch 2.7 via the another_hunk function, which causes a Denial of Service.
6.9.3	CVE-2023-6277	An out-of-memory flaw was found in libtiff. Passing a crafted tiff file to TIFFOpen() API may allow a remote attacker to cause a denial of service via a craft input with size smaller than 379 KB.
6.9.3	CVE-2024-20921	Vulnerability in the Oracle Java SE, Oracle GraalVM for JDK, Oracle GraalVM Enterprise Edition product of Oracle Java SE (component: Hotspot). Supported versions that are affected are Oracle Java SE: 8u391, 8u391-perf, 11.0.21, 17.0.9, 21.0.1; Oracle GraalVM for JDK: 17.0.9, 21.0.1; Oracle GraalVM Enterprise Edition: 20.3.12, 21.3.8 and 22.3.4. Difficult to exploit vulnerability allows unauthenticated attacker with network access via multiple protocols to compromise Oracle Java SE, Oracle GraalVM for JDK, Oracle GraalVM Enterprise Edition. Successful attacks of this vulnerability can result in unauthorized access to critical data or complete access to all Oracle Java SE, Oracle GraalVM for JDK, Oracle GraalVM Enterprise Edition accessible data. Note: This vulnerability can be exploited by using APIs in the specified Component, e.g., through a web service which supplies data to the APIs. This vulnerability also applies to Java deployments, typically in clients running sandboxed Java Web Start applications or sandboxed Java applets, that load and run untrusted code (e.g., code that comes from the internet) and rely on the Java sandbox for security. CVSS 3.1 Base Score 5.9 (Confidentiality impacts). CVSS Vector: (CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:N/A:N).

Release	CVE Fix	Description
6.9.3	CVE-2024-25269	libheif <= 1.17.6 contains a memory leak in the function JpegEncoder::Encode. This flaw allows an attacker to cause a denial of service attack.
6.9.3	CVE-2024-20952	Vulnerability in the Oracle Java SE, Oracle GraalVM for JDK, Oracle GraalVM Enterprise Edition product of Oracle Java SE (component: Security). Supported versions that are affected are Oracle Java SE: 8u391, 8u391-perf, 11.0.21, 17.0.9, 21.0.1; Oracle GraalVM for JDK: 17.0.9, 21.0.1; Oracle GraalVM Enterprise Edition: 20.3.12, 21.3.8 and 22.3.4. Difficult to exploit vulnerability allows unauthenticated attacker with network access via multiple protocols to compromise Oracle Java SE, Oracle GraalVM for JDK, Oracle GraalVM Enterprise Edition. Successful attacks of this vulnerability can result in unauthorized creation, deletion or modification access to critical data or all Oracle Java SE, Oracle GraalVM for JDK, Oracle GraalVM Enterprise Edition accessible data as well as unauthorized access to critical data or complete access to all Oracle Java SE, Oracle GraalVM for JDK, Oracle GraalVM Enterprise Edition accessible data. Note: This vulnerability applies to Java deployments, typically in clients running sandboxed Java Web Start applications or sandboxed Java applets, that load and run untrusted code (e.g., code that comes from the internet) and rely on the Java sandbox for security. This vulnerability does not apply to Java deployments, typically in servers, that load and run only trusted code (e.g., code installed by an administrator). CVSS 3.1 Base Score 7.4 (Confidentiality and Integrity impacts). CVSS Vector: (CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:N).
6.9.3	CVE-2017-9117	In LibTIFF 4.0.7, the program processes BMP images without verifying that biWidth and biHeight in the bitmap-information header match the actual input, leading to a heap-based buffer over-read in bmp2tiff.
6.9.3	CVE-2024-28757	libexpat through 2.6.1 allows an XML Entity Expansion attack when there is isolated use of external parsers (created via XML_ExternalEntityParserCreate).
6.9.3	CVE-2022-4899	A vulnerability was found in zstd v1.4.10, where an attacker can supply empty string as an argument to the command line tool to cause buffer overrun.
6.9.3	CVE-2023-39130	GNU gdb (GDB) 13.0.50.20220805-git was discovered to contain a heap buffer overflow via the function pe_as16() at /gdb/coff-pe-read.c.
6.9.3	CVE-2017-11164	In PCRE 8.41, the OP_KETRMAX feature in the match function in pcre_exec.c allows stack exhaustion (uncontrolled recursion) when processing a crafted regular expression.
6.9.3	CVE-2020-15719	libldap in certain third-party OpenLDAP packages has a certificate-validation flaw when the third-party package is asserting RFC6125 support. It considers CN even when there is a non-matching subjectAltName (SAN). This is fixed in, for example, openldap-2.4.46-10.el8 in Red Hat Enterprise Linux.
6.9.3	CVE-2021-40633	A memory leak (out-of-memory) in gif2rgb in util/gif2rgb.c in giflib 5.1.4 allows remote attackers trigger an out of memory exception or denial of service via a gif format file.
6.9.3	CVE-2022-3857	A flaw was found in libpng 1.6.38. A crafted PNG image can lead to a segmentation fault and denial of service in png_setup_paeth_row() function.
6.9.3	CVE-2021-33574	The mq_notify function in the GNU C Library (aka glibc) versions 2.32 and 2.33 has a use-after-free. It may use the notification thread attributes object (passed through its struct sigevent parameter) after it has been freed by the caller, leading to a denial of service (application crash) or possibly unspecified other impact.
6.9.3	CVE-2024-2236	A timing-based side-channel flaw was found in libgcrypt's RSA implementation. This issue may allow a remote attacker to initiate a Bleichenbacher-style attack, which can lead to the decryption of RSA ciphertexts.
6.9.3	CVE-2023-26966	libtiff 4.5.0 is vulnerable to Buffer Overflow in uv_encode() when libtiff reads a corrupted little-endian TIFF file and specifies the output to be big-endian.
6.9.3	CVE-2024-26458	Kerberos 5 (aka krb5) 1.21.2 contains a memory leak in /krb5/src/lib/rpc/pmap_rmt.c.
6.9.3	CVE-2023-7008	A vulnerability was found in systemd-resolved. This issue may allow systemd-resolved to accept records of DNSSEC-signed domains even when they have no signature, allowing man-in-the-middles (or the upstream DNS resolver) to manipulate records.

Release	CVE Fix	Description
6.9.3	CVE-2024-28834	A flaw was found in GnuTLS. The Minerva attack is a cryptographic vulnerability that exploits deterministic behavior in systems like GnuTLS, leading to side-channel leaks. In specific scenarios, such as when using the GNUTLS_PRIVKEY_FLAG_REPRODUCIBLE flag, it can result in a noticeable step in nonce size from 513 to 512 bits, exposing a potential timing side-channel.
6.9.3	CVE-2022-47008	An issue was discovered function make_tempdir, and make_tempname in bucomm.c in Binutils 2.34 thru 2.38, allows attackers to cause a denial of service due to memory leaks.
6.9.3	CVE-2024-27982	**Overview** node **is a JavaScript runtime built on Chrome's V8 JavaScript engine. Affected versions of this package are vulnerable to HTTP Request Smuggling via content length ofuscation. An attacker can smuggle an HTTP request by including a space before a Content-Length header. ## Remediation Upgrade** `node` **to version 18.20.1, 20.12.1, 21.7.2 or higher. ## References -** GitHub Commit **-** GitHub Commit **-** GitHub Commit **-** Node.js Release Notes **-** Vulnerability Report
6.9.3	CVE-2022-28506	There is a heap-buffer-overflow in GIFLIB 5.2.1 function DumpScreen2RGB() in gif2rgb.c:298:45.
6.9.3	CVE-2024-22365	linux-pam (aka Linux PAM) before 1.6.0 allows attackers to cause a denial of service (blocked login process) via mkfifo because the openat call (for protect_dir) lacks O_DIRECTORY.
6.9.3	CVE-2023-5678	Generating excessively long X9.42 DH keys or checking excessively long X9.42 DH keys or parameters may be very slow.
6.9.3	CVE-2016-2568	pkexec, when used with --user nonpriv, allows local users to escape to the parent session via a crafted TIOCSTI ioctl call, which pushes characters to the terminal's input buffer.
6.9.3	CVE-2024-0553	A vulnerability was found in GnuTLS. The response times to malformed ciphertexts in RSA-PSK ClientKeyExchange differ from the response times of ciphertexts with correct PKCS#1 v1.5 padding. This issue may allow a remote attacker to perform a timing side-channel attack in the RSA-PSK key exchange, potentially leading to the leakage of sensitive data. CVE-2024-0553 is designated as an incomplete resolution for CVE-2023-5981.
6.9.3	CVE-2011-4116	_is_safe in the File::Temp module for Perl does not properly handle symlinks.
6.9.3	CVE-2022-3219	GnuPG can be made to spin on a relatively small input by (for example) crafting a public key with thousands of signatures attached, compressed down to just a few KB.
6.9.3	CVE-2023-3618	A flaw was found in libtiff. A specially crafted tiff file can lead to a segmentation fault due to a buffer overflow in the Fax3Encode function in libtiff/tif_fax3.c, resulting in a denial of service.
6.9.3	CVE-2023-50495	NCurse v6.4-20230418 was discovered to contain a segmentation fault via the component _nc_wrap_entry().
6.9.3	CVE-2023-2953	A vulnerability was found in openldap. This security flaw causes a null pointer dereference in ber_memalloc_x() function.
6.9.3	CVE-2007-5686	initscripts in rPath Linux 1 sets insecure permissions for the /var/log/btmp file, which allows local users to obtain sensitive information regarding authentication attempts. NOTE: because sshd detects the insecure permissions and does not log certain events, this also prevents sshd from logging failed authentication attempts by remote attackers.
6.9.3	CVE-2023-34969	D-Bus before 1.15.6 sometimes allows unprivileged users to crash dbus-daemon. If a privileged user with control over the dbus-daemon is using the org.freedesktop.DBus.Monitoring interface to monitor message bus traffic, then an unprivileged user with the ability to connect to the same dbus-daemon can cause a dbus-daemon crash under some circumstances via an unreplyable message. When done on the well-known system bus, this is a denial-of-service vulnerability. The fixed versions are 1.12.28, 1.14.8, and 1.15.6.
6.9.3	CVE-2022-45703	Heap buffer overflow vulnerability in binutils readelf before 2.40 via function display_debug_section in file readelf.c.

Release	CVE Fix	Description
6.9.3	CVE-2010-4756	The glob implementation in the GNU C Library (aka glibc or libc6) allows remote authenticated users to cause a denial of service (CPU and memory consumption) via crafted glob expressions that do not match any pathnames, as demonstrated by glob expressions in STAT commands to an FTP daemon, a different vulnerability than CVE-2010-2632.
6.9.2	CVE-2009-4487	nginx 0.7.64 writes data to a log file without sanitizing non-printable characters, which might allow remote attackers to modify a window's title, or possibly execute arbitrary commands or overwrite files, via an HTTP request containing an escape sequence for a terminal emulator.
6.9.2	CVE-2019-8457	SQLite3 from 3.6.0 to and including 3.27.2 is vulnerable to heap out-of-bound read in the rtreenode() function when handling invalid rtree tables.
6.9.2	CVE-2017-7245	Stack-based buffer overflow in the pcre32_copy_substring function in pcre_get.c in libpcre1 in PCRE 8.40 allows remote attackers to cause a denial of service (WRITE of size 4) or possibly have unspecified other impact via a crafted file.
6.9.2	CVE-2019-20838	libpcre in PCRE before 8.43 allows a subject buffer over-read in JIT when UTF is disabled, and \X or \R has more than one fixed quantifier, a related issue to CVE-2019-20454.
6.9.2	CVE-2021-31239	An issue found in SQLite SQLite3 v.3.35.4 that allows a remote attacker to cause a denial of service via the appendvfs.c function.
6.9.2	CVE-2023-39615	Xmlsoft Libxml2 v2.11.0 was discovered to contain an out of bounds read via the xmlSAX2StartElement() function at /libxml2/SAX2.c. This vulnerability allows attackers to cause a Denial of Service (DoS) via supplying a crafted XML file. NOTE: the vendor's position is that the product does not support the legacy SAX1 interface with custom callbacks; there is a crash even without crafted input.
6.9.2	CVE-2021-36085	The CIL compiler in SELinux 3.2 has a use-after-free in __cil_verify_classperms (called from __verify_map_perm_classperms and hashtab_map).
6.9.2	CVE-2023-5981	A vulnerability was found that the response times to malformed ciphertexts in RSA-PSK ClientKeyExchange differ from response times of ciphertexts with correct PKCS#1 v1.5 padding.
6.9.2	CVE-2023-39804	*This vulnerability has not been analyzed by NVD yet.*
6.9.2	CVE-2016-3709	Possible cross-site scripting vulnerability in libxml after commit 960f0e2.
6.9.2	CVE-2023-36054	lib/kadm5/kadm_rpc_xdr.c in MIT Kerberos 5 (aka krb5) before 1.20.2 and 1.21.x before 1.21.1 frees an uninitialized pointer. A remote authenticated user can trigger a kadmind crash. This occurs because _xdr_kadm5_principal_ent_rec does not validate the relationship between n_key_data and the key_data array count.
6.9.2	CVE-2023-4911	A buffer overflow was discovered in the GNU C Library's dynamic loader ld.so while processing the GLIBC_TUNABLES environment variable. This issue could allow a local attacker to use maliciously crafted GLIBC_TUNABLES environment variables when launching binaries with SUID permission to execute code with elevated privileges.
6.9.2	CVE-2023-3817	Checking excessively long DH keys or parameters may be very slow.
6.9.2	CVE-2022-35737	SQLite 1.0.12 through 3.39.x before 3.39.2 sometimes allows an array-bounds overflow if billions of bytes are used in a string argument to a C API.
6.9.2	CVE-2021-36690	A segmentation fault can occur in the sqlite3.exe command-line component of SQLite 3.36.0 via the idxGetTableInfo function when there is a crafted SQL query. NOTE: the vendor disputes the relevance of this report because a sqlite3.exe user already has full privileges (e.g., is intentionally allowed to execute commands). This report does NOT imply any problem in the SQLite library.

Release	CVE Fix	Description
6.9.2	CVE-2022-2309	NULL Pointer Dereference allows attackers to cause a denial of service (or application crash). This only applies when lxml is used together with libxml2 2.9.10 through 2.9.14. libxml2 2.9.9 and earlier are not affected. It allows triggering crashes through forged input data, given a vulnerable code sequence in the application. The vulnerability is caused by the iterwalk function (also used by the canonicalize function). Such code shouldn't be in wide-spread use, given that parsing + iterwalk would usually be replaced with the more efficient iterparse function. However, an XML converter that serialises to C14N would also be vulnerable, for example, and there are legitimate use cases for this code sequence. If untrusted input is received (also remotely) and processed via iterwalk function, a crash can be triggered.
6.9.2	CVE-2021-36086	The CIL compiler in SELinux 3.2 has a use-after-free in cil_reset_classpermission (called from cil_reset_classperms_set and cil_reset_classperms_list).
6.9.2	CVE-2017-16231	In PCRE 8.41, after compiling, a pcretest load test PoC produces a crash overflow in the function match() in pcre_exec.c because of a self-recursive call. NOTE: third parties dispute the relevance of this report, noting that there are options that can be used to limit the amount of stack that is used.
6.9.2	CVE-2023-29491	ncurses before 6.4 20230408, when used by a setuid application, allows local users to trigger security-relevant memory corruption via malformed data in a terminfo database file that is found in $HOME/.terminfo or reached via the TERMINFO or TERM environment variable.
6.9.2	CVE-2023-3446	Checking excessively long DH keys or parameters may be very slow.
6.9.2	CVE-2021-36087	The CIL compiler in SELinux 3.2 has a heap-based buffer over-read in ebitmap_match_any (called indirectly from cil_check_neverallow). This occurs because there is sometimes a lack of checks for invalid statements in an optional block.
6.9.2	CVE-2020-13529	An exploitable denial-of-service vulnerability exists in Systemd 245. A specially crafted DHCP FORCERENEW packet can cause a server running the DHCP client to be vulnerable to a DHCP ACK spoofing attack. An attacker can forge a pair of FORCERENEW and DCHP ACK packets to reconfigure the server.
6.9.2	CVE-2023-46218	This flaw allows a malicious HTTP server to set "super cookies" in curl that are then passed back to more origins than what is otherwise allowed or possible. This allows a site to set cookies that then would get sent to different and unrelated sites and domains.
6.9.2	CVE-2023-4806	A flaw was found in glibc. In an extremely rare situation, the getaddrinfo function may access memory that has been freed, resulting in an application crash. This issue is only exploitable when a NSS module implements only the *nss_gethostbyname2_r and nss*_getcanonname_r hooks without implementing the *nss**_gethostbyname3_r hook. The resolved name should return a large number of IPv6 and IPv4, and the call to the getaddrinfo function should have the AF_INET6 address family with AI_CANONNAME, AI_ALL and AI_V4MAPPED as flags.
6.9.2	CVE-2017-7246	Stack-based buffer overflow in the pcre32_copy_substring function in pcre_get.c in libpcre1 in PCRE 8.40 allows remote attackers to cause a denial of service (WRITE of size 268) or possibly have unspecified other impact via a crafted file.
6.9.2	CVE-2023-4813	A flaw was found in glibc. In an uncommon situation, the gaih_inet function may use memory that has been freed, resulting in an application crash. This issue is only exploitable when the getaddrinfo function is called and the hosts database in /etc/nsswitch.conf is configured with SUCCESS=continue or SUCCESS=merge.
6.9.2	CVE-2022-48303	GNU Tar through 1.34 has a one-byte out-of-bounds read that results in use of uninitialized memory for a conditional jump. Exploitation to change the flow of control has not been demonstrated. The issue occurs in from_header in list.c via a V7 archive in which mtime has approximately 11 whitespace characters.
6.9.2	CVE-2013-0337	The default configuration of nginx, possibly 1.3.13 and earlier, uses world-readable permissions for the (1) access.log and (2) error.log files, which allows local users to obtain sensitive information by reading the files.
6.9.2	CVE-2022-1304	An out-of-bounds read/write vulnerability was found in e2fsprogs 1.46.5. This issue leads to a segmentation fault and possibly arbitrary code execution via a specially crafted filesystem.

Release	CVE Fix	Description
6.9.2	CVE-2024-20932	Vulnerability in the Oracle Java SE, Oracle GraalVM for JDK, Oracle GraalVM Enterprise Edition product of Oracle Java SE (component: Security). Supported versions that are affected are Oracle Java SE: 17.0.9; Oracle GraalVM for JDK: 17.0.9; Oracle GraalVM Enterprise Edition: 21.3.8 and 22.3.4. Easily exploitable vulnerability allows unauthenticated attacker with network access via multiple protocols to compromise Oracle Java SE, Oracle GraalVM for JDK, Oracle GraalVM Enterprise Edition. Successful attacks of this vulnerability can result in unauthorized creation, deletion or modification access to critical data or all Oracle Java SE, Oracle GraalVM for JDK, Oracle GraalVM Enterprise Edition accessible data. Note: This vulnerability applies to Java deployments, typically in clients running sandboxed Java Web Start applications or sandboxed Java applets, that load and run untrusted code (e.g., code that comes from the internet) and rely on the Java sandbox for security. This vulnerability does not apply to Java deployments, typically in servers, that load and run only trusted code (e.g., code installed by an administrator). CVSS 3.1 Base Score 7.5 (Integrity impacts). CVSS Vector: (CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:H/A:N).
6.9.2	CVE-2023-47038	A vulnerability was found in perl. This issue occurs when a crafted regular expression is compiled by perl, which can allow an attacker controlled byte buffer overflow in a heap allocated buffer.
6.9.2	CVE-2015-9019	In libxslt 1.1.29 and earlier, the EXSLT math.random function was not initialized with a random seed during startup, which could cause usage of this function to produce predictable outputs.
6.9.2	CVE-2021-36084	The CIL compiler in SELinux 3.2 has a use-after-free in __cil_verify_classperms (called from __cil_verify_classpermission and __cil_pre_verify_helper).
6.9.2	CVE-2024-0567	A vulnerability was found in GnuTLS, where a cockpit (which uses gnuTLS) rejects a certificate chain with distributed trust. This issue occurs when validating a certificate chain with cockpit-certificate-ensure. This flaw allows an unauthenticated, remote client or attacker to initiate a denial of service attack.
6.9.2	CVE-2023-44487	The HTTP/2 protocol allows a denial of service (server resource consumption) because request cancellation can reset many streams quickly, as exploited in the wild in August through October 2023.
6.9.2	CVE-2020-16156	CPAN 2.28 allows Signature Verification Bypass.
6.9.2	CVE-2023-46219	When saving HSTS data to an excessively long file name, curl could end up removing all contents, making subsequent requests using that file unaware of the HSTS status they should otherwise use.
6.9.2	CVE-2023-45322	libxml2 through 2.11.5 has a use-after-free that can only occur after a certain memory allocation fails. This occurs in xmlUnlinkNode in tree.c. NOTE: the vendor's position is "I don't think these issues are critical enough to warrant a CVE ID ... because an attacker typically can't control when memory allocations fail."
6.9.2	CVE-2021-33560	Libgcrypt before 1.8.8 and 1.9.x before 1.9.3 mishandles ElGamal encryption because it lacks exponent blinding to address a side-channel attack against mpi_powm, and the window size is not chosen appropriately. This, for example, affects use of ElGamal in OpenPGP.
6.9.1	CVE-2024-21626	AWS is aware of CVE-2024-21626, an issue affecting the runc component of several open source container management systems. Under certain conditions, an actor could leverage a specially crafted container or container configuration to access files or directories outside the container's file system namespace. An updated version of runc that addresses the issue is available for Amazon Linux 1 (runc-1.1.11-1.0.amzn1), Amazon Linux 2 (runc-1.1.11-1.amzn2) and for Amazon Linux 2023 (runc-1.1.11-1.amzn2023). AWS recommends that customers using runc or any container-related software apply those updates or a newer version.
6.9.1	CVE-2023-6606	An out-of-bounds read vulnerability was found in smbCalcSize in fs/smb/client/netmisc.c in the Linux Kernel. This issue could allow a local attacker to crash the system or leak internal kernel information.
6.9.1	CVE-2023-49569	A path traversal vulnerability was discovered in go-git versions prior to v5.11. This vulnerability allows an attacker to create and amend files across the filesystem. In the worse case scenario, remote code execution could be achieved.Applications are only affected if they are using the ChrootOS "https://pkg.go.dev/github.com/go-git/go-billy/v5/osfs#ChrootOS" , which is th default when using "Plain" versions of Open and Clone funcs (e.g. PlainClone). Applications using BoundOS "https://pkg.go.dev/github.com/go-git/go-billy/v5/osfs#BoundOS" or in-memory filesystems are not affected by this issue. This is a go-git implementation issue and does not affect the upstream git cli.

Release	CVE Fix	Description
6.9.1	CVE-2023-50447	Pillow through 10.1.0 allows PIL.ImageMath.eval Arbitrary Code Execution via the environment parameter, a different vulnerability than CVE-2022-22817 (which was about the expression parameter).
6.9.1	CVE-2023-6817	A use-after-free vulnerability in the Linux kernel's netfilter nf_tables component can be exploited to achieve local privilege escalation. The function nft_pipapo_walk did not skip inactive elements during set walk which could lead double deactivations of PIPAPO (Pile Packet Policies) elements, leading to use-after-free. We recommend upgrading past commit 317eb9685095678f2c9f5a8189de698c5354316a.
6.9.1	CVE-2024-0193	A use-after-free flaw was found in the netfilter subsystem of the Linux kernel. If the catchall element is garbage-collected when the pipapo set is removed, the element can be deactivated twice. This can cause a use-after-free issue on an NFT_CHAIN object or NFT_OBJECT object, allowing a local unprivileged user with CAP_NET_ADMIN capability to escalate their privileges on the system.
6.9.1	CVE-2023-6932	A use-after-free vulnerability in the Linux kernel's ipv4 igmp component can be exploited to achieve local privilege escalation. A race condition can be exploited to cause a timer be mistakenly registered on a RCU read locked object which is freed by another thread. We recommend upgrading past commit e2b706c691905fe78468c361aaabc719d0a496f1.
6.9.1	CVE-2024-1086	A use-after-free vulnerability in the Linux kernel's netfilter nf_tables component can be exploited to achieve local privilege escalation. The nft_verdict_init() function allows positive values as drop error within the hook verdict, and hence the nf_hook_slow() function can cause a double free vulnerability when NF_DROP is issued with a drop error which resembles NF_ACCEPT. We recommend upgrading past commit f342de4e2f33e0e39165d8639387aa6c19dff660.
6.9.1	CVE-2023-6931	A heap out-of-bounds write vulnerability in the Linux kernel's Performance Events system component can be exploited to achieve local privilege escalation. A perf_event's read_size can overflow, leading to an heap out-of-bounds increment or write in perf_read_group(). We recommend upgrading past commit 382c27f4ed28f803b1f1473ac2d8db0afc795a1b.
6.9.1	CVE-2023-7104	A vulnerability was found in SQLite SQLite3 up to 3.43.0 and classified as critical. This issue affects the function sessionReadRecord of the file ext/session/sqlite3session.c of the component make alltest Handler. The manipulation leads to heap-based buffer overflow. It is recommended to apply a patch to fix this issue. The associated identifier of this vulnerability is VDB-248999.NOTE https://sqlite.org/forum/forumpost/5bcbf4571cNOTE Fixed by https://sqlite.org/src/info/0e4e7a05c4204b47
6.9.1	CVE-2023-6040	An out-of-bounds access vulnerability involving netfilter was reported and fixed as f1082dd31fe4 (netfilter nf_tables Reject tables of unsupported family); While creating a new netfilter table, lack of a safeguard against invalid nf_tables family (pf) values within `nf_tables_newtable` function enables an attacker to achieve out-of-bounds access.
6.9.1	CVE-2023-39198	A race condition leading to a use-after-free issue was found in the QXL driver in the Linux kernel.
6.9.1	CVE-2023-52340	When a router encounters an IPv6 packet too big to transmit to the next-hop, it returns an ICMP6 "Packet Too Big" (PTB) message to the sender. The sender caches this updated Maximum Transmission Unit (MTU) so it knows not to exceed this value when subsequently routing to the same host. In Linux kernels prior to 6.3, garbage collection is run on the IPv6 Destination Route Cache if the number of entries exceeds a threshold when adding the destination to the cache. This garbage collection examines every entry in the cache while holding a lock. In these affected kernel versions, a flood of the IPv6 ICMP6 PTB messages could cause high lock contention and increased CPU usage, leading to a Denial-of-Service. The fix backports the garbage collection improvements from Linux kernel 6.3 by bringing the IPv6 code closer to the IPv4 code, which does not have this issue. Patch https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit?id=af6d10345ca76670c1b7c37799f0d5576ccef277
6.9.1	CVE-2023-39325	The HTTP/2 protocol allows a denial of service (server resource consumption) because request cancellation can reset many streams quickly, as exploited in the wild in August through October 2023.
6.9.1	CVE-2022-32149	An attacker may cause a denial of service by crafting an Accept-Language header which ParseAcceptLanguage will take significant time to parse.
6.9.1	CVE-2020-36773	Artifex Ghostscript before 9.53.0 has an out-of-bounds write and use-after-free in devices/vector/gdevtxtw.c (for txtwrite) because a single character code in a PDF document can map to more than one Unicode code point (e.g., for a ligature).

Release	CVE Fix	Description
6.9.1	CVE-2024-20932	Vulnerability in the Oracle Java SE, Oracle GraalVM for JDK, Oracle GraalVM Enterprise Edition product of Oracle Java SE (component Security). Supported versions that are affected are Oracle Java SE 17.0.9; Oracle GraalVM for JDK 17.0.9; Oracle GraalVM Enterprise Edition 21.3.8 and 22.3.4. Easily exploitable vulnerability allows unauthenticated attacker with network access via multiple protocols to compromise Oracle Java SE, Oracle GraalVM for JDK, Oracle GraalVM Enterprise Edition. Successful attacks of this vulnerability can result in unauthorized creation, deletion or modification access to critical data or all Oracle Java SE, Oracle GraalVM for JDK, Oracle GraalVM Enterprise Edition accessible data. Note This vulnerability applies to Java deployments, typically in clients running sandboxed Java Web Start applications or sandboxed Java applets, that load and run untrusted code (e.g., code that comes from the internet) and rely on the Java sandbox for security. This vulnerability does not apply to Java deployments, typically in servers, that load and run only trusted code (e.g., code installed by an administrator).
6.9.1	CVE-2023-46751	An issue was discovered in the function gdev_prn_open_printer_seekable() in Artifex Ghostscript through 10.02.0 allows remote attackers to crash the application via a dangling pointer.
6.9.1	CVE-2023-51764	Postfix through 3.8.4 allows SMTP smuggling unless configured with smtpd_data_restrictions=reject_unauth_pipelining and smtpd_discard_ehlo_keywords=chunking (or certain other options that exist in recent versions). Remote attackers can use a published exploitation technique to inject e-mail messages with a spoofed MAIL FROM address, allowing bypass of an SPF protection mechanism. This occurs because Postfix supports . but some other popular e-mail servers do not. To prevent attack variants (by always disallowing without), a different solution is required the smtpd_forbid_bare_newline=yes option with a Postfix minimum version of 3.5.23, 3.6.13, 3.7.9, 3.8.4, or 3.9.
6.9.1	CVE-2023-47108	OpenTelemetry-Go Contrib is a collection of third-party packages for OpenTelemetry-Go. Prior to version 0.46.0, the grpc Unary Server Interceptor out of the box adds labels `net.peer.sock.addr` and `net.peer.sock.port` that have unbound cardinality. It leads to the server's potential memory exhaustion when many malicious requests are sent. An attacker can easily flood the peer address and port for requests. Version 0.46.0 contains a fix for this issue. As a workaround to stop being affected, a view removing the attributes can be used. The other possibility is to disable grpc metrics instrumentation by passing `otelgrpc.WithMeterProvider` option with `noop.NewMeterProvider`.
6.9.1	CVE-2023-29406	The HTTP/1 client does not fully validate the contents of the Host header. A maliciously crafted Host header can inject additional headers or entire requests. With fix, the HTTP/1 client now refuses to send requests containing an invalid Request.Host or Request.URL.Host value.
6.9.1	CVE-2024-23849	In rds_recv_track_latency in net/rds/af_rds.c in the Linux kernel through 6.7.1, there is an off-by-one error for an RDS_MSG_RX_DGRAM_TRACE_MAX comparison, resulting in out-of-bounds access.
6.9.1	CVE-2024-20952	Vulnerability in the Oracle Java SE, Oracle GraalVM for JDK, Oracle GraalVM Enterprise Edition product of Oracle Java SE (component Security). Supported versions that are affected are Oracle Java SE 8u391, 8u391-perf, 11.0.21, 17.0.9, 21.0.1; Oracle GraalVM for JDK 17.0.9, 21.0.1; Oracle GraalVM Enterprise Edition 20.3.12, 21.3.8 and 22.3.4. Difficult to exploit vulnerability allows unauthenticated attacker with network access via multiple protocols to compromise Oracle Java SE, Oracle GraalVM for JDK, Oracle GraalVM Enterprise Edition. Successful attacks of this vulnerability can result in unauthorized creation, deletion or modification access to critical data or all Oracle Java SE, Oracle GraalVM for JDK, Oracle GraalVM Enterprise Edition accessible data as well as unauthorized access to critical data or complete access to all Oracle Java SE, Oracle GraalVM for JDK, Oracle GraalVM Enterprise Edition accessible data. Note This vulnerability applies to Java deployments, typically in clients running.
6.9.1	CVE-2024-20918	A vulnerability that allows an attacker to execute arbitrary java code from the javascript engine even though the option "--no-java" was set.
6.9.1	CVE-2024-0565	An out-of-bounds memory read flaw was found in receive_encrypted_standard in fs/smb/client/smb2ops.c in the SMB Client sub-component in the Linux Kernel. This issue occurs due to integer underflow on the memcpy length, leading to a denial of service.
6.9.1	CVE-2023-6531	A use-after-free flaw was found in the Linux Kernel due to a race problem in the unix garbage collector's deletion of SKB races with unix_stream_read_generic() on the socket that the SKB is queued on.
6.9.1	CVE-2023-42465	Sudo before 1.9.15 might allow row hammer attacks (for authentication bypass or privilege escalation) because application logic sometimes is based on not equaling an error value (instead of equaling a success value), and because the values do not resist flips of a single bit.

Release	CVE Fix	Description
6.9.1	CVE-2024-0646	An out-of-bounds memory write flaw was found in the Linux kernel's Transport Layer Security functionality in how a user calls a function splice with a ktls socket as the destination. This flaw allows a local user to crash or potentially escalate their privileges on the system.
6.9.1	CVE-2020-19187	Buffer Overflow vulnerability in fmt_entry function in progs/dump_entry.c:1100 in ncurses 6.1 allows remote attackers to cause a denial of service via crafted command.
6.9.1	CVE-2020-19185	Buffer Overflow vulnerability in one_one_mapping function in progs/dump_entry.c:1373 in ncurses 6.1 allows remote attackers to cause a denial of service via crafted command.
6.9.1	CVE-2020-19186	Buffer Overflow vulnerability in _nc_find_entry function in tinfo/comp_hash.c:66 in ncurses 6.1 allows remote attackers to cause a denial of service via crafted command.
6.9.1	CVE-2016-0775	Buffer overflow in the ImagingFliDecode function in libImaging/FliDecode.c in Pillow before 3.1.1 allows remote attackers to cause a denial of service (crash) via a crafted FLI file.
6.9.1	CVE-2020-19188	Buffer Overflow vulnerability in fmt_entry function in progs/dump_entry.c:1116 in ncurses 6.1 allows remote attackers to cause a denial of service via crafted command.
6.9.1	CVE-2020-19189	Buffer Overflow vulnerability in postprocess_terminfo function in tinfo/parse_entry.c:997 in ncurses 6.1 allows remote attackers to cause a denial of service via crafted command.
6.9.1	CVE-2023-6915	A Null pointer dereference problem was found in ida_free in lib/idr.c in the Linux Kernel. This issue may allow an attacker using this library to cause a denial of service problem due to a missing check at a function return.
6.9.1	CVE-2023-46838	A flaw has been found in Xen. An unprivileged guest can cause Denial of Service (DoS) of the host by sending network packets to the backend, causing the backend to crash.
6.9.1	CVE-2023-46218	This flaw allows a malicious HTTP server to set "super cookies" in curl that are then passed back to more origins than what is otherwise allowed or possible. This allows a site to set cookies that then would get sent to different and unrelated sites and domains.It could do this by exploiting a mixed case flaw in curl's function that verifies a given cookie domain against the Public Suffix List (PSL). For example a cookie could be set with domain=co.UK when the URL used a lowercase hostname curl.co.uk, even though co.uk is listed as a PSL domain.
6.9.1	CVE-2024-0584	A use-after-free issue was found in igmp_start_timer in net/ipv4/igmp.c in the network sub-component in the Linux Kernel. This flaw allows a local user to observe a refcnt use-after-free issue when receiving an igmp query packet, leading to a kernel information leak.
6.9.1	CVE-2023-38473	A reachable assertion was found in avahi_alternative_host_name.
6.9.1	CVE-2023-43786	A vulnerability was found in libX11 due to an infinite loop within the PutSubImage() function. This flaw allows a local user to consume all available system resources and cause a denial of service condition.
6.9.1	CVE-2023-34969	D-Bus before 1.15.6 sometimes allows unprivileged users to crash dbus-daemon. If a privileged user with control over the dbus-daemon is using the org.freedesktop.DBus.Monitoring interface to monitor message bus traffic, then an unprivileged user with the ability to connect to the same dbus-daemon can cause a dbus-daemon crash under some circumstances via an unreplyable message. When done on the well-known system bus, this is a denial-of-service vulnerability. The fixed versions are 1.12.28, 1.14.8, and 1.15.6.
6.9.1	CVE-2023-38472	A reachable assertion was found in avahi_rdata_parse.
6.9.1	CVE-2023-25153	Containerd is an open source container runtime. Before versions 1.6.18 and 1.5.18, when importing an OCI image, there was no limit on the number of bytes read for certain files. A maliciously crafted image with a large file where a limit was not applied could cause a denial of service. This bug has been fixed in containerd 1.6.18 and 1.5.18. Users should update to these versions to resolve the issue. As a workaround, ensure that only trusted images are used and that only trusted users have permissions to import images.

Release	CVE Fix	Description
6.9.1	CVE-2023-30630	Dmidecode before 3.5 allows -dump-bin to overwrite a local file. This has security relevance because, for example, execution of Dmidecode via Sudo is plausible.
6.9.1	CVE-2024-0607	netfilter nf_tables fix pointer math issue in nft_byteorder_eval().
6.9.1	CVE-2023-5388	It was discovered that the numerical library used in NSS for RSA cryptography leaks information whether high order bits of the RSA decryption result are zero. This information can be used to mount a Bleichenbacher or Manger like attack against all RSA decryption operations. As the leak happens before any padding operations, it affects all padding modes PKCS#1 v1.5, OAEP, and RSASVP. Both API level calls and TLS server operation are affected.
6.9.1	CVE-2022-48566	An issue was discovered in compare_digest in Lib/hmac.py in Python through 3.9.1. Constant-time-defeating optimisations were possible in the accumulator variable in hmac.compare_digest.
6.9.1	CVE-2024-20921	Loop optimizations are not correct when induction variable overflows.
6.9.1	CVE-2023-6135	Multiple NSS NIST curves were susceptible to a side-channel attack known as "Minerva". This attack could potentially allow an attacker to recover the private key. This vulnerability affects Firefox < 121.
6.9.1	CVE-2023-48795	AWS is aware of CVE-2023-48795, also known as Terrapin, which is found in the SSH protocol and affects SSH channel integrity. A protocol extension has been introduced by OpenSSH which needs to be applied to both the client and the server in order to address this issue. We recommend customers update to the latest version of SSH.
6.9.1	CVE-2024-20919	With carefully crafted custom bytecodes, arbitrary unverified bytecodes could be executed.
6.9.1	CVE-2022-41721	A request smuggling attack is possible when using MaxBytesHandler. When using MaxBytesHandler, the body of an HTTP request is not fully consumed. When the server attempts to read HTTP2 frames from the connection, it will instead be reading the body of the HTTP request, which could be attacker-manipulated to represent arbitrary HTTP2 requests.
6.9.1	CVE-2022-47007	An issue was discovered function stab_demangle_v3_arg in stabs.c in Binutils 2.34 thru 2.38, allows attackers to cause a denial of service due to memory leaks.
6.9.1	CVE-2022-47008	An issue was discovered function make_tempdir, and make_tempname in bucomm.c in Binutils 2.34 thru 2.38, allows attackers to cause a denial of service due to memory leaks.
6.9.1	CVE-2024-22667	Vim before 9.0.2142 has a stack-based buffer overflow because did_set_langmap in map.c calls sprintf to write to the error buffer that is passed down to the option callback functions.
6.9.1	CVE-2022-47010	An issue was discovered function pr_function_type in prdbg.c in Binutils 2.34 thru 2.38, allows attackers to cause a denial of service due to memory leaks.
6.9.1	CVE-2021-46174	Heap-based Buffer Overflow in function bfd_getl32 in Binutils objdump 3.37.
6.9.1	CVE-2016-9189	Pillow before 3.3.2 allows context-dependent attackers to obtain sensitive information by using the "crafted image file" approach, related to an "Integer Overflow" issue affecting the Image.core.map_buffer in map.c component.
6.9.1	CVE-2022-35205	An issue was discovered in Binutils readelf 2.38.50, reachable assertion failure in function display_debug_names allows attackers to cause a denial of service.
6.9.1	CVE-2023-46862	An issue was discovered in the Linux kernel through 6.5.9. During a race with SQ thread exit, an io_uring/fdinfo.c io_uring_show_fdinfo NULL pointer dereference can occur.
6.9.1	CVE-2020-19724	A memory consumption issue in get_data function in binutils/nm.c in GNU nm before 2.34 allows attackers to cause a denial of service via crafted command.
6.9.1	CVE-2022-48064	GNU Binutils before 2.40 was discovered to contain an excessive memory consumption vulnerability via the function bfd_dwarf2_find_nearest_line_with_alt at dwarf2.c. The attacker could supply a crafted ELF file and cause a DNS attack.

Release	CVE Fix	Description
6.9.1	CVE-2023-3978	Text nodes not in the HTML namespace are incorrectly literally rendered, causing text which should be escaped to not be. This could lead to an XSS attack.
6.9.1	CVE-2019-17595	There is a heap-based buffer over-read in the fmt_entry function in tinfo/comp_hash.c in the terminfo library in ncurses before 6.1-20191012.
6.9.1	CVE-2024-22195	Jinja is an extensible templating engine. Special placeholders in the template allow writing code similar to Python syntax. It is possible to inject arbitrary HTML attributes into the rendered HTML template, potentially leading to Cross-Site Scripting (XSS). The Jinja `xmlattr` filter can be abused to inject arbitrary HTML attribute keys and values, bypassing the auto escaping mechanism and potentially leading to XSS. It may also be possible to bypass attribute validation checks if they are blacklist-based.
6.9.1	CVE-2023-25173	containerd is an open source container runtime. A bug was found in containerd prior to versions 1.6.18 and 1.5.18 where supplementary groups are not set up properly inside a container. If an attacker has direct access to a container and manipulates their supplementary group access, they may be able to use supplementary group access to bypass primary group restrictions in some cases, potentially gaining access to sensitive information or gaining the ability to execute code in that container. Downstream applications that use the containerd client library may be affected as well. This bug has been fixed in containerd v1.6.18 and v.1.5.18. Users should update to these versions and recreate containers to resolve this issue. Users who rely on a downstream application that uses containerd's client library should check that application for a separate advisory and instructions. As a workaround, ensure that the `"USER $USERNAME"` Dockerfile instruction is not used. Instead, set the container entrypoint to a value similar to `ENTRYPOINT ["su", "-", "user"]` to allow `su` to properly set up supplementary groups.
6.9.1	CVE-2019-17594	There is a heap-based buffer over-read in the _nc_find_entry function in tinfo/comp_hash.c in the terminfo library in ncurses before 6.1-20191012.
6.9.1	CVE-2023-29409	Extremely large RSA keys in certificate chains can cause a client/server to expend significant CPU time verifying signatures. With fix, the size of RSA keys transmitted during handshakes is restricted to <= 8192 bits. Based on a survey of publicly trusted RSA keys, there are currently only three certificates in circulation with keys larger than this, and all three appear to be test certificates that are not actively deployed. It is possible there are larger keys in use in private PKIs, but we target the web PKI, so causing breakage here in the interests of increasing the default safety of users of crypto/tls seems reasonable.
6.9.1	CVE-2023-50495	NCurse v6.4-20230418 was discovered to contain a segmentation fault via the component _nc_wrap_entry().
6.9.1	CVE-2023-1972	Potential heap based buffer overflow found in _bfd_elf_slurp_version_tables() in bfd/elf.c.
6.9.1	CVE-2024-20945	Crypto key may be leaked via debug logging in some cases.
6.9.1	CVE-2023-39326	A malicious HTTP sender can use chunk extensions to cause a receiver reading from a request or response body to read many more bytes from the network than are in the body. A malicious HTTP client can further exploit this to cause a server to automatically read a large amount of data (up to about 1GiB) when a handler fails to read the entire body of a request. Chunk extensions are a little-used HTTP feature which permit including additional metadata in a request or response body sent using the chunked encoding. The net/http chunked encoding reader discards this metadata. A sender can exploit this by inserting a large metadata segment with each byte transferred. The chunk reader now produces an error if the ratio of real body to encoded bytes grows too small.
6.9.1	CVE-2023-6121	An out-of-bounds read vulnerability was found in the NVMe-oF/TCP subsystem in the Linux kernel. This flaw allows a remote attacker to send a crafted TCP packet, triggering a heap-based buffer overflow that results in kmalloc data to be printed (and potentially leaked) to the kernel ring buffer (dmesg).

Release	CVE Fix	Description
6.9.1	CVE-2023-45803	urllib3 is a user-friendly HTTP client library for Python. urllib3 previously wouldn't remove the HTTP request body when an HTTP redirect response using status 301, 302, or 303 after the request had its method changed from one that could accept a request body (like `POST`) to `GET` as is required by HTTP RFCs. Although this behavior is not specified in the section for redirects, it can be inferred by piecing together information from different sections and we have observed the behavior in other major HTTP client implementations like curl and web browsers. Because the vulnerability requires a previously trusted service to become compromised in order to have an impact on confidentiality we believe the exploitability of this vulnerability is low. Additionally, many users aren't putting sensitive data in HTTP request bodies, if this is the case then this vulnerability isn't exploitable. Both of the following conditions must be true to be affected by this vulnerability 1. Using urllib3 and submitting sensitive information in the HTTP request body (such as form data or JSON) and 2. The origin service is compromised and starts redirecting using 301, 302, or 303 to a malicious peer or the redirected-to service becomes compromised. This issue has been addressed in versions 1.26.18 and 2.0.7 and users are advised to update to resolve this issue. Users unable to update should disable redirects for services that aren't expecting to respond with redirects with `redirects=False` and disable automatic redirects with `redirects=False` and handle 301, 302, and 303 redirects manually by stripping the HTTP request body.
6.9.1	CVE-2020-19190	Buffer Overflow vulnerability in _nc_find_entry in tinfo/comp_hash.c:70 in ncurses 6.1 allows remote attackers to cause a denial of service via crafted command.
6.9.1	CVE-2023-46219	When saving HSTS data to an excessively long file name, curl could end up removing all contents, making subsequent requests using that file unaware of the HSTS status they should otherwise use.
6.9.1	CVE-2023-48706	Vim is a UNIX editor that, prior to version 9.0.2121, has a heap-use-after-free vulnerability. When executing a `:s` command for the very first time and using a sub-replace-special atom inside the substitution part, it is possible that the recursive `:s` call causes free-ing of memory which may later then be accessed by the initial `:s` command. The user must intentionally execute the payload and the whole process is a bit tricky to do since it seems to work only reliably for the very first 😳 command. It may also cause a crash of Vim. Version 9.0.2121 contains a fix for this issue.
6.9.1	CVE-2022-41409	Integer overflow vulnerability in pcre2test before 10.41 allows attackers to cause a denial of service or other unspecified impacts via negative input.
6.9.1	CVE-2024-22365	A vulnerability was found in Linux PAM. An unprivileged user that is not yet in a corresponding mount namespace with ~/tmp mounted as a polyinstantiated dir can place a FIFO there, and a subsequent attempt to login as this user with `pam_namespace` configured will cause the `openat()` in `protect_dir()` to block the attempt, causing a local denial of service.
6.9.1	CVE-2023-46316	In buc Traceroute 2.0.12 through 2.1.2 before 2.1.3, the wrapper scripts do not properly parse command lines.
6.9.1	CVE-2024-20925	There are several integer overflows in the media handling.
6.9.1	CVE-2024-20923	Missing validation may cause unexpected issues.
6.9.1	CVE-2023-39804	It was discovered that tar incorrectly handled extended attributes in PAX archives. An attacker could supply a specially crafted file and cause tar to crash, resulting in a denial of service.
6.9.1	CVE-2024-20922	Vulnerability in the Oracle Java SE, Oracle GraalVM Enterprise Edition product of Oracle Java SE (component JavaFX). Supported versions that are affected are Oracle Java SE 8u391; Oracle GraalVM Enterprise Edition 20.3.12 and 21.3.8. Difficult to exploit vulnerability allows unauthenticated attacker with logon to the infrastructure where Oracle Java SE, Oracle GraalVM Enterprise Edition executes to compromise Oracle Java SE, Oracle GraalVM Enterprise Edition. Successful attacks require human interaction from a person other than the attacker. Successful attacks of this vulnerability can result in unauthorized update, insert or delete access to some of Oracle Java SE, Oracle GraalVM Enterprise Edition accessible data. Note This vulnerability applies to Java deployments, typically in clients running sandboxed Java Web Start applications or sandboxed Java applets, that load and run untrusted code (e.g., code that comes from the internet) and rely on the Java sandbox for security. This vulnerability does not apply to Java deployments, typically in servers, that load and run only trusted code (e.g., code installed by an administrator).
6.9.1	CVE-2022-40090	An issue was discovered in function TIFFReadDirectory libtiff before 4.4.0 allows attackers to cause a denial of service via crafted TIFF file.

Release	CVE Fix	Description
6.9.1	CVE-2021-38115	read_header_tga in gd_tga.c in the GD Graphics Library (aka LibGD) through 2.3.2 allows remote attackers to cause a denial of service (out-of-bounds read) via a crafted TGA file.
6.9.1	CVE-2024-0985	Late privilege drop in REFRESH MATERIALIZED VIEW CONCURRENTLY in PostgreSQL allows an object creator to execute arbitrary SQL functions as the command issuer. The command intends to run SQL functions as the owner of the materialized view, enabling safe refresh of untrusted materialized views. The victim is a superuser or member of one of the attacker's roles. The attack requires luring the victim into running REFRESH MATERIALIZED VIEW CONCURRENTLY on the attacker's materialized view. As part of exploiting this vulnerability, the attacker creates functions that use CREATE RULE to convert the internally-built temporary table to a view. Versions before PostgreSQL 15.6, 14.11, 13.14, and 12.18 are affected. The only known exploit does not work in PostgreSQL 16 and later. For defense in depth, PostgreSQL 16.2 adds the protections that older branches are using to fix their vulnerability.
6.9.1	CVE-2022-1056	Out-of-bounds Read error in tiffcrop in libtiff 4.3.0 allows attackers to cause a denial-of-service via a crafted tiff file. For users that compile libtiff from sources, the fix is available with commit 46dc8fcd.
6.9.1	CVE-2023-30775	A vulnerability was found in the libtiff library. This security flaw causes a heap buffer overflow in extractContigSamples32bits, tiffcrop.c.
6.9.1	CVE-2022-31782	ftbench.c in FreeType Demo Programs through 2.12.1 has a heap-based buffer overflow.
6.9.1	CVE-2019-6129	**DISPUTED** png_create_info_struct in png.c in libpng 1.6.36 has a memory leak, as demonstrated by pngcp. NOTE a third party has stated "I don't think it is libpng's job to free this buffer."
6.9.1	CVE-2021-40145	**DISPUTED** gdImageGd2Ptr in gd_gd2.c in the GD Graphics Library (aka LibGD) through 2.3.2 has a double free. NOTE the vendor's position is "The GD2 image format is a proprietary image format of libgd. It has to be regarded as being obsolete, and should only be used for development and testing purposes."
6.9.1	CVE-2021-46822	The PPM reader in libjpeg-turbo through 2.0.90 mishandles use of tjLoadImage for loading a 16-bit binary PPM file into a grayscale buffer and loading a 16-bit binary PGM file into an RGB buffer. This is related to a heap-based buffer overflow in the get_word_rgb_row function in rdppm.c.
6.9.1	CVE-2013-0340	expat 2.1.0 and earlier does not properly handle entities expansion unless an application developer uses the XML_SetEntityDeclHandler function, which allows remote attackers to cause a denial of service (resource consumption), send HTTP requests to intranet servers, or read arbitrary files via a crafted XML document, aka an XML External Entity (XXE) issue. NOTE it could be argued that because expat already provides the ability to disable external entity expansion, the responsibility for resolving this issue lies with application developers; according to this argument, this entry should be REJECTed, and each affected application would need its own CVE.
6.9.1	CVE-2021-40812	The GD Graphics Library (aka LibGD) through 2.3.2 has an out-of-bounds read because of the lack of certain gdGetBuf and gdPutBuf return value checks.
6.9.1	CVE-2020-36309	ngx_http_lua_module (aka lua-nginx-module) before 0.10.16 in OpenResty allows unsafe characters in an argument when using the API to mutate a URI, or a request or response header.
6.9.0	CVE-2017-8291	Artifex Ghostscript through 2017-04-26 allows -dSAFER bypass and remote command execution via .rsdparams type confusion with a "/OutputFile (%pipe%" substring in a crafted .eps document that is an input to the gs program, as exploited in the wild in April 2017.
6.9.0	CVE-2017-8779	rpcbind through 0.2.4, LIBTIRPC through 1.0.1 and 1.0.2-rc through 1.0.2-rc3, and NTIRPC through 1.4.3 do not consider the maximum RPC data size during memory allocation for XDR strings, which allows remote attackers to cause a denial of service (memory consumption with no subsequent free) via a crafted UDP packet to port 111, aka rpcbomb.
6.9.0	CVE-2017-1000117	A malicious third-party can give a crafted "ssh://..." URL to an unsuspecting victim, and an attempt to visit the URL can result in any program that exists on the victim's machine being executed. Such a URL could be placed in the .gitmodules file of a malicious project, and an unsuspecting victim could be tricked into running "git clone --recurse-submodules" to trigger the vulnerability.

Release	CVE Fix	Description
6.9.0	CVE-2017-14492	Heap-based buffer overflow in dnsmasq before 2.78 allows remote attackers to cause a denial of service (crash) or execute arbitrary code via a crafted IPv6 router advertisement request.
6.9.0	CVE-2017-12615	When running Apache Tomcat 7.0.0 to 7.0.79 on Windows with HTTP PUTs enabled (e.g. via setting the readonly initialisation parameter of the Default to false) it was possible to upload a JSP file to the server via a specially crafted request. This JSP could then be requested and any code it contained would be executed by the server.
6.9.0	CVE-2017-12617	When running Apache Tomcat versions 9.0.0.M1 to 9.0.0, 8.5.0 to 8.5.22, 8.0.0.RC1 to 8.0.46 and 7.0.0 to 7.0.81 with HTTP PUTs enabled (e.g. via setting the readonly initialisation parameter of the Default servlet to false) it was possible to upload a JSP file to the server via a specially crafted request. This JSP could then be requested and any code it contained would be executed by the server.
6.9.0	CVE-2017-17405	Ruby before 2.4.3 allows Net::FTP command injection. Net::FTP#get, getbinaryfile, gettextfile, put, putbinaryfile, and puttextfile use Kernel#open to open a local file. If the localfile argument starts with the "|" pipe character, the command following the pipe character is executed. The default value of localfile is File.basename(remotefile), so malicious FTP servers could cause arbitrary command execution.
6.9.0	CVE-2018-1111	DHCP packages in Red Hat Enterprise Linux 6 and 7, Fedora 28, and earlier are vulnerable to a command injection flaw in the NetworkManager integration script included in the DHCP client. A malicious DHCP server, or an attacker on the local network able to spoof DHCP responses, could use this flaw to execute arbitrary commands with root privileges on systems using NetworkManager and configured to obtain network configuration using the DHCP protocol.
6.9.0	CVE-2018-16509	An issue was discovered in Artifex Ghostscript before 9.24. Incorrect "restoration of privilege" checking during handling of /invalidaccess exceptions could be used by attackers able to supply crafted PostScript to execute code using the "pipe" instruction.
6.9.0	CVE-2018-17456	Git before 2.14.5, 2.15.x before 2.15.3, 2.16.x before 2.16.5, 2.17.x before 2.17.2, 2.18.x before 2.18.1, and 2.19.x before 2.19.1 allows remote code execution during processing of a recursive "git clone" of a superproject if a .gitmodules file has a URL field beginning with a '-' character.
6.9.0	CVE-2020-1938	When using the Apache JServ Protocol (AJP), care must be taken when trusting incoming connections to Apache Tomcat. Tomcat treats AJP connections as having higher trust than, for example, a similar HTTP connection. If such connections are available to an attacker, they can be exploited in ways that may be surprising.
6.9.0	CVE-2020-8617	Using a specially-crafted message, an attacker may potentially cause a BIND server to reach an inconsistent state if the attacker knows (or successfully guesses) the name of a TSIG key used by the server. Since BIND, by default, configures a local session key even on servers whose configuration does not otherwise make use of it, almost all current BIND servers are vulnerable. In releases of BIND dating from March 2018 and after, an assertion check in tsig.c detects this inconsistent state and deliberately exits. Prior to the introduction of the check the server would continue operating in an inconsistent state, with potentially harmful results.
6.9.0	CVE-2023-3576	A memory leak flaw was found in Libtiff's tiffcrop utility. This issue occurs when tiffcrop operates on a TIFF image file, allowing an attacker to pass a crafted TIFF image file to tiffcrop utility, which causes this memory leak issue, resulting an application crash, eventually leading to a denial of service.
6.9.0	CVE-2022-1725	NULL Pointer Dereference in GitHub repository vim/vim prior to 8.2.4959.
6.9.0	CVE-2023-48237	Vim is an open source command line text editor. In affected versions when shifting lines in operator pending mode and using a very large value, it may be possible to overflow the size of integer. Impact is low, user interaction is required and a crash may not even happen in all situations. This issue has been addressed in commit `6bf131888` which has been included in version 9.0.2112. Users are advised to upgrade. There are no known workarounds for this vulnerability.
6.9.0	CVE-2020-19726	An issue was discovered in binutils libbfd.c 2.36 relating to the auxiliary symbol data allows attackers to read or write to system memory or cause a denial of service.
6.9.0	CVE-2022-1771	Uncontrolled Recursion in GitHub repository vim/vim prior to 8.2.4975.
6.9.0	CVE-2023-47471	Buffer Overflow vulnerability in strukturag libde265 v1.10.12 allows a local attacker to cause a denial of service via the slice_segment_header function in the slice.cc component.

Release	CVE Fix	Description
6.9.0	CVE-2022-2042	Use After Free in GitHub repository vim/vim prior to 8.2.
6.9.0	CVE-2023-5868	A memory disclosure vulnerability was found in PostgreSQL that allows remote users to access sensitive information by exploiting certain aggregate function calls with 'unknown'-type arguments. Handling 'unknown'-type values from string literals without type designation can disclose bytes, potentially revealing notable and confidential information. This issue exists due to excessive data output in aggregate function calls, enabling remote users to read some portion of system memory.
6.9.0	CVE-2023-48235	Vim is an open source command line text editor. When parsing relative ex addresses one may unintentionally cause an overflow. Ironically this happens in the existing overflow check, because the line number becomes negative and LONG_MAX - lnum will cause the overflow. Impact is low, user interaction is required and a crash may not even happen in all situations. This issue has been addressed in commit `060623e` which has been included in release version 9.0.2110. Users are advised to upgrade. There are no known workarounds for this vulnerability.
6.9.0	CVE-2023-39417	IN THE EXTENSION SCRIPT, a SQL Injection vulnerability was found in PostgreSQL if it uses @extowner@, @extschema@, or @extschema:...@ inside a quoting construct (dollar quoting, '', or ""). If an administrator has installed files of a vulnerable, trusted, non-bundled extension, an attacker with database-level CREATE privilege can execute arbitrary code as the bootstrap superuser.
6.9.0	CVE-2022-35205	An issue was discovered in Binutils readelf 2.38.50, reachable assertion failure in function display_debug_names allows attackers to cause a denial of service.
6.9.0	CVE-2023-48236	Vim is an open source command line text editor. When using the z= command, the user may overflow the count with values larger than MAX_INT. Impact is low, user interaction is required and a crash may not even happen in all situations. This vulnerability has been addressed in commit `73b2d379` which has been included in release version 9.0.2111. Users are advised to upgrade. There are no known workarounds for this vulnerability.
6.9.0	CVE-2023-43887	Libde265 v1.0.12 was discovered to contain multiple buffer overflows via the num_tile_columns and num_tile_row parameters in the function pic_parameter_set::dump.
6.9.0	CVE-2023-48231	Vim is an open source command line text editor. When closing a window, vim may try to access already freed window structure. Exploitation beyond crashing the application has not been shown to be viable. This issue has been addressed in commit `25aabc2b` which has been included in release version 9.0.2106. Users are advised to upgrade. There are no known workarounds for this vulnerability.
6.9.0	CVE-2023-46246	Vim is an improved version of the good old UNIX editor Vi. Heap-use-after-free in memory allocated in the function `ga_grow_inner` in in the file `src/alloc.c` at line 748, which is freed in the file `src/ex_docmd.c` in the function `do_cmdline` at line 1010 and then used again in `src/cmdhist.c` at line 759. When using the `:history` command, it's possible that the provided argument overflows the accepted value. Causing an Integer Overflow and potentially later an use-after-free. This vulnerability has been patched in version 9.0.2068.
6.9.0	CVE-2021-46174	Heap-based Buffer Overflow in function bfd_getl32 in Binutils objdump 3.37.
6.9.0	CVE-2022-2000	Out-of-bounds Write in GitHub repository vim/vim prior to 8.2.
6.9.0	CVE-2023-27102	Libde265 v1.0.11 was discovered to contain a segmentation violation via the function decoder_context::process_slice_segment_header at decctx.cc.
6.9.0	CVE-2023-48234	Vim is an open source command line text editor. When getting the count for a normal mode z command, it may overflow for large counts given. Impact is low, user interaction is required and a crash may not even happen in all situations. This issue has been addressed in commit `58f9befca1` which has been included in release version 9.0.2109. Users are advised to upgrade. There are no known workarounds for this vulnerability.
6.9.0	CVE-2023-5870	A flaw was found in PostgreSQL involving the pg_cancel_backend role that signals background workers, including the logical replication launcher, autovacuum workers, and the autovacuum launcher. Successful exploitation requires a non-core extension with a less-resilient background worker and would affect that specific background worker only. This issue may allow a remote high privileged user to launch a denial of service (DoS) attack.

Release	CVE Fix	Description
6.9.0	CVE-2023-48233	Vim is an open source command line text editor. If the count after the ⍀ command is larger than what fits into a (signed) long variable, abort with e_value_too_large. Impact is low, user interaction is required and a crash may not even happen in all situations. This issue has been addressed in commit `ac6378773` which has been included in release version 9.0.2108. Users are advised to upgrade. There are no known workarounds for this vulnerability.
6.9.0	CVE-2023-5869	A flaw was found in PostgreSQL that allows authenticated database users to execute arbitrary code through missing overflow checks during SQL array value modification. This issue exists due to an integer overflow during array modification where a remote user can trigger the overflow by providing specially crafted data. This enables the execution of arbitrary code on the target system, allowing users to write arbitrary bytes to memory and extensively read the server's memory.
6.9.0	CVE-2022-1897	Out-of-bounds Write in GitHub repository vim/vim prior to 8.2.
6.9.0	CVE-2023-40745	LibTIFF is vulnerable to an integer overflow. This flaw allows remote attackers to cause a denial of service (application crash) or possibly execute an arbitrary code via a crafted tiff image, which triggers a heap-based buffer overflow.
6.9.0	CVE-2023-27103	Libde265 v1.0.11 was discovered to contain a heap buffer overflow via the function derive_collocated_motion_vectors at motion.cc.
6.9.0	CVE-2023-28531	ssh-add in OpenSSH before 9.3 adds smartcard keys to ssh-agent without the intended per-hop destination constraints. The earliest affected version is 8.9.
6.9.0	CVE-2023-41175	A vulnerability was found in libtiff due to multiple potential integer overflows in raw2tiff.c. This flaw allows remote attackers to cause a denial of service or possibly execute an arbitrary code via a crafted tiff image, which triggers a heap-based buffer overflow.
6.8.0	CVE-2023-35945	Envoy is a cloud-native high-performance edge/middle/service proxy. Envoy's HTTP/2 codec may leak a header map and bookkeeping structures upon receiving `RST_STREAM` immediately followed by the `GOAWAY` frames from an upstream server. In nghttp2, cleanup of pending requests due to receipt of the `GOAWAY` frame skips de-allocation of the bookkeeping structure and pending compressed header. The error return [code path] is taken if connection is already marked for not sending more requests due to `GOAWAY` frame. The clean-up code is right after the return statement, causing memory leak. Denial of service through memory exhaustion. This vulnerability was patched in versions(s) 1.26.3, 1.25.8, 1.24.9, 1.23.11.
6.8.0	CVE-2023-2975	The AES-SIV cipher implementation contains a bug that causes it to ignore empty associated data entries which are unauthenticated as a consequence.
6.8.0	CVE-2023-38545	This flaw makes curl overflow a heap based buffer in the SOCKS5 proxy handshake.
6.8.0	CVE-2022-48174	There is a stack overflow vulnerability in ash.c:6030 in busybox before 1.35. In the environment of Internet of Vehicles, this vulnerability can be executed from command to arbitrary code execution.
6.8.0	CVE-2023-2603	A vulnerability was found in libcap. This issue occurs in the _libcap_strdup() function and can lead to an integer overflow if the input string is close to 4GiB.
6.8.0	CVE-2023-2602	A vulnerability was found in the pthread_create() function in libcap. This issue may allow a malicious actor to use cause __real_pthread_create() to return an error, which can exhaust the process memory.
6.8.0	CVE-2023-38039	When curl retrieves an HTTP response, it stores the incoming headers so that they can be accessed later via the libcurl headers API.
6.8.0	CVE-2022-41862	In PostgreSQL, a modified, unauthenticated server can send an unterminated string during the establishment of Kerberos transport encryption. In certain conditions a server can cause a libpq client to over-read and report an error message containing uninitialized bytes.
6.8.0	CVE-2023-38546	This flaw allows an attacker to insert cookies at will into a running program using libcurl, if the specific series of conditions are met.

Release	CVE Fix	Description
6.8.0	CVE-2022-23491	Certifi is a curated collection of Root Certificates for validating the trustworthiness of SSL certificates while verifying the identity of TLS hosts. Certifi 2022.12.07 removes root certificates from "TrustCor" from the root store. These are in the process of being removed from Mozilla's trust store. TrustCor's root certificates are being removed pursuant to an investigation prompted by media reporting that TrustCor's ownership also operated a business that produced spyware. Conclusions of Mozilla's investigation can be found in the linked google group discussion.
6.8.0	CVE-2023-32803	*This vulnerability has not been analyzed by NVD yet.*
6.8.0	CVE-2023-2650	Processing some specially crafted ASN.1 object identifiers or data containing them may be very slow.
6.7.0	CVE-2015-20107	In Python (aka CPython) up to 3.10.8, the mailcap module does not add escape characters into commands discovered in the system mailcap file. This may allow attackers to inject shell commands into applications that call mailcap.findmatch with untrusted input (if they lack validation of user-provided filenames or arguments). The fix is also back-ported to 3.7, 3.8, 3.9
6.7.0	CVE-2020-10735	A flaw was found in python. In algorithms with quadratic time complexity using non-binary bases, when using int("text"), a system could take 50ms to parse an int string with 100,000 digits and 5s for 1,000,000 digits (float, decimal, int.from_bytes(), and int() for binary bases 2, 4, 8, 16, and 32 are not affected). The highest threat from this vulnerability is to system availability.
6.7.0	CVE-2020-19185	Buffer Overflow vulnerability in one_one_mapping function in progs/dump_entry.c:1373 in ncurses 6.1 allows remote attackers to cause a denial of service via crafted command.
6.7.0	CVE-2020-19186	Buffer Overflow vulnerability in _nc_find_entry function in tinfo/comp_hash.c:66 in ncurses 6.1 allows remote attackers to cause a denial of service via crafted command.
6.7.0	CVE-2020-19187	Buffer Overflow vulnerability in fmt_entry function in progs/dump_entry.c:1100 in ncurses 6.1 allows remote attackers to cause a denial of service via crafted command.
6.7.0	CVE-2020-19188	Buffer Overflow vulnerability in fmt_entry function in progs/dump_entry.c:1116 in ncurses 6.1 allows remote attackers to cause a denial of service via crafted command.
6.7.0	CVE-2020-19189	Buffer Overflow vulnerability in postprocess_terminfo function in tinfo/parse_entry.c:997 in ncurses 6.1 allows remote attackers to cause a denial of service via crafted command.
6.7.0	CVE-2020-19190	Buffer Overflow vulnerability in _nc_find_entry in tinfo/comp_hash.c:70 in ncurses 6.1 allows remote attackers to cause a denial of service via crafted command.
6.7.0	CVE-2020-27619	In Python 3 through 3.9.0, the Lib/test/multibytecodec_support.py CJK codec tests call eval() on content retrieved via HTTP.
6.7.0	CVE-2020-35448	An issue was discovered in the Binary File Descriptor (BFD) library (aka libbfd), as distributed in GNU Binutils 2.35.1. A heap-based buffer over-read can occur in bfd_getl_signed_32 in libbfd.c because sh_entsize is not validated in _bfd_elf_slurp_secondary_reloc_section in elf.c.
6.7.0	CVE-2021-20197	There is an open race window when writing output in the following utilities in GNU binutils version 2.35 and earlier:ar, objcopy, strip, ranlib. When these utilities are run as a privileged user (presumably as part of a script updating binaries across different users), an unprivileged user can trick these utilities into getting ownership of arbitrary files through a symlink.
6.7.0	CVE-2021-20284	A flaw was found in GNU Binutils 2.35.1, where there is a heap-based buffer overflow in _bfd_elf_slurp_secondary_reloc_section in elf.c due to the number of symbols not calculated correctly. The highest threat from this vulnerability is to system availability.
6.7.0	CVE-2021-29921	In Python before 3,9,5, the ipaddress library mishandles leading zero characters in the octets of an IP address string. This (in some situations) allows attackers to bypass access control that is based on IP addresses.
6.7.0	CVE-2021-33294	In elfutils 0.183, an infinite loop was found in the function handle_symtab in readelf.c .Which allows attackers to cause a denial of service (infinite loop) via crafted file.

Release	CVE Fix	Description
6.7.0	CVE-2021-3426	There's a flaw in Python 3's pydoc. A local or adjacent attacker who discovers or is able to convince another local or adjacent user to start a pydoc server could access the server and use it to disclose sensitive information belonging to the other user that they would not normally be able to access. The highest risk of this flaw is to data confidentiality. This flaw affects Python versions before 3.8.9, Python versions before 3.9.3 and Python versions before 3.10.0a7.
6.7.0	CVE-2021-3487	There's a flaw in the BFD library of binutils in versions before 2.36. An attacker who supplies a crafted file to an application linked with BFD, and using the DWARF functionality, could cause an impact to system availability by way of excessive memory consumption.
6.7.0	CVE-2021-3530	A flaw was discovered in GNU libiberty within demangle_path() in rust-demangle.c, as distributed in GNU Binutils version 2.36. A crafted symbol can cause stack memory to be exhausted leading to a crash.
6.7.0	CVE-2021-3549	An out of bounds flaw was found in GNU binutils objdump utility version 2.36. An attacker could use this flaw and pass a large section to avr_elf32_load_records_from_section() probably resulting in a crash or in some cases memory corruption. The highest threat from this vulnerability is to integrity as well as system availability.
6.7.0	CVE-2021-36368	DISPUTED An issue was discovered in OpenSSH before 8.9. If a client is using public-key authentication with agent forwarding but without -oLogLevel=verbose, and an attacker has silently modified the server to support the None authentication option, then the user cannot determine whether FIDO authentication is going to confirm that the user wishes to connect to that server, or that the user wishes to allow that server to connect to a different server on the user's behalf. NOTE the vendor's position is "this is not an authentication bypass, since nothing is being bypassed.
6.7.0	CVE-2021-3733	There's a flaw in urllib's AbstractBasicAuthHandler class. An attacker who controls a malicious HTTP server that an HTTP client (such as web browser) connects to, could trigger a Regular Expression Denial of Service (ReDOS) during an authentication request with a specially crafted payload that is sent by the server to the client. The greatest threat that this flaw poses is to application availability.
6.7.0	CVE-2021-3737	A flaw was found in python. An improperly handled HTTP response in the HTTP client code of python may allow a remote attacker, who controls the HTTP server, to make the client script enter an infinite loop, consuming CPU time. The highest threat from this vulnerability is to system availability.
6.7.0	CVE-2021-3826	Heap/stack buffer overflow in the dlang_lname function in d-demangle.c in libiberty allows attackers to potentially cause a denial of service (segmentation fault and crash) via a crafted mangled symbol.
6.7.0	CVE-2021-41617	sshd in OpenSSH 6.2 through 8.x before 8.8, when certain non-default configurations are used, allows privilege escalation because supplemental groups are not initialized as expected. Helper programs for AuthorizedKeysCommand and AuthorizedPrincipalsCommand may run with privileges associated with group memberships of the sshd process, if the configuration specifies running the command as a different user.
6.7.0	CVE-2021-45078	stab_xcoff_builtin_type in stabs.c in GNU Binutils through 2.37 allows attackers to cause a denial of service (heap-based buffer overflow) or possibly have unspecified other impact, as demonstrated by an out-of-bounds write. NOTE this issue exists because of an incorrect fix for CVE-2018-12699.
6.7.0	CVE-2021-46195	GCC v12.0 was discovered to contain an uncontrolled recursion via the component libiberty/rust-demangle.c. This vulnerability allows attackers to cause a Denial of Service (DoS) by consuming excessive CPU and memory resources.
6.7.0	CVE-2022-2208	NULL Pointer Dereference in GitHub repository vim/vim prior to 8.2.5163.
6.7.0	CVE-2022-2210	Out-of-bounds Write in GitHub repository vim/vim prior to 8.2.
6.7.0	CVE-2022-2257	Out-of-bounds Read in GitHub repository vim/vim prior to 9.0.
6.7.0	CVE-2022-2264	Heap-based Buffer Overflow in GitHub repository vim/vim prior to 9.0.
6.7.0	CVE-2022-2284	Heap-based Buffer Overflow in GitHub repository vim/vim prior to 9.0.

Release	CVE Fix	Description
6.7.0	CVE-2022-2285	Integer Overflow or Wraparound in GitHub repository vim/vim prior to 9.0.
6.7.0	CVE-2022-2286	Out-of-bounds Read in GitHub repository vim/vim prior to 9.0.
6.7.0	CVE-2022-2287	Out-of-bounds Read in GitHub repository vim/vim prior to 9.0.
6.7.0	CVE-2022-2289	Use After Free in GitHub repository vim/vim prior to 9.0.
6.7.0	CVE-2022-2598	Out-of-bounds Write to API in GitHub repository vim/vim prior to 9.0.0100.
6.7.0	CVE-2022-3016	Use After Free in GitHub repository vim/vim prior to 9.0.0286.
6.7.0	CVE-2022-3037	Use After Free in GitHub repository vim/vim prior to 9.0.0322.
6.7.0	CVE-2022-3099	Use After Free in GitHub repository vim/vim prior to 9.0.0360.
6.7.0	CVE-2022-37454	The Keccak XKCP SHA-3 reference implementation before fdc6fef has an integer overflow and resultant buffer overflow that allows attackers to execute arbitrary code or eliminate expected cryptographic properties. This occurs in the sponge function interface.
6.7.0	CVE-2022-38533	In GNU Binutils before 2.40, there is a heap-buffer-overflow in the error function bfd_getl32 when called from the strip_main function in strip-new via a crafted file.
6.7.0	CVE-2022-40433	An issue was discovered in function ciMethodBlocks::make_block_at in Oracle JDK (HotSpot VM) 11, 17 and OpenJDK (HotSpot VM) 8, 11, 17, allows attackers to cause a denial of service.
6.7.0	CVE-2022-4285	An illegal memory access flaw was found in the binutils package. Parsing an ELF file containing corrupt symbol version information may result in a denial of service. This issue is the result of an incomplete fix for CVE-2020-16599.
6.7.0	CVE-2022-42919	Python 3.9.x before 3.9.16 and 3.10.x before 3.10.9 on Linux allows local privilege escalation in a non-default configuration. The Python multiprocessing library, when used with the forkserver start method on Linux, allows pickles to be deserialized from any user in the same machine local network namespace, which in many system configurations means any user on the same machine. Pickles can execute arbitrary code. Thus, this allows for local user privilege escalation to the user that any forkserver process is running as. Setting multiprocessing.util.abstract_sockets_supported to False is a workaround. The forkserver start method for multiprocessing is not the default start method. This issue is Linux specific because only Linux supports abstract namespace sockets. CPython before 3.9 does not make use of Linux abstract namespace sockets by default. Support for users manually specifying an abstract namespace socket was added as a bugfix in 3.7.8 and 3.8.3, but users would need to make specific uncommon API calls in order to do that in CPython before 3.9.
6.7.0	CVE-2022-45061	An issue was discovered in Python before 3.11.1. An unnecessary quadratic algorithm exists in one path when processing some inputs to the IDNA (RFC 3490) decoder, such that a crafted, unreasonably long name being presented to the decoder could lead to a CPU denial of service. Hostnames are often supplied by remote servers that could be controlled by a malicious actor; in such a scenario, they could trigger excessive CPU consumption on the client attempting to make use of an attacker-supplied supposed hostname. For example, the attack payload could be placed in the Location header of an HTTP response with status code 302. A fix is planned in 3.11.1, 3.10.9, 3.9.16, 3.8.16, and 3.7.16.
6.7.0	CVE-2022-47673	An issue was discovered in Binutils addr2line before 2.39.3, function parse_module contains multiple out of bound reads which may cause a denial of service or other unspecified impacts.
6.7.0	CVE-2022-47696	An issue was discovered Binutils objdump before 2.39.3 allows attackers to cause a denial of service or other unspecified impacts via function compare_symbols.
6.7.0	CVE-2023-1579	Heap based buffer overflow in binutils-gdb/bfd/libbfd.c in bfd_getl64.
6.7.0	CVE-2023-2222	This vulnerability has not been analyzed by NVD yet.

Release	CVE Fix	Description
6.7.0	CVE-2023-22603	ConsultIDs none. Reason This candidate was withdrawn by its CNA. Further investigation showed that it was not a security issue. Notes none.
6.7.0	CVE-2023-22604	ConsultIDs- none. Reason-This candidate was withdrawn by its CNA. Further investigation showed that it was not a security issue. Notes- none.
6.7.0	CVE-2023-22605	ConsultIDs- none. Reason- This candidate was withdrawn by its CNA. Further investigation showed that it was not a security issue. Notes- none
6.7.0	CVE-2023-22606	ConsultIDs- none. Reason- This candidate was withdrawn by its CNA. Further investigation showed that it was not a security issue. Notes- none
6.7.0	CVE-2023-22609	ConsultIDs- none. Reason- This candidate was withdrawn by its CNA. Further investigation showed that it was not a security issue. Notes- none
6.7.0	CVE-2023-24535	Parsing invalid messages can panic. Parsing a text-format message which contains a potential number consisting of a minus sign, one or more characters of whitespace, and no further input will cause a panic.
6.7.0	CVE-2023-25584	An out-of-bounds read flaw was found in the parse_module function in bfd/vms-alpha.c in Binutils.
	CVE-2023-25585	A flaw was found in Binutils. The use of an uninitialized field in the struct module *module may lead to application crash and local denial of service.
	CVE-2023-25586	A flaw was found in Binutils. A logic fail in the bfd_init_section_decompress_status function may lead to the use of an uninitialized variable that can cause a crash and local denial of service.
6.7.0	CVE-2023-25588	A flaw was found in Binutils. The field `the_bfd` of `asymbol` struct is uninitialized in the `bfd_mach_o_get_synthetic_symtab` function, which may lead to an application crash and local denial of service.
6.7.0	CVE-2023-32002	The use of `Module._load()` can bypass the policy mechanism and require modules outside of the policy.json definition for a given module. This vulnerability affects all users using the experimental policy mechanism in all active release lines- 16.x, 18.x and, 20.x. Please note that at the time this CVE was issued, the policy is an experimental feature of Node.js.
6.7.0	CVE-2023-32006	The use of `module.constructor.createRequire()` can bypass the policy mechanism and require modules outside of the policy.json definition for a given module. This vulnerability affects all users using the experimental policy mechanism in all active release lines- 16.x, 18.x, and, 20.x. Please note that at the time this CVE was issued, the policy is an experimental feature of Node.js.
6.7.0	CVE-2023-32559	A privilege escalation vulnerability exists in the experimental policy mechanism in all active release lines- 16.x, 18.x and, 20.x. The use of the deprecated API `process.binding()` can bypass the policy mechanism by requiring internal modules and eventually take advantage of `process.binding('spawn_sync')` run arbitrary code, outside of the limits defined in a `policy.json` file. Please note that at the time this CVE was issued, the policy is an experimental feature of Node.js.
6.7.0	CVE-2023-38408	The PKCS 11 feature in ssh-agent in OpenSSH before 9.3p2 has an insufficiently trustworthy search path, leading to remote code execution if an agent is forwarded to an attacker-controlled system. (Code in /usr/lib is not necessarily safe for loading into ssh-agent.) NOTE this issue exists because of an incomplete fix for CVE-2016-10009.
6.7.0	CVE-2023-40217	An issue was discovered in Python before 3.8.18, 3.9.x before 3.9.18, 3.10.x before 3.10.13, and 3.11.x before 3.11.5. It primarily affects servers (such as HTTP servers) that use TLS client authentication. If a TLS server-side socket is created, receives data into the socket buffer, and then is closed quickly, there is a brief window where the SSLSocket instance will detect the socket as "not connected" and won't initiate a handshake, but buffered data will still be readable from the socket buffer. This data will not be authenticated if the server-side TLS peer is expecting client certificate authentication, and is indistinguishable from valid TLS stream data. Data is limited in size to the amount that will fit in the buffer. (The TLS connection cannot directly be used for data exfiltration because the vulnerable code path requires that the connection be closed on initialization of the SSLSocket.)

Release	CVE Fix	Description
6.7.0	CVE-2023-4863	Heap buffer overflow in libwebp in Google Chrome prior to 116.0.5845.187 and libwebp 1.3.2 allowed a remote attacker to perform an out of bounds memory write via a crafted HTML page. (Chromium security severity- Critical)
6.7.0	CVE-2023-5129	This CVE- ID has been rejected or withdrawn by its CVE- Numbering Authority. Duplicate of CVE-2023-4863.
6.7.0	CVE-2023-5156	A flaw was found in the GNU C Library. A recent fix for CVE-2023-4806 introduced the potential for a memory leak, which may result in an application crash.
6.5.0	CVE-2023-2650	Possible DoS translating ASN.1 object identifiers
6.5.0	CVE-2023-2975	AES-SIV implementation ignores empty associated data entries
6.5.0	CVE-2023-3446	Excessive time spent checking DH keys and parameters
6.5.0	CVE-2023-3817	Excessive time spent checking DH q parameter value
6.5.0	CVE-2023-35945	Envoy vulnerable to HTTP/2 memory leak in nghttp2 codec
6.5.0	CVE-2022-29458	ncurses 6.3 before patch 20220416 has an out-of-bounds read and segmentation violation in convert_strings in tinfo/read_entry.c in the terminfo library.
6.5.0	CVE-2022-3715	A flaw was found in the bash package, where a heap-buffer overflow can occur in valid parameter_transform. This issue may lead to memory problems.
6.5.0	CVE-2022-41409	Integer overflow vulnerability in pcre2test before 10.41 allows attackers to cause a denial of service or other unspecified impacts via negative input.
6.5.0	CVE-2022-4899	A vulnerability was found in zstd v1.4.10, where an attacker can supply empty string as an argument to the command line tool to cause buffer overrun.
6.5.0	CVE-2016-1585	In all versions of AppArmor mount rules are accidentally widened when compiled.
6.5.0	CVE-2016-2568	pkexec, when used with --user nonpriv, allows local users to escape to the parent session via a crafted TIOCSTI ioctl call, which pushes characters to the terminal's input buffer.
6.5.0	CVE-2023-34969	D-Bus before 1.15.6 sometimes allows unprivileged users to crash dbus-daemon.