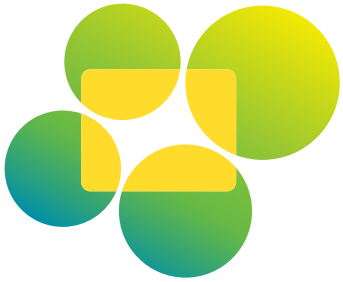
Yellowbrick Data Warehouse



Product Documentation

Version 5.2.32

5.2.32-43426.02f7540e

8/28/2025

© 2018 - 2025 Yellowbrick Data, Inc. All Rights Reserved.

What's New in Version 5.2

Version 5.2.0 contains the following new features and enhancements. There are also some significant changes in behavior. For features and changes introduced in 5.2.x releases later than 5.2.0, see the Release Notes.

Query Planning Enhancements

A number of enhancements have been made in query planning for this release:

- General improvements in join ordering
- Better use of declared primary and foreign keys in query planning
- Improved estimates of join cost and cardinality of matching key values
- Various improvements and adjustments to the `EXPLAIN` command output

Minimum Configuration for Andromeda Platform

The Andromeda platform supports a new three-node minimum configuration.

System View Updates

The following system views are changed in Version 5.2:

- `sys.replica` has a new `is_table_level_restart` column.
- `sys.replica_status` and `sys.log_replica_status` both have a new `retries` column.

Encryption Keystore Alert

A new user-defined system alert is triggered when the encryption keystore is locked.

ybcli Changes

The SSD section of the `blade status` output has been reformatted and now includes the SSD firmware version. Note that on the Andromeda platform the `blade status` output includes an additional line with BMC and FPGA information.

---

ybtools Changes

In previous releases, RPMs for `ybtools` on RHEL and AIX systems had to be installed in the default location (`/opt/ybtools`). In Version 5.2, you can specify an alternative path.

The `ybdumpschema` and `ybdumproles` help text has been updated for consistency. In previous releases, there were some unnecessary differences in the output for the two tools.

SQL Changes

In `CREATE TABLE`, the ON keyword is now supported for `DISTRIBUTE ON RANDOM` and `DISTRIBUTE ON REPLICATE`. Previously, the `ON` keyword was only supported for hash distribution (`DISTRIBUTE ON` column).

The default transaction isolation level can be changed only by internal system users. The only supported transaction isolation level that superusers can set is `read committed`.

The `DEALLOCATE` command now has an `IF EXISTS` option.

The `ADD_MONTHS` function always returns the last day of the result month when the input expression is the last day of the month. For example, 2000 was a leap year. Therefore, 2000-02-28 + 1 month returns 2000-03-28, and 2000-02-29 + 1 month returns 2000-03-31.

Parent topic: Yellowbrick Documentation

Contents

- Documentation Version 5.2.32
- What's New in Version 5.2
- System Management
  - External Interfaces
  - System Maintenance
  - System Hardware Encryption
    - Setting Up Encryption
      - Example: Set Up the Keystore and Enable Encryption
      - Example: Use Multiple Unlock Keys
      - Example: Force Keystore Setup
    - Managing Encrypted Drives
    - Example: Rotate the Keystore with Multiple Unlock Keys
  - Manager Node Failover
  - Starting and Stopping the Database
  - Powering the Appliance Off and On
  - Checking the Logs
  - User Accounts
  - Using the System Management Console
    - Console Login
    - Introduction to the SMC
    - Configuration Settings in the SMC
  - Chassis Management
  - System Alerts
    - Alert Types
    - Creating an Alert Endpoint
      - Endpoint Types
      - Alert Templates
    - Managing Alerts
  - Remote Diagnostics
    - Diagnostics Service Levels
    - Enabling Remote Diagnostics
    - Configuring Remote Diagnostics
  - Using Network Tracing
  - Manager Node Ports
  - ybcli Reference
    - blade
    - clear
    - config
      - config bmc
      - config dba
      - config firewall
      - config hostname
      - config keystore
      - config network
      - config ntp
      - config phonehome
      - config timezone
      - config user
    - database
    - encryption
    - exit, quit
    - health
    - help
    - keystore

        - log
        - manager
        - status
        - system
- Client Tools and Drivers
    - Installing the Java Runtime Environment
    - Installing ybtools
        - Installing ybtools on Linux
        - Installing ybtools on AIX
        - Installing ybtools on macOS
        - Installing ybtools on Windows
        - Installing the "generic" ybtools
    - Uninstalling ybtools
    - Opening Network Ports for Clients
    - Downloading Drivers
    - Configuring SSL/TLS for Tools and Drivers
        - SSL/TLS Overview
        - Enabling and Verifying SSL/TLS Encryption
        - Secure Connections for ODBC/JDBC Clients and ybsql
            - Secure Connections for ODBC Clients
            - Secure Connections for JDBC Clients
            - Secure Connections for ybsql (libpq)
            - Secure Connections for DBeaver Clients
        - Secure Connections for Java-based ybtools
        - Obtaining a Certificate Chain
        - Troubleshooting SSL Issues
    - Using the ADO.NET Driver
- ybsql Reference
    - ybsql Connections
    - ybsql Environment Variables
    - ybsql Startup File
    - ybsql Command-Line Options
    - ybsql Slash (\) Commands
    - ybsql \copy Command
        - \copy Parameters
        - \copy Examples
    - ybsql Examples
    - ybsql Properties and Variables
        - ybsql Properties
        - ybsql Display Properties
        - ybsql User Variables
        - ybsql Prompt Variables
        - Examples with ybsql Variables
- Configuration Parameters
    - General
        - rowstore_block_threshold_percent
        - rowstore_read_only_threshold_percent
        - rowstore_warning_threshold_percent
    - YRS Alerting Parameters
        - yrs_data_files_count_minor_threshold
        - yrs_data_files_count_major_threshold
        - yrs_data_files_count_critical_threshold
        - yrs_commit_records_count_minor_threshold
        - yrs_commit_records_count_major_threshold
        - yrs_commit_records_count_critical_threshold
        - yrs_delete_records_count_minor_threshold
        - yrs_delete_records_count_major_threshold

        - yrs_delete_records_count_critical_threshold
        - yrs_unused_files_count_minor_threshold
        - yrs_unused_files_count_major_threshold
        - yrs_unused_files_count_critical_threshold
- Database Administration
    - Database Objects
    - Database Limits
    - Creating Databases
    - Managing Tables and Views
        - Distributing Data
        - Partitioning Tables
        - Auto-Analyzing Tables
        - External Tables
        - Generating Values with Sequences
        - System Columns in Tables
        - Disk Quotas
    - Managing the Row Store
    - Generating DDL
    - Running Queries
    - Cross-Database Queries
    - Canceling Queries
    - Managing Idle Sessions
    - Encrypting Sensitive Data
        - SQL Encryption Examples
        - Encryption Key Rotation
        - Deprecated Encryption Functions
    - Creating Stored Procedures
    - Managing Database Users
    - LDAP Integration
        - Importing Certificates
        - LDAP Sample Schema
        - LDAP Authentication
            - LDAP Authentication Modes
            - Setting Up Authentication
            - LDAP Authentication Settings
            - Bind Examples
            - Search, then Bind Example
        - Synchronizing Users and Groups
            - LDAP Synchronization Settings
            - LDAP Synchronization: Basic Example
            - LDAP Synchronization: Examples with Filters
    - Checking the Server Version
    - System Views
        - sys.alert
        - sys.backup
        - sys.backup_chain
        - sys.backup_legacy
        - sys.backup_snapshot
        - sys.blade
        - sys.column
        - sys.const
        - sys.database
        - sys.drive_summary
        - sys.key
        - sys.lock
        - sys.load
        - sys.log_alert

        - sys.log_analyze
        - sys.log_audit
        - sys.log_authentication
        - sys.log_backup
        - sys.log_backup_legacy
        - sys.log_error
        - sys.log_load
        - sys.log_query
        - sys.log_query_alert
        - sys.log_query_analyze
        - sys.log_query_explain
        - sys.log_replica_status
        - sys.log_restore
        - sys.log_restore_legacy
        - sys.log_session
        - sys.log_unload
        - sys.log_warning
        - sys.mount
        - sys.procedure
        - sys.query
        - sys.query_analyze
        - sys.query_explain
        - sys.query_rule_event
        - sys.remote_server
        - sys.replica
        - sys.replica_status
        - sys.restore
        - sys.restore_legacy
        - 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_rule
        - sys.wlm_pending_pool
        - sys.worker
- Loading and Unloading Data
    - Bulk Loading Tables
        - Running a Bulk Load
            - Using the Dry Run Option
            - Setting the Commit Interval
            - Saving Load Options to a File
            - Loading Generated Key Values
            - Specifying NULL Behavior for Different Columns
            - Deletes, Updates, and Upserts
            - Handling Duplicate Rows

        - Monitoring Load Operations
        - Analyzing Rejected Rows
        - Resuming a Partial Load
        - Loading Data Exported from SQL Server
    - Bulk Load Examples
        - Bulk Load Insert Examples
        - Delete, Update, and Upsert Examples
        - Examples with Duplicate Rows
    - Loading Tables from Object Storage
        - Loading from Amazon S3
            - S3 Authentication Methods
            - S3 Object Storage URIs
            - S3 Object Storage Examples
        - Loading from S3-Compatible Object Storage
        - Loading from Azure Blob Storage
            - Azure Authentication Methods
            - Azure Blob Storage URIs
            - Azure Blob Storage Examples
        - ybload Object Storage Options
    - ybload Command
        - ybload Options
        - Setting the --format Option
        - NULL and Empty Markers
        - Character Escape Sequences
        - ybload Field Options
        - ybload Date Formats
        - ybload Time Formats
        - ybload Timestamp Formats
        - ybload Advanced Processing Options
        - Escaping Quotes in Windows Clients
- Trickle Loading Data via JDBC
    - Performance Factors
    - Performance Tuning
    - Example of a Trickle Load Java Program
- Loading Tables with Spark
    - Setting Up the ybrelay Service
        - Running the ybrelay-init Script
        - TLS Support for ybrelay and Spark
        - Creating a Java Keystore
        - Starting the ybrelay Service Manually
    - Setting up and Running a Spark Job
        - Spark Application Options
        - Spark Format for Reading Schemas
        - Error Tolerance
        - Examples of Spark Jobs
        - Output for a Spark Job
    - ybrelay Options
    - Spark and ybrelay Glossary
- Unloading Data
    - Setting Up an Unload
    - Saving Unload Options to a File
    - Unloading Data to an S3 Bucket
    - ybunload Command
        - ybunload Options
        - ybunload Output Files
        - ybunload Examples
- Workload Management

- Sharing System Resources
- How Queries Are Executed
- How WLM Works
  - Creating WLM Profiles
  - Creating WLM Resource Pools
    - Allocating Query Memory
    - Example: Memory Allocation Among Resource Pools
    - Examples of Flex Pools
  - Creating WLM Rules
    - Rule Types
    - Rule Conditions
    - Rule Actions
    - Rule Processing
    - Rule Examples
      - Submit Rule Examples
      - Assemble Rule Examples
      - Compile Rule Examples
      - Runtime Rule Examples
      - Completion Rule Examples
      - Restart Rule Examples
      - Examples of JavaScript for Rules
    - Quick Rules in the SMC
    - JavaScript Properties for WLM Rules
    - Recoverable Error Codes
  - Throttling Concurrency
  - Moving Queries
  - Restarting Queries
  - Using Query Tags
  - Exporting and Importing Profiles
  - Monitoring WLM Activity
- A Simple WLM Example
  - WLM Example: Using the SMC
  - WLM Example: Using SQL
- Backup and Restore
  - Overview
    - Backup Strategies
    - Restore Operations
    - Backup and Restore Clients
    - Backup and Restore Glossary
  - ybbackup Commands
    - ybbackup Options
    - ybbackup Examples
  - ybrestore Commands
    - ybrestore Options
    - ybrestore Examples
    - Running Incremental Restores
    - Restoring Users and Roles
  - ybbackupctl Commands
    - ybbackupctl Options
    - ybbackupctl Examples
  - Monitoring Backup and Restore Operations
  - Decrypting Columns in a Restored Database
  - HOT_STANDBY and READONLY Modes
- Database Replication
  - Overview
  - Setting Up Replication
    - Setting Up Loopback Replication

        - Configuring SSL Trust
        - Seeding a Replica
    - Monitoring Replication
    - Managing Replication
        - Pausing and Resuming Replication
        - Dropping a Replica
        - Replication Failover and Failback
        - Rolling Back a Replicated Database
    - Replication SQL Commands
    - Replication Glossary
- SQL Reference
    - SQL Data Types
        - BIGINT
        - BOOLEAN
        - CHAR
        - DATE
        - DECIMAL
            - Calculations with DECIMAL Values
            - Examples of DECIMAL Calculations
        - DOUBLE PRECISION
        - INTEGER
        - IPV4
        - IPV6
        - MACADDR
        - MACADDR8
        - REAL
        - SMALLINT
        - TIME
        - TIMESTAMP
        - TIMESTAMP WITH TIME ZONE
        - UUID
        - VARCHAR
        - Trailing Blanks in Character Data
        - Data Type Casting
            - Explicit Casting
            - Implicit Casting
            - Implicit Casting Examples
        - SQL String Constants
            - C-style Escape Sequences in String Constants
            - Unicode Escape Sequences in String Constants
            - Escape Sequences for Non-Printable Characters
    - SQL Commands
        - SQL Identifiers
        - ABORT
        - ALTER DATABASE
        - ALTER DATABASE ADD REPLICA
        - ALTER DATABASE ALTER REPLICA
        - ALTER DATABASE DROP REPLICA
        - ALTER DEFAULT PRIVILEGES
        - ALTER PROCEDURE
        - ALTER REMOTE SERVER
        - ALTER ROLE
        - ALTER SCHEMA
        - ALTER SEQUENCE
        - ALTER SYSTEM
        - ALTER TABLE
        - ALTER USER

- ALTER VIEW
- ALTER WLM PROFILE
- ALTER WLM RESOURCE POOL
- ALTER WLM RULE
- BEGIN
- CALL
- CANCEL
- CLOSE
- COMMENT ON
- COMMIT
- CREATE DATABASE
- CREATE EXTERNAL MOUNT
- CREATE EXTERNAL TABLE
  - USING Options
  - External Table Examples
- CREATE KEY
- CREATE PROCEDURE
- CREATE REMOTE SERVER
- CREATE ROLE
- CREATE SEQUENCE
- CREATE SCHEMA
- CREATE TABLE
  - CREATE TABLE Examples
  - Distribution Options
  - Partitioning Options
  - Sorted and Clustered Tables
  - Disabling Join Elimination for LEFT Joins
- 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 DATABASE
- DROP EXTERNAL MOUNT
- 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

- GRANT
  - ON DATABASE
  - ON KEY
  - ON PROCEDURE
  - ON ROLE
  - ON SCHEMA
  - ON SEQUENCE
  - ON SYSTEM
  - ON TABLE
  - ROLE
  - Abbreviations for ACLs
- HELP
- IMPORT SSL TRUST
- INSERT
- LOCK SEQUENCE
- MOVE cursor
- MOVE query
- PREPARE
- REASSIGN OWNED BY
- RELEASE SAVEPOINT
- RESET
- RESTART query
- REVOKE
- REVOKE SSL TRUST
- ROLLBACK
- ROLLBACK DATABASE
- ROLLBACK TO SAVEPOINT
- SAVEPOINT
- SELECT
  - WITH Clause
  - SELECT List
  - FROM Clause
  - WHERE Clause
  - GROUP BY Clause
  - HAVING Clause
  - WINDOW Clause
  - ORDER BY Clause
  - LIMIT Clause
  - OFFSET Clause
  - UNION, INTERSECT, EXCEPT
  - Subqueries
    - Subquery Examples
    - Correlated Subqueries
- SELECT INTO
- SET
- SET ROLE
- SET SESSION AUTHORIZATION
- SHOW
- SHOW SSL CA
- SHOW SSL IDENTITY
- SHOW SSL SYSTEM
- SHOW SSL TRUST
- START TRANSACTION
- TABLE
- TRUNCATE
- UPDATE
- SQL Functions and Operators

- Supported Functions and Return Types
- SQL Conditions
  - Logical Operators
  - Comparison Operators
  - ANY
  - BETWEEN
  - EXISTS
  - IN
- String Functions
  - ASCII
  - BIT_LENGTH
  - BTRIM
  - CHAR_LENGTH
  - CHR
  - CONCAT, ||
  - CONCAT_WS
  - DECRYPT (deprecated)
  - DECRYPT_KS
  - DLE_DST
  - ENCODE
  - ENCRYPT (deprecated)
  - ENCRYPT_KS
    - ENCRYPT_KS Examples
    - Encryption and Decryption Algorithms
  - HASH
  - HASH4
  - HASH8
  - HMAC
  - HMAC_KS
  - INITCAP
  - INSTR
  - LE_DST
  - LEFT
  - LENGTH
  - LOWER
  - LPAD
  - LTRIM, RTRIM
  - MD5
  - NYSIIS
  - OCTET_LENGTH
  - POSITION
  - REPEAT
  - REPLACE
  - REVERSE
  - RIGHT
  - RPAD
  - SPLIT_PART
  - STRPOS
  - SUBSTR
  - SUBSTRING
  - TO_ASCII
  - TO_HEX
  - TRANSLATE
  - TRIM
  - UPPER
- Pattern Matching
  - SQL Operators and Pattern Matching Functions

        - SQL Regex Operators
        - REGEXP_REPLACE
        - REGEXP_INSTR
        - REGEXP_EXTRACT
        - REGEXP_LIKE
        - LIKE
        - SIMILAR TO
        - SIMILAR TO: Best Practices
        - SUBSTRING (Pattern Match)
        - Simple Regex Examples
      - Regular Expression Details
        - Character Classes and Escaped Special Characters
        - Quantifiers
        - Anchors
        - Grouping Constructs
        - Alternation Constructs
        - Back References
        - Flags
- Datetime Functions
  - ADD_MONTHS
  - AGE
  - AT TIME ZONE
  - CURRENT_DATE
  - CURRENT_TIMESTAMP
  - CURRENT_UTC_TIMESTAMP()
  - DATEADD
  - DATEDIFF
  - DATENAME
  - DATE_PART
  - DATE_TRUNC
  - DAY
  - DAYS_BETWEEN
  - EOMONTH
  - EXTRACT
  - GETDATE
  - JUSTIFY_DAYS, _HOURS, _INTERVAL
  - LAST_DAY
  - LOCALTIME
  - LOCALTIMESTAMP
  - MAKE_DATE
  - MAKE_INTERVAL
  - MAKE_TIME
  - MAKE_TIMESTAMP
  - MAKE_TIMESTAMPTZ
  - MONTH
  - MONTHS_BETWEEN
  - NEXT_DAY
  - NOW()
  - OVERLAPS
  - TIMEOFDAY()
  - TIMEZONE
  - YEAR
  - Supported Dateparts
- Mathematical Functions
  - Mathematical Operators
  - ABS
  - ACOS

    - ASIN
    - CBRT
    - CEIL
    - COS
    - DEGREES
    - DIV
    - ERF
    - EXP
    - FLOOR
    - LN
    - LOG
    - MOD
    - POWER
    - PROBNORM
    - RADIANS
    - RANDOM()
    - ROUND
    - SIGN
    - SIN
    - SQRT
    - TRUNC
- Aggregate Functions
    - AVG
    - COUNT
    - COUNT(DISTINCT)
    - GROUP_CONCAT
    - LISTAGG
    - MAX
    - MEDIAN()
    - MIN
    - PERCENTILE_CONT
    - PERCENTILE_DISC
    - STDDEV_SAMP and STDDEV_POP
    - STRING_AGG
    - SUM
    - VAR_SAMP and VAR_POP
- Window Functions
    - Syntax for Window Functions
    - 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)
    - VAR_SAMP and VAR_POP (window functions)
- Conditional Expressions

    - CASE
    - COALESCE
    - DECODE
    - GREATEST and LEAST
    - IIF
    - ISNULL
    - NULLIF
    - NVL
    - NVL2
- Formatting Functions
    - FLOAT4, FLOAT8
    - INT2, INT4, INT8
    - TO_CHAR
    - TO_DATE
    - TO_NUMBER
    - TO_TIMESTAMP
    - Formats for Datetime Values
    - Formats for Numeric Values
- Type-Safe Casting Functions
    - SAFE_TO_DATE
    - SAFE_TO_DOUBLE
    - SAFE_TO_TIMESTAMP
- JSON Functions
    - JSON_ARRAY_STR
    - JSON_INGEST
    - JSON_LOOKUP
    - JSON_OBJECT_STR
- CONVERT (SQL Server Migration Function)
- GETBIT Function
- NEXTVAL Function
- System Functions
    - CURRENT_DATABASE()
    - CURRENT_QUERY()
    - CURRENT_SCHEMA
    - CURRENT_SCHEMAS()
    - CURRENT_SETTING
    - CURRENT_USER
    - GETDATABASEENCODING()
    - HAS_DATABASE_PRIVILEGE
    - HAS_FUNCTION_PRIVILEGE
    - HAS_KEY_PRIVILEGE
    - HAS_ROLE_PRIVILEGE
    - HAS_SCHEMA_PRIVILEGE
    - HAS_SYSTEM_PRIVILEGE
    - HAS_TABLE_PRIVILEGE
    - PG_ADVISORY_LOCK
    - PG_ADVISORY_UNLOCK
    - PG_HAS_ROLE
    - PG_RELOAD_CONF()
    - PG_TRY_ADVISORY_LOCK
    - SESSION_USER
    - sys.execution_id()
    - sys.filesys_number_from_rowid
    - sys.gen_random_uuid()
    - sys.inode_number_from_rowid
    - sys.ldap_sync
    - sys.oldest_rollback_point_id_in_replica

        - sys.row_number_from_rowid
        - sys.worker_id()
        - sys.worker_id_from_rowid
        - sys.worker_uuid
        - VERSION()
        - yb_terminate_session
      - Network Address Functions
        - CONTAINS
        - HOSTMASK
        - INET_MERGE
        - IPV4_GET_INET, IPV6_GET_INET
        - IPV4_GET_MASKBITS, IPV6_GET_MASKBITS
        - IPV4_NETMASK, IPV6_NETMASK
        - MACADDR8_SET7BIT
        - TEXT
        - TRUNC
    - SQL Reserved Words
- Error Codes
- Glossary
- Notices and Licenses
  - Third Party Licenses
  - OpenSSL License
  - Third Party Library Notices

System Management

This section covers Yellowbrick appliance tasks from a system management perspective, including details on maintenance and failover operations.

In This Section

- External Interfaces
- System Maintenance
- System Hardware Encryption
- Manager Node Failover
- Starting and Stopping the Database
- Powering the Appliance Off and On
- Checking the Logs
- User Accounts
- Using the System Management Console
- Chassis Management
- System Alerts
- Remote Diagnostics
- Using Network Tracing
- Manager Node Ports
- ybcli Reference

Parent topic: Yellowbrick Documentation

External Interfaces

The following Yellowbrick external interfaces are built-in parts of the system that are intended for use by application developers, system administrators, and DBAs. Customers typically set up their own BI tools and applications for end-user access.

Applications that connect to the appliance include SQL tools, the bulk loader, and the System Management Console (SMC).

System Management Console (SMC)

The SMC is a user interface for system configuration, cluster management, and monitoring activities on Yellowbrick databases. You can use the SMC to monitor resource utilization, configure workload management (WLM), analyze graphical query plans, and monitor query performance. See Using the System Management Console.

---

ybcli

The `ybcli` is the system management CLI for Yellowbrick appliances. This CLI runs only on the manager nodes and is the main tool for managing the appliance at the system and hardware level. See ybcli Reference.

SQL Client (ybsql)

The `ybsql` tool is the command-line SQL interface for Yellowbrick databases. You can use `ybsql` as a standard SQL entry tool to create and modify database objects (users, tables, views), run queries, and maintain the database. Yellowbrick supports ANSI SQL with some customizations. See ybsql Reference.

Bulk Load and Unload Clients (ybload, ybunload)

The `ybload` and ybunload utilities are Java-based programs that run on systems outside the Yellowbrick appliance.

You use `ybload` to run bulk loads into database tables, reading the data from local or remote storage systems. Bulk loads bypass the manager node and distribute data directly to the worker nodes. You can download the client from the SMC. Bulk Loading Tables.

You use `ybunload` to unload the contents of a specific table or the results of a given query. The output files are exported to local storage or object storage systems. See Unloading Data.

Backup and Restore Clients

The backup and restore clients are Java-based programs that run on client systems:

- `ybbackup` backs up one or more databases
- `ybrestore` restores databases from backups
- `ybbackupctl` returns information about existing backups

See Backup and Restore.

ODBC, JDBC, and ADO.NET Drivers

These drivers provide standard modes of connectivity to Yellowbrick databases from analytic applications. You can download each driver from the SMC. See Client Tools and Drivers.

Parent topic: System Management

System Maintenance

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: TAA16(

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 the event 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.

Parent topic: System Management

System Hardware Encryption

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 the event 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.

In This Section

- Setting Up Encryption
- Managing Encrypted Drives
- Example: Rotate the Keystore with Multiple Unlock Keys

Parent topic: System Management

Setting Up Encryption

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 wish 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.

In This Section

- Example: Set Up the Keystore and Enable Encryption
- Example: Use Multiple Unlock Keys
- Example: Force Keystore Setup

Parent topic: System Hardware Encryption

Example: Set Up the Keystore and Enable Encryption

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 administra
using a combination of keys. No single key has to be distributed to a single admini

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 secu

The following key is used to authenticate to the keystore (required by any encrypti
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: 5278a49d2180e1b2fa841eebc01a1ba9ad551e33ebef1f218fcef3db9f28

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 encryptic
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: 5278a49d2180e1b2fa841eebc01a1ba9ad551e33ebef1f218fcef3db9f28
```

---

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@keysad

keysadmin@keysadmin.yellowbrick.io's password:
ybd-ks-11-03-2019-11-27-59.tar.gz                                    100%   22KB  2
```

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
Also make sure to have the keys for the keystore available in a secure place. These

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 no

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 e

Determining system status
Retrieving blade details. Standby...

No blades are currently using encryption. Enabling encryption initially will take th

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: REDACTE
        Enabling encryption. Standby... Encryption enabled
    Drive-sdb - Serial: S3D2NX0J507787F -> Generating key. Standby...  KEY: REDACTE
        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
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 no

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 e

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
...
```

Parent topic: Setting Up Encryption

Example: Use Multiple Unlock Keys

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 administra
using a combination of keys. No single key has to be distributed to a single admini

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 secu

The following 3 keys are used to unlock the keystore after system bootup or failove
Note: 3 of 5 key(s) are required for unlocking the keystore.
Keystore unlock key 1: 6bfeae96d9953905ba36a7683b7c2c76648cc91d71adff29ce3155089d06
Keystore unlock key 2: 514f10d8aec2d6719bbaf898fdb236b6eedb716a2df8d04e05fb076631db
Keystore unlock key 3: 69cea90c851a8bf7abab7684677f8fa6db036c1794f03da6137ce084a0c1
Keystore unlock key 4: c50b770062675d6716e5313390dc6e29077e87eb388dacc8ed1d75a6cddd
Keystore unlock key 5: a8127657aa0b4a973e151351b9c511e2f112ff2fdd1056cb25b6bf055113
```

```
The following key is used to authenticate to the keystore (required by any encryptio
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
Also make sure to have the keys for the keystore available in a secure place. These
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
```

```
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
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
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 e
```

Parent topic: Setting Up Encryption

Example: Force Keystore Setup

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 c
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 y
 -> If you cannot restore the keystore, each encrypted drive will have to be manuall

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 administra
using a combination of keys. No single key has to be distributed to a single adminis

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 secur

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: 1c980bf78d027facb30e5ca2df401c1227d3292690a527566e24098955e01
```

```
The following key is used to authenticate to the keystore (required by any encryptio
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
```

Parent topic: Setting Up Encryption

Managing Encrypted Drives

The following procedures explain what you need to do to return encrypted drives to an operational state in the event 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.

Parent topic: System Hardware Encryption

Example: Rotate the Keystore with Multiple Unlock Keys

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 keys
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.ta
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 (ct
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 administra
using a combination of keys. No single key has to be distributed to a single admini

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 unlo
```

```
New keystore unlock key 1: fc243401f27cb29d8da94ff03df9facc5f1806511a3b05f02971d495
New keystore unlock key 2: 778d2a3724b7f4421c54d3dc4323809c6253b5cea706478c1160702b
New keystore unlock key 3: 5c83c05756909d209b130210a39b2693316c17a0ec55684f98ff28df

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
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 pro

Do you want to unlock the keystore again?
Note: This is not required unless you intend to run additional encryption commands.
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 l
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
```

Parent topic: System Hardware Encryption

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.

In other words, `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 into the primary node directly when running the `system failover` command because the floating IP 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.

Parent topic: System Management

Starting and Stopping the Database

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 worker 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.

Parent topic: System Management

Powering the Appliance Off and On

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 cert warning because BMC uses a self-signed cert. You can ignore the cert 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-hand 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 into 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.

Parent topic: System Management

Checking the Logs

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.

Parent topic: System Management

User Accounts

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.

Type	Username	Description
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
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`).

Type	Username	Description
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. - 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.

Parent topic: System Management

Using the System Management Console

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

- Console Login
- Introduction to the SMC
- Configuration Settings in the SMC

Parent topic: System Management

Console Login

To log into 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.

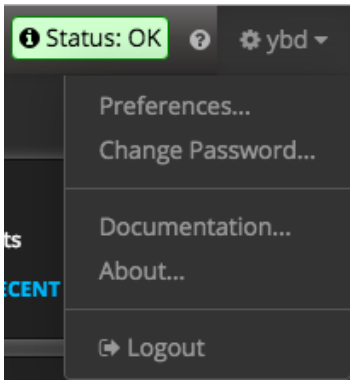


Note: The SMC is started automatically as part of the database software stack when the database is started with:

```
ybcli database start
```

See ybcli Reference.

Tip: You can access the complete product documentation from the menu in the top-right corner of the console:



Parent topic: Using the System Management Console

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 worker level.

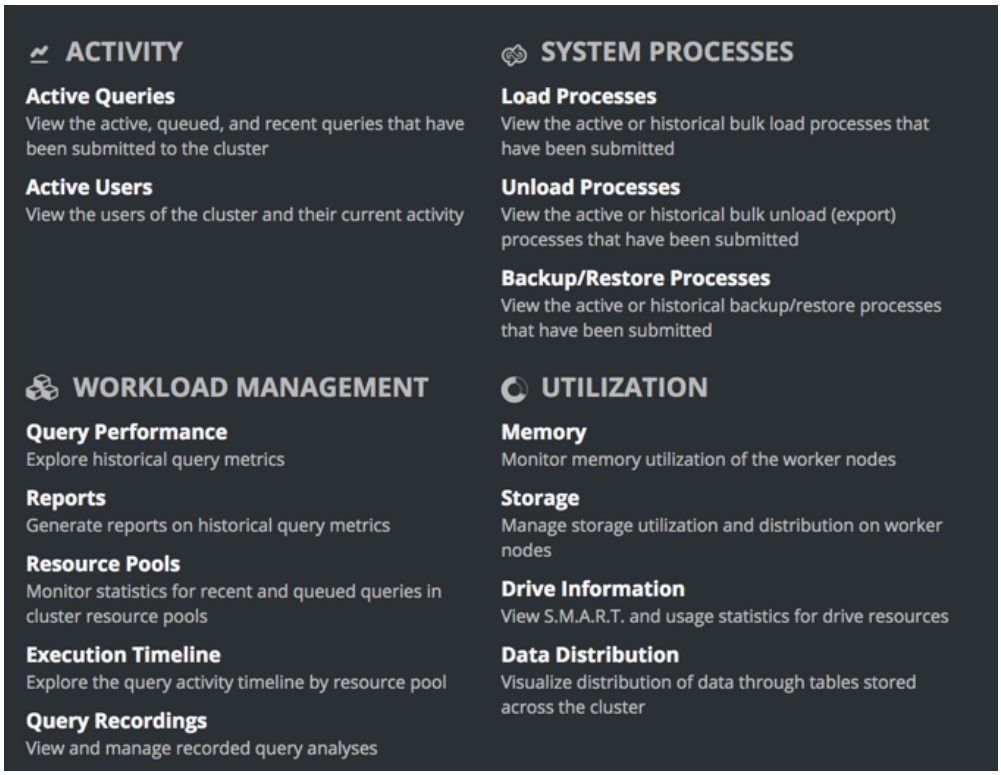
Databases

Connect to existing databases and create new databases. Look at database users and groups and the data distribution statistics for tables.

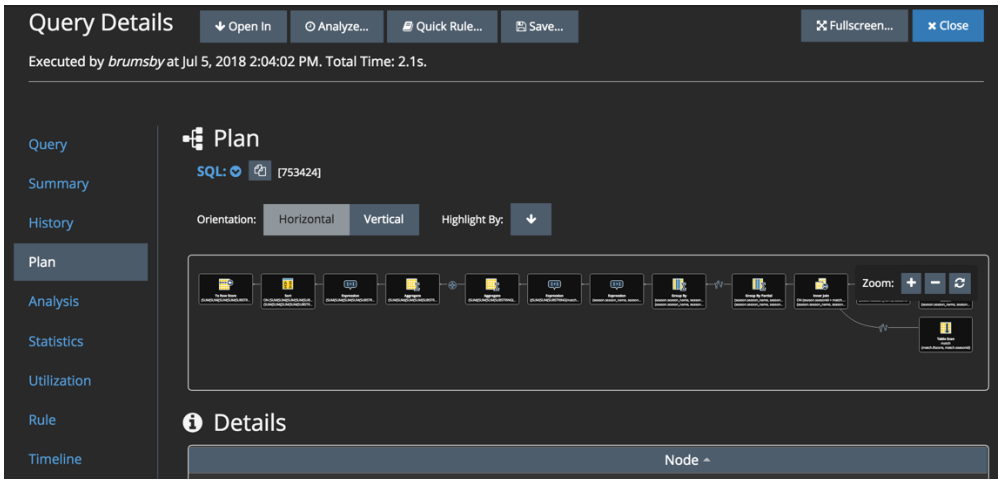


Monitor

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.

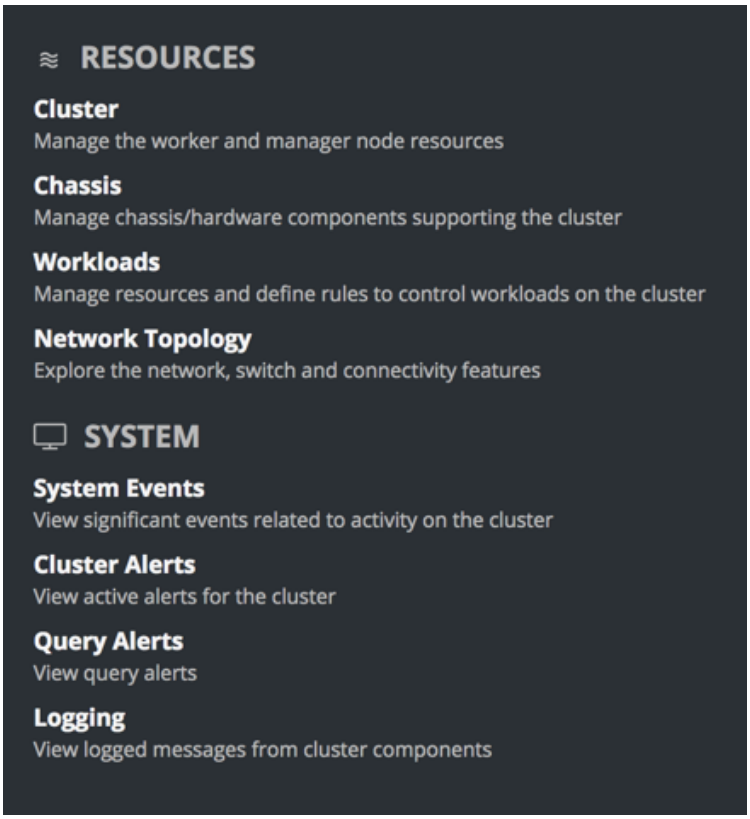


For example, here is the graphical query plan for a specific query that was run by a client user (**Monitor** > **Query Performance** > **Query Details** > **Plan**):



Manage

Manage cluster resources, including the manager and worker 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.

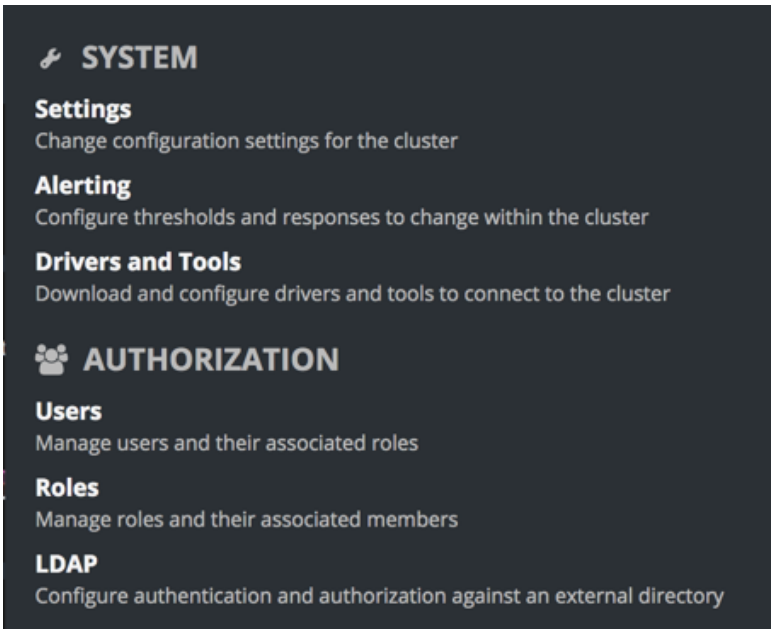
Download the following drivers and tools:

- 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 into 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.

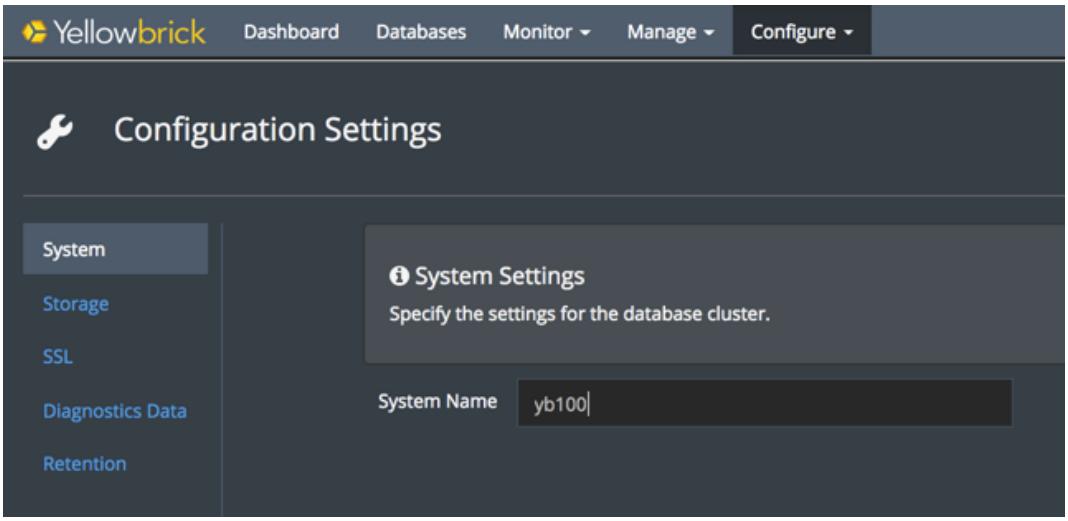


Passwords for SMC users must be at least 8 characters long and contain at least one uppercase character, one lowercase character, one numeric character, and one punctuation character. Special characters are also allowed (such as `@` and `$`).

Parent topic: Using the System Management Console

Configuration Settings in the SMC

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 top-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.

Parent topic: Using the System Management Console

Chassis Management

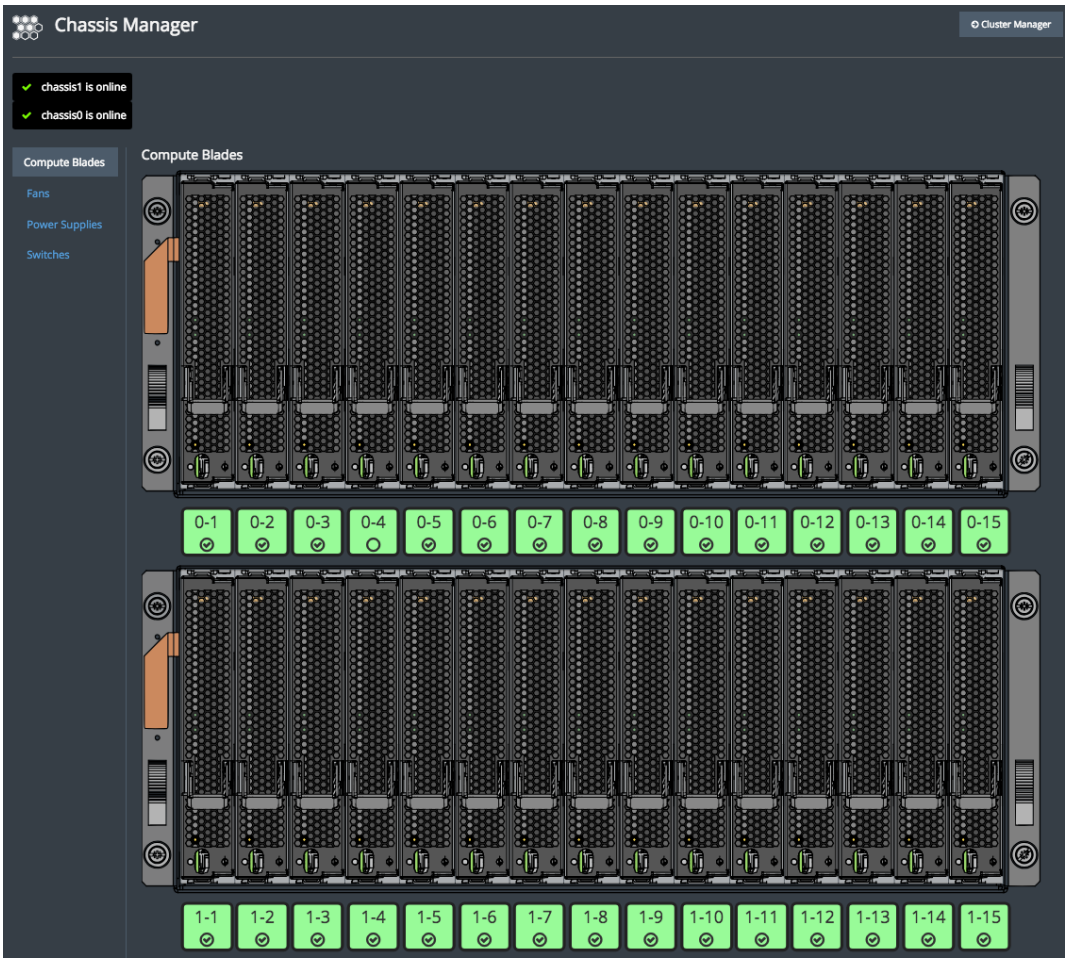
The Chassis Manager page displays the real-time status of each hardware component in the appliance (in the SMC, go to **Manage** > **Chassis**). You can use this page to monitor the current state of the appliance and investigate error conditions.

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

Graphic	Physical Status	Power Status
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 appliance itself. Each blade has three LED lights: green, red, and blue. The states of each light are: Off, On, Fast blink, Slow blink. 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 software worker 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 worker online| |Red|Software worker in error state| |Dark gray|Software worker offline| |Light gray|Compute blade physically removed| |Blue|New compute blade physically inserted but not added to the cluster|

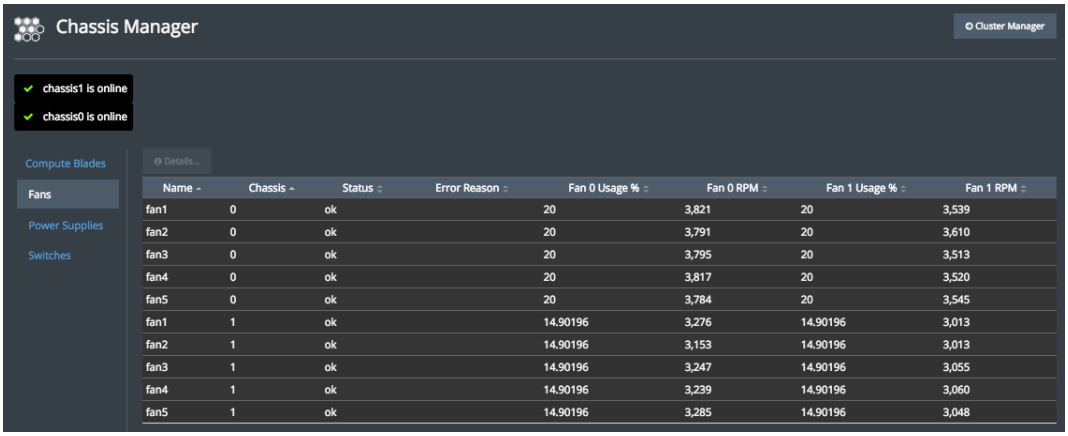
Icon Indicator

|Icon|Description| |----|-----------| |Circle with checkbox|Software worker registered OK| |Circle with no checkbox|Software worker registered as spare| |Circle with `X`|Cluster unable to communicate with software worker on compute blade| |Triangle with `!`|Software worker 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:



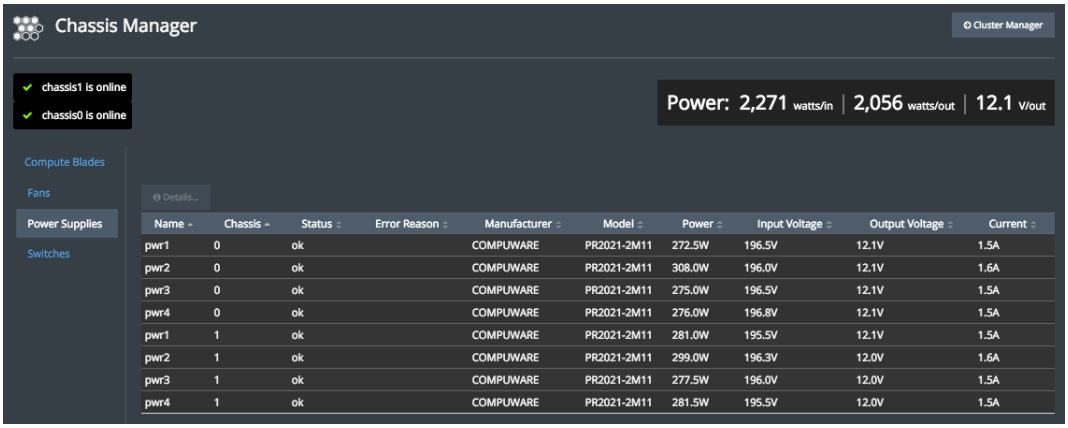
Name	Chassis	Status	Error Reason	Fan 0 Usage %	Fan 0 RPM	Fan 1 Usage %	Fan 1 RPM
fan1	0	ok		20	3,821	20	3,539
fan2	0	ok		20	3,791	20	3,610
fan3	0	ok		20	3,795	20	3,513
fan4	0	ok		20	3,817	20	3,520
fan5	0	ok		20	3,784	20	3,545
fan1	1	ok		14.90196	3,276	14.90196	3,013
fan2	1	ok		14.90196	3,153	14.90196	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

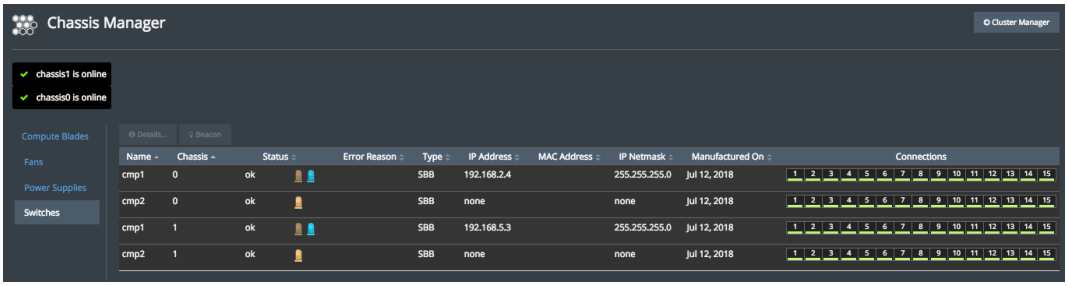
This tab shows the status of each power supply in the appliance. Appliances may have 2 to 4 power supplies per chassis depending on the number of compute blades that are installed. Power supplies are redundant and provide n+2 protection in 15-node configurations. Information on the status, manufacturer 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:



Parent topic: System Management

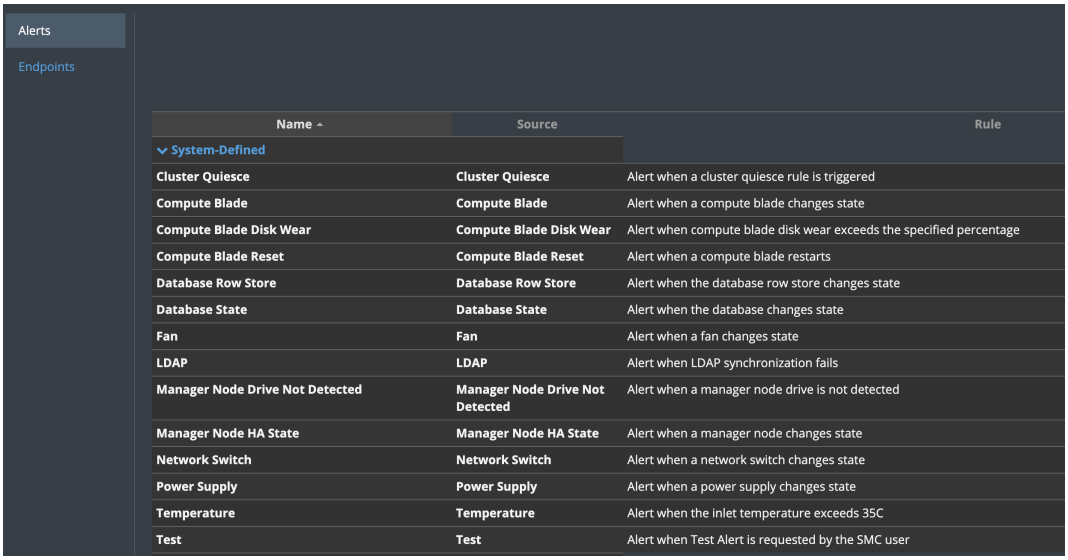
System Alerts

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 into 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.

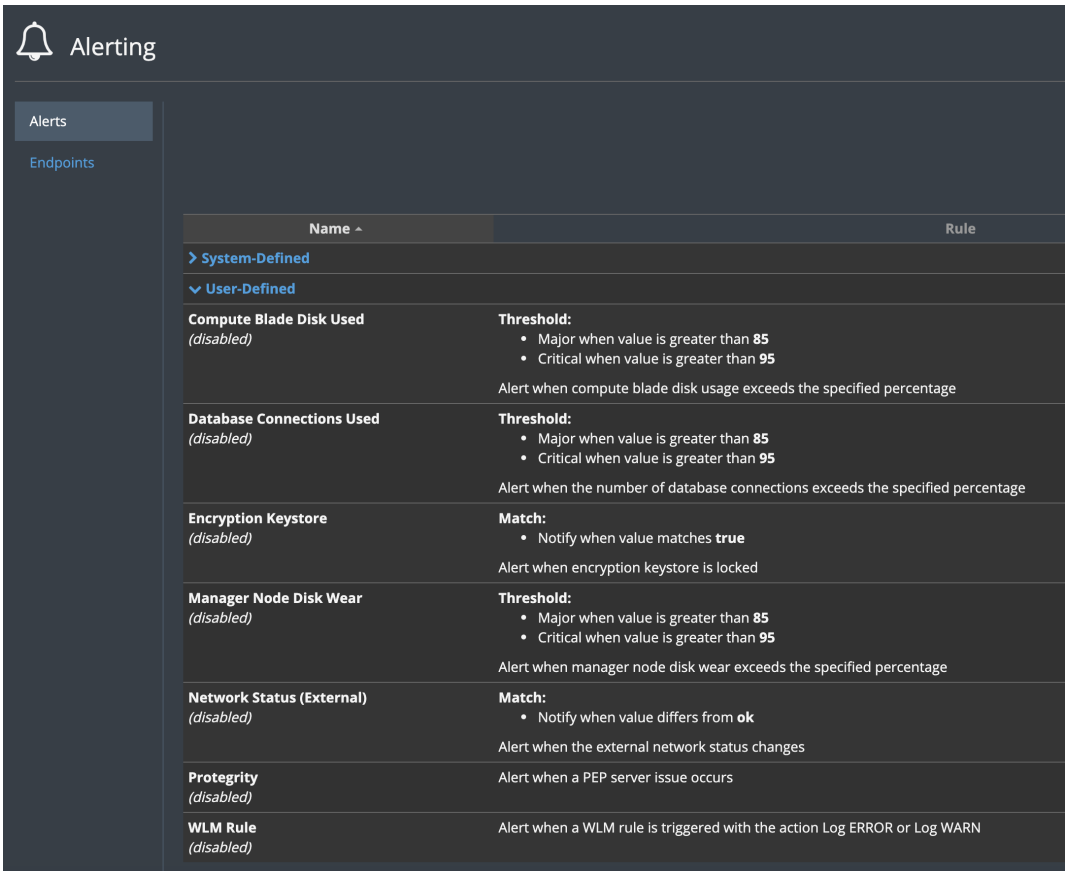
System-Defined and User-Defined Alerts

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.

In This Section

- Alert Types
- Creating an Alert Endpoint
- Managing Alerts

Parent topic: System Management

Alert Types

This section describes the specific system-defined and user-defined alerts you can configure on your appliance.

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 cancelled. 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`

Alert Name	Rule	Resource ID in Alert Messages
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`
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`

Alert Name	Rule	Resource ID in Alert Messages
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`

Alert Name	Rule	Resource ID in Alert Messages
Temperature	Alert when the inlet temperature for the system exceeds 35C.	`database:temperature`
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

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_`
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 numb number, then `usage`. For e `chassis0:blade9:drive`
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 numb number, then `wear`. For ex `chassis0:blade9:drive`
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` depending on which manage the primary.
Network Status (External)	Alert when the external network status changes.	`manager#:external_bo`

Alert Name	Rule	Resource ID in Messages
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` where `SELECT * rule` is of a WLM rule that was trigge 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`

Alert Name	Rule	Resource ID in Messages
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**.

Parent topic: System Alerts

Creating an Alert Endpoint

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.

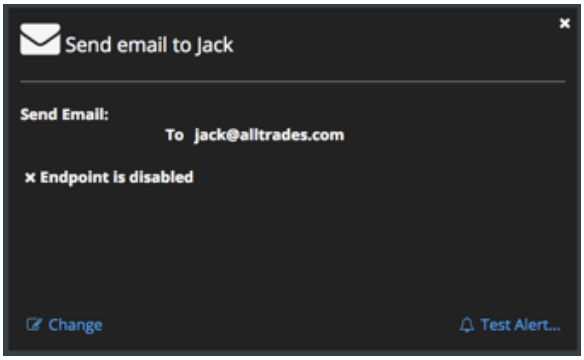
- Send Email
- Send an SNMP Trap
- Post to Web Service (http)
- Notify Yellowbrick Support

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 top-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**.

In This Section

- Endpoint Types
- Alert Templates

Parent topic: System Alerts

Endpoint Types

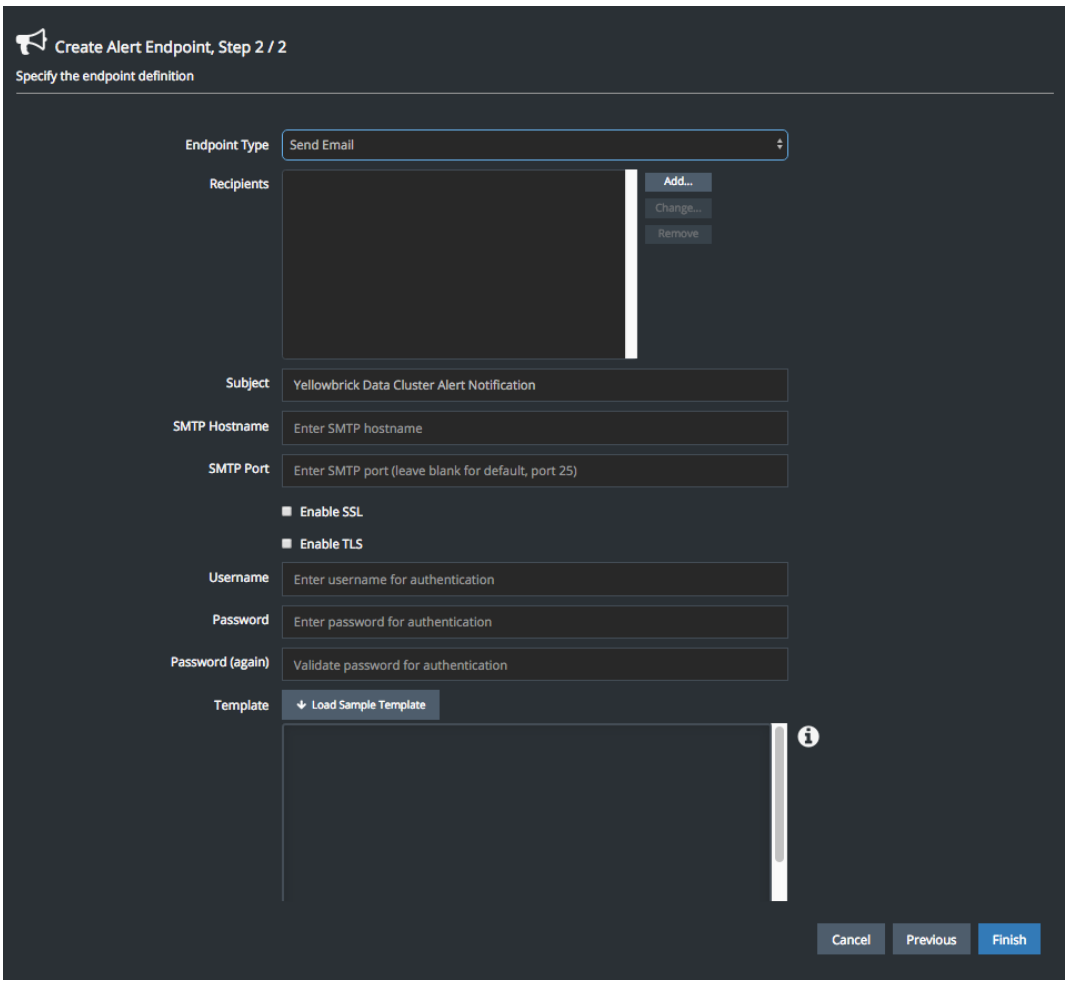
Yellowbrick supports four different endpoint types:

- 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.

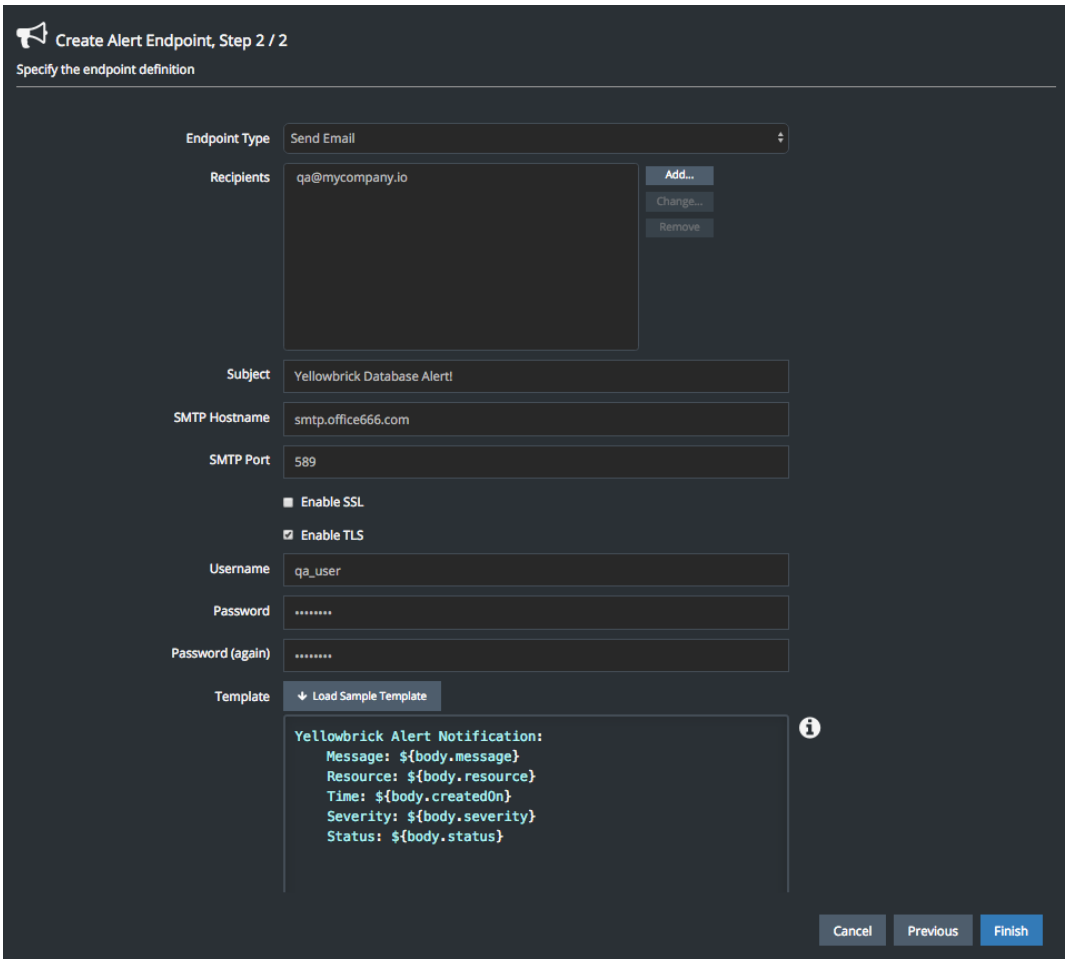
Send Email

The following configuration details are required for this endpoint type:

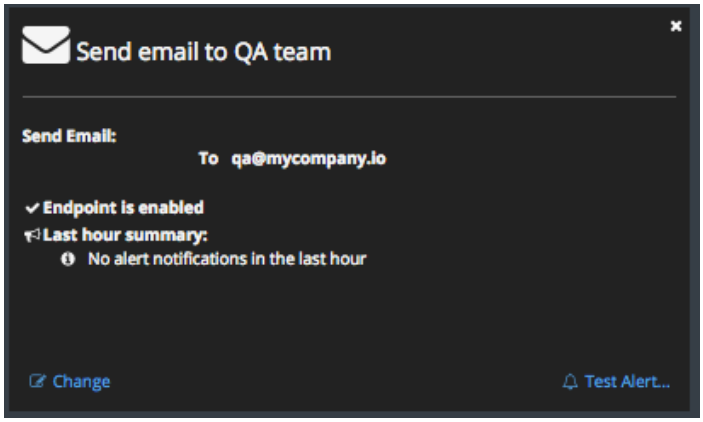


Add email addresses by clicking the **Add** button. You cannot type into the `Recipients` field. Be sure to provide accurate credentials (username/password) and the correct SMTP hostname and port number for your mail server.

You can change the subject line that will be used for email alerts. For example:

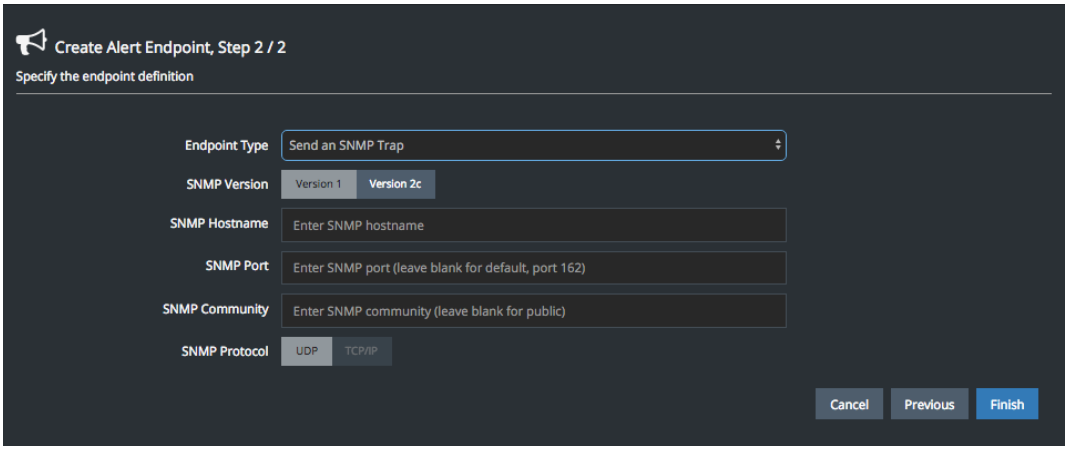


For information about Load Sample Template choices, see Alert Templates. After clicking **Finish** on this screen, you can see the endpoint on the **Endpoints** tab of the main **Alerting** screen:



Send an SNMP Trap

The following configuration details are required for this endpoint type:

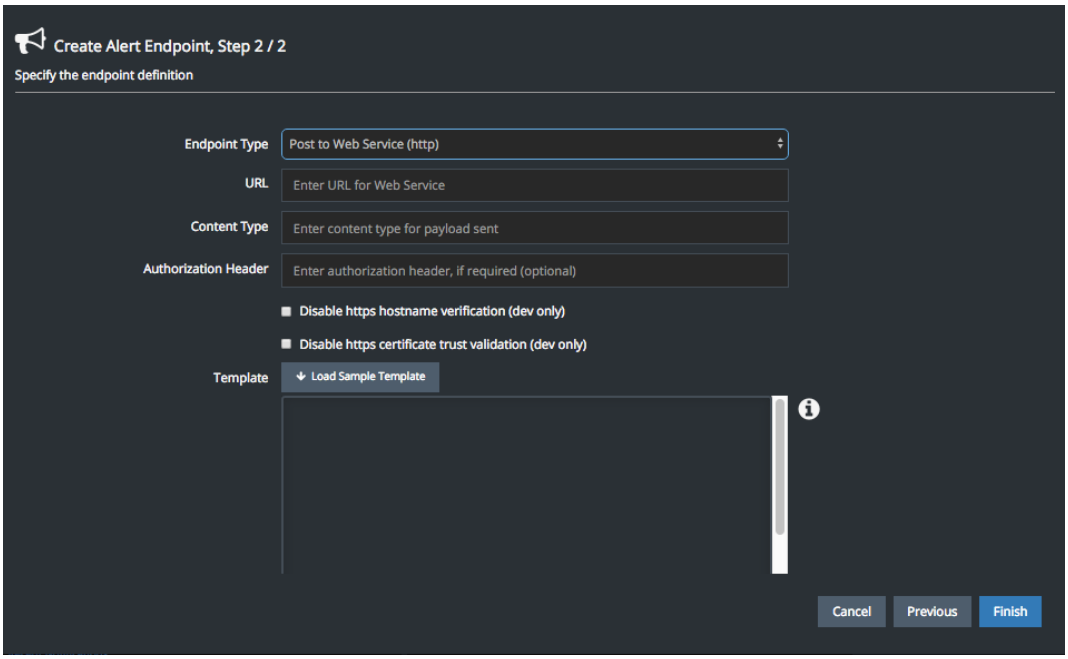


SNMP Versions 1 and 2c are supported. Version 1 supports the UDP protocol. Version 2c supports UDP or TCP/IP.

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.

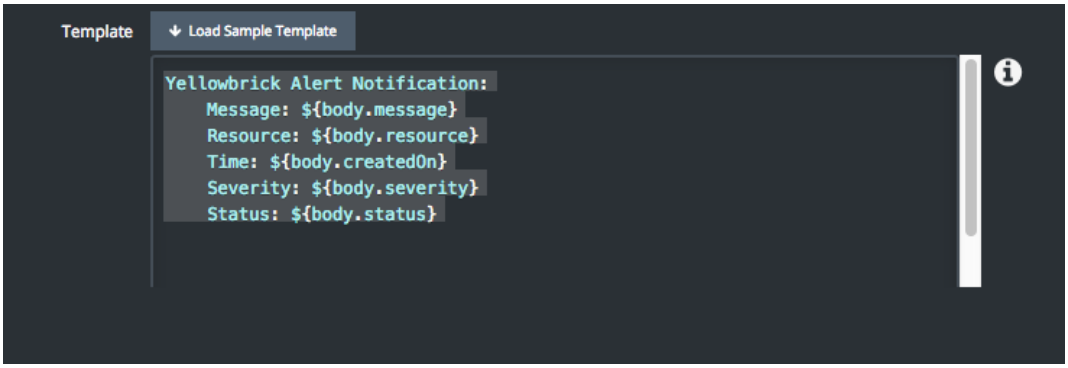
Parent topic: Creating an Alert Endpoint

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
   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.

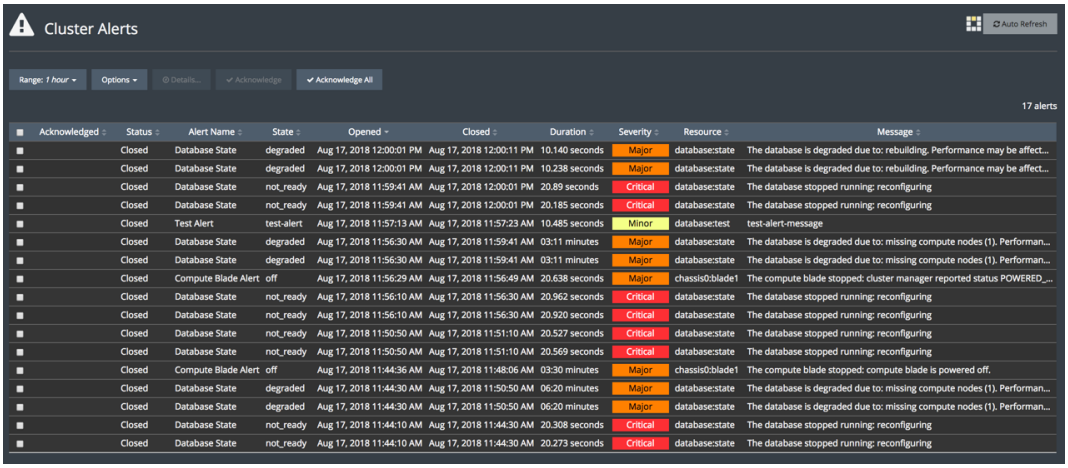
Parent topic: Creating an Alert Endpoint

Managing Alerts

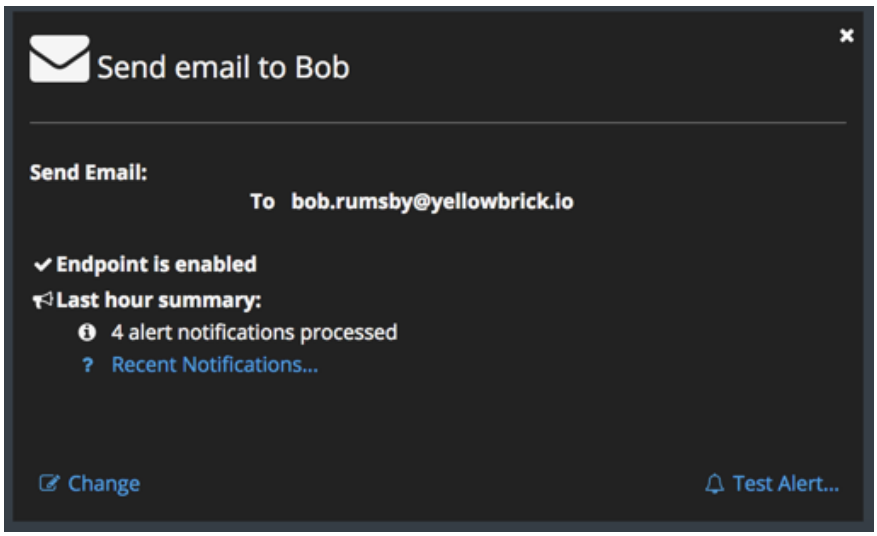
To manage active and logged alerts, you can use the SMC views, or you can query system views.

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]
...
```

Parent topic: System Alerts

Remote Diagnostics

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.

In This Section

- Diagnostics Service Levels
- Enabling Remote Diagnostics
- Configuring Remote Diagnostics

Parent topic: System Management

Diagnostics Service Levels

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.

Service Level	Description
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.

Parent topic: Remote Diagnostics

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.

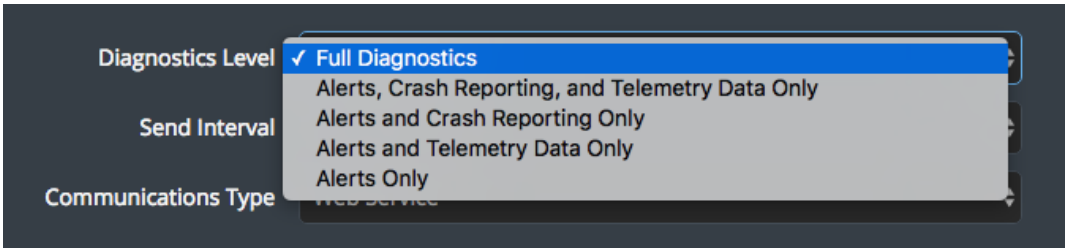
Parent topic: Remote Diagnostics

Configuring 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 into 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**.)

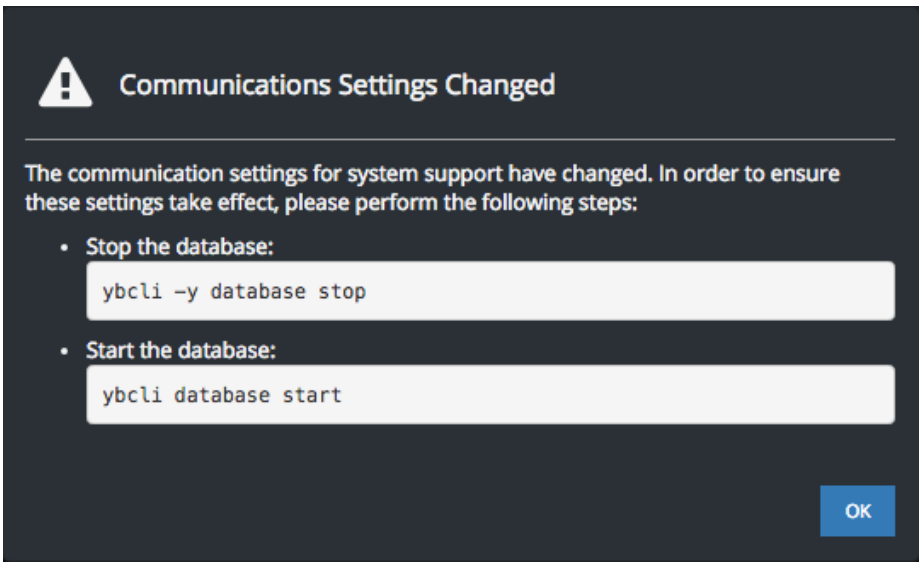
- If you selected **Email**, fill out the requested details, including the SMTP server and the recipient.

  Note: When the **Recipients** field is left blank, `phonehome@yellowbrick.io` is the default recipient. If you enter any other email addresses where you would like 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 top-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.



Parent topic: Remote Diagnostics

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.

Parent topic: System Management

Manager Node Ports

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.` - 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 contact a different NTP server
Red Hat Subscription	`subscription.rhn.redhat.com :443`	TCP	Can be blocked; not needed.

Service	Host:port	Protocol	Purpose/Notes
Yellowbrick remote diagnostics (phonehome)	`phonehome.yellowbrick.com:443`	TCP	Only needed if phonehome is enabled; it is not enabled by default. See Remote Diagnostic for configuration instructions.

Parent topic: System Management

ybcli Reference

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  regis
```

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.

In This Section

- blade
- clear
- config
- database
- encryption
- exit, quit
- health
- help

- keystore
- log
- manager
- status
- system

Parent topic: System Management

blade

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 worker 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: 00000000-0000-0000-0000-38B8

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 worker 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 recov
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
              Pass 1 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-
              Pass 2 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-
              Pass 3 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-
              DRIVE-0 - Serial: S3EWNX0JC12534E -> ALL 3 ERASE PASSES SUCCEEDE
       DRIVE-1 - Serial: S3EWNX0K200853L -> 3 pass erase starting [2019-09-19
              Pass 1 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-
              Pass 2 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-
              Pass 3 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-
              DRIVE-1 - Serial: S3EWNX0K200853L -> ALL 3 ERASE PASSES SUCCEEDE
       DRIVE-2 - Serial: S3EWNX0K202426R -> 3 pass erase starting [2019-09-19
              Pass 1 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-
              Pass 2 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-
```

```
               Pass 3 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-
               DRIVE-2 - Serial: S3EWNX0K202426R -> ALL 3 ERASE PASSES SUCCEEDE
        DRIVE-3 - Serial: S3EWNX0K200684P -> 3 pass erase starting [2019-09-19
               Pass 1 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-
               Pass 2 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-
               Pass 3 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-
               DRIVE-3 - Serial: S3EWNX0K200684P -> ALL 3 ERASE PASSES SUCCEEDE
        DRIVE-4 - Serial: S3EWNX0JC12542B -> 3 pass erase starting [2019-09-19
               Pass 1 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-
               Pass 2 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-
               Pass 3 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-
               DRIVE-4 - Serial: S3EWNX0JC12542B -> ALL 3 ERASE PASSES SUCCEEDE
        DRIVE-5 - Serial: S3EWNX0K100445J -> 3 pass erase starting [2019-09-19
               Pass 1 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-
               Pass 2 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-
               Pass 3 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-
               DRIVE-5 - Serial: S3EWNX0K100445J -> ALL 3 ERASE PASSES SUCCEEDE
        DRIVE-6 - Serial: S3EWNX0K100372P -> 3 pass erase starting [2019-09-19
               Pass 1 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-
               Pass 2 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-
               Pass 3 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-
               DRIVE-6 - Serial: S3EWNX0K100372P -> ALL 3 ERASE PASSES SUCCEEDE
        DRIVE-7 - Serial: S3EWNX0JC12602K -> 3 pass erase starting [2019-09-19
               Pass 1 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-
               Pass 2 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-
               Pass 3 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 800-
               DRIVE-7 - Serial: S3EWNX0JC12602K -> ALL 3 ERASE PASSES SUCCEEDE
        All 8 drives on blade in chassis: 1 bay:  1 successfully erased with 3
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 zeroes 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 recov
Clear erase will destroy all data on the blade by writing zeros to all bytes o
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
        Retrieving drive details. Standby... Done
        Affected Inventory of Disks found on the Yellowbrick Appliance host:yb
        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 st
                Pass 1 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 80
                  512110190592 bytes written to Drive 0 (Clear) completed at 2
                  Cryptographic Key Delete and Disk Erase (Purge) completed at
                Pass 2 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 80
                  1024220381184 bytes written to Drive 0 (Clear) completed at
                  Cryptographic Key Delete and Disk Erase (Purge) completed at
                Pass 3 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 80
                  1536330571776 bytes written to Drive 0 (Clear) completed at
                  Cryptographic Key Delete and Disk Erase (Purge) completed at
                DRIVE-0 - Serial: S469NF0K510756B -> ALL 3 ERASE PASSES SUCCEE
        DRIVE-1, Chassis 0, Bay 1 - Serial: S469NF0K510762E -> 3 pass erase st
                Pass 1 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 80
                  512110190592 bytes written to Drive 1 (Clear) completed at 2
                  Cryptographic Key Delete and Disk Erase (Purge) completed at
                Pass 2 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 80
                  1024220381184 bytes written to Drive 1 (Clear) completed at
                  Cryptographic Key Delete and Disk Erase (Purge) completed at
                Pass 3 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 80
                  1536330571776 bytes written to Drive 1 (Clear) completed at
                  Cryptographic Key Delete and Disk Erase (Purge) completed at
                DRIVE-1 - Serial: S469NF0K510762E -> ALL 3 ERASE PASSES SUCCEE
        DRIVE-2, Chassis 0, Bay 1 - Serial: S469NF0K501083Z -> 3 pass erase st
                Pass 1 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 80
                  512110190592 bytes written to Drive 2 (Clear) completed at 2
                  Cryptographic Key Delete and Disk Erase (Purge) completed at
                Pass 2 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 80
                  1024220381184 bytes written to Drive 2 (Clear) completed at
                  Cryptographic Key Delete and Disk Erase (Purge) completed at
                Pass 3 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 80
                  1536330571776 bytes written to Drive 2 (Clear) completed at
                  Cryptographic Key Delete and Disk Erase (Purge) completed at
                DRIVE-2 - Serial: S469NF0K501083Z -> ALL 3 ERASE PASSES SUCCEE
        DRIVE-3, Chassis 0, Bay 1 - Serial: S469NF0K501067B -> 3 pass erase st
                Pass 1 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 80
                  512110190592 bytes written to Drive 3 (Clear) completed at 2
                  Cryptographic Key Delete and Disk Erase (Purge) completed at
                Pass 2 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 80
                  1024220381184 bytes written to Drive 3 (Clear) completed at
                  Cryptographic Key Delete and Disk Erase (Purge) completed at
                Pass 3 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 80
                  1536330571776 bytes written to Drive 3 (Clear) completed at
                  Cryptographic Key Delete and Disk Erase (Purge) completed at
                DRIVE-3 - Serial: S469NF0K501067B -> ALL 3 ERASE PASSES SUCCEE
        DRIVE-4, Chassis 0, Bay 1 - Serial: S469NF0K501097L -> 3 pass erase st
```

```
                Pass 1 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 80
                  512110190592 bytes written to Drive 4 (Clear) completed at 2
                  Cryptographic Key Delete and Disk Erase (Purge) completed at
                Pass 2 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 80
                  1024220381184 bytes written to Drive 4 (Clear) completed at
                  Cryptographic Key Delete and Disk Erase (Purge) completed at
                Pass 3 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 80
                  1536330571776 bytes written to Drive 4 (Clear) completed at
                  Cryptographic Key Delete and Disk Erase (Purge) completed at
                DRIVE-4 - Serial: S469NF0K501097L -> ALL 3 ERASE PASSES SUCCEE
        DRIVE-5, Chassis 0, Bay 1 - Serial: S469NF0K510767N -> 3 pass erase st
                Pass 1 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 80
                  512110190592 bytes written to Drive 5 (Clear) completed at 2
                  Cryptographic Key Delete and Disk Erase (Purge) completed at
                Pass 2 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 80
                  1024220381184 bytes written to Drive 5 (Clear) completed at
                  Cryptographic Key Delete and Disk Erase (Purge) completed at
                Pass 3 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 80
                  1536330571776 bytes written to Drive 5 (Clear) completed at
                  Cryptographic Key Delete and Disk Erase (Purge) completed at
                DRIVE-5 - Serial: S469NF0K510767N -> ALL 3 ERASE PASSES SUCCEE
        DRIVE-6, Chassis 0, Bay 1 - Serial: S469NF0K510759X -> 3 pass erase st
                Pass 1 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 80
                  512110190592 bytes written to Drive 6 (Clear) completed at 2
                  Cryptographic Key Delete and Disk Erase (Purge) completed at
                Pass 2 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 80
                  1024220381184 bytes written to Drive 6 (Clear) completed at
                  Cryptographic Key Delete and Disk Erase (Purge) completed at
                Pass 3 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 80
                  1536330571776 bytes written to Drive 6 (Clear) completed at
                  Cryptographic Key Delete and Disk Erase (Purge) completed at
                DRIVE-6 - Serial: S469NF0K510759X -> ALL 3 ERASE PASSES SUCCEE
        DRIVE-7, Chassis 0, Bay 1 - Serial: S469NF0K501064H -> 3 pass erase st
                Pass 1 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 80
                  512110190592 bytes written to Drive 7 (Clear) completed at 2
                  Cryptographic Key Delete and Disk Erase (Purge) completed at
                Pass 2 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 80
                  1024220381184 bytes written to Drive 7 (Clear) completed at
                  Cryptographic Key Delete and Disk Erase (Purge) completed at
                Pass 3 of 3 -> DRIVE ERASED SUCCESSFULLY using method: NIST 80
                  1536330571776 bytes written to Drive 7 (Clear) completed at
                  Cryptographic Key Delete and Disk Erase (Purge) completed at
                DRIVE-7 - Serial: S469NF0K501064H -> ALL 3 ERASE PASSES SUCCEE
        All 8 drives on blade in chassis: 0 bay:  1 successfully erased with 3
```

```
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

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: P
```

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:
         BIOS: v05.04.21.0038.00.011 - Memory total/free: 65587652/1870728 KiB
         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: S3EWNWAJ312
                       OK/0% 2B6QCXP7        OK/0% 2B6QCXP7        OK/0% 2B6QC
                    0: S3EWNWAJ311166N    1: S3EWNWAJ330207R    2: S3EWNWAJ311
                       OK/0% 2B6QCXP7        OK/0% 2B6QCXP7        OK/0% 2B6QC
         Cluster status: OPERATIONAL - Cluster role: MEMBER - Last seen: just
```

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 - Versi
         BIOS: Athn_1.01.3 - Memory total/free: 1056744476/18735500 KiB
         BMC: 00.17.01 - Kalidah-1: 0x0DE38309 - Kalidah-2: 0x0DE38309 - IB-FW
         CPU: AMD EPYC 7702P 64-Core Processor - Microcode: 0x8301034 - Cores:
         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: S439NE0MA00
                           OK/0% EDA5502Q        OK/0% EDA5502Q        OK/0% EDA55
                       0: S439NE0MA00442     1: S439NE0MA00253     2: S439NE0MA00
                           OK/0% EDA5502Q        OK/0% EDA5502Q        OK/0% EDA55
          Cluster status: OPERATIONAL - Cluster role: MEMBER - Last seen: just
```

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
```

Parent topic: ybcli Reference

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`.

Parent topic: ybcli Reference

config

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 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
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>
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> <ga
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.

In This Section

- config bmc
- config dba
- config firewall
- config hostname
- config keystore
- config network
- config ntp
- config passwords
- config phonehome

- config timezone
- config user

Parent topic: ybcli Reference

config bmc

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 intru
   3 | 02/24/2020 | 14:44:32 | Physical Security #0xaa | General Chassis intru
   4 | 02/24/2020 | 14:44:39 | Temperature #0x28 | Lower Critical going low |
   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.
```

Parent topic: config

config dba

Change the password of any database user account, except reserved system users (`sys_*`) and `ybadmin` .

```
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, c
You are now connected to database "yellowbrick_test" as user "ybd_test".
```

Parent topic: config

config firewall

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:
```

Parent topic: config

config hostname

Get or set host names on the manager nodes.

```
config hostname get
config hostname set [ local | remote ] <hostname>
```

get

For example:

```
YBCLI(13325) (PRIMARY - yb100-mgr0)> config hostname get

Local node : yb100-mgr0.ybroad.io
Remote node: yb100-mgr1.ybroad.io
```

set

For example:

```
YBCLI(13325) (PRIMARY - yb100-mgr0)> config hostname set local yb100-mgr0.ybro

Changing the hostname of a manager node will cause the network stack to be rec
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.ybr
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
```

Parent topic: config

config keystore

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.
```

Parent topic: config

config network

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 | activ
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>
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> <ga
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 b
Before changing the MTU, please verify the size chosen is supported on this ne

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

Successfully set default gateway
```

route create [all | local | remote] [vlanid] <ip> <subnet> <gateway>

Create a static route for the network. For example:

```
YBCLI(1484) (PRIMARY - yb100-mgr1)> config network route create local 4000 172

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> <subnet> <gateway>

Delete a static route for the network. For example:

```
YBCLI(1484) (PRIMARY - yb99-mgr1)> config network route delete local 4000 172.

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 min
       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

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 a
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 configuration (cluster floating IP) for the VLAN. For example:

```
YBCLI(1484) (PRIMARY - yb100-mgr1)> config network vlan system set 1000 172.16

Are you sure you want to set the VLAN ID:1000 floating system IP to: 172.16.20
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: 1

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
```

Parent topic: config

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. Standb

Remote manager node
-------------------

Successfully added 10.21.41.139 as NTP server. Restarting NTP services. Standb
```

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.

Remote manager node
--------------------

Removed 10.21.41.139 from NTP configuration. Restarting NTP services. Standby.
```

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
```

Parent topic: config

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)>
```

Parent topic: config

config phonehome

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.
...
```

Parent topic: config

config timezone

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 wa
Note: After this command is run, the database can be started again using the '

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)>
```

Parent topic: config

config user

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 into 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 a
```

```
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 w
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 aft
Continue (yes/no)? yes

This command allows setting individual password policies for users on this sys

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 t
Please enter the WARN time reminder (in days) for password changes (entering 0
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 man
```

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 sys

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 t
Please enter the WARN time reminder (in days) for password changes (entering 0
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)>
```

Parent topic: config

database

Start or stop the Yellowbrick database software. These commands do not start up or shut down the system. The manager and worker 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.

Parent topic: ybcli Reference

encryption

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 f

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 k
    Drive-sdb - Serial: S3D2NX0J507787F -> Disabling encryption and removing k

...

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
    Drive-1 - Serial: S3EWNWAJ318656W -> Disabling encryption and removing key
    Drive-2 - Serial: S3EWNWAJ318660F -> Disabling encryption and removing key
    Drive-3 - Serial: S3EWNWAJ318658M -> Disabling encryption and removing key
    Drive-4 - Serial: S3EWNWAJ318653X -> Disabling encryption and removing key
    Drive-5 - Serial: S3EWNWAJ318652L -> Disabling encryption and removing key
    Drive-6 - Serial: S3EWNWAJ318657P -> Disabling encryption and removing key
    Drive-7 - Serial: S3EWNWAJ318651N -> Disabling encryption and removing key

Blade in bay: 2 - UUID: 00000000-0000-0000-0000-38B8EBD0053C
    Drive-0 - Serial: S3EWNWAJ316127E -> Disabling encryption and removing key
    Drive-1 - Serial: S3EWNWAJ330178F -> Disabling encryption and removing key
    Drive-2 - Serial: S3EWNWAJ330176R -> Disabling encryption and removing key
...
```

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 syste

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 w

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 Y

Rotating keys on manager node(s). Standby...

Manager: yb00-mgr0.yellowbrick.io (local node)
    Drive-sda - Serial: S3D2NX0J601378A -> Generating key. Standby... KEY: TxX
        Rotating key. Standby...Rotated
    Drive-sdb - Serial: S3D2NX0J601380W -> Generating key. Standby... KEY: ZP2
        Rotating key. Standby...Rotated

Manager: yb00-mgr1.yellowbrick.io (remote node)
    Drive-sda - Serial: S3D2NX0J507797Y -> Generating key. Standby... KEY: 84E
        Rotating key. Standby...Rotated
    Drive-sdb - Serial: S3D2NX0J507787F -> Generating key. Standby... KEY: Hmd
        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 w

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
If the system is restarted or failed over, all keys will have to be provided a

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 (worker node) has one or more drives that are locked, that worker 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.

Parent topic: ybcli Reference

exit, quit

Exit the `ybcli` session. The `exit` and `quit` commands are synonymous.

```
YBCLI (PRIMARY)> exit
Exiting. Standby...
Bye

YBCLI (PRIMARY)> quit

Exiting. Standby...
Bye
```

Parent topic: ybcli Reference

health

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
Bay:  2 HW Status: ok    Power: on LED: [ OFF  ON   OFF  ] FRU: C4-0005-01 R01
Bay:  3 HW Status: ok    Power: on LED: [ OFF  ON   OFF  ] FRU: C4-0005-01 R01
Bay:  4 HW Status: ok    Power: on LED: [ OFF  ON   OFF  ] FRU: C4-0005-01 R01
Bay:  5 HW Status: ok    Power: on LED: [ OFF  ON   OFF  ] FRU: C4-0005-01 R01
Bay:  6 HW Status: ok    Power: on LED: [ OFF  ON   OFF  ] FRU: C4-0005-01 R01
...
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

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 R0
CMP2 Secondary Status: ok Temp: 58C LED: [ FAST SLOW OFF  ] FRU: C4-0001-03 R0

CMP alerts reported: None

Chassis:  1
```

```
-----------
CMP1 Primary   Status: ok Temp: 45C LED: [ OFF  ON   OFF  ] FRU: C4-0001-01 R0
CMP2 Secondary Status: ok Temp: 58C LED: [ OFF  SLOW OFF  ] FRU: C4-0001-04 R0

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 %

...
```

Parent topic: ybcli Reference

help

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 ]
```

Parent topic: ybcli Reference

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 encry

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 syste
    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 Y
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 querie

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 rotation requires both the authentication and unlock keys to be prese

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 syste
       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 admi
using a combination of keys. No single key has to be distributed to a single a

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
New keystore unlock key 1: 86fdbd9599bb83c732e3c4eeab2d47a1c6325c85d1c72547c4f

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 syste
       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
```

Parent topic: ybcli Reference

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> | "<s
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 YBD
2018-07-10T17:53:42.042423-07:00 ybos-00000000-0000-0000-0000-38B8EBD00C1C YBD
2018-07-10T17:53:42.042434-07:00 ybos-00000000-0000-0000-0000-38B8EBD00C1C YBD
...
```

To return logs for all blades (for all hours), use the following command:

```
log blade all all
```

You can also specify a number of hours for all blades. For example:

```
log blade 1 all
```

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 u
2017-03-14 15:05:03.788 PDT 26246 DETAIL:  Connection matched pg_hba.conf line
...
```

install

Return the log for when the installer 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

Yellowbrick Data appliance installer v2.0.0
Installing: v0.0.1-11160
Installing using the following parameters: {'core': False, 'fieldreset': False

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
2017-10-25T13:59:34.419245-07:00 yb00-mgr1 journal: YBCLI user ybdadmin (10.20
2017-10-25T13:59:57.254574-07:00 yb00-mgr1 journal: YBCLI user ybdadmin (10.20
2017-10-25T13:59:57.258328-07:00 yb00-mgr1 journal: YBCLI user ybdadmin (10.20
2017-10-25T14:00:37.265908-07:00 yb00-mgr1 journal: YBCLI user ybdadmin (10.20
2017-10-25T14:00:37.270577-07:00 yb00-mgr1 journal: YBCLI user ybdadmin (10.20
2017-10-25T14:04:32.013941-07:00 yb00-mgr1 journal: YBCLI user ybdadmin (10.20
2017-10-25T14:04:32.016892-07:00 yb00-mgr1 journal: YBCLI user ybdadmin (10.20
2017-10-25T14:04:37.550317-07:00 yb00-mgr1 journal: YBCLI user ybdadmin (10.20
2017-10-25T14:04:37.552872-07:00 yb00-mgr1 journal: YBCLI user ybdadmin (10.20
...
```

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="rsyslog
2017-10-31T13:01:01.973358-07:00 yb00-mgr1 crmd[3756]:  notice: Operation ybdk
2017-10-31T13:01:01.974923-07:00 yb00-mgr1 crmd[3756]:  notice: Initiating act
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 ybdf
2017-11-27T19:27:54.225340+01:00 yb00-mgr1 crmd[3725]:   error: pcmkRegisterNo
2017-11-27T19:27:54.569690+01:00 yb00-mgr1 ntpd[7901]: Deleting interface #35
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 instal
2019-04-16T14:41:37.304598-07:00 yb00-mgr0 installer2[10894]: Current version instal
2019-04-16T14:44:41.598975-07:00 yb00-mgr0 installer2[10894]: New YBD appliance pacl
```

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 instal
2019-04-16T14:41:37.304598-07:00 yb00-mgr0 installer2[10894]: Current version instal
```

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
2019-04-16T14:44:42.960413-07:00 ybos-00000000-0000-0000-0000-38B8EBD0061D YBDB[4423
...
2019-04-16T14:47:52.951151-07:00 ybos-00000000-0000-0000-0000-38B8EBD0061D kernel:
2019-04-16T14:47:52.951151-07:00 ybos-00000000-0000-0000-0000-38B8EBD0061D kernel:
...
2019-04-16T14:47:52.951730-07:00 ybos-00000000-0000-0000-0000-38B8EBD0061D kernel:
2019-04-16T14:47:52.951730-07:00 ybos-00000000-0000-0000-0000-38B8EBD0061D kernel:
```

```
Press any key for next section. 'q' to stop
...
```

Parent topic: ybcli Reference

manager

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 the event 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
If the remote node is unavailable, resetting the local node causes a database
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,
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 dat

ERROR: YBCLI cannot perform a failover while connected to the floating cluster
To perform a cluster failover, please connect to the IP address of the current

The IP addresses and hostnames of the manager nodes can be obtained using the
    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

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:
         BIOS: v05.04.21.0038.00.011 - Memory total/free: 65587652/1870728 KiB
         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: S3EWNWAJ312
                        OK/0% 2B6QCXP7        OK/0% 2B6QCXP7        OK/0% 2B6QC
                    0: S3EWNWAJ311166N    1: S3EWNWAJ330207R    2: S3EWNWAJ311
                        OK/0% 2B6QCXP7        OK/0% 2B6QCXP7        OK/0% 2B6QC
         Cluster status: OPERATIONAL - Cluster role: MEMBER - Last seen: just
```

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 - Versi
         BIOS: Athn_1.01.3 - Memory total/free: 1056744476/18735500 KiB
         BMC: 00.17.01 - Kalidah-1: 0x0DE38309 - Kalidah-2: 0x0DE38309 - IB-FW
         CPU: AMD EPYC 7702P 64-Core Processor - Microcode: 0x8301034 - Cores:
         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: S439NE0MA00
                       OK/0% EDA5502Q        OK/0% EDA5502Q        OK/0% EDA55
                    0: S439NE0MA00442     1: S439NE0MA00253     2: S439NE0MA00
                       OK/0% EDA5502Q        OK/0% EDA5502Q        OK/0% EDA55
         Cluster status: OPERATIONAL - Cluster role: MEMBER - Last seen: just
```

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:idl
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
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 blade
```

```
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:idl
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
Encryption keystore                : Available - Status: READY - Locked: YES
Chassis configuration              : Found: 1 - Configured: 1
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.

Parent topic: ybcli Reference

system

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 a
Note: Only use this command if a previous install was interrupted due to loss
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
       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
       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
       5.1.0-32781 -> 5.1.0-32872  (succeeded - FACTORY) - 11-06-2020 at 15:27
```

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 additio
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 Yellowb
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/201
```

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:

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 Yellowb
Are you sure you want to do this?
Response (yes/no): yes
WARNING: The database may be unresponsive, and may reconfigure while full syst
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..
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/202
```

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 abort 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 i
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 worker 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 wor
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. Stan

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 into 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.p1

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 manuall
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:idl
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
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 blade
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:idl
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
Encryption keystore                 : Available - Status: READY - Locked: YES
Chassis configuration               : Found: 1 - Configured: 1
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:
         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: 6
         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

Execution End: Fri Apr  5 12:51:15 PDT 2019
```

Parent topic: ybcli Reference

Client Tools and Drivers

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`,` ybbackupctl
- ybdumpschema
- ybdumproles
- ybrelay

Supported Client Platforms

The client tools are supported on the following platforms:

- Ubuntu 16.04 LTS and 18.04 LTS
- RHEL and CentOS 7, 8 and 9
- AIX 7.1 and 7.2
- macOS 10.10 and later
- Windows Server 2012 and later, and Windows 10

OpenSSL and OpenLDAP Libraries

The `ybtools` package on Linux client platforms depends on:

- OpenSSL: `libssl 1.0`
- OpenLDAP: `libldap-2.4-2`

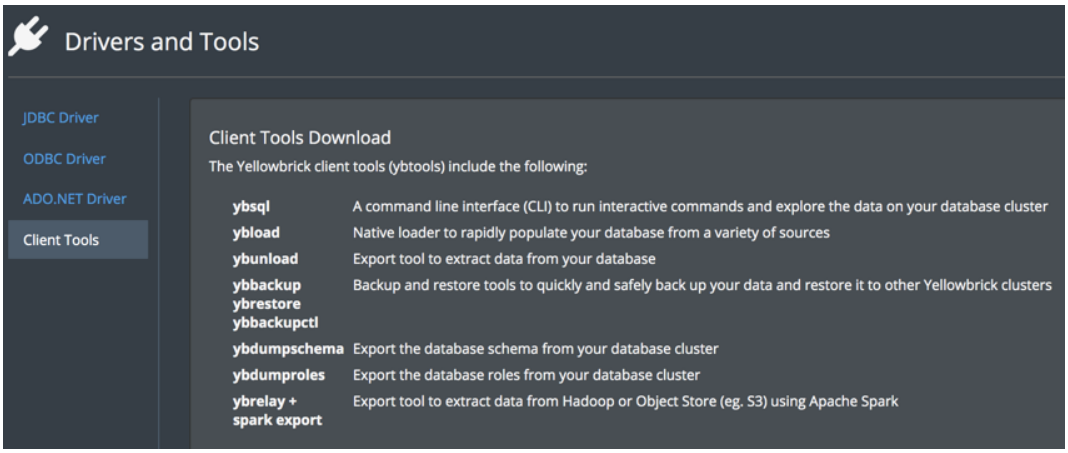
If necessary, you can install these libraries as follows:

- Ubuntu: `apt-get install openssl`
- RHEL and CentOS 7 and 8: `yum install openssl`
- CentOS 9: openssl 1.0 is not available anymore in the base repositories, however the CentOS 8 RPM can still be used `dnf install -y http://mirror.centos.org/centos/8-stream/AppStream/x86_64/os/Packages/compat-openssl10-1.0.2o-3.el8.x86_64.rpm`

Client Downloads

You can download the appropriate `ybtools` package for your operating system either from the SMC 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 system administrator.

To download `ybtools` from the SMC, go to **Configure** > **Drivers and Tools** > **Client 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.

In This Section

- Installing the Java Runtime Environment

- Installing ybtools
- Uninstalling ybtools
- Opening Network Ports for Clients
- Downloading Drivers
- Configuring SSL/TLS for Tools and Drivers
- Using the ADO.NET Driver

Parent topic: Yellowbrick Documentation

Installing the Java Runtime Environment

The following client tools require the 64-bit version of Java 8 (also known as Java 1.8):

- `ybbackup`
- `ybbackupctl`
- `ybrestore`
- `ybload`
- `ybunload`
- `ybrelay`

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
java version "1.8.0_101"
Java(TM) SE Runtime Environment (build 1.8.0_101-b13)
Java HotSpot(TM) 64-Bit Server VM (build 25.101-b13, mixed mode)
```

The build numbers may vary; what you are looking for is `1.8` 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 8.

Note: For Windows clients, a Windows Offline (64-bit) installation is recommended.

3. Follow the Oracle installation instructions for your client platform.

4. Check the Java version again when the installation is complete.

Parent topic: Client Tools and Drivers

Installing ybtools

The following sections explain how to install `ybtools` on the following supported client platforms:

Platform	Installer File Name	Supported OS Versions
AIX 7	`ybtools-*.aix7.1.ppc.rpm`	AIX 7.1 and 7.2
EL7	`ybtools-*.el7.x86_64.rpm`	Red Hat Enterprise Linux, CentOS 7, 8 and 9 (64-bit)
Ubuntu 16	`ybtools-*.ubuntu.16.04.x86_64.deb`	Ubuntu 16.04 LTS (64-bit)
Ubuntu 18	`ybtools-*.ubuntu.18.04.x86_64.deb`	Ubuntu 18.04 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.

In This Section

- Installing ybtools on Linux
- Installing ybtools on AIX
- Installing ybtools on macOS
- Installing ybtools on Windows

- Installing the "generic" ybtools

Parent topic: Client Tools and Drivers

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 SMC. Go to **Configure** > **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 16.04 LTS**: Install the `ybtools` package:

```
$ sudo apt-get install ./ybtools-5.2.4-20201208023004.ubuntu.16.04.x86_64.deb
```

  where the package name is `ybtools-5.2.4-20201208023004.ubuntu.16.04.x86_64.deb` in this example.

- **Ubuntu 18.04 LTS**: Install the `ybtools` package:

```
$ sudo apt-get install ./ybtools-5.2.4-20201208023004.ubuntu.18.04.x86_64.deb
```

  where the package name is `ybtools-5.2.4-20201208023004.ubuntu.18.04.x86_64.deb` in this example.

- **RHEL/CentOS 7.0, 8.0 and Stream 9**:

  1. Remove the old version of `ybtools`:

```
$ sudo yum remove ybtools
```

2. Install the new `ybtools` package:

```
$ sudo yum install ybtools-5.2.4-20201208023004.el7.x86_64.rpm
```

where the package name is `ybtools-5.2.4-20201208023004.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-5.2.4-20201208023004.ubuntu.16.04.x86_64.deb
Reading package lists... Done
Building dependency tree
Reading state information... Done
Note, selecting 'ybtools' instead of './ybtools-5.2.4-20201208023004.ubuntu.16.04.x8
The following packages were automatically installed and are no longer required:
  bsh libasm3-java libganymed-ssh2-java libisl13 libjackrabbit-java libjline-java li
  libjsch-agent-proxy-java libjsoncpp0v5 liblldb-3.8 libllvm3.8 libmaven-scm-java li
  llvm-3.8-dev llvm-3.8-runtime 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 330 not upgraded.
Need to get 0 B/72.2 MB of archives.
After this operation, 0 B of additional disk space will be used.
Get:1 /home/brumsby/ybtools-5.2.4-20201208023004.ubuntu.16.04.x86_64.deb ybtools amd
(Reading database ... 153992 files and directories currently installed.)
Preparing to unpack .../ybtools-5.2.4-20201208023004.ubuntu.16.04.x86_64.deb ...
Unpacking ybtools (5.2.4-20201208023004) over (5.2.4-20201201022005) ...
Setting up ybtools (5.2.4-20201208023004) ...
update-alternatives: using /opt/ybtools/bin/ybbackup to provide /usr/bin/ybbackup (y
update-alternatives: using /opt/ybtools/bin/ybbackup-list-old to provide /usr/bin/yb
update-alternatives: using /opt/ybtools/bin/ybbackup-old to provide /usr/bin/ybbacku
update-alternatives: using /opt/ybtools/bin/ybbackupctl to provide /usr/bin/ybbackup
update-alternatives: using /opt/ybtools/bin/ybdumproles to provide /usr/bin/ybdumpro
update-alternatives: using /opt/ybtools/bin/ybdumpschema to provide /usr/bin/ybdumps
update-alternatives: using /opt/ybtools/bin/ybload to provide /usr/bin/ybload (yblo
update-alternatives: using /opt/ybtools/bin/ybrelay to provide /usr/bin/ybrelay (ybr
update-alternatives: using /opt/ybtools/bin/ybrestore to provide /usr/bin/ybrestore
update-alternatives: using /opt/ybtools/bin/ybrestore-old to provide /usr/bin/ybrest
update-alternatives: using /opt/ybtools/bin/ybsql to provide /usr/bin/ybsql (ybsql)
```

```
update-alternatives: using /opt/ybtools/bin/ybunload to provide /usr/bin/ybunload (y
update-alternatives: using /opt/ybtools/config/ybrelay-init to provide /usr/bin/ybre
```

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.

Parent topic: Installing ybtools

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 SMC. Go to **Configure** > **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-5.2.4-20201208023004.aix7.1.ppc.rpm
ebb49ee3ed738d6471cc34bf8fdfd3ea  ybtools-5.2.4-20201208023004.aix7.1.ppc.rpm
```

4. Install the `ybtools` package:

```
$ yum install ybtools-5.2.4-20201208023004.aix7.1.ppc.rpm
```

where the package name is `ybtools-5.2.4-20201208023004.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/fre
```

```
$ 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-5.2.4-20201208023004.aix7.1.ppc.rpm
Failed to set locale, defaulting to C
AIX_Toolbox
AIX_Toolbox/primary_db
AIX_Toolbox_71
```

```
AIX_Toolbox_71/primary_db
AIX_Toolbox_noarch
AIX_Toolbox_noarch/primary_db
Setting up Install Process
Examining ybtools-5.2.4-20201208023004.aix7.1.ppc.rpm: ybtools-5.2.4-20201208023004
Marking ybtools-5.2.4-20201208023004.aix7.1.ppc.rpm to be installed
Resolving Dependencies
--> Running transaction check
---> Package ybtools.ppc 0:5.2.4-20201208023004 will be installed
--> Processing Dependency: libldap.a(libldap-2.4.so.2) for package: ybtools-5.2.4-2
--> 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.pp
--> 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
...
Running Transaction Check
Running Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing : libgcc-6.3.0-2.ppc
  Installing : cyrus-sasl-2.1.26-3.ppc
  Installing : openldap-2.4.48-1.ppc
Group "ldap" does not exist.
  Installing : ybtools-5.2.4-20201208023004.aix7.1.ppc
Installed:
  ybtools.ppc 0:5.2.4-20201208023004
Dependency Installed:
  cyrus-sasl.ppc 0:2.1.26-3                          libgcc.ppc 0:6.3.0-2
Complete!
```

Parent topic: Installing ybtools

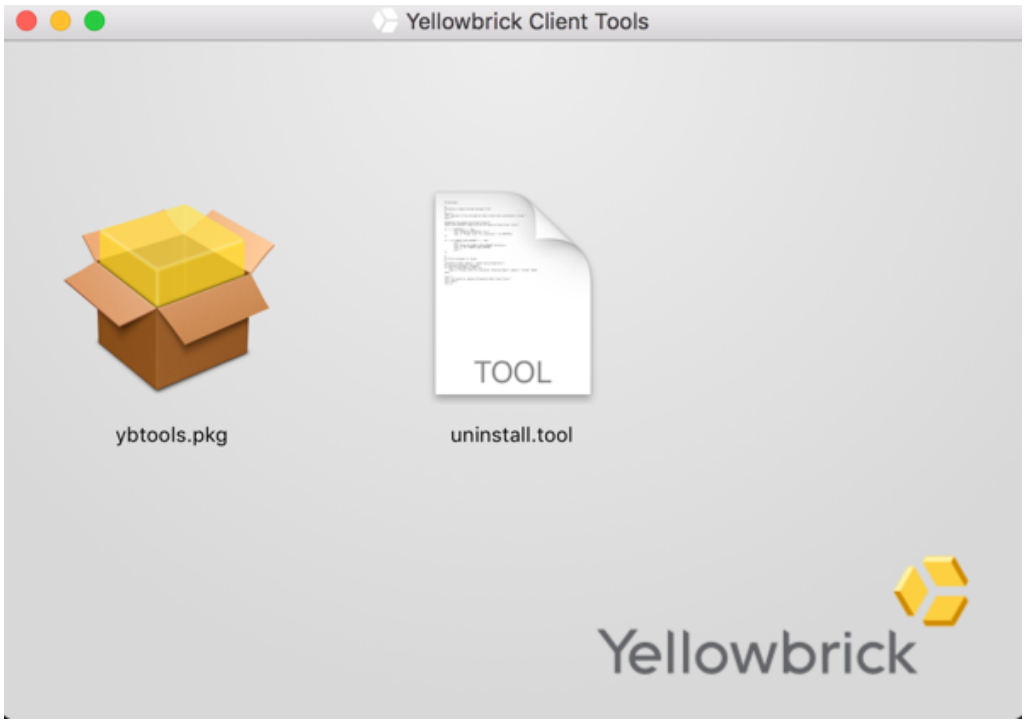
Installing ybtools on macOS

You can install `ybtools` on macOS versions 10.10 and later.

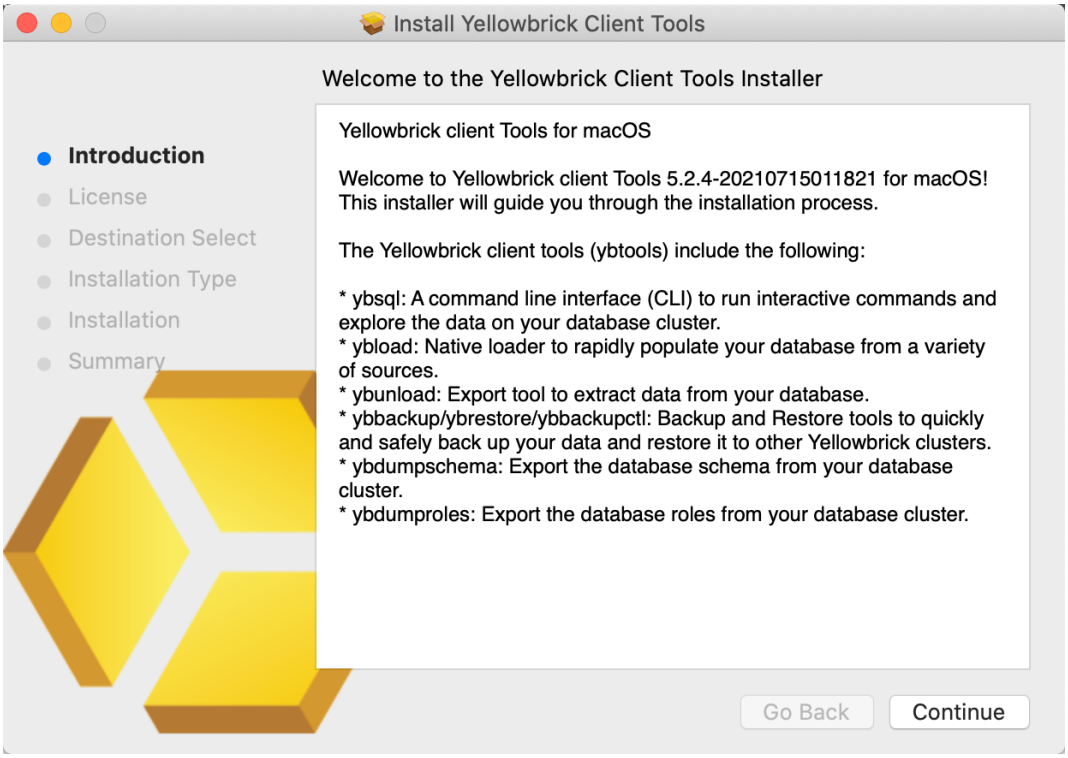
1. Download the macOS package from the SMC. Go to **Configure** > **Drivers and Tools** > **Client Tools**. For example: `ybtools-5.2.4-20201208023004.macos.x86_64.dmg` .

2. If necessary, copy the downloaded package to the macOS client system.

3. Double-click the `.dmg` file. The Yellowbrick Client Tools installer opens.

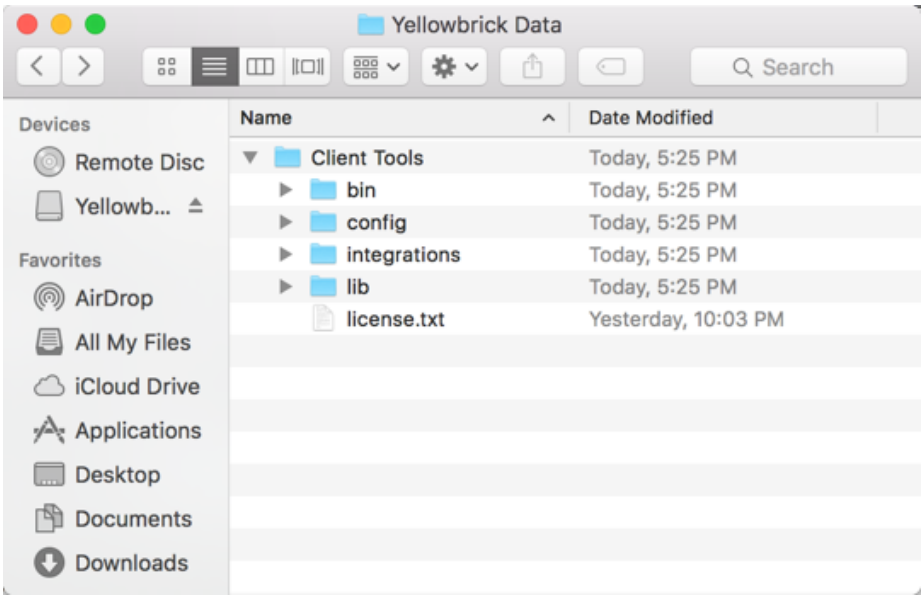


4. Double-click the `ybtools.pkg` icon and step through the installer as directed. Be sure to accept the license agreement.



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:



Parent topic: Installing ybtools

Installing ybtools on Windows

Make sure that Java 1.8 64-bit is installed on your client system. See Installing the Java Runtime Environment.

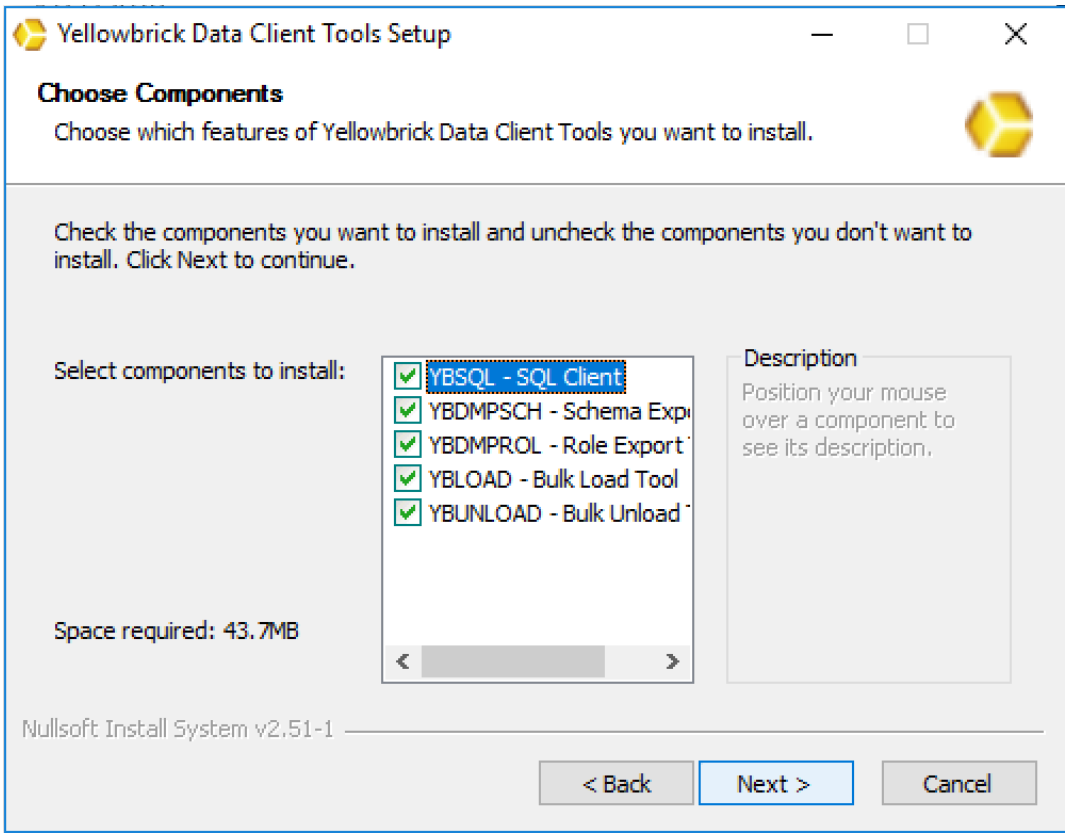
Follow these steps to install the client tools on a Windows client system:

1. Download the Windows package from the SMC. Go to **Configure** > **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 installer (for example, `ybtools-5.2.4-20201208023004.windows.x86_64.exe`).

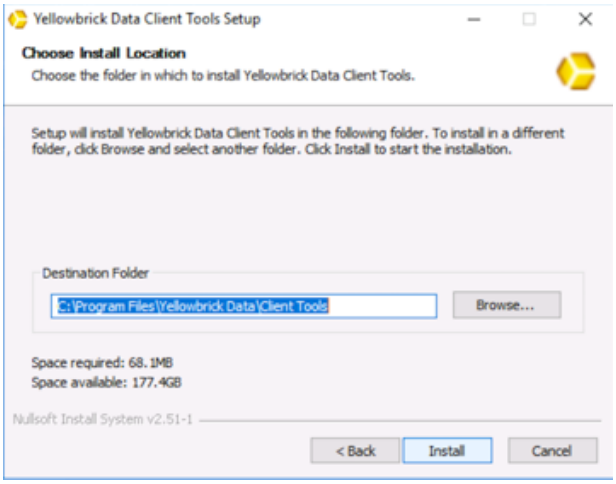
The Client Tools Setup Wizard starts up.

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:

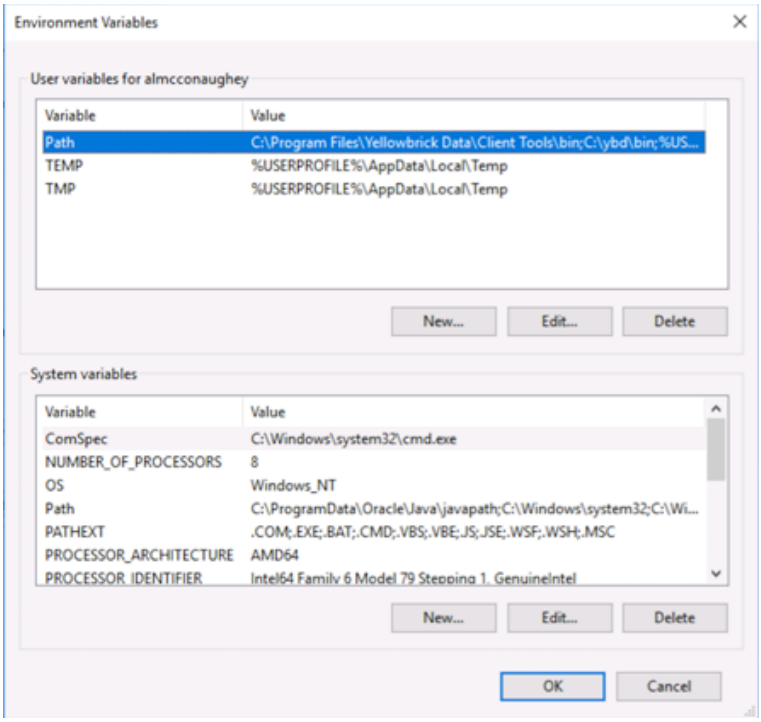
```
C:\Program Files\Yellowbrick Data\Client Tools
```

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.



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.

Parent topic: Installing ybtools

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 SMC. Go to **Configure** > **Drivers and Tools** > **Client Tools**. For example: `ybtools-5.2.4-20201208023004.generic.noarch.tar.gz`

2. If necessary, copy the downloaded package to the client system.

3. Unpack the `generic.noarch.tar.gz` file to a directory of your choice. The result will be a top-level `ybtools` directory with some subdirectories.

- 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.

Parent topic: Installing ybtools

Uninstalling ybtools

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 li
  libmaven-scm-java libnetbeans-cvsclient-java libruby2.2 libsisu-guice-java llvm-3
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 (5.2.4-20201208023004) ...
```

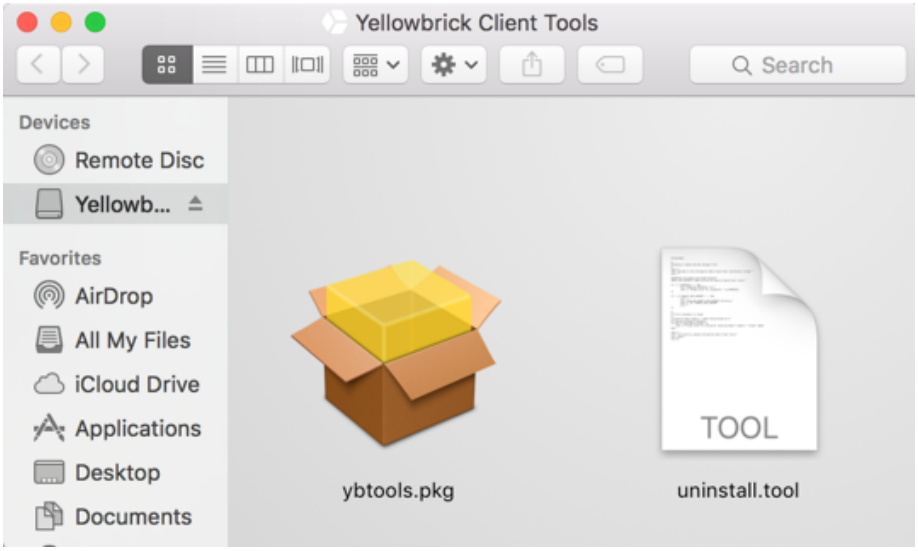
Uninstall on Windows

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]
```

Parent topic: Client Tools and Drivers

Opening Network Ports for Clients

The Yellowbrick client applications (`ybtools`) run on customer computers and need to communicate with the Yellowbrick appliance in order to send or receive data, using the customer's existing network infrastructure. The appliance is usually installed in a server room or a data center, while the clients run on desktop machines throughout the company or on other servers that may or may not be in the same physical location as the appliance.

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.

For example, if database port `5432` is not open, `ybsql` returns the following error:

```
$ ybsql -h premdb yellowbrick
ybsql: could not connect to server: No such file or directory
    Is the server running locally and accepting
    connections on Unix domain socket "/tmp/.s.PGSQL.5432"?
```

Port Numbers

Yellowbrick client tools communicate with the appliance by specifying both a hostname (or IP address) and a port. The reserved ports are as follows:

- `5432` : normal database port (used by all clients)
- `11111` : control port, unsecured HTTP (used by `ybload` , `ybunload` , `ybbackup` , `ybrestore`)
- `11112` : control port, secured HTTPS (used by `ybload` , `ybunload` , `ybbackup` , `ybrestore` when SSL-only mode is in effect or the `--secured` option is specified via one of the Yellowbrick client tools)

When SSL-only mode is used for client connections, both `11111` and `11112` need to be open. Connections may be automatically redirected from `11111` to `11112`.

- `31000` and `31001`: two ports for sending and receiving data (used by `ybload`, `ybunload`, `ybbackup`, `ybrestore`, and so on)

Configure the network firewall to allow traffic from the client computer to the appliance 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.

Parent topic: Client Tools and Drivers

Downloading Drivers

You can connect ODBC, JDBC, and ADO.NET applications to a Yellowbrick database, using drivers that you can download and install.

Start by downloading the appropriate driver from the Customer Center Downloads page or the System Management Console (SMC). In the SMC, select **Drivers and Tools** from the **Configure** menu, then select from the list of drivers:

JDBC

- PostgreSQL 9.4, Java 8 (`postgresql-9.4.1208.jar`
- PostgreSQL 9.4, Java 7 (`postgresql-9.4.1208.jre7.jar`
- PostgreSQL 9.4, Java 6 (`postgresql-9.4.1208.jre6.jar`
- PostgreSQL PGJDBC-NG (`pgjdbc-ng-0.8.2-all.jar`)

ODBC for Windows

The SMC provides the following downloads for Windows platforms.

- PostgreSQL 09.05.400 32-bit MSI: `psqlodbc_09_05_0400-x86.zip`
- PostgreSQL 09.05.400 64-bit MSI: `psqlodbc_09_05_0400-x64.zip`
- PostgreSQL 09.05.400 Windows Setup Executable: `psqlodbc_09_05_0400.zip`

You can also download the same drivers here: https://www.postgresql.org/ftp/odbc/versions/msi/

ODBC for Linux and macOS

For general instructions on using ODBC on Linux and macOS platforms, go to: http://www.unixodbc.org/

To install and use ODBC, you need to configure the unixODBC subsystem, using a PostgreSQL 9.5 driver. Follow the instructions in the SMC by going to **Drivers and Tools** > **ODBC**.

ADO.NET

NPGSQL 3.1.6 (`Npgsql-3.1.6.msi`)

Alternatively, you can select and download the same drivers directly from the following web sites:

- **JDBC**: https://jdbc.postgresql.org/download/
- **ODBC**: https://www.postgresql.org/ftp/odbc/versions/msi/
- **ADO.NET**: https://github.com/npgsql/npgsql/releases/tag/v3.1.6

Parent topic: Client Tools and Drivers

Configuring SSL/TLS for Tools and Drivers

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.

In This Section

- SSL/TLS Overview
- Enabling and Verifying SSL/TLS Encryption
- Secure Connections for ODBC/JDBC Clients and ybsql
- Secure Connections for Java-based ybtools
- Obtaining a Certificate Chain
- Troubleshooting SSL Issues

Parent topic: Client Tools and Drivers

SSL/TLS Overview

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

If 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 can be configured to allow TLS/SSL connections only, but the server cannot require the client to do trust verification.

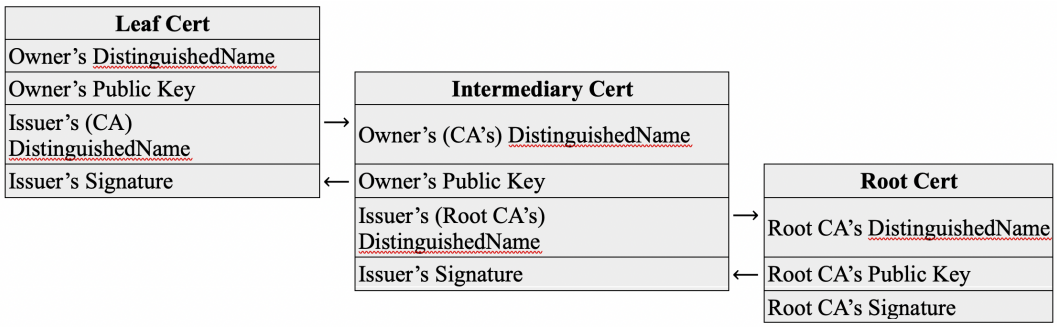
SSL Chain of Trust

A chain of trust is an ordered list of certificates, available on the client host:

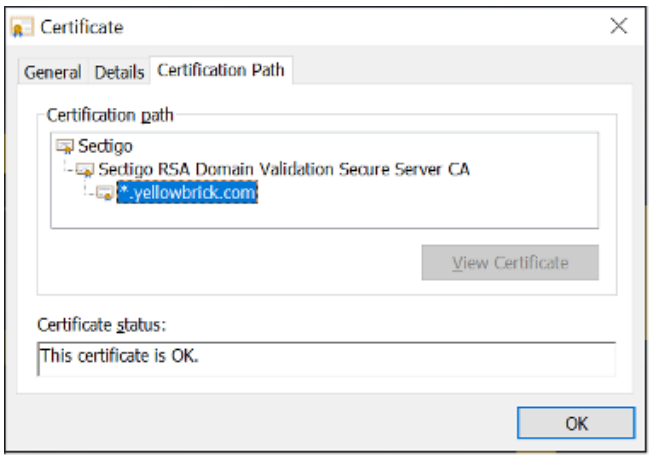
- An initial leaf certificate, which is the certificate installed on your Yellowbrick data warehouse
- Intermediate certificates, which are issued to your organization by a trusted Certificate Authority (CA)
- The root certificate of the CA, which is publicly available and typically included in the trusted 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 ships with 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 appliances 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, SMC access, and other SSL/TLS communication with the Yellowbrick appliance 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.

Parent topic: Configuring SSL/TLS for Tools and Drivers

Enabling and Verifying SSL/TLS Encryption

The Yellowbrick server supports SSL/TLS encryption for client communications and passwords, including remote connections to the SMC and connections via client tools and applications.

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
```

SSL-Only Mode

If you want to automatically enforce SSL mode for all client connections, you can set the appliance to SSL-only mode. When the appliance is running in this mode, all client connections from all tools, including connections to the SMC, the front-end PostgreSQL database, and `ybtools`, must connect via HTTPS. All non-SSL connections are rejected.

Enable SSL-Only Mode and Import Certificates

Follow these steps to enable SSL-only mode in the SMC. If necessary, you can import certificates as part of this procedure.

The initial SSL certificate for a Yellowbrick database is established with a self-signed, untrusted SSL certificate. Your organization may choose to change this database certificate to align with its internal policies by importing a certificate signed with a private signing authority/CA or by using an external signing authority. Without a Certificate Authority (CA)-signed certificate, you may get errors such as:

```
There is a problem with this website's security certificate.
```

The **Import Certificate** button in the SMC allows customers to import a new SSL certificate for the database. The same certificate is used for establishing trust to both the HTTPS listener and the Postgres TLS listener. The import must be done with a PEM-encoded certificate issued by a third-party CA or a custom CA trusted by the organization's standard browser.

To enable SSL-only mode:

1. Log into the SMC as the `yellowbrick` user.
2. Go to **Configure** > **Settings** > **SSL**.
3. To import a new certificate, click **Import Certificate** and import the public key and the private key. You must specify the password for the private key. Be prepared to provide all three values: both keys and the private key password.

Although the format of the files being imported is text, it is encoded as follows:

```
-- BEGIN PRIVATE KEY -- <data> -- END PRIVATE KEY
...
-- BEGIN CERTIFICATE -- <data> -- END CERTIFICATE KEY --
```

4. Select the **Enable SSL-only communications** checkbox. (Skip this step if you only want to import a new certificate.)
5. If required by your certificate, enter a fully-qualified hostname for the SMC in the **Automatic Redirect** field. Redirection is commonly used to match the Common Name (CN) or Subject Alternative Name (SAN) in an SSL certificate to the URL a user visits in order to establish trust. If browsers attempt to log into the SMC via an HTTP URL, those connections will be redirected to the hostname you entered, using HTTPS.

For example:

```
http://yb100
```

will be redirected to:

```
https://yb100.redirectme.com
```

assuming you entered `yb100.redirectme.com` in the **Automatic Redirect** field.

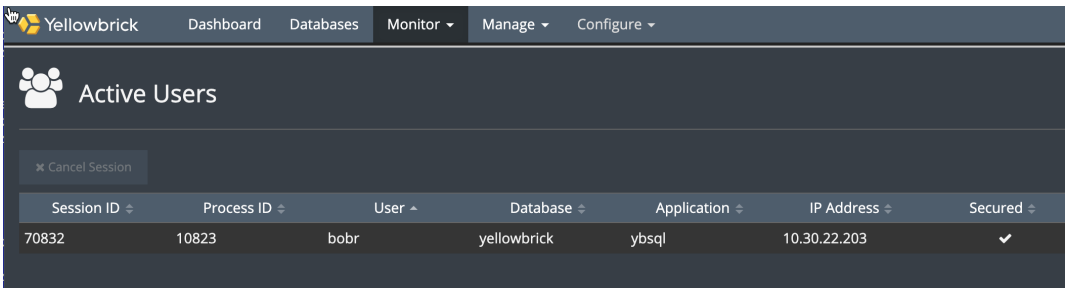
6. Click **Save Settings** in the top-right corner of the Configuration Settings screen.
7. Restart the database as prompted, using `ybcli`:

```
$ ybcli -y database stop
...
$ybcli database start
```

Verifying That Connections Are Secure

You can verify that a session is using a secure SSL connection in two ways:

- Check the SMC. Go to **Monitor** > **Active Users** and note the **Secured** column. For example:



- Query the sys.session view. For example, the following query returns information for the current session user, showing that the connection is secure:

```
yellowbrick=> select su.name, ss.session_id, ss.client_ip_address as client_ip, ss.u
from sys.session ss join sys.user su on ss.user_id=su.user_id
where su.name=session_user;
 name | session_id |    client_ip    | user_id | secure |                secure_deta
------+------------+-----------------+---------+--------+---------------------------
 bobr |      70832 | 10.30.22.203/32 |   16397 | t      | TLSv1.2/ECDHE-RSA-AES256-(
(1 row)
```

Parent topic: Configuring SSL/TLS for Tools and Drivers

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.

SSL Modes

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 a connection property, depending on how you are connecting.

Whether you use the environment variable or a connection property, you can set `YBSSLMODE` to any of the following values:

SSL Mode	Description
`allow`	Attempt a non-SSL connection. If it fails, attempt an SSL connection.
`disable`	Only attempt a non-SSL connection.
`prefer`	Attempt an SSL connection. If it 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).

SSL Mode	Description
`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 cert(s) and root cert 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.
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` trus 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`
Windows PowerShell	`$env:YBSSLMODE = 'require'`

In This Section

- Secure Connections for ODBC Clients
- Secure Connections for JDBC Clients
- Secure Connections for ybsql (libpq)
- Secure Connections for DBeaver Clients

Parent topic: Configuring SSL/TLS for Tools and Drivers

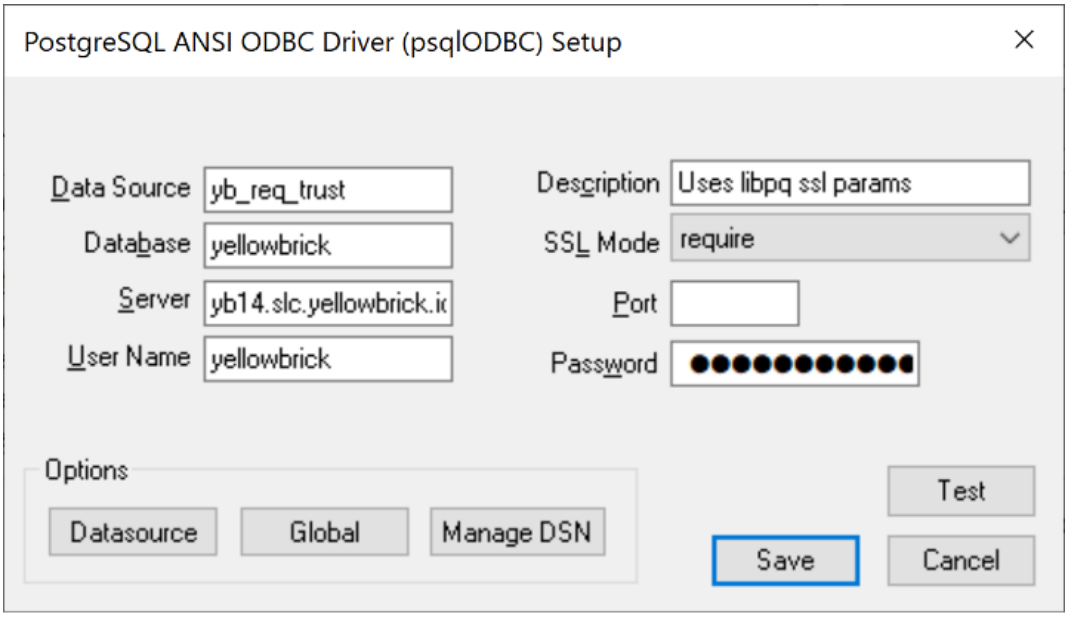
Secure Connections for ODBC Clients

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).

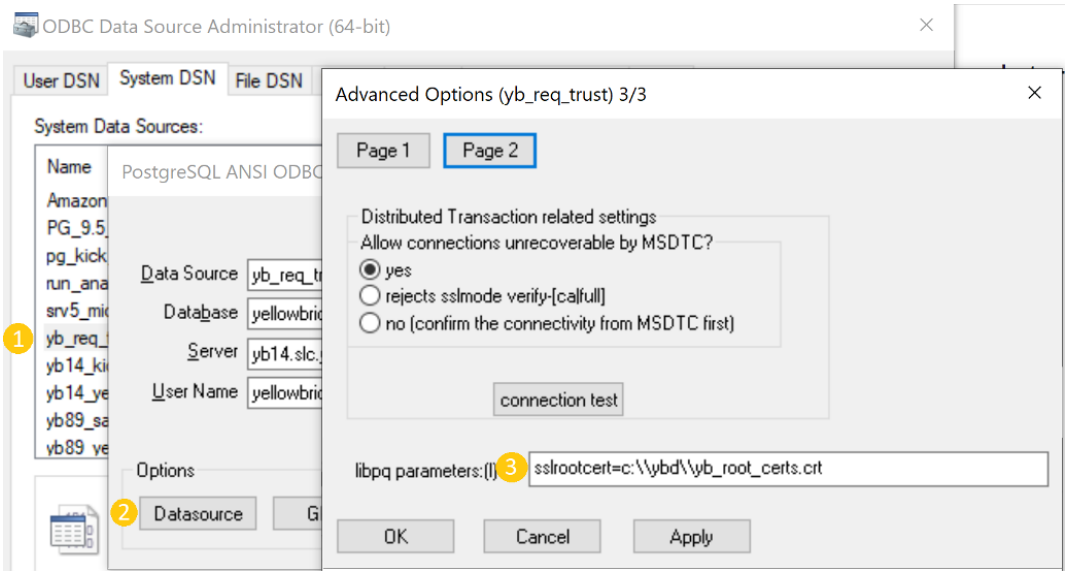
Example: Windows UI

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 cert.

```
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 cert.

```
[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
```

Parent topic: Secure Connections for ODBC/JDBC Clients and ybsql

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. If SSL is going to be used, the connection uses the root cert 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&sslrootce
```

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

Parent topic: Secure Connections for ODBC/JDBC Clients and ybsql

Secure Connections for ybsql (libpq)

Assuming you are using the default locations, you can test the basic connection 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=yb007.nyc.yellowbrick.io dbname=yellowbrick user=yellow
```

You should be prompted for your password and a secure connection will be attempted.

Example: ybsql on Windows (verify-ca, using a non-default root cert)

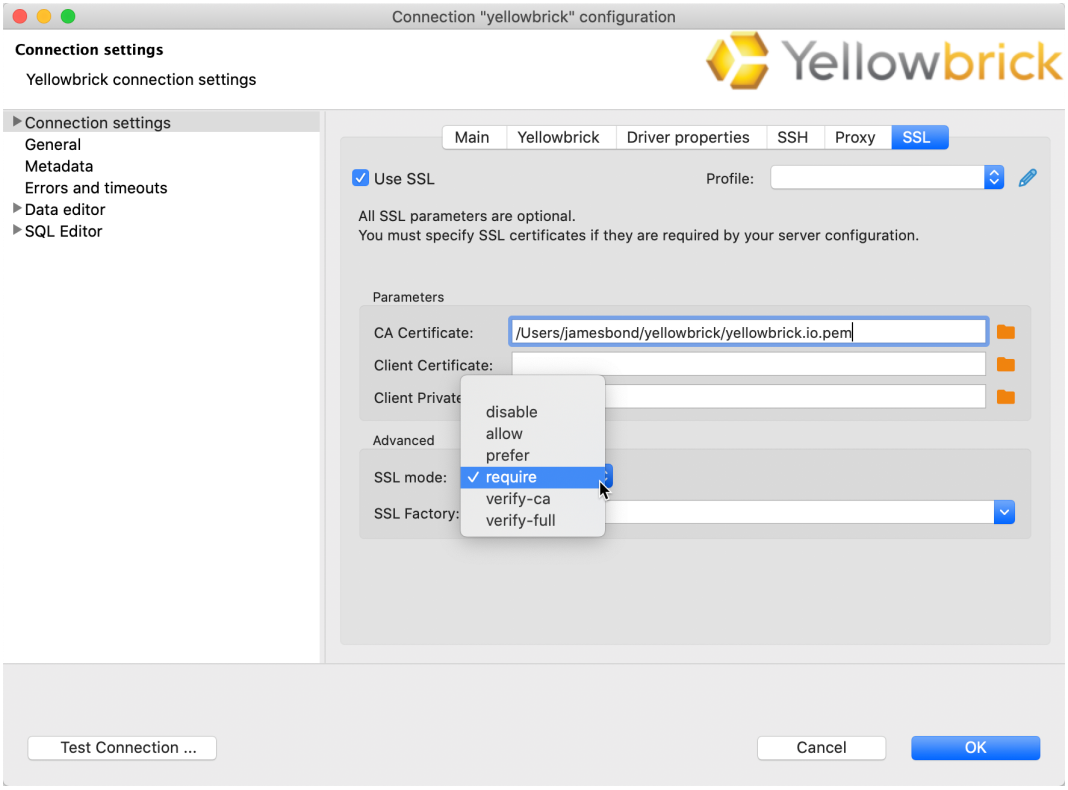
You can precede the `libpq` connection string with options such as `-c` to execute a command. For example, this connection runs a query against the `sys.session` view and returns the secure details for the session:

```
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.yello
...
secure_details
-----------------------------------------------
TLSv1.2/ECDHE-RSA-AES256-GCM-SHA384/256 bits
(1 row)
```

Parent topic: Secure Connections for ODBC/JDBC Clients and ybsql

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 cert, which only needs to be set if you are not using the default root cert file.

Parent topic: Secure Connections for ODBC/JDBC Clients and ybsql

Secure Connections for Java-based ybtools

The following Java-based client tools support 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 SSL/TLS but with different options. See ybsql Connections.

These tools 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. The default setting is not secured; no encryption is used.
- `--cacert` : supply a custom root ca bundle for trusting the cert installed under Yellowbrick
- Note that this is not a server cert to be used in two-way trust.
- `--disable-trust` : do not require root cert 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.

Note: When SSL-only mode is enabled, you do not need to use the `--secured` option in `ybtools` commands. However, the behavior described here for the `--cacert` and `--disable-trust` options still applies.

See Opening Network Ports for Clients for a list of the port numbers that the client tools use for data control and data transfer. The `ybtools` data transfer ports will remain unencrypted if the `--secured` option is not set or SSL-only mode is not enabled.

When you are making a secure connection with Java-based `ybtools` :

- The default connection mode is determined by whether **Enable SSL-only communications** is set for the Yellowbrick data warehouse. If SSL-only is enabled, connections must be secure. To check this setting, log into

the SMC and go to **Configure** > **Settings** > **SSL** > **Configuration**.

- To require connections to be secure if they are not by default, use the `--secured` option within each Java-based tool.
- 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 cert bundle file only for intermediary certificates issued by your organization.
- If you do need certificates included in a custom root cert 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 ybqsl. 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 cert 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`
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`	

Property	Value / From	Notes
Default root cert file	`root.cert` or `root.crt`	See Creating a Client-Side root CA File.

Alternative root cert file	`--cacert`
	`YBCACERT`
	`YBSSLROOTCERT`
	`PGSSLROOTCERT`
		Alternative root cert will be looked for in this order of preference.

Parent topic: Configuring SSL/TLS for Tools and Drivers

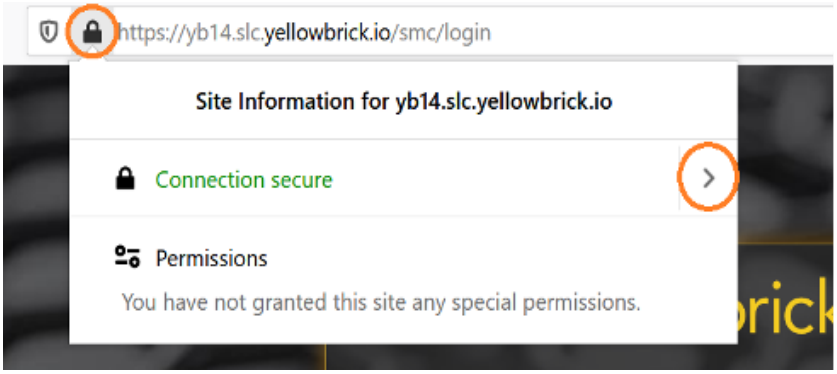
Obtaining a Certificate Chain

If you require certificates to be included in your root cert file and your DBA or IT staff has not provided them, you can often extract them yourself by using a web browser or the `openssl` application. If you do this, keep in mind that while it does no harm to include them, you do not need to include the leaf certificates in the file. You only the intermediary certificates and, for `ybsql` connections, root certificates.

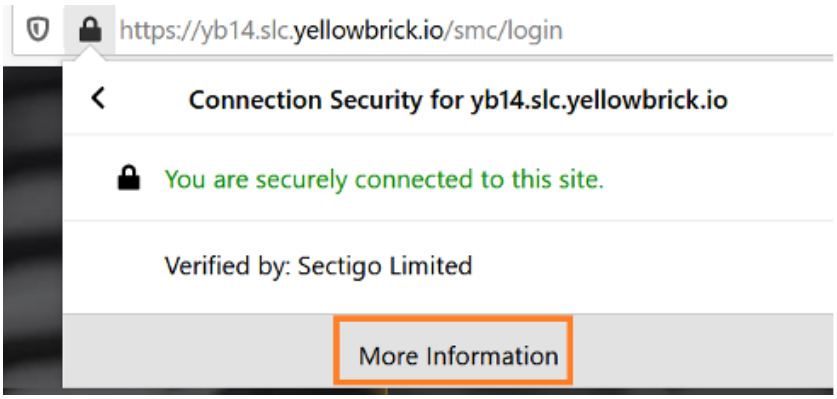
Export the chain of trust certs using FireFox

Firefox has the most convenient interface for exporting certificates to a file as it has an option to export the entire certificate chain with a single click.

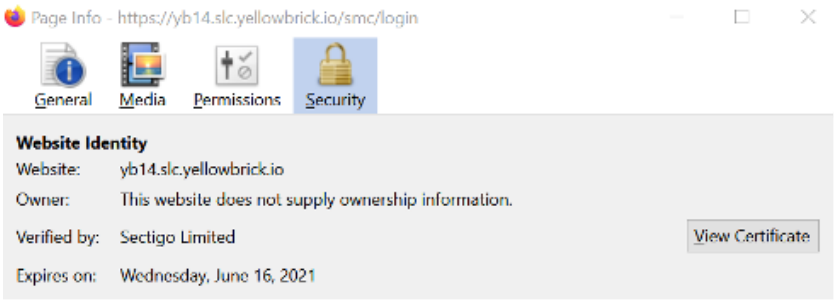
1. Click the lock icon next to the URL for the SMC login, then click the **Connection secure** expansion arrow.



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.



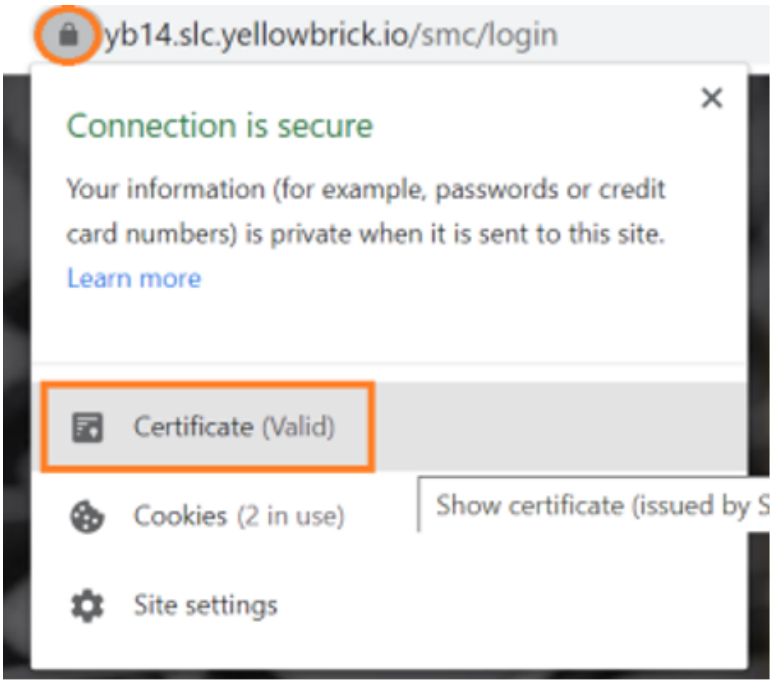
4. Click the PEM (chain) link, which generates the certificate bundle for the entire chain of trust.



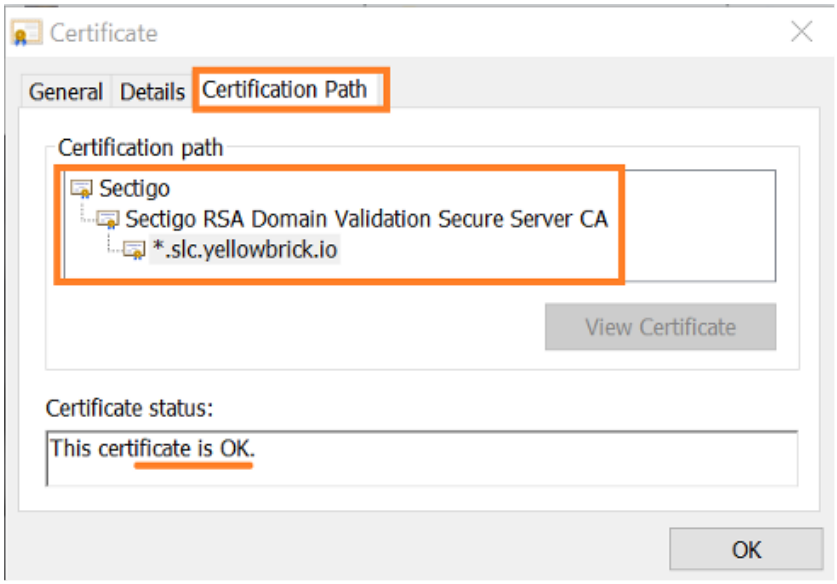
Export the chain of trust certs using Chrome

You have to follow these steps for each certificate in the chain (unlike the Firefox procedure, where you can export the entire certificate chain at once).

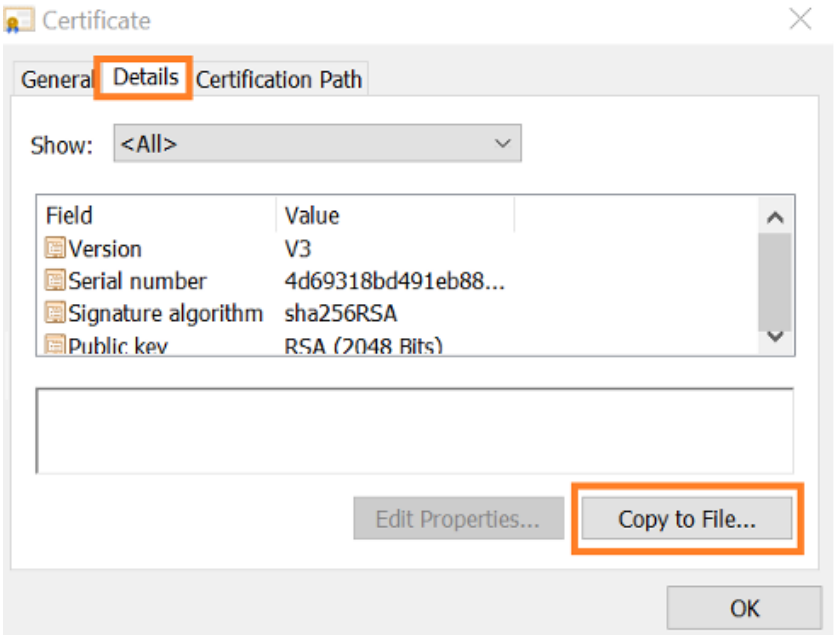
1. Click the lock icon next to the URL for the SMC login, then **Certificate**.



2. Find all the certs that need to be exported by clicking **Certification Path**.

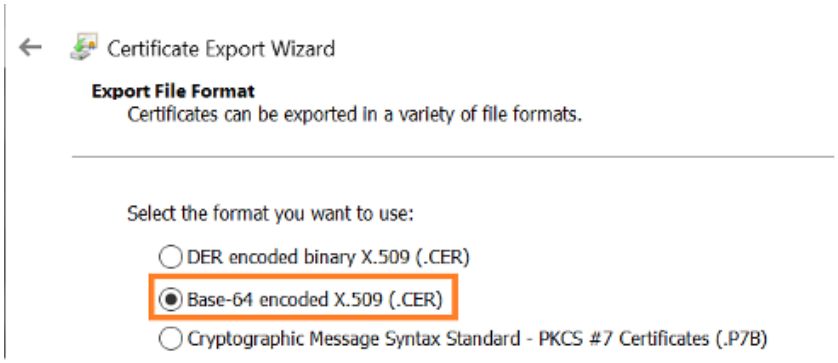


3. For each cert to be exported, click **Details**. If the message says `This certificate is OK`, click **Copy to File**.



4. Follow the steps in the Certificate Export Wizard.

- Select **Base-64 encoded x.509 (.CER)**.



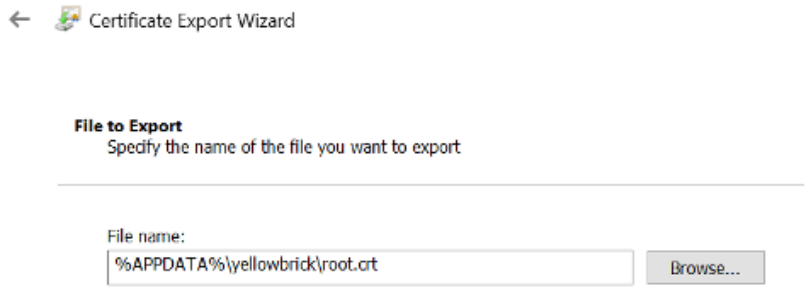
- Specify the path and name of the cert file. The default locations and names are commonly used. For example:

  Linux:

  ```
  /.yellowbrick/root.cert
  ```

  Windows:

```
%APPDATA%\yellowbrick\root.crt
```



Make sure you are not overwriting existing certificate information. If you already have a file, you should *append* the new certs to that file.

Export the chain of trust certs using openssl

To show all certificate information, including the metadata, echo the results of an `openssl` command. For example:

```
$ echo -n | openssl s_client -connect yb007.nyc.yellowbrick.io:443 -showcerts
```

To show only the certificates themselves, run a command like this:

```
$ echo -n | openssl s_client -connect yb007.nyc.yellowbrick.io:443 -showcerts \
 | sed -ne '/-BEGIN CERTIFICATE-/,/-END CERTIFICATE-/p' > yb007.pem
```

Parent topic: Configuring SSL/TLS for Tools and Drivers

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 cert 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=ye
ybsql: server certificate for "*.slc.yellowbrick.io" (and 1 other name) does not mat
..
ybsql "sslmode=verify-full sslrootcert=/mnt/c/ybd/my.pem host=yb14.yellowbrick.io db
ybsql: server certificate for "*.slc.yellowbrick.io" (and 1 other name) does not mat
```

"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, keep in mind 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 cert in your root cert file.

For example, a Yellowbrick data warehouse ships with 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
ybsql: SSL error: certificate verify failed
```

Parent topic: Configuring SSL/TLS for Tools and Drivers

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=********;
```

Parent topic: Client Tools and Drivers

ybsql Reference

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 SMC as part of the Yellowbrick client tools package (`ybtools`). Go to **Configure** > **Drivers and Tools** > **Client Tools**.

This section describes the commands and options that are available in `ybsql`.

In This Section

- ybsql Connections
- ybsql Environment Variables
- ybsql Startup File
- ybsql Command-Line Options
- ybsql Slash (\) Commands
- ybsql \copy Command
- ybsql Examples
- ybsql Properties and Variables

Parent topic: Yellowbrick Documentation

ybsql Connections

This section describes how to connect to a database with `ybsql`.

Syntax Summary

```
ybsql [ connection_options ] [ 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 [ -U | --username username ] ]
[ -W | --password ]
[ -w | --no-password ]
```

-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
```

-U, --username

Name of the database user for the connection. Alternatively, you can specify `username` by itself as long as 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.

-h, --host

Host name of the appliance.

-p, --port

Port number for the connection (`5432` by default).

Environment Variables

You can use the following environment variables to set connection parameters for `ybsql` connections. Note that the following examples are for Windows. For Linux, precede each example with `export`.

Environment Variable	ybsql Option	Description	Example

Environment Variable	ybsql Option	Description	Example
`YBDATABASE`	`-d` or `--dbname`	Destination database name. Default: `yellowbrick` See also SQL Identifiers.	`YBDATABASE=premdb`
`YBHOST`	`-h` or `--host`	Destination server host name. Default: `localhost`	`YBHOST=test.ybsystem.io`
`YBPASSWORD`	`-W` or `--password`	Interactive prompt for the database user's password. No default.	`YBPASSWORD=********`
`YBPORT`		Destination server port number. Default: `5432`	`YBPORT=5433`
`YBSSLMODE`	See SSL Modes.	Determines whether or not SSL/TLS is used when connecting to the server and, if so, how.	`YBSSLMODE=require`
`YBUSER`	`-U` or `--username`	Database login username. No default.	`YBUSER=bobr`

SSL/TLS Connection Modes

When SSL is enabled on the Yellowbrick appliance, the appropriate `SSLMODE` must be chosen:

- By default, `ybsql` will not perform any server certificate verification, which can be used if a custom certificate has not been installed.
- If you have chosen an `SSLMODE` requiring verification, the certificate(s) of one or more trusted CAs must be located in `~/.yellowbrick/root.cert` for Linux or `%APPDATA%\yellowbrick\root.crt` for Windows.
- These modes are used to prevent being a victim of server identity spoofing.

`YBSSLMODE` environment variable can be set to several modes. For details, see SSL Modes.

Important: If a `YBSSLMODE` environment variable is not set but an `SSLMODE` environment variable is, `ybsql` will use the `SSLMODE` environment variable.

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 clear 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 clear text and may appear in log messages.

Parent topic: ybsql Reference

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.

YBDATABASE

Name of the destination database. Default: `yellowbrick`. See ybsql Connections.

YBHOST

Name of the destination server host. Default: `localhost`. See ybsql Connections.

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 ybsql Connections.

YBPORT

The destination server port number. Default: `5432`. See ybsql Connections.

YBSSLMODE

Determines whether or not SSL/TLS is used when connecting to the server and, if so, how. See ybsql Connections.

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. No default. See ybsql Connections.

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
...
```

Parent topic: ybsql Reference

ybsql Startup File

Startup files are useful for preserving settings and shortcuts that users want to enable every time they log into `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.

Parent topic: ybsql Reference

ybsql Command-Line Options

In addition to the connection options, you can specify a number of 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.

Parent topic: ybsql Reference

ybsql Slash (\) Commands

Within a `ybsql` session, you can use a number of "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.

```
premdb=# \c yellowbrick bobr localhost 5432
Password for user bobr:
You are now connected to database "yellowbrick" as user "bobr" on host "localh
```

\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:

```
premdb=# \ddp
                  Default access privileges
    Owner    | Schema | Type  |        Access privileges
-------------+--------+-------+---------------------------------
 yellowbrick |        | table | =r/yellowbrick                 +
             |        |       | yellowbrick=arwdDxt/yellowbrick+
             |        |       | bobr=a/yellowbrick
(1 row)
```

The `\ddp` command returns no rows if no default access privileges have been defined.

Access privileges are logged in `sys.database`; see ../systemtables/sys.database.md#section_hnr_qng_1mb.

\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_
--------+------+------+--------+----------+--------+---------+----------+-----
     48 |   49 |   98 | [NULL] | [NULL]   | [NULL] | [NULL]  |        0 |   0.
     49 |   50 |   99 | [NULL] | [NULL]   | [NULL] | [NULL]  |        0 |   0.
     50 |   51 |  100 | [NULL] | [NULL]   | [NULL] | [NULL]  |        0 |   0.
(3 rows)

                    Watch every 5s Tue Mar 20 18:14:26 2018

 teamid | htid | atid |  name  | nickname |  city  | stadium | capacity | avg_
--------+------+------+--------+----------+--------+---------+----------+-----
     48 |   49 |   98 | [NULL] | [NULL]   | [NULL] | [NULL]  |        0 |   0.
     49 |   50 |   99 | [NULL] | [NULL]   | [NULL] | [NULL]  |        0 |   0.
     50 |   51 |  100 | [NULL] | [NULL]   | [NULL] | [NULL]  |        0 |   0.
(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 quote 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).

Parent topic: ybsql Reference

ybsql \copy Command

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. No superuser privileges are required. For example, the user running this command only requires `INSERT` privileges on the `team` table:

```
premdb=# \copy team from newteam.csv with (delimiter ',');
COPY 50
```

In This Section

- \copy Parameters
- \copy Examples

Parent topic: ybsql Reference

\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.

Parent topic: ybsql \copy Command

\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 '/ho
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, forc
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, nu
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 |
```

Parent topic: ybsql \copy Command

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 | a
--------+------+------+--------------+----------+-------+--------------+----------+-
    21 |   22 |   71 | Leeds United | Whites   | Leeds | Elland Road |    39460 |
(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=r
<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\_p
\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|
```

-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,
CREATE TABLE
create table team(teamid smallint, htid smallint, atid smallint, name varchar(30), r
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 smalli
CREATE TABLE
...
```

Parent topic: ybsql Reference

ybsql Properties and Variables

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.

In This Section

- ybsql Properties
- ybsql Display Properties
- ybsql User Variables
- ybsql Prompt Variables
- Examples with ybsql Variables

Parent topic: ybsql Reference

ybsql Properties

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 fatal 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 a number of `%` 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
```

Parent topic: ybsql Properties and Variables

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-hand margin. When the `wrapped` format wraps data from one line to the next without a newline character, a dot (`.`) is shown in the right-hand margin of the first line, and again in the left-hand margin of the following line.
- `unicode` : Unicode box-drawing characters. Newlines in data are shown using a carriage return symbol in the right-hand margin. When the data is wrapped from one line to the next without a newline, an ellipsis is shown in the right-hand margin of the first line, and again in the left-hand 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)
```

Parent topic: ybsql Properties and Variables

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.

Parent topic: ybsql Properties and Variables

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

Substitution	Description
`%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
```

Parent topic: ybsql Properties and Variables

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::times
   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
```

Parent topic: ybsql Properties and Variables

Configuration Parameters

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.

Parent topic: Yellowbrick Documentation

In This Section

- General
- YRS Alerting Parameters

General Parameters

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.

Parent topic: Configuration Parameters

In This Section

- rowstore_block_threshold_percent
- rowstore_read_only_threshold_percent
- rowstore_warning_threshold_percent

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 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).

Parent topic: General Parameters

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 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).

Parent topic: General Parameters

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 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).

Parent topic: General Parameters

Yellowbrick Row Store (YRS) YRS Alerting Parameters

> **NOTE**
>
> Advanced usage: Please contact Customer Support if you need to change the default values for these configuration parameters.

Yellowbrick implements a number of 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.

Parent topic: Configuration Parameters

In This Section

- yrs_data_files_count_minor_threshold
- yrs_data_files_count_major_threshold

- yrs_data_files_count_critical_threshold
- yrs_commit_records_count_minor_threshold
- yrs_commit_records_count_major_threshold
- yrs_commit_records_count_critical_threshold
- yrs_delete_records_count_minor_threshold
- yrs_delete_records_count_major_threshold
- yrs_delete_records_count_critical_threshold
- yrs_unused_files_count_minor_threshold
- yrs_unused_files_count_major_threshold
- yrs_unused_files_count_critical_threshold

yrs_data_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 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();
```

Parent topic: YRS Alerting Parameters

yrs_data_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 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();
```

Parent topic: YRS Alerting Parameters

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();
```

Parent topic: YRS Alerting Parameters

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();
```

Parent topic: YRS Alerting Parameters

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();
```

Parent topic: YRS Alerting Parameters

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();
```

Parent topic: YRS Alerting Parameters

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();
```

Parent topic: YRS Alerting Parameters

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();
```

Parent topic: YRS Alerting Parameters

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();
```

Parent topic: YRS Alerting Parameters

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();
```

Parent topic: YRS Alerting Parameters

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();
```

Parent topic: YRS Alerting Parameters

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();
```

Parent topic: YRS Alerting Parameters

Database Administration

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 *system* administration and maintenance tasks at the cluster level, see System Management.

For information about bulk loading tables, see Bulk Loading Tables.

In This Section

- Database Objects
- Database Limits
- Creating Databases
- Managing Tables and Views
- Managing the Row Store
- Generating DDL
- Running Queries
- Cross-Database Queries
- Canceling Queries
- Managing Idle Sessions
- Encrypting Sensitive Data
- Creating Stored Procedures
- Managing Database Users
- LDAP Integration
- Checking the Server Version
- System Views

Parent topic: Yellowbrick Documentation

Database Objects

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 functions (UDFs)
- User-defined types

Parent topic: Database Administration

Database Limits

This section lists the limits that are applied to database objects in Yellowbrick databases.

Database Limit	Value	Notes
Databases per appliance	1,000	The maximum number of databases includes the `yellowbrick` database, so the effective limit is 999 user-created databases.
Tables per appliance	70,000	There is no limit on the number of tables per database, only a limit on the total number per appliance. This limit applies to persistent tables only.
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 or view	2000	Increased in Version 5.2.9. In addition to being a storage limit, this limit defines the maximum number of columns that can be placed in a row. Therefore, the output of a `UNION` query, for example, is limited to 2000 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 2000 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)`

Database Limit	Value	Notes
		`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	264	
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.

Database Limit	Value	Notes
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.
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.
Replicas per database	1	See Database Replication.

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 appliance*, 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.

Parent topic: Database Administration

Creating 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
```

Database Modes

The backup and restore and database replication features both make use of special modes that make databases available or unavailable for write operations. See HOT_STANDBY and READONLY Modes.

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.

Parent topic: Database Administration

Managing Tables and Views

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.

External Tables

Yellowbrick also supports external tables, which are tables that are stored in files outside the database. You can create an external table by "unloading" the results of any valid Yellowbrick query into a specific file on an NFS-mounted file system. You can also read from an external table by selecting from it directly or by using `CTAS` or `INSERT` statements to select external data that creates or updates other tables.

See CREATE EXTERNAL TABLE.

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

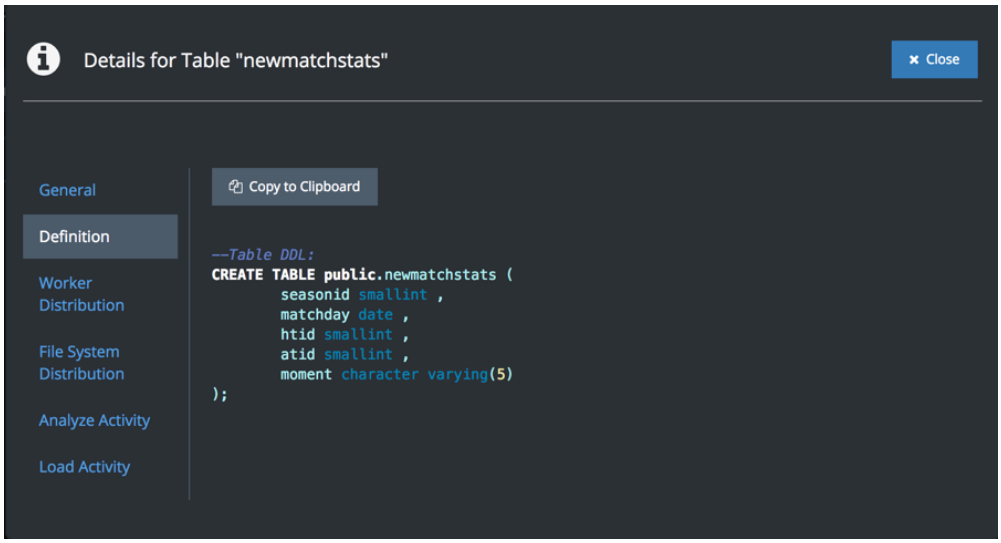
By default, tables are automatically analyzed. This analysis updates column-level statistics that the planner uses to optimize query execution. See Auto-Analyzing Tables.

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

Use the SMC to copy the DDL for a table to the clipboard. Go to **Databases** and select a database. Then double-click the table ID. On the Details screen, click **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` as long as 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.

In This Section

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

Parent topic: Database Administration

Distributing Data

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 worker 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 worker 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 var
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 varch
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 worker 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.

Parent topic: Managing Tables and Views

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 a number of 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 worker 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 resul
```

- 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.

Keep in mind 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? In other words, 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, phone 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 number 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.

Parent topic: Managing Tables and Views

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 the SMC. In the SMC, go to **Databases**, select the database, double-click the row for the table you want to check, and click **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, you can disable the immediate ANALYZE operations that happen after each INSERT. To disable these ANALYZE operations, set the `ybd_analyze_after_writes` configuration parameter to `OFF`. This parameter defaults to `ON`.

After changing this parameter, you have to reload the system configuration (or restart the database) before the new behavior takes effect. To set the parameter globally, use an ALTER SYSTEM command:

```
yellowbrick=# alter system set ybd_analyze_after_writes to off;
```

To set the parameter at the session level, you can use ALTER ROLE. For example:

```
premdb=# create role insert_select;
CREATE ROLE
premdb=# alter role insert_select set ybd_analyze_after_writes to off;
ALTER ROLE
```

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.

Disabling Immediate ANALYZE Operations for Bulk Loads

If your workload consists of a large number of continuous trickle-feed loads using `ybload`, you may need to disable the immediate ANALYZE operations that happen after each `ybload` operation. To disable these ANALYZE operations, set the `ybd_analyze_after_loads` configuration parameter to `OFF`. This parameter defaults to `ON`.

After changing this parameter, you have to reload the system configuration (or restart the database) before the new behavior takes effect. To set the parameter globally, use an ALTER SYSTEM command:

```
yellowbrick=# alter system set ybd_analyze_after_loads to off;
```

To set the parameter at the session level, you can use ALTER ROLE. For example:

```
premdb=# create role ybload_user;
CREATE ROLE
premdb=# alter role ybload_user set ybd_analyze_after_loads to off;
ALTER ROLE
```

After a configuration reload, immediate ANALYZE operations will be disabled for all `ybload` sessions run by `ybload_user`. The advantage to disabling the parameter in this way is that it will remain enabled for other write operations.

Note that setting this parameter to `OFF` does not prevent an immediate ANALYZE from being run after intermediate load operations, such as after an INSERT into a target table from a staging table. However, setting `ybd_analyze_after_writes` to `OFF` will prevent an immediate ANALYZE in these cases.

Parent topic: Managing Tables and Views

External 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, u
```

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 as long as 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.

Parent topic: Managing Tables and Views

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 worker nodes. Each value is stored as a 64-bit `BIGINT` number. The last 10 bits identify the worker. 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 worker nodes on the cluster. Each worker 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 worker 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.

Parent topic: Managing Tables and Views

System Columns in 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
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 worker 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. You cannot select row IDs from external tables.

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=
     rowid       | row_number_from_rowid | seasonid |      matchday       | htid | a
-----------------+-----------------------+----------+---------------------+------+-
 1126011584388774 |                  7846 |       21 | 2013-02-23 00:00:00 |    2 |
 1126011584388775 |                  7847 |       21 | 2012-09-29 00:00:00 |    2 |
 1126011584388776 |                  7848 |       21 | 2013-04-16 00:00:00 |    2 |
 1126011584388777 |                  7849 |       21 | 2012-11-10 00:00:00 |    2 |
 1126011584388778 |                  7850 |       21 | 2013-01-30 00:00:00 |    2 |
 1126011584388779 |                  7851 |       21 | 2013-01-13 00:00:00 |    2 |
 1126011584388780 |                  7852 |       21 | 2013-04-28 00:00:00 |    2 |
 1126011584388781 |                  7853 |       21 | 2012-12-29 00:00:00 |    2 |
 1126011584388782 |                  7854 |       21 | 2013-04-13 00:00:00 |    2 |
 1126011584388783 |                  7855 |       21 | 2012-10-27 00:00:00 |    2 |
(10 rows)
```

The following example finds out where (on which worker 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 worker 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_
            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)
```

Parent topic: Managing Tables and Views

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. In the event of trying to consume more space than 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 impacts 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 usage objects. You can change the maximum to a value from 100 to 20000000 using an ALTER SYSTEM command. For example, if you wanted to increase the maximum from 500k objects to 1000000:

```
yellowbrick=# alter system set max_quota_objects to 1000000;
WARNING: Will be effective after the next server configuration reload, or after the
ALTER SYSTEM
```

Parent topic: Managing Tables and Views

Managing the Row Store

A Yellowbrick appliance contains a specific storage drive on the manager node known as the *row store*. The row store is a storage device for data that is not loaded directly to the *column store* on the worker nodes. 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 (file systems on the worker 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 disk 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 worker 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.

The maximum size depends on the limit set for the row store drive during installation. The default row store size is 500GB (reported as `549755813888` bytes in sys.table).

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.
- **Run `ybcli` commands**: status storage `and` system status.
- **Check for system alerts**: At 100% full, the system sends a `MAJOR` alert with the message:

```
The yellowbrick row store is full, user inserts will be slower.
```

Parent topic: Database Administration

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 'md5
...
```

Parent topic: Database Administration

Running Queries

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:

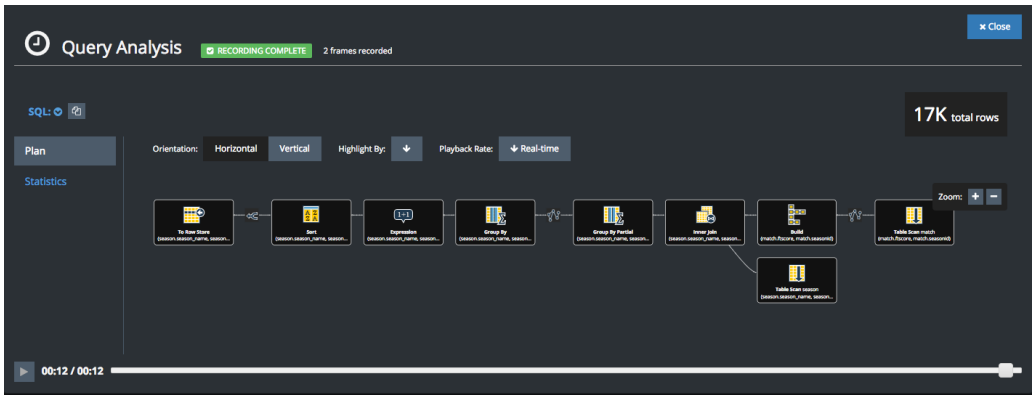
- You can run read queries across physical databases, but you cannot write to remote databases. See Cross-Database Queries.

- 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.

Recording and Playing Back Queries in the SMC

Administrators can use the Query Analysis feature in the SMC to record the progress of a query and play it back for analysis.

1. Go to **Monitor** > **Query Performance** and select a query. On the **Query Details** screen, click **Analyze**, then **Execute and Record**.
2. The analysis will be recorded as the query runs.
3. Play back the query recording by clicking the play icon in the bottom-left corner of the Query Analysis screen. For example:



Note: The Query Analysis feature is not supported with Firefox Private Browsing.

Parent topic: Database Administration

Cross-Database Queries

You can write queries that select from tables in different databases on the same appliance. You can run cross-database queries from any database that belongs to the same appliance as the tables listed in the `FROM` clause. The database you are connected to 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 *read* from remote databases are supported. However, writing data to remote 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 the 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 the 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.matc
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 on the same appliance:

```
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
     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
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
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
```

Parent topic: Database Administration

Canceling Queries

You can cancel a running query by using the CANCEL command. This command takes the `query_id` of an active query as its only argument. You can use the **Monitor** > **Active Queries** feature in the SMC or query the `sys.query` view to get the ID of the query you want to cancel.

To cancel an active query:

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 cancelled by an administrator.

Aborting a Query with WLM Rules

You can cancel queries conditionally that 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 seasor
ERROR:  canceling statement due to statement timeout
```

Parent topic: Database Administration

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 appliance 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. 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`.
- `idle_in_transaction_session_timeout`: 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`.

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 appliance.

Both parameters accept settings with the following units: `ms`, `s`, `min`, `h`, and `d`.

You can modify these parameters with the following commands:

- ALTER SYSTEM: Only superusers 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:

```
premdb=# alter system set idle_session_timeout to '2h';
ALTER SYSTEM
premdb=# 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:

```
premdb=# set idle_session_timeout to 999999;
SET
premdb=# 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 superuser to run this command, but you must have permissions to alter the role in question.

```
premdb=# alter role bobr set idle_session_timeout to 999999;
ALTER ROLE
premdb=# show idle_session_timeout;
 idle_session_timeout
----------------------
 1h
(1 row)
```

Parent topic: Database Administration

Encrypting Sensitive Data

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.

Configuring Encryption for SQL Access

The `ENCRYPT_KS` and `DECRYPT_KS` functions are backed by an integrated "vault" on the manager node, which functions as a key manager (or keystore). The same vault is used in the manager node to secure keys for unlocking encrypted drives; see System Hardware Encryption.

Before you can use the SQL encryption functions, you have to unlock the keystore file and complete the following configuration steps:

1. Enable SQL encryption for the system 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.

2. 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
```

3. 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)
```

4. 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`.

5. 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 f
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:

- ybcli config keystore
- 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 parameter 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=RAND
weekly_wages varchar(100) encrypted with(COLUMN_ENCRYPTION_KEY=playerkey,ENCRYPTION_
avg_mins_per_match double precision,
matches_played real,
cob varchar(100) encrypted with(COLUMN_ENCRYPTION_KEY=playerkey,ENCRYPTION_TYPE=RAND
CREATE TABLE
premdb=# \d player
                                                                    Table "pub
      Column       |          Type          |
-------------------+------------------------+-------------------------------------
 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
 weekly_wages       | character varying(100) | encrypted with (column_encryption_key
 avg_mins_per_match | double precision       |
 matches_played     | real                   |
 cob                | character varying(100) | encrypted with (column_encryption_key

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 clear 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     | weekl
----------+--------+----------+-----------+----------+----------+------------+------
       15 |     23 |       27 | Virgil    | Van Dijk | D        | 07-08-1991 | 180000
       16 |     24 |       27 | David     | Silva    | M        | 01-08-1986 | 250000
       17 |     22 |       27 | Jamie     | Vardy    | F        | 01-11-1987 | 80000
(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. For information about accessing encrypted data in a restored database, see Decrypting Columns in a Restored Database.

In This Section

- SQL Encryption Examples
- Encryption Key Rotation
- Deprecated Encryption Functions

Parent topic: Database Administration

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      | ca
------------+----------------------+-----------+-------------+------------------+-
```

```
 cdofaC71    | Bournemouth          | Cherries  | Bournemouth | Vitality Stadium |
 ddohkmN2c3  | Bradford City        | Bantams   | Bradford    | Valley Parade    |
 edIgaCd2    | Blackpool            | Seasiders | Blackpool   | Bloomfield Road  |
 fdYfaCt2    | Bolton Wanderers     | Trotters  | Bolton      | Macron Stadium   |
 g32fg8d1cp3 | West Bromwich Albion | Baggies   | Birmingham  | The Hawthorns    |
(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
"select encrypt_ks(concat_ws(':',teamid,htid,atid),yb100key) as ids, name, nickname,
Password for user yb100:
13:01:14.577 [ INFO] ABOUT CLIENT:
   app.cli_args         = "-d" "premdb" "--username" "yb100" "-W" "--format" "csv" "
   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, mixe
   jvm.options          = "-Xms512m, -Xmx6g, -XX:+UseG1GC, -Dapp.name=ybunload, -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
```

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 decrypte
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
```

Parent topic: Encrypting Sensitive Data

Encryption Key Rotation

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
```

Parent topic: Encrypting Sensitive Data

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      | ca
--------+----------------------+----------------+------------+------------------+--
      2 | Aston Villa          | RuCsrql{7K2    | Birmingham | Villa Park       |
      4 | Birmingham City      | FiSmyyE        | Birmingham | St. Andrew's     |
     43 | West Bromwich Albion | FOyq{KlzQ0     | Birmingham | The Hawthorns    |
      1 | Arsenal              | A8jstaVuQ0     | London     | Emirates Stadium |
     12 | Charlton Athletic    | CCCqmyF{Q0     | London     | The Valley       |
     13 | Chelsea              | T8isgKF|7CpX21 | London     | Stamford Bridge  |
     15 | Crystal Palace       | 8OyqraFu       | London     | Selhurst Park    |
     18 | Fulham               | EWCmjqFzCGYX   | London     | Craven Cottage   |
     32 | Queens Park Rangers  | 5WysfyE        | London     | Loftus Road      |
     41 | Tottenham Hotspur    | USTmhyE        | London     | White Hart Lane  |
     44 | West Ham United      | 5OSsqaVuQ0     | London     | Upton Park       |
     46 | Wimbledon            | 9Wisg0         | London     | Selhurst Park    |
(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 o
from encrypted_team_table
...
```

Note: This example presents the actual key value in clear 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
      encrypt
---------------------
 CyftgWky|C2f10
 V8zzguku5qYmwD8
 T8jpjKkwi0
 BuyvqKkwi0
 R0VtqKkviC2
...
```

Parent topic: Encrypting Sensitive Data

Creating Stored Procedures

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.

Parent topic: Database Administration

Managing Database Users

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.

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. 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 superusers, but then revoke specific privileges that a superuser would normally hold. Conversely, a regular user (non-superuser) can be granted certain privileges that would normally belong to a superuser. 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.

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
- sys.database

The following functions are useful for listing the privileges that have been granted to specific users and roles:

- HAS_DATABASE_PRIVILEGE
- HAS_FUNCTION_PRIVILEGE
- HAS_KEY_PRIVILEGE
- HAS_SCHEMA_PRIVILEGE
- HAS_SYSTEM_PRIVILEGE
- HAS_TABLE_PRIVILEGE

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 via an LDAP server, see LDAP Integration.

Parent topic: Database Administration

LDAP Integration

Lightweight Directory Access Protocol (LDAP) is an application protocol for directory access (like a phone 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 Versus LDAP Users and Groups

Yellowbrick users and groups may be local to the Yellowbrick appliance, LDAP-based, or a combination of both. *Enabling LDAP on your appliance 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 appliance. 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.

In This Section

- Importing Certificates
- LDAP Sample Schema
- LDAP Authentication
- Synchronizing Users and Groups

Parent topic: Database Administration

Importing Certificates

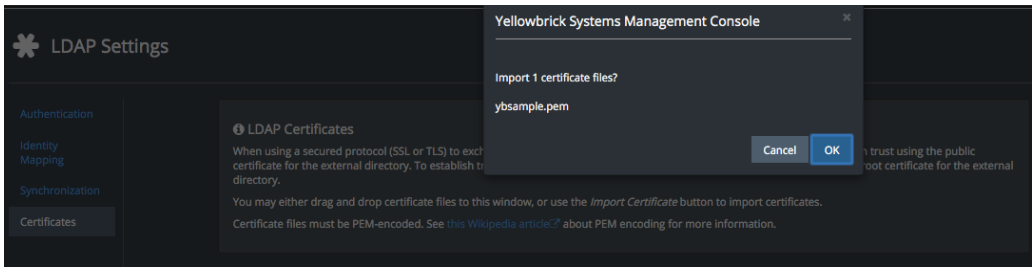
If you are using LDAP over Secure Sockets Layer (SSL) or LDAP with Transport Layer Security (TLS), you must follow these steps before setting up authentication.

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.

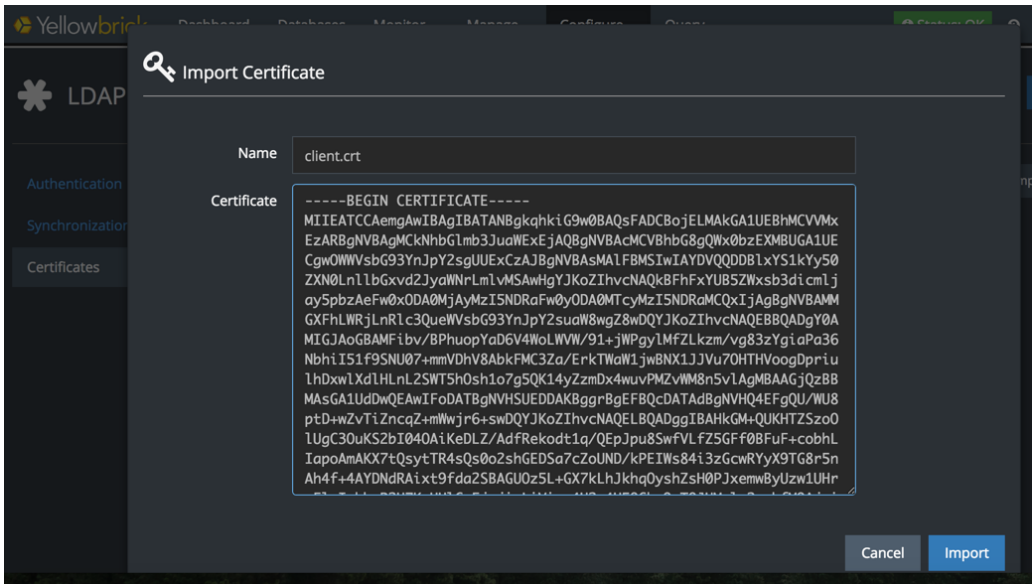
Certificate files must be PEM-encoded.

1. Export the SSL certificate from the LDAP Server.
2. Log into 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:

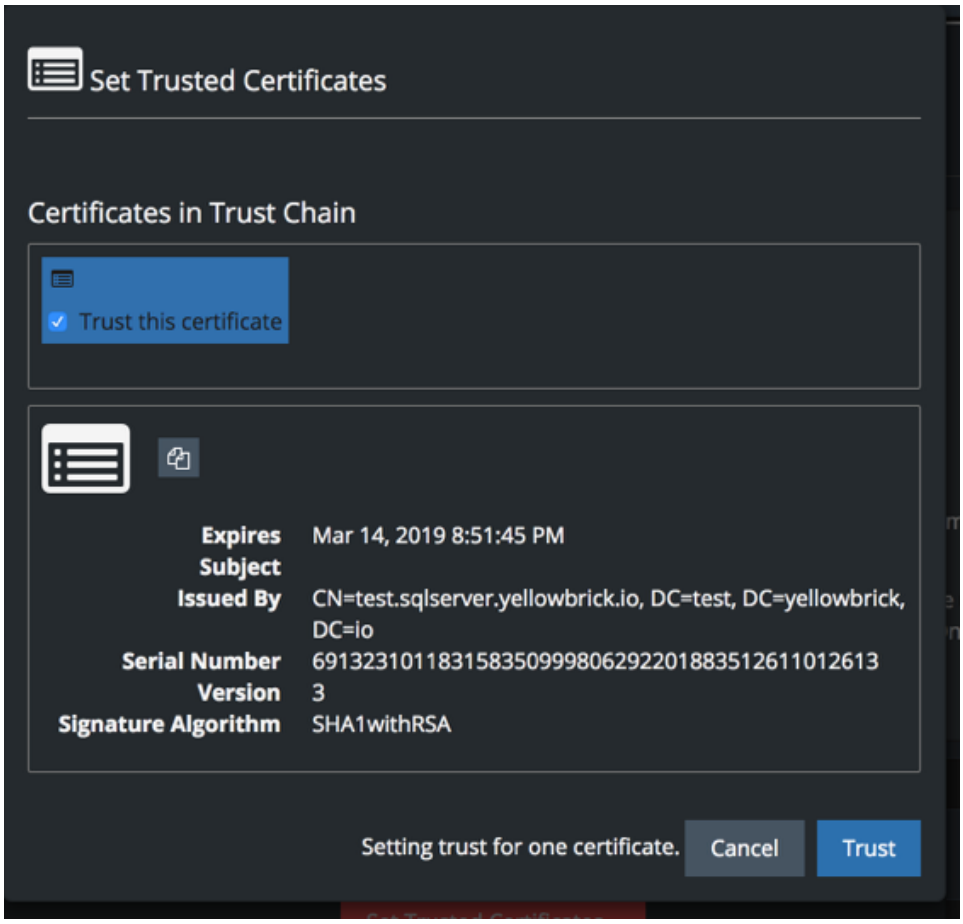


- Click **Import Certificate**, then copy and paste the contents of the certificate. The certificate content must begin with `-----BEGIN CERTIFICATE-----` and end with `-----END CERTIFICATE-----`. For example:



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.

Parent topic: LDAP Integration

LDAP Sample Schema

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.yellowbr
InetOrgPerson	yb_db_developers	developer_1	developer1	developer1@test.yellow
InetOrgPerson	db_marketing_role	marketing_1	marketing1	marketing1@test.yellow
InetOrgPerson	db_sales_role	sales_1	sales1	sales1@test.yellowbric
InetOrgPerson	yb_db_admins	user_1_dba	user1-dba	user1-dba@test.yellow
InetOrgPerson	yb_prod_users	elt_user	eltuser	eltuser@test.yellowbric
group	yb_svc_logins	web_user	webuser	webuser@test.yellowbr
user	none	ad_search	ad_search	ad_search@test.yellow

Parent topic: LDAP Integration

LDAP Authentication

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 into 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 the 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"
```

In This Section

- LDAP Authentication Modes
- Setting Up Authentication
- LDAP Authentication Settings
- Bind Examples
- Search, then Bind Example

Parent topic: LDAP Integration

LDAP Authentication Modes

Yellowbrick supports two modes for LDAP authentication:

- Bind
- Search, then Bind

See also LDAP Authentication Settings.

Bind Mode

In Bind mode, the Yellowbrick appliance 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 appliance, 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 first and last name 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 appliance 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`.

Parent topic: LDAP Authentication

Setting Up Authentication

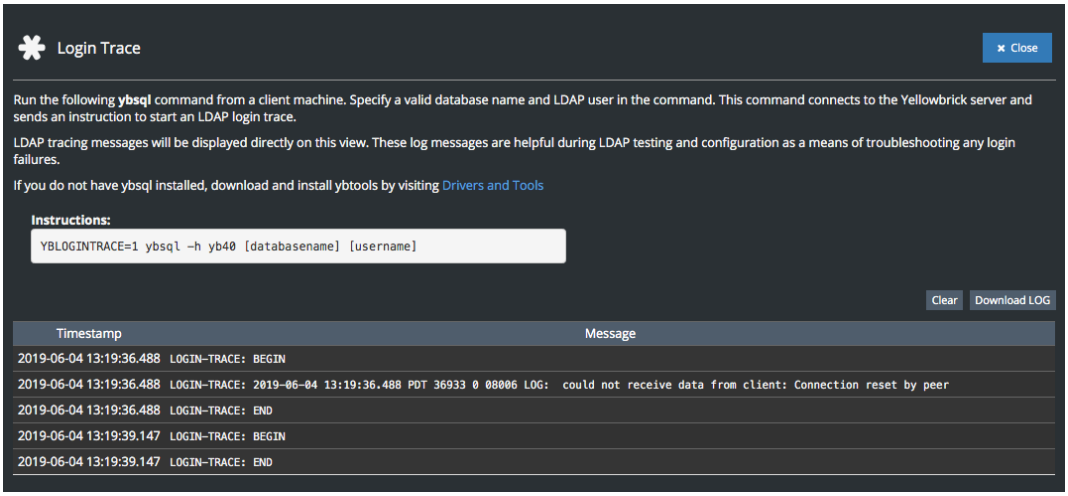
To set up LDAP authentication, follow these steps:

1. Log into a database on your appliance 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 top-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 synch 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 into the database as a new LDAP-authenticated superuser and use that account to create all of the regular users on the appliance. 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 into the database without LDAP authentication.

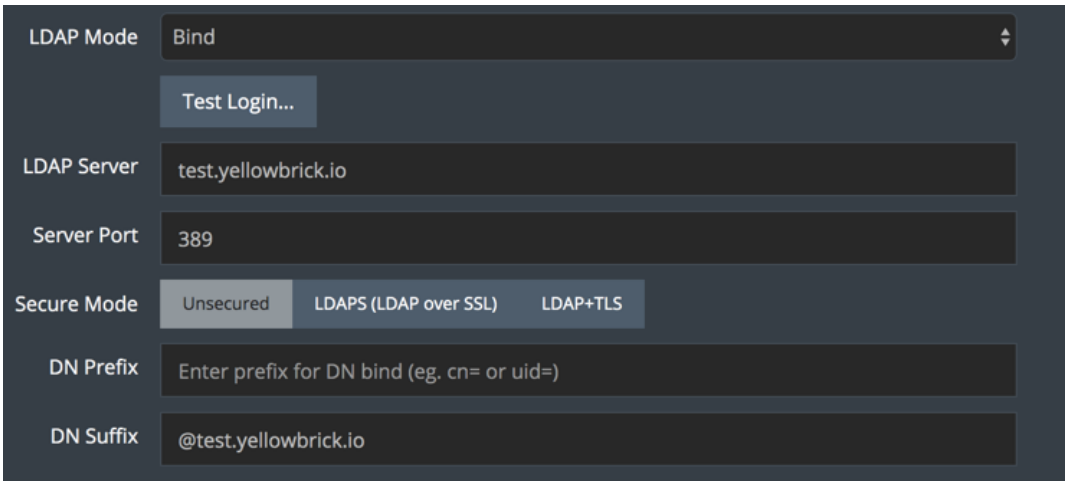
Parent topic: LDAP Authentication

LDAP Authentication Settings

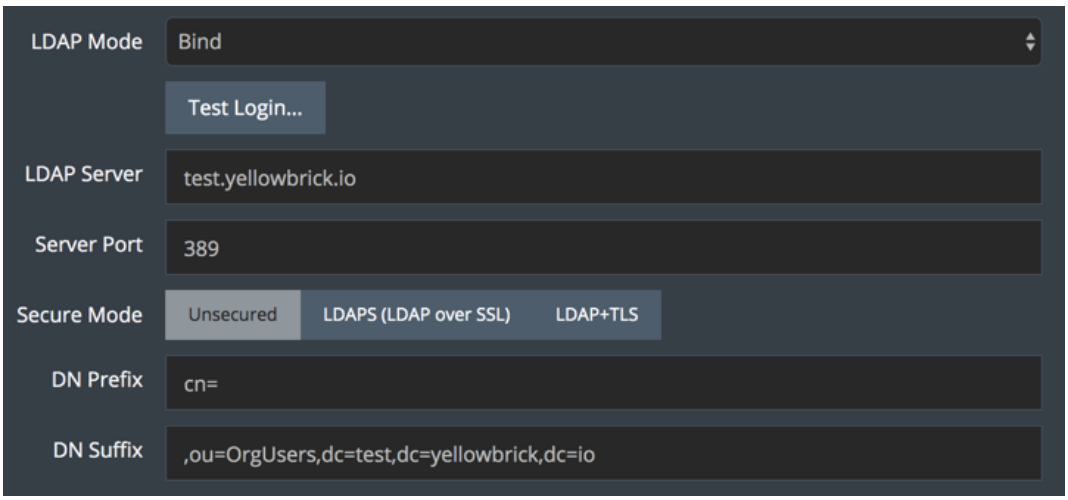
Use this section for reference when setting up LDAP authentication.

Bind Settings

The following settings apply to the Bind mode. See also LDAP Authentication Modes. The first example is for a user of a Microsoft Active Directory Server:



The second example is for a user of an OpenLDAP Server:



Test Login

Click this button to test the LDAP login credentials after filling out the fields below it.

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 first and last name 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.

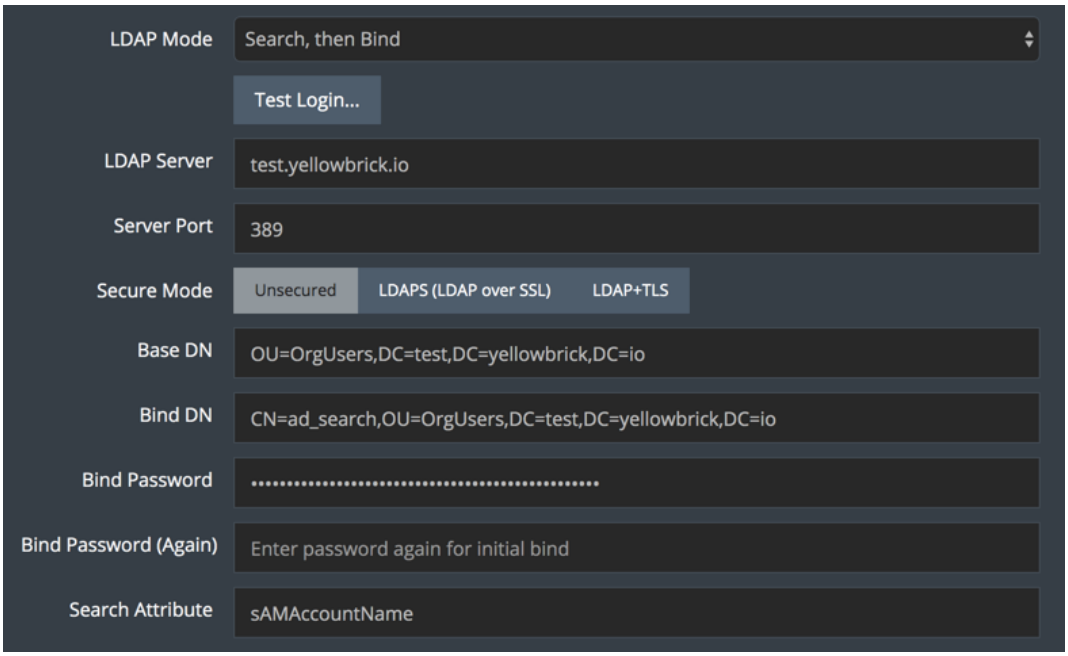
Note: No prefix is required for Active Directory configurations.

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.

Search, then Bind Settings

The following settings apply only to the Search, then Bind mode. See also LDAP Authentication Modes. Here is an example for a user of a Microsoft Active Directory Server:



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.

Parent topic: LDAP Authentication

Bind Examples

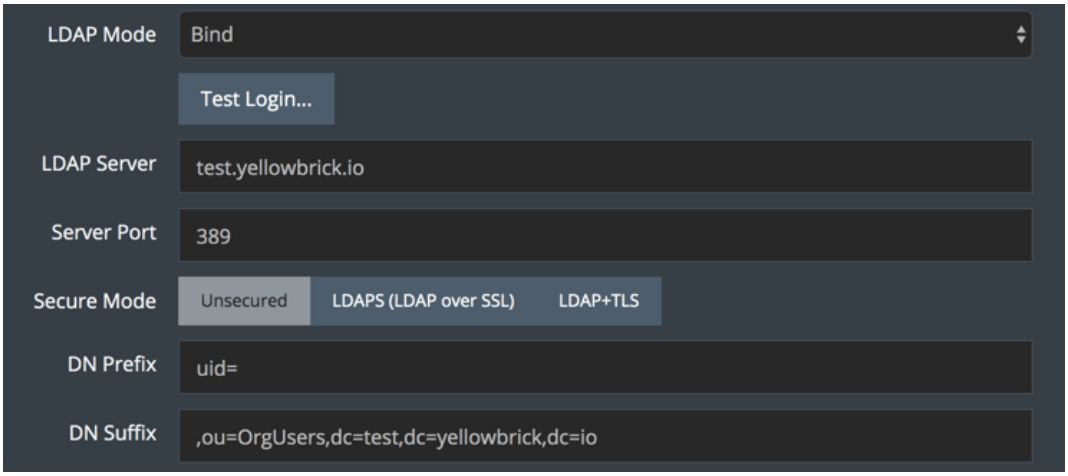
This section contains LDAP authentication examples for Bind mode. See LDAP Sample Schema for information about the LDAP groups and users associated with these examples.

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 appliance, the generated LDAP username is submitted to the LDAP server as follows:

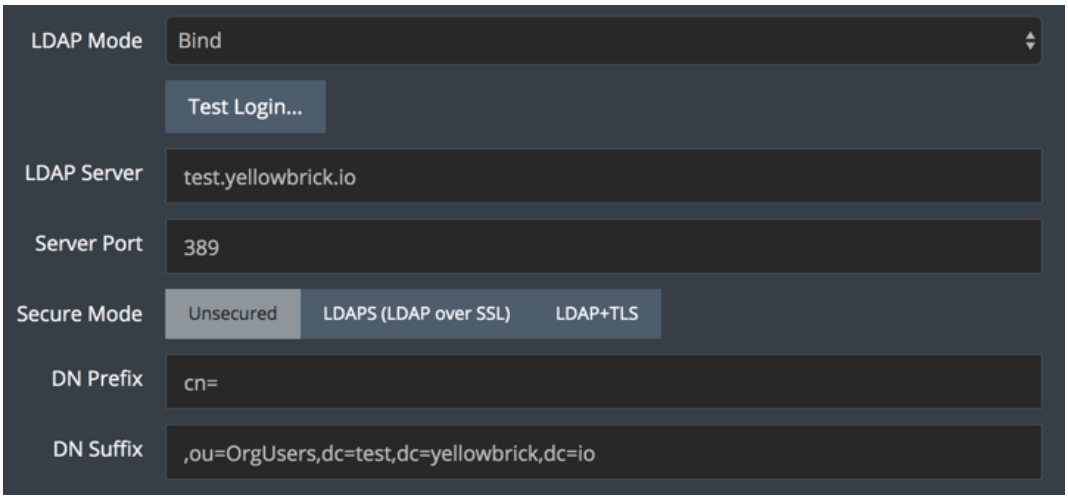
```
uid=analyst1,ou=OrgUsers,dc=test,dc=yellowbrick,dc=io
```

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`

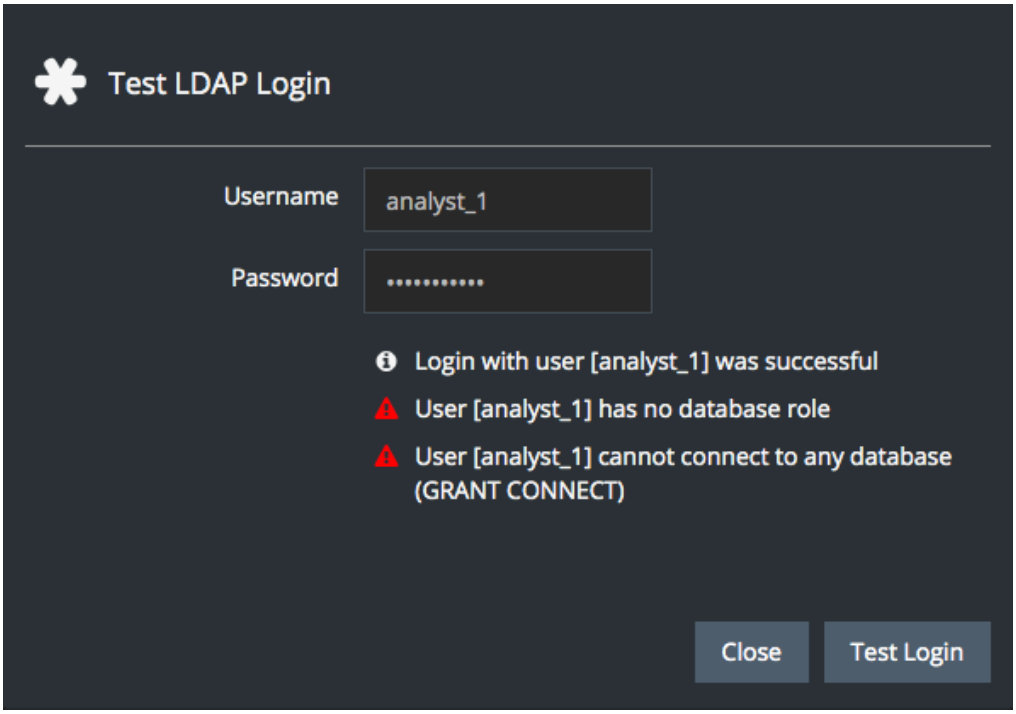
Note: You must include the comma before `ou=` in the suffix string. The `dc` entries are the components of the company domain.



When `analyst_1` logs into the appliance, the generated LDAP username is submitted to the LDAP server as follows:

```
cn=analyst_1,ou=OrgUsers,dc=test,dc=yellowbrick,dc=io
```

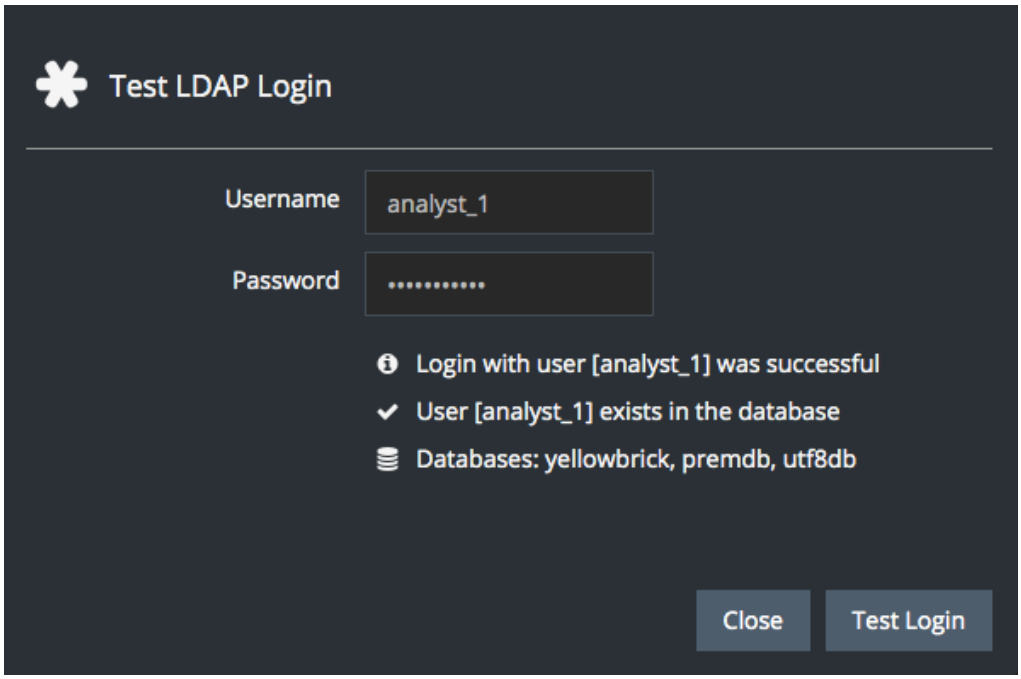
Note that `analyst_1` exists only in LDAP. There is no equivalent database user. Therefore, the login is successful, but two warnings are displayed:



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:

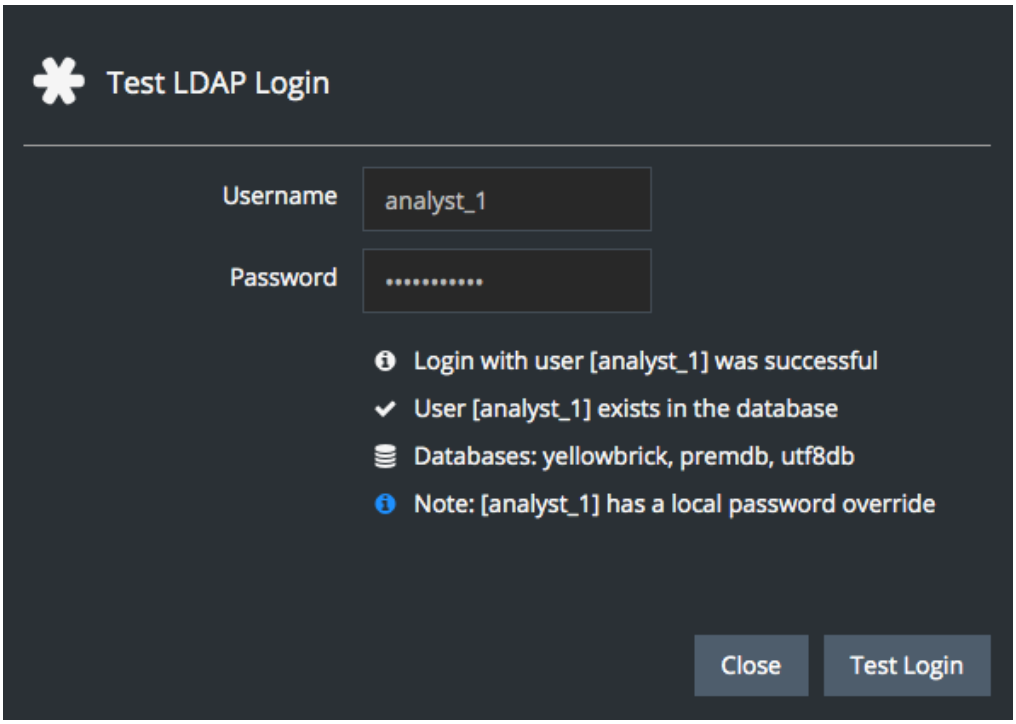


The user can now log in via LDAP, exists as a database user, and can connect to databases.

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:

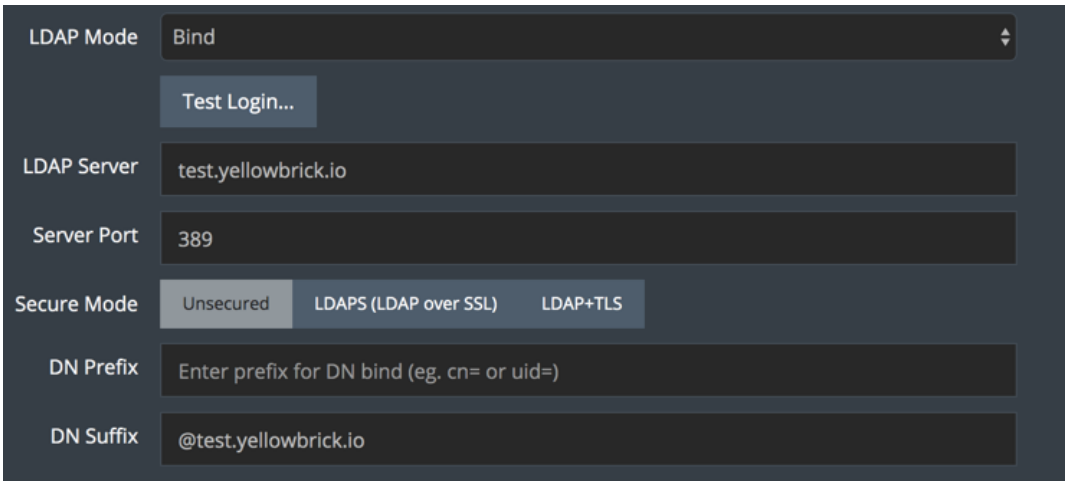


Active Directory (AD) Example with a User Principal Name (UPN)

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:

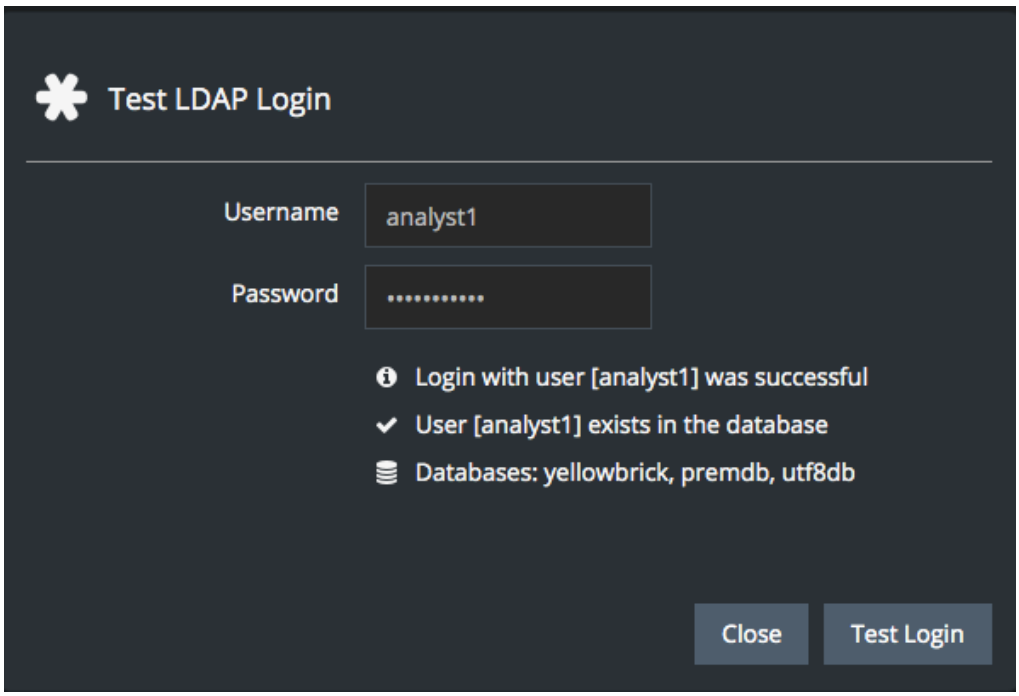
- **DN Prefix**: *no prefix*
- **DN Suffix**: `@test.yellowbrick.io`



When `analyst1` logs into the appliance, 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:



Parent topic: LDAP Authentication

Search, then Bind Example

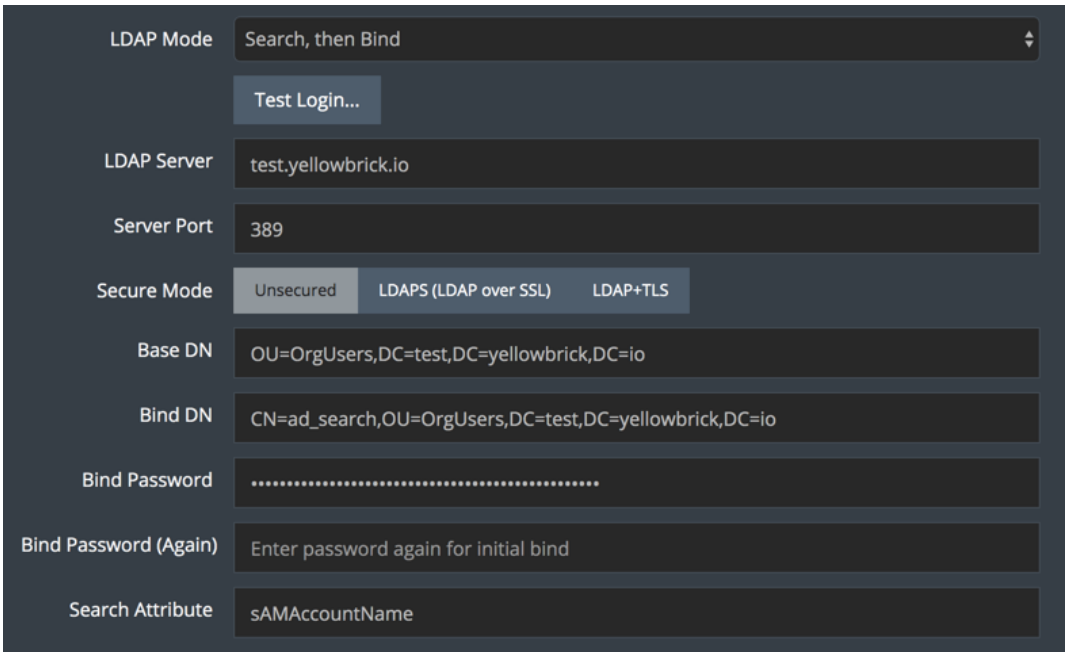
This section contains an LDAP authentication example for Search, then Bind mode with an Active Directory (AD) LDAP server. See LDAP Sample Schema for information about the LDAP groups and users associated with this example.

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**).

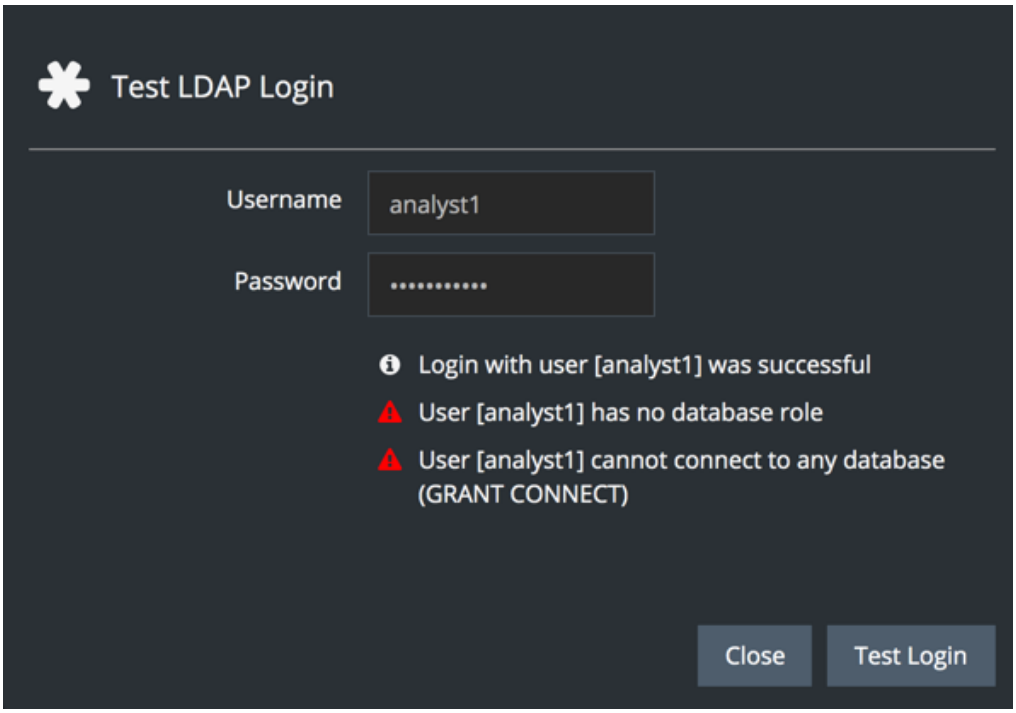
When a login is attempted, the server binds to the LDAP directory with this fixed username and password, then performs a search for the user who is trying to log into the database. The search tries to find an exact match on a specified "search attribute." In this example, the username supplied when the 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.

Parent topic: LDAP Authentication

Synchronizing Users and Groups

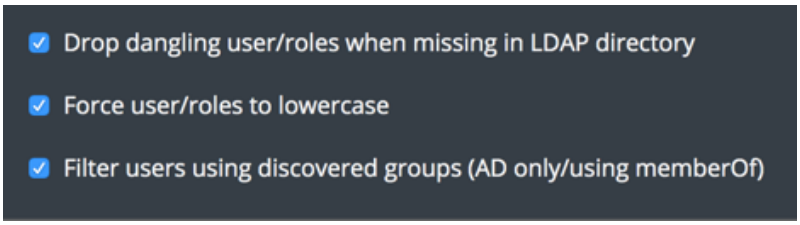
First complete the steps under Setting Up Authentication.

This procedure is an alternative to creating non-superusers manually in the database. The Yellowbrick appliance 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 SMC, which provides a number of synchronization options.

1. In the SMC, go to **Configure** > **LDAP** > **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:

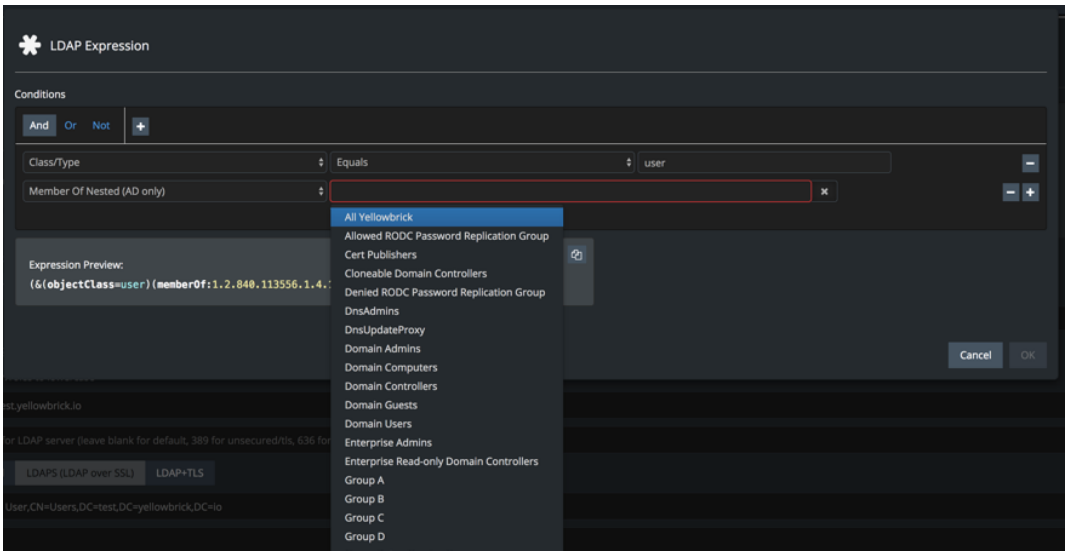


See LDAP Synchronization Settings for details.

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**.

Note: LDAP synchronization failures trigger a system alert. Both the SMC and the `ybcli system status` command show synchronization status.

In This Section

- LDAP Synchronization Settings
- LDAP Synchronization: Basic Example
- LDAP Synchronization: Examples with Filters

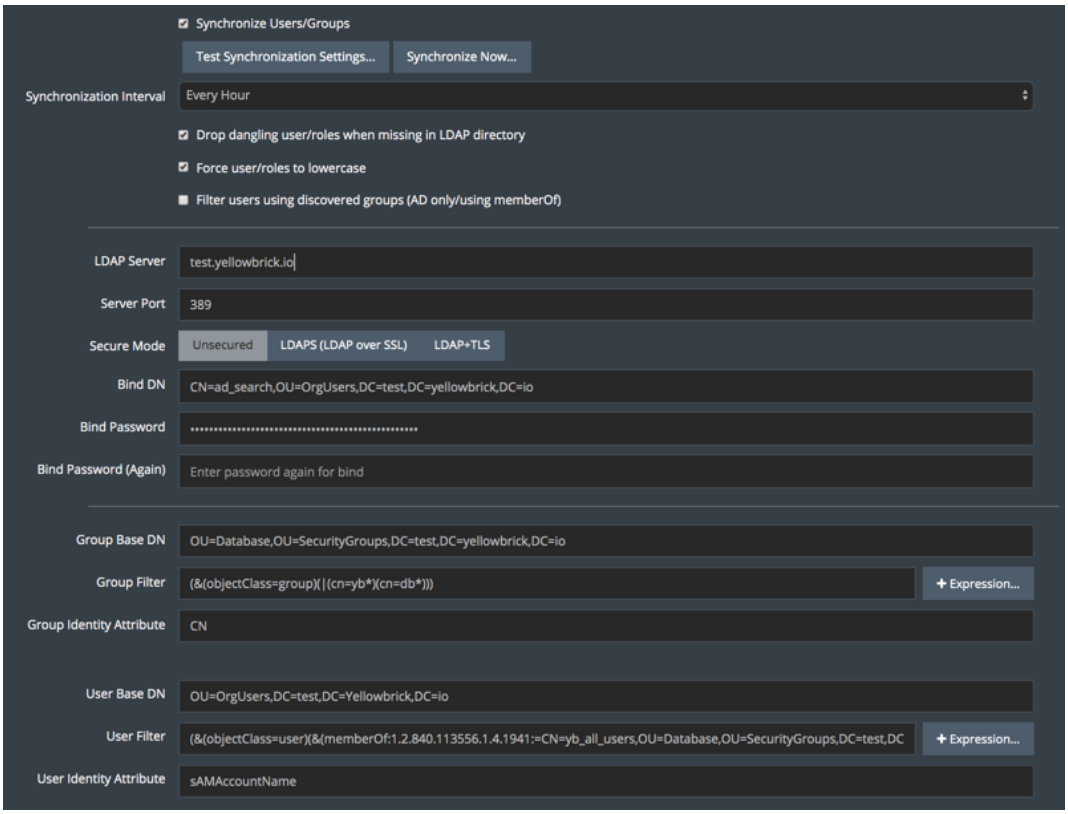
Parent topic: LDAP Integration

LDAP Synchronization Settings

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
        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.

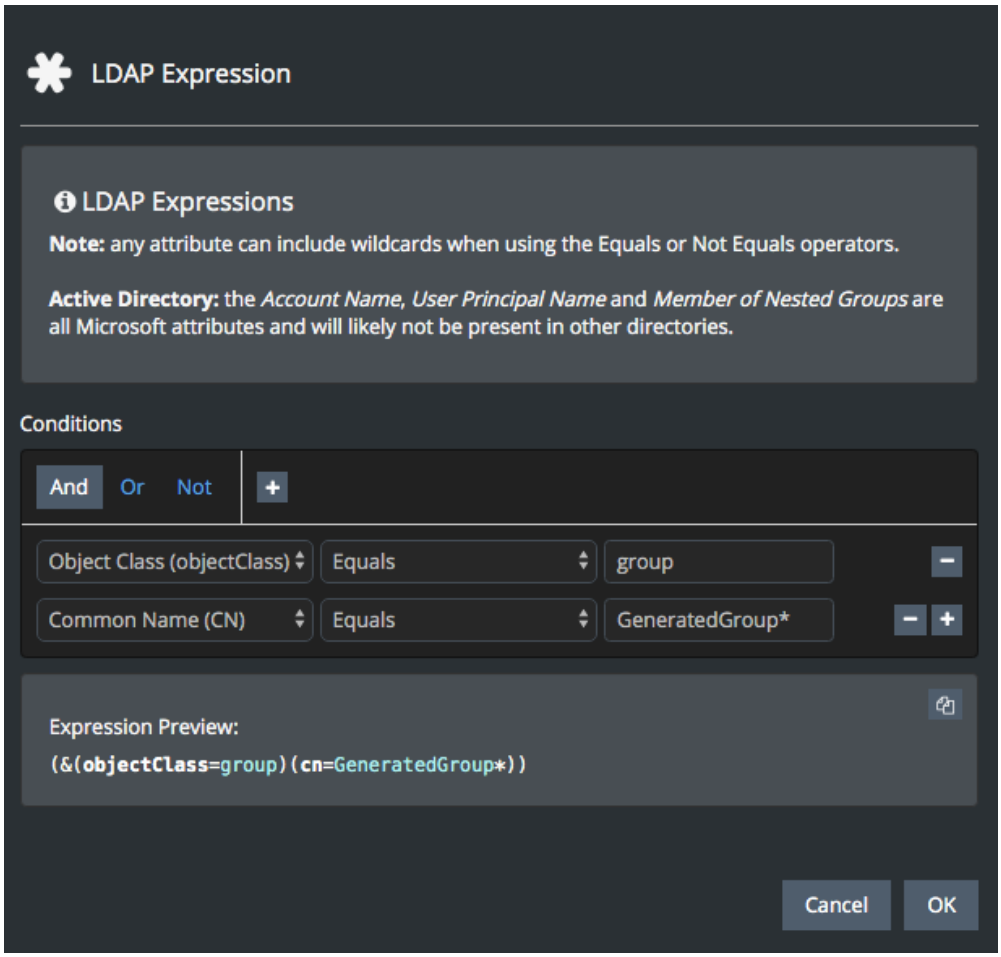
Group Base DN

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`.

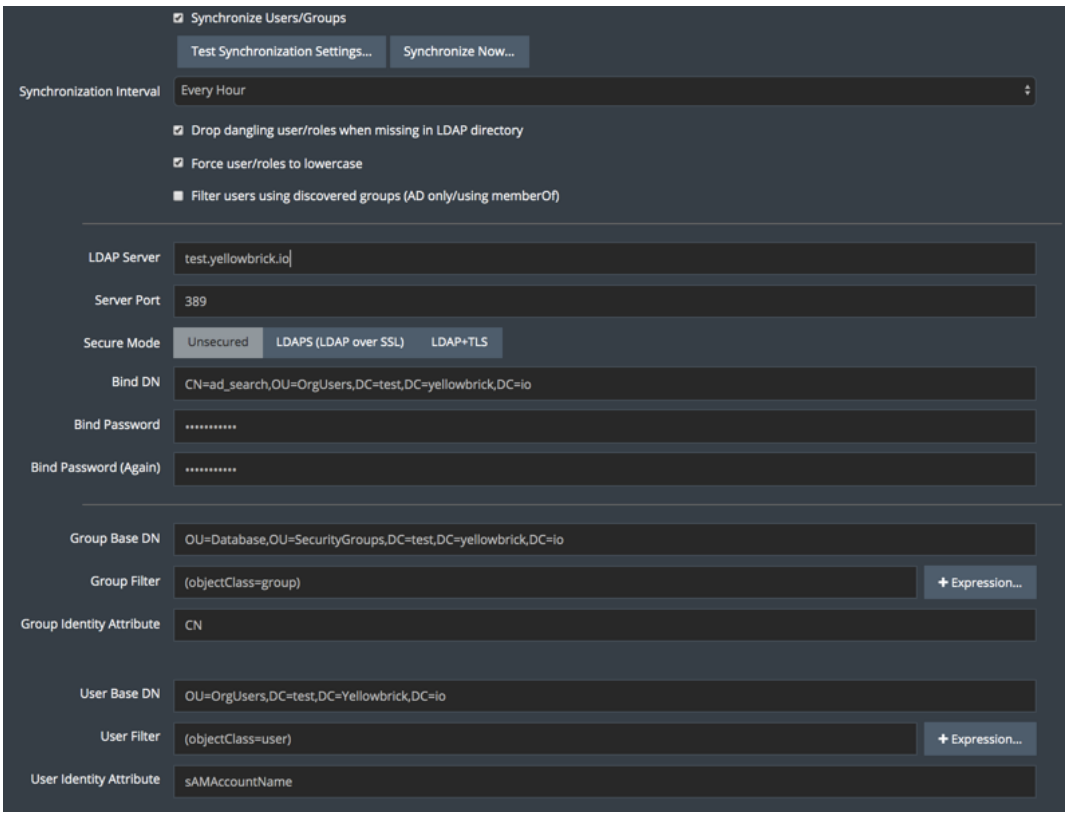
Parent topic: Synchronizing Users and Groups

LDAP Synchronization: Basic Example

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.

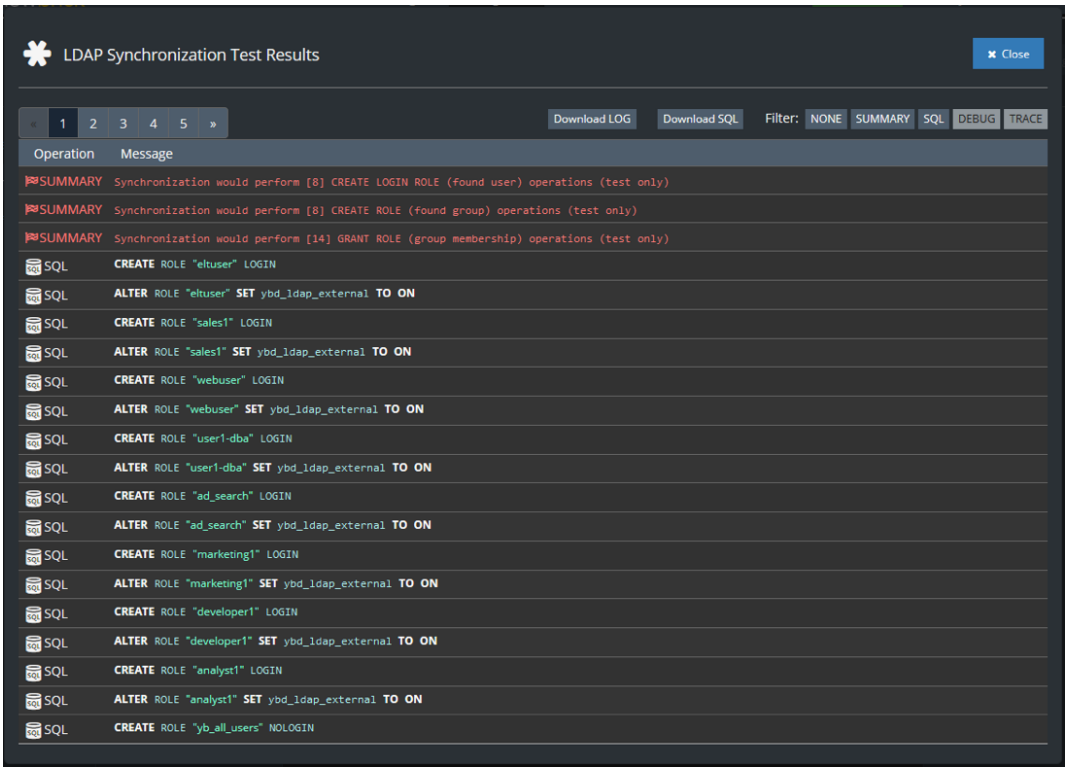
1. Set up LDAP authentication as directed in one of the previous AD examples.

2. Go to **Configure** > **LDAP** > **Synchronization**, then select **Synchronize Users/Groups**. Fill out the details as shown and click **Save Settings**.



In this case, **Group Filter** and **User Filter** are left blank, so they default to `(objectClass=group)` and `(objectClass=user)`. In other words, the server will synch *all groups and users* under the trees specified in **Group Base DN** and **User Base DN**.

3. Click **Test Synchronization Settings**. A test results page is displayed.

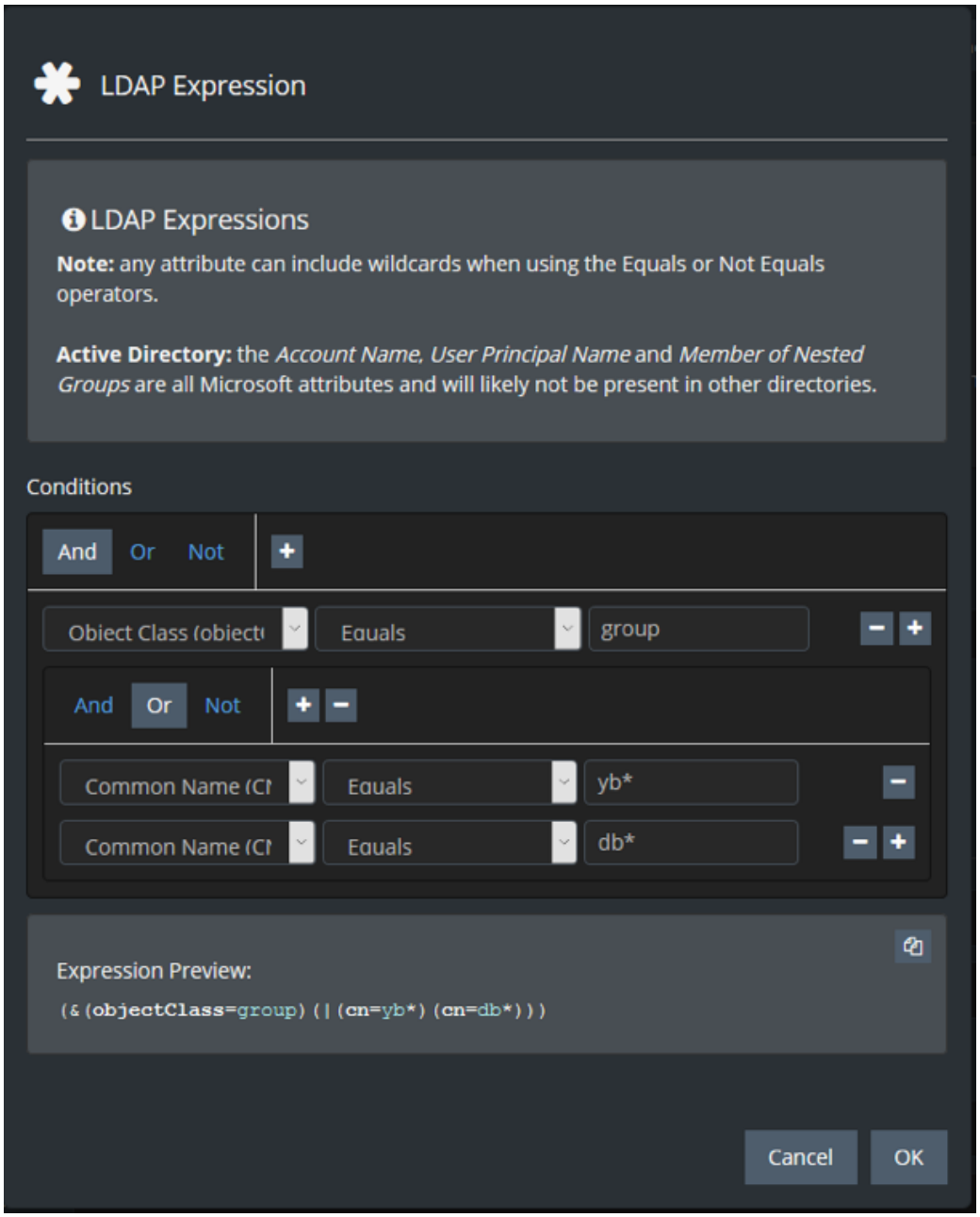


Parent topic: Synchronizing Users and Groups

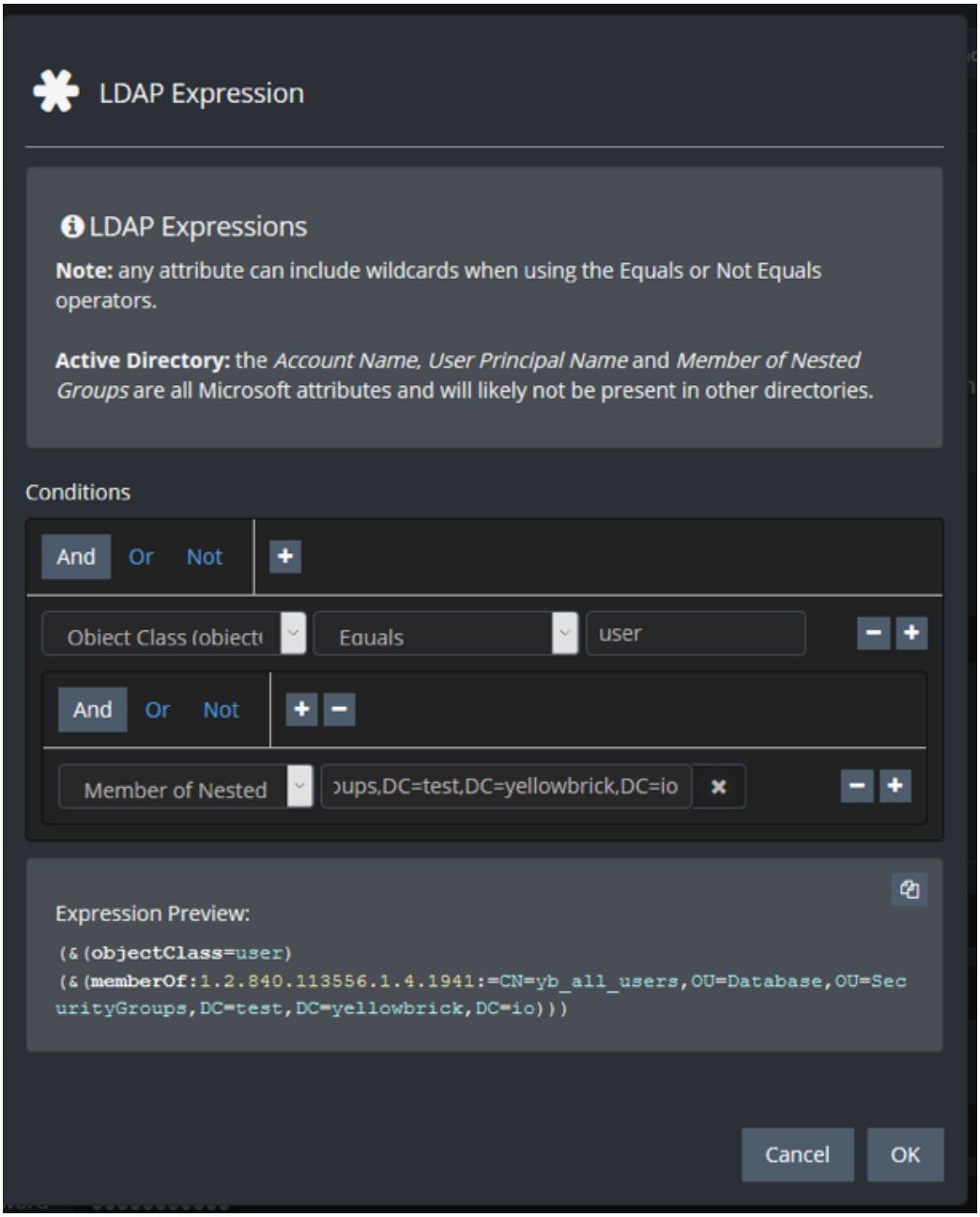
LDAP Synchronization: Examples with Filters

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.

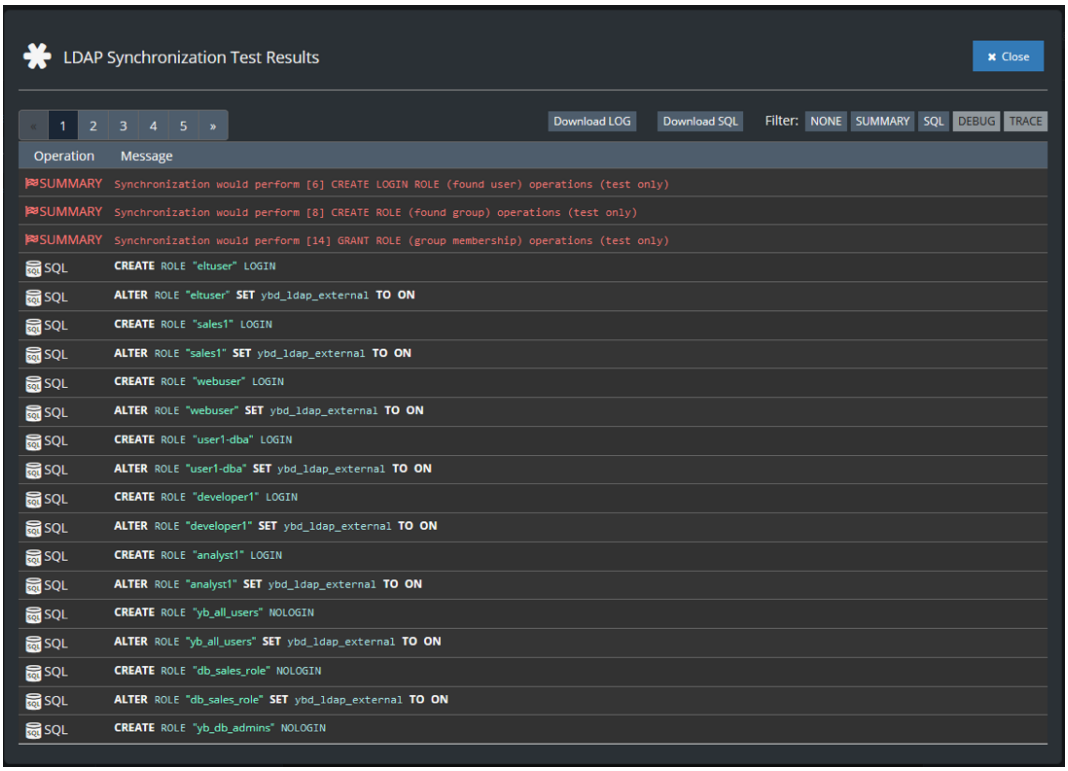
In this case, assume that your organization's LDAP directory contains additional non-Yellowbrick groups under `Database/SecurityGroups`, such as `mstr_users` and `tableau_users`, that you want to filter out during synchronization. To synchronize only those groups whose names begin with `yb` or `db`, 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 descendent groups (**Member of Nested**).



After defining the group and user filters, click **Test Synchronization Settings**. A test results page is displayed.



Parent topic: Synchronizing Users and Groups

Checking the Server Version

To find out what version of the Yellowbrick database server is running, return the current value of the `yb_server_version` configuration parameter. Use the following SHOW command:

```
premdb=# show yb_server_version;
 yb_server_version
-------------------
 4.1.0
(1 row)
```

Alternatively, you can use the `VERSION()` function:

```
premdb=# select version() from sys.const;
                version
-----------------------------------------
 Yellowbrick Database version 4.1.0-13610
(1 row)
```

Parent topic: Database Administration

System Views

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 `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, run this command:

```
premdb=# set schema 'sys';
SET
```

Then list the views that belong to the schema:

```
yellowbrick=# \dvS
              List of relations
 Schema |         Name          | Type |  Owner
--------+-----------------------+------+----------
 sys    | alert                 | view | ybdadmin
 sys    | backup                | view | ybdadmin
 sys    | backup_chain          | view | ybdadmin
 sys    | backup_legacy         | view | ybdadmin
 sys    | backup_snapshot       | view | ybdadmin
 sys    | blade                 | view | ybdadmin
 sys    | column                | view | ybdadmin
 sys    | database              | view | ybdadmin
...
```

Alternatively, you can identify 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.

A log retention policy determines the amount 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

The retention policy is adjustable in the SMC. Go to **Configure** > **Configuration Settings** > **Retention**.

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 a number 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.

System View(s)	Regular Users Can See:
`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.

In This Section

- sys.alert
- sys.backup

- sys.backup_chain
- sys.backup_legacy
- sys.backup_snapshot
- sys.blade
- sys.column
- sys.const
- sys.database
- sys.drive_summary
- sys.key
- sys.lock
- sys.load
- sys.log_alert
- sys.log_analyze
- sys.log_audit
- sys.log_authentication
- sys.log_backup
- sys.log_backup_legacy
- sys.log_error
- sys.log_load
- sys.log_query
- sys.log_query_alert
- sys.log_query_analyze
- sys.log_query_explain

- sys.log_replica_status
- sys.log_restore
- sys.log_restore_legacy
- sys.log_session
- sys.log_unload
- sys.log_warning
- sys.mount
- sys.procedure
- sys.query
- sys.query_analyze
- sys.query_explain
- sys.query_rule_event
- sys.remote_server
- sys.replica
- sys.replica_status
- sys.restore
- sys.restore_legacy
- 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_rule
- sys.wlm_pending_pool
- sys.worker

Parent topic: Database Administration

sys.alert

This system view returns information about active alerts on the system. 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 the appliance, 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.

Column Name	Data Type	Description
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 |
```

Parent topic: System Views

sys.backup

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           | DzTlIEcb4E1EYW0BfHbo1mzMj9vuPskwWriWw3JowgAVscEKXVHO3y0gx5K
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           | DzTlIEcb4E1EYW0BfHbo1mzMj9vuPskwWriWw3JowgAVscEKXVHO3y0gx5K
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           | DzTlIEcb4E1EYW0BfHbo1mzMj9vuPskwWriWw3JowgAVscEKXVHO3y0gx5K
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           | DzTlIEcb4E1EYW0BfHbo1mzMj9vuPskwWriWw3JowgAVscEKXVHO3y0gx5K0
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           | DzTlIEcb4E1EYW0BfHbo1mzMj9vuPskwWriWw3JowgAVscEKXVHO3y0gx5K0
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
...
```

Parent topic: System Views

sys.backup_chain

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_poi
-------------+--------------+--------+------------------------+--------------------
```

```
      16391 | December2019 | {}     | Dec17Full              | [NULL]
(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
-------------+-----------------------+--------+-----------------------------------
      16432 | premdb_local_replica1 | {}     | premdb_local_replica1_20_01_14_00_50_
(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_bac
-------------+------------+------------------------------------------+-------------
      16396 | May2022    | {"excludedSchemas":["sys","matchstats"]} | May23FullExclu
(1 row)
```

Parent topic: System Views

sys.backup_legacy

This system view captures information about active backup (`ybbackup`) operations that are run with pre-Version 4.0 legacy tools.

Column Name	Data Type	Description
session_key	text	Unique session key generated for the backup operation.
state	text	Current state of the backup operation, such as `RUNNING` or `DONE` .
error_string	text	Error message reported for a failed backup. For example, if a backup is cancelled before it completes, the view may report an `External data socket error` .
object_id	bigint	Unique ID of the database that was backed up.
user_id	bigint	Unique ID of the database user.
session_id	bigint	Unique ID of the `ybbackup` session.
object_name	text	Name of the database that was backed up, as passed in on the `ybbackup` command line.
user_name	text	Name of the database user, as passed in on the `ybbackup` command line (or set with `YBUSER`).
client_hostname	text	Client host name, if available.
client_username	text	Operating system user on the client system who is running the backup tool.
start_time	timestamptz	When the `ybbackup` command started running.
last_activity_time	timestamptz	When `ybbackup` activity was last logged.
transaction_id	bigint	Unique transaction ID for the backup.

Column Name	Data Type	Description
backup_name	text	Backup name as specified with the `ybbackup --name` option.
directories	text	Directory location where the backup is stored.
sent_bytes	bigint	Number of bytes transferred over the network.
sent_bytes_per_second	double precision	Number of bytes per second transferred over the network.
nodata	boolean	Whether this backup was a "nodata" backup (metadata only).
groups_remaining	bigint	Number of backup "groups" remaining to be processed. When a backup starts, the data that needs to be backed up is batched into some number of groups. Although the data is logically organized into groups, a group is not equivalent to a table. The number of groups remaining drops as the backup proceeds.
total_groups	bigint	Total number of groups for this backup operation.

Examples

```
yellowbrick=# select * from sys.backup;
-[ RECORD 1 ]---------+-----------------------------------------------------------------
session_key           | Dmct_TYz45AEyl35aWYjXW7bs_x54zRv2_-bYTaRzpy04oHULShYFlB3WPv
state                 | DONE
error_string          | [NULL]
object_id             | 16456
user_id               | 16445
session_id            | 16413
object_name           | premdb
user_name             | yb100
client_hostname       | localhost
client_username       | brumsby
start_time            | 2018-12-10 17:56:41-08
last_activity_time    | 2018-12-10 17:56:43-08
transaction_id        | 6145
```

```
backup_name           | dec10
directories           | [/home/yb100/yb100_backups]
sent_bytes            | 5028997
sent_bytes_per_second | 2514498.5
nodata                | f
groups_remaining      | 0
total_groups          | 1
-[ RECORD 2 ]---------+-----------------------------------------------------------
session_key           | AUJCoeuc7kxJI-KkRJfwQ7gKIrYk5vD9q5Cmx7Epz3g_RxIh1IKEGOoZ8Et
state                 | DONE
error_string          | [NULL]
object_id             | 16384
user_id               | 16445
session_id            | 16413
object_name           | premdb
user_name             | yb100
client_hostname       | localhost
client_username       | brumsby
start_time            | 2018-12-10 17:32:17-08
last_activity_time    | 2018-12-10 17:32:24-08
transaction_id        | 3688
backup_name           | dec10
directories           | [/home/yb100/yb100_backups]
sent_bytes            | 112814
sent_bytes_per_second | 16116.2857142857
nodata                | f
groups_remaining      | 0
total_groups          | 5
```

The following queries track the progress of a single backup operation.

```
premdb=# select * from sys.backup;
(0 rows)

premdb=# \watch 3
(0 rows)

Watch every 3s  Wed Dec  2 16:31:32 2018

-[ RECORD 1 ]---------+-----------------------------------------------------------
session_key           | B2QQzmMmFBqMDnLhda1Ds14flpztGp38ib6uyTDYBxpSqFkNeUrIDRwJ5Ta
state                 | RUNNING
error_string          | [NULL]
```

```
object_id             | 16423
user_id               | 16434
session_id            | 16772
object_name           | premdb
user_name             | bobr
client_hostname       | localhost
client_username       | yb100
start_time            | 2018-12-02 16:31:29-07
last_activity_time    | 2018-12-02 16:31:31-07
transaction_id        | 46230
backup_name           | dec2018
directories           | [/home/ybdata/premdb_backups]
sent_bytes            | 0
sent_bytes_per_second | 0
nodata                | f
groups_remaining      | 9
total_groups          | 9
...
-[ RECORD 1 ]---------+-------------------------------------------------------------
session_key           | B2QQzmMmFBqMDnLhda1Ds14flpztGp38ib6uyTDYBxpSqFkNeUrIDRwJ5TaC
state                 | DONE
error_string          | [NULL]
object_id             | 16423
user_id               | 16434
session_id            | 16772
object_name           | premdb
user_name             | bobr
client_hostname       | localhost
client_username       | yb100
start_time            | 2018-12-02 16:31:29-07
last_activity_time    | 2018-12-02 16:31:38-07
transaction_id        | 46230
backup_name           | dec2018
directories           | [/home/ybdata/premdb_backups]
sent_bytes            | 154559519
sent_bytes_per_second | 17173279.8888889
nodata                | f
groups_remaining      | 0
total_groups          | 9
...
```

Parent topic: System Views

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
-------------+-------------+---------------+----------------------------------------
       16391 |        1260 | 3:3:          | ybd_af97315a-622b-ab81-ad4d-59ac80ec6fc5
       16391 |        1262 | 1262:1262:    | Dec17Full
(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
-------------+-------------+------------------+-----------------------------------
       16432 |        4119 | 0:3:3:           | ybd_0035c89e-ccec-9fe5-f1f4-59fe687a
       16432 |        4123 | 0:4123:4123:     | premdb_local_replica1_20_01_14_00_07
       16432 |        4542 | 0:4542:4542:     | premdb_local_replica1_20_01_14_00_08
       16432 |        4581 | 0:4581:4581:     | premdb_local_replica1_20_01_14_00_09
       16432 |        4624 | 0:4624:4624:     | premdb_local_replica1_20_01_14_00_10
       16432 |        4709 | 0:4709:4709:     | premdb_local_replica1_20_01_14_00_11
       16432 |        4749 | 0:4749:4749:     | premdb_local_replica1_20_01_14_00_12
       16432 |        4788 | 0:4788:4788:     | premdb_local_replica1_20_01_14_00_13
       16432 |        4827 | 0:4827:4827:     | premdb_local_replica1_20_01_14_00_14
       16432 |        4866 | 0:4866:4866:     | premdb_local_replica1_20_01_14_00_15
       16432 |        4926 | 0:4926:4926:     | premdb_local_replica1_20_01_14_00_16
       16432 |        5043 | 0:5043:5043:     | premdb_local_replica1_20_01_14_00_17
       16432 |        5222 | 0:5222:5222:     | premdb_local_replica1_20_01_14_00_18
...
```

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
-------------+-------------+------------------+-----------------------------------
       16395 |      137080 | 0:3:3:           | ybd_f89f1179-08cc-43f9-a1c2-1ed724d88
       16395 |      186750 | 0:186750:186750: | April1Full
       16395 |      186855 | 0:186855:186855: | premrep_20_04_01_13_20_42
       16395 |      189364 | 0:189364:189364: | premrep_20_04_01_13_22_12
       16395 |      189495 | 0:189495:189495: | April1Inc
       16395 |      189537 | 0:189537:189537: | premrep_20_04_01_13_23_42
       16395 |      189659 | 0:189659:189659: | premrep_20_04_01_13_25_12
       16395 |      190025 | 0:190025:190025: | premrep_20_04_01_13_26_42
       16395 |      190211 | 0:190211:190211: | April1Inc2
       16395 |      190300 | 0:190300:190300: | premrep_20_04_01_13_28_12
       16395 |      190351 | 0:190351:190351: | premrep_20_04_01_13_29_42
       16395 |      190400 | 0:190400:190400: | premrep_20_04_01_13_31_12
       16395 |      190528 | 0:190528:190528: | premrep_20_04_01_13_32_42
...
```

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
-------------+-------------+------------------+-----------------------------------
       16395 |      137080 | 0:3:3:           | ybd_f89f1179-08cc-43f9-a1c2-1ed724d88
       16395 |      191574 | 0:191574:191574: | April1Cumu
       16395 |      191621 | 0:191621:191621: | premrep_20_04_01_13_52_13
(3 rows)
```

The new backup, `April1Cumu` resets the "horizon" for backup and restore operations. Snapshots prior to `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
-------------+-------------+------------------+--------------------------------------
       16395 |      137080 | 0:3:3:           | ybd_f89f1179-08cc-43f9-a1c2-1ed724d88
       16395 |      191574 | 0:191574:191574: | April1Cumu
       16395 |      191621 | 0:191621:191621: | premrep_20_04_01_13_52_13
       16395 |      191761 | 0:191761:191761: | premrep_20_04_01_13_53_42
       16395 |      191810 | 0:191810:191810: | premrep_20_04_01_13_55_12
(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
       FULL                                 April1Full 2020-04-01T20:19:34.955359Z
 INCREMENTAL                                  April1Inc 2020-04-01T20:23:22.82759Z
 INCREMENTAL                                 April1Inc2 2020-04-01T20:26:54.542625Z
  CUMULATIVE                                 April1Cumu 2020-04-01T20:52:05.523348Z
```

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.

Parent topic: System Views

sys.blade

This system view returns status information about the compute blades that are used as worker nodes.

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 worker is starting up.

NETWORK_READY

The network link is ready.

WORKER_FAILED

Worker failed and is restarting.

WORKER_FAILED_REGISTRATION

The worker was unable to register with the cluster manager.

WORKER_ERROR

Worker error, reported from the Lime compiler.

WORKER_REGISTERED

The worker 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.

Parent topic: System Views

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	name	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	integer	`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 | colu
-------------+---------------+----------+-------------+------------+----------+----
       16459 | premdb        |    16007 | public      | match      |    16512 | seas
       16459 | premdb        |    16007 | public      | match      |    16512 | matc
       16459 | premdb        |    16007 | public      | match      |    16512 | htid
       16459 | premdb        |    16007 | public      | match      |    16512 | atid
       16459 | premdb        |    16007 | public      | match      |    16512 | ftsc
       16459 | premdb        |    16007 | public      | match      |    16512 | htsc
(6 rows)
```

Parent topic: System Views

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.

Parent topic: System Views

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. See HOT_STANDBY and READONLY Modes.
readonly_reason	text	Why the database is in `READONLY` mode. The database may have been set to `READONLY` explicitly by a database administrator. The system may put a database in this mode if the connection between the primary and secondary appliances is lost (for example, because of a network failure), or if the target database for replication is not in `HOT_STANDBY` mode.
is_hot_standby	boolean	Whether the database is in `HOT_STANDBY` mode. See HOT_STANDBY and READONLY Modes.

Column Name	Data Type	Description
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.
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.

Examples

Return information for a database named `newpremdb`:

```
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]
```

Return information about any readonly databases:

```
premdb=# select database_id, name, is_readonly, readonly_reason, rows_columnstore fr
 database_id |  name  | is_readonly |                                        readonly_r
-------------+--------+-------------+-------------------------------------------------
       31384 | repldb | t           | Replication topology validation failure. Target
(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, c
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, c
You are now connected to database "premdb" as user "yellowbrick".
premdb=# revoke connect, temp on database yellowbrick, yellowbrick_test, yellowbrick
REVOKE
premdb=# \c premdb noprivs
Password for user noprivs:
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, c
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`.

Parent topic: System Views

sys.drive_summary

This system view returns a summary of storage information for the drives on each worker node.

Column Name	Data Type	Description
worker_id	uuid	Worker 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 worker
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_u
-------------------------------------+------------+-------+---------------+-------
 00000000-0000-0000-0000-38b8ebd00c1c |          0 |     0 | 2048347078656 | 356606(
 00000000-0000-0000-0000-38b8ebd00c6c |          0 |     0 | 2048347078656 | 345254(
 00000000-0000-0000-0000-38b8ebd00bf9 |          0 |     0 | 2048347078656 | 339386(
 00000000-0000-0000-0000-38b8ebd007fd |          0 |     0 | 2048347078656 | 357673(
 00000000-0000-0000-0000-38b8ebd00c35 |          0 |     0 | 2048347078656 | 429966(
 00000000-0000-0000-0000-38b8ebd00c0d |          0 |     0 | 2048347078656 | 365672(
 00000000-0000-0000-0000-38b8ebd00c58 |          0 |     0 | 2048347078656 | 315751(
 00000000-0000-0000-0000-38b8ebd00c53 |          0 |     0 | 2048347078656 | 326126(
 00000000-0000-0000-0000-38b8ebd00ca8 |          0 |     0 | 2048347078656 | 328330(
 00000000-0000-0000-0000-38b8ebd00c30 |          0 |     0 | 2048347078656 | 327034(
 00000000-0000-0000-0000-38b8ebd00c2b |          0 |     0 | 2048347078656 | 359581(
 00000000-0000-0000-0000-38b8ebd00c2b |          0 |     1 | 2048347078656 | 356343(
 00000000-0000-0000-0000-38b8ebd00c6c |          0 |     1 | 2048347078656 | 344658(
 00000000-0000-0000-0000-38b8ebd00c1c |          0 |     1 | 2048347078656 | 363593(
 00000000-0000-0000-0000-38b8ebd00c58 |          0 |     1 | 2048347078656 | 317699(
```

```
  00000000-0000-0000-0000-38b8ebd00c53 |          0 |     1 | 2048347078656 | 327791
  00000000-0000-0000-0000-38b8ebd007fd |          0 |     1 | 2048347078656 | 360108
  00000000-0000-0000-0000-38b8ebd00bf9 |          0 |     1 | 2048347078656 | 342580
  00000000-0000-0000-0000-38b8ebd00c35 |          0 |     1 | 2048347078656 | 427101
  00000000-0000-0000-0000-38b8ebd00c30 |          0 |     1 | 2048347078656 | 331408
  00000000-0000-0000-0000-38b8ebd00c0d |          0 |     1 | 2048347078656 | 362717
  00000000-0000-0000-0000-38b8ebd00ca8 |          0 |     1 | 2048347078656 | 330909
...
```

Parent topic: System Views

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)
```

Parent topic: System Views

sys.lock

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	Always `TABLE`.
session_id	integer	Unique database session ID.
is_granted	boolean	Whether the requested lock was granted.
lock_type	text	Type of lock requested.
blocked_by_session_id	integer	Session ID that is blocking the requested lock (`NULL` if not blocked).

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
-----------+-------------+-------------+------------+------------+------------------
     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           | ShareUpdateExclusi
      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    |
----------+-------------+-------------+------------+------------+-----------------+
  1812902 |       17106 | TABLE       |      28200 | f          | AccessShareLock |
   17137 |       17106 | TABLE       |      28770 | f          | ExclusiveLock   |
   17137 |       17106 | TABLE       |      28770 | f          | ExclusiveLock   |
(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_
 table_id |                  name                   |    lock_type
----------+-----------------------------------------+-----------------
   56889 | ThirdPartyDataGeneralPopulationOverlap | AccessShareLock
   17395 | TargetingDataUniqueThirdPartyUsers     | AccessShareLock
   17395 | TargetingDataUniqueThirdPartyUsers     | ExclusiveLock
   17386 | TargetingData                          | AccessShareLock
   56865 | LookAlikeModelResult                   | AccessShareLock
   56865 | LookAlikeModelResult                   | ExclusiveLock
...
```

Parent topic: System Views

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-SYTi
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
user_name                | 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-SYTi
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
user_name                 | 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

...
```

Parent topic: System Views

sys.log_alert

This system view returns information about alerts that have been recorded on the system. 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 ]-----+----------------------------------------
...
```

Parent topic: System Views

sys.log_analyze

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_t
----------+-------------+---------+-------------------------------+----------------
   16884 |       16874 |   16006 | 2017-06-19 14:17:09.448803-07 | 2017-06-19 14:17
   16881 |       16874 |   16006 | 2017-06-19 14:17:11.431026-07 | 2017-06-19 14:17
   16893 |       16874 |   16006 | 2017-06-19 15:37:15.650933-07 | 2017-06-19 15:37
   16887 |       16874 |      10 | 2017-06-19 17:54:36.917482-07 | 2017-06-19 17:54
   16887 |       16874 |      10 | 2017-06-20 12:49:15.774682-07 | 2017-06-20 12:49
(5 rows)
```

Note: The `yellowbrick` user and internal system users have five-digit user IDs that begin with `16`.

Parent topic: System Views

sys.log_audit

This view records audit events that occur on the system 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 | sever
-------------------------------------+----------------------------+----------+----
 001d42ea-b97a-43cb-8f3a-e831d2790ad5 | 2017-06-19 14:12:07.406-07 |          |
 f1b61530-dc7d-4eaa-be2f-94904b2d5764 | 2017-06-19 14:12:14.134-07 |          |
 fd4e0ace-b020-4615-941e-74a58c40fa85 | 2017-06-19 14:12:14.142-07 |          |
 966a4ca0-06ff-4605-afd2-65a4fee15877 | 2017-06-19 14:12:17.191-07 |          |
 2ada1d60-8162-4def-9095-d2309c9fd79c | 2017-06-19 14:12:17.202-07 |          |
 798a36a8-78e0-4e70-97dd-e2bc3bcc671d | 2017-06-19 14:12:17.204-07 |          |
 55429d42-85e9-4962-90ec-7a43c9f53d11 | 2017-06-19 14:12:17.221-07 |          |
 b733219c-791e-4206-9458-f81771919ab0 | 2017-06-19 14:12:17.229-07 |          |
 c75ba733-7d32-4ade-b0e5-546e3552cdda | 2017-06-19 14:12:17.277-07 |          |
 8fcd9daf-c902-48ec-82d0-26057181a27e | 2017-06-19 14:12:17.315-07 |          |
```

```
 b1b70644-7434-48ff-9710-2cb7aa107008 | 2017-06-19 14:12:17.465-07 |          |
(11 rows)
```

Parent topic: System Views

sys.log_authentication

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_
---------------------------+-------------+---------------+----------------+--------
 2018-07-20 11:26:52.798-07 | brumsby     | yellowbrick   | Authentication | [local
 2018-07-20 11:26:59.22-07  | brumsby     | premdb        | Authentication | [local
 2018-07-20 18:43:31.995-07 | brumsby     | premdb        | Authentication | [local
 2018-07-20 18:44:37.141-07 | bobr        | premdb        | Authentication | [local
 2018-07-20 18:46:12.87-07  | brumsby     | premdb        | Authentication | [local
 2018-07-23 18:10:19.87-07  | brumsby     | premdb        | Authentication | [local
 2018-07-23 18:24:56.439-07 | yellowbrick | yellowbrick   | Authentication | 127.0.
```

```
 2018-07-24 11:33:45.312-07 | brumsby     | premdb        | Authentication | [local]
 2018-07-24 18:16:17.993-07 | bobr        | premdb        | Authentication | [local]
(9 rows)
```

Return failed authentication records:

```
premdb=# select * from sys.log_authentication where status !=0;
        start_time         | user_name | database_name |     event      | client_ip
---------------------------+-----------+---------------+----------------+---------
 2018-07-24 18:16:17.993-07 | bobr      | premdb        | Authentication | [local]
(1 row)
```

Parent topic: System Views

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 cancelled 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).

Column Name	Data Type	Description
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_
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_lrOHWLu
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
```

Parent topic: System Views

sys.log_backup_legacy

This view records the history of backup (`ybbackup`) operations that are run with pre-Version 4.0 legacy tools. A unique row is defined by the transaction ID assigned to the backup operation.

Column Name	Data Type	Description
session_key	text	Unique session key generated for the backup operation.
state	text	Current state of the backup operation, such as `RUNNING` or `DONE` .
error_string	text	Error message reported for a failed backup. For example, if a backup is cancelled before it completes, the view may report an `External data socket error` .
object_id	bigint	Unique ID of the database that was backed up.
user_id	bigint	Unique ID of the database user.
session_id	bigint	Unique ID of the `ybbackup` session.
object_name	text	Name of the database that was backed up, as passed in on the `ybbackup` command line.
user_name	text	Name of the database user, as passed in on the `ybbackup` command line (or set with `YBUSER`).
client_hostname	text	Client host name, if available.
client_username	text	Operating system user on the client system who is running the backup tool.
start_time	timestamptz	When the `ybbackup` command started running.
end_time	timestamptz	When the `ybbackup` command completed.
transaction_id	bigint	Unique transaction ID for the backup.

Column Name	Data Type	Description
backup_name	text	Backup name as specified with the `ybbackup --name` option.
directories	text	Directory location where the backup is stored.
sent_bytes	bigint	Number of bytes transferred over the network.
sent_bytes_per_second	double precision	Number of bytes per second transferred over the network.
nodata	boolean	Whether this backup was a "nodata" backup (metadata only).

Example

```
premdb=# select * from sys.log_backup;

-[ RECORD 1 ]---------+------------------------------------------------------------
session_key           | B2QQzmMmFBqMDnLhda1Ds14flpztGp38ib6uyTDYBxpSqFkNeUrIDRwJ5TaC
state                 | DONE
error_string          | [NULL]
object_id             | 16423
user_id               | 16434
session_id            | 16772
object_name           | premdb
user_name             | bobr
client_hostname       | localhost
client_username       | yb100
start_time            | 2018-12-02 16:31:29.795-07
end_time              | 2018-12-02 16:31:38.569-07
transaction_id        | 46230
backup_name           | dec2018
directories           | [/home/brumsby/premdb_backups]
sent_bytes            | 154559519
sent_bytes_per_second | 19121553
nodata                | f
...
```

Parent topic: System Views

sys.log_error

This view captures system-level errors that are logged during database activities. 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 | ackn
-------------------------------------+------------------------------+----------+----
 0194213f-f847-46ec-b97a-cce8b392ea98 | 2017-06-13 17:08:33.58-07    |          | f
 3571ce6e-dbe0-4aac-988e-a5e4e63d6939 | 2017-06-14 19:58:58.972-07 |          | f
 fd64a358-4e6a-4efd-9946-d3ed26247717 | 2017-06-14 20:13:38.975-07 |          | f
(3 rows)
```

Parent topic: System Views

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.

Column Name	Data Type	Description
user_name	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.
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.

Column Name	Data Type	Description
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.

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-Y9IE8k30Fa
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
user_name                 | 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 duratio
from sys.log_load where table_name='newmatchstats' and session_id=18913;
 table_id |         start_time         |          end_time          | duration | ins
----------+----------------------------+----------------------------+----------+---
   16496 | 2018-03-26 16:42:09.233-07 | 2018-03-26 16:42:16.012-07 |     6.77 |
(1 row)
```

Parent topic: System Views

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.

Column Name	Data Type	Description
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.

Column Name	Data Type	Description
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.
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.

Column Name	Data Type	Description
num_workers	integer	Total number of workers 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-worker queries after the `assemble` phase. For multi-worker queries, the actual worker 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 worker that is currently taking the longest time to finish executing this query (or the worker that was the last to finish when the query was done). `NULL` until the query starts executing. In a distributed system, workers may run for different durations. This column helps identify which of the workers matter from a tuning perspective. The "longest-running worker" is the worker that finished last when the query was completed and moved to `sys.log_query`. While the query is running, the longest-running worker is either the worker with the highest `io_spill_space_bytes` (if spilling is occurring) or the worker 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.

Column Name	Data Type	Description
cpu_percent	integer	How effectively the CPU was used on the worker that dominated runtime (the longest worker). `(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-worker queries, this value should be equal to `cpu_percent` on that worker.
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.

Column Name	Data Type	Description
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.
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.

Column Name	Data Type	Description
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 worker slots. `NULL` before entering the `acquire_resources` state, then initialized to 0.

Column Name	Data Type	Description

run_ms	decimal(18,3)	How long the query was running on the *longest-running* worker 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 worker. `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 worker. `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 worker 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.

Column Name	Data Type	Description
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 workers). This value does not include temporary storage reads.
io_write_bytes	bigint	Number of bytes written by the query (aggregated across all workers). This value does not include temporary storage writes.

Column Name	Data Type	Description
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 worker nodes without spilling. For queries that did not reach the worker 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 worker nodes without spilling. For queries that did not reach the worker nodes, this column returns `NULL`.
io_network_bytes	bigint	Number of bytes moved internally over the network on the cluster (manager node to workers 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`.

Column Name	Data Type	Description
rows_returned	bigint	Number of rows the query returned to the client.
memory_bytes	bigint	Amount of memory consumed by the longest-running worker that ran this query.
memory_bytes_max	bigint	Maximum amount of memory used by any worker that ran this query.
io_spill_space_bytes	bigint	Amount of temporary space consumed by the longest-running worker 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 worker 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 worker* from the query plan. `NULL` until the query exits the `compile` state.
memory_required_bytes	bigint	The minimum amount of memory required *per worker* to start the query. `NULL` until the query exits the `compile` state.
memory_granted_bytes	bigint	Amount of memory *per worker* 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.

Column Name	Data Type	Description
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_returr
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
-----------+------------------------------------+---------------+------------------
 ybtools10 | DBeaver 22.0.3 - Main <premdb>     | premdb        | bobrum-beta1-clust
 ybtools10 | DBeaver 22.0.3 - Metadata <premdb> | premdb        | bobrum-beta1-clust
 ybtools10 | DBeaver 22.0.3 - Metadata <premdb> | premdb        | bobrum-beta1-clust
 ybtools10 | ybsql                              | premdb        | bobrum-beta1-clust
 ybtools10 | ybsql                              | premdb        | bobrum-beta1-clust
 ybtools10 | ybsql                              | premdb        | bobrum-beta1-clust
 ybtools10 | ybsql                              | premdb        | bobrum-beta1-clust
 ybtools10 | ybsql                              | premdb        | bobrum-beta1-clust
(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, homegoa
```

```
   247096 | done  | bobr     |  115.999 | select season_name, numteams, sum(substr(t
(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
from sys.log_query
where type='update' and username='bobr'
order by 1 desc;
 query_id | username | application_name |  type  | state |          qrytxt
----------+----------+------------------+--------+-------+--------------------------
  1143640 | bobr     | ybsql            | update | done  | update newmatchstats set
  1141949 | bobr     | ybsql            | update | done  | update newmatchstats set
  1130126 | bobr     | ybsql            | update | done  | update newmatchstats set
  1095945 | bobr     | ybsql            | update | done  | update newmatchstats set
(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 sy
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
                     | 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
                     | 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
                    | 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
 query_id |  type  |        submit_time         |         done_time          | total
----------+--------+----------------------------+----------------------------+------
   258345 | insert | 2020-09-09 17:59:01.871-07 | 2020-09-09 17:59:07.801-07 | 5930
   116626 | select | 2020-09-09 16:28:15.35-07  | 2020-09-09 16:28:18.41-07  | 3059
   258043 | insert | 2020-09-09 17:58:57.191-07 | 2020-09-09 17:59:00.22-07  | 3029
   140372 | select | 2020-09-09 16:47:55.137-07 | 2020-09-09 16:47:58.087-07 | 2950
   130453 | select | 2020-09-09 16:39:08.884-07 | 2020-09-09 16:39:11.778-07 | 2894
   234651 | select | 2020-09-09 17:48:01.518-07 | 2020-09-09 17:48:04.087-07 | 2568
   282685 | select | 2020-09-09 18:11:07.228-07 | 2020-09-09 18:11:09.47-07  | 2241
   276565 | select | 2020-09-09 18:08:03.255-07 | 2020-09-09 18:08:05.291-07 | 2036
   265009 | select | 2020-09-09 18:02:18.264-07 | 2020-09-09 18:02:20.3-07   | 2035
   150281 | select | 2020-09-09 16:52:50.278-07 | 2020-09-09 16:52:52.312-07 | 2033
(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_res
 query_id | transaction_id |  state  |      pool_id      | num_restart |        subm
----------+----------------+---------+-------------------+-------------+-----------
  2573711 |     4294974926 | done    | bobr_pool         |           1 | 2020-09-18
  2573711 |     4294974926 | restart | bobr_profile: mix |           0 | 2020-09-18
(2 rows)
```

Parent topic: System Views

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	Whether the user has acknowledged the alert on the SMC Manage Query Alerts screen, where this data is also visible.
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         | ackno
-------------------------------------+----------+----------------------------+----
 e44d60f0-5cab-48da-b0d0-77e7262cf809 |    49040 | 2018-09-11 14:55:04.421-07 | f
 17e4a5d0-c985-4627-8397-0b0691e3a6f4 |    48998 | 2018-09-11 14:55:04.356-07 | f
 6830c650-a5a8-4c33-af7e-705fbdd6ee0e |    52054 | 2018-09-11 14:57:20.431-07 | f
 02fb578d-bb94-4324-9af1-5e58c11c61c2 |    49029 | 2018-09-11 14:55:04.394-07 | f
(4 rows)
```

Parent topic: System Views

sys.log_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 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_analyz
 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)
```

Parent topic: System Views

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	text	Workers processing the plan node: `single` or `all` for each node of the query plan
query_plan	text	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 |                       subst
---------------------------------------------+---------+--------------------------
 0XeN2C-FjU6lxnkPh9ry4jOuU9Ad+s5O0dNO+T7uOdA= |       0 | SELECT
                                             |         | (match.matchday, DATEADD(m
 0XeN2C-FjU6lxnkPh9ry4jOuU9Ad+s5O0dNO+T7uOdA= |       1 | APPEND SCAN
                                             |         | (match.matchday, DATEADD(m
 0XeN2C-FjU6lxnkPh9ry4jOuU9Ad+s5O0dNO+T7uOdA= |       2 | | SCAN match
```

```
                                                |          |   (match.matchday, match.at
 0XeN2C-FjU6lxnkPh9ry4jOuU9Ad+s5O0dNO+T7uOdA= |        3 | |-TRANSPOSE
                                                |          |   (match.matchday, DATEADD(
 0XeN2C-FjU6lxnkPh9ry4jOuU9Ad+s5O0dNO+T7uOdA= |        4 | |-DISTRIBUTE RANDOM
                                                |          |   (match.matchday, DATEADD(
 0XeN2C-FjU6lxnkPh9ry4jOuU9Ad+s5O0dNO+T7uOdA= |        5 |   SCAN ROW STORE
                                                |          |   (match.matchday, match.ht
 0XeN2C-FjU6lxnkPh9ry4jOuU9Ad+s5O0dNO+T7uOdA= |        6 |   FILTER WHERE ((match.hti
 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
where slq.plan_id=slqe.plan_id and slq.query_text like 'select season_name, numteams
                  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
...
```

Parent topic: System Views

sys.log_replica_status

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_byte
-[ 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-2YVbj
restore_session_key | AxtTYw1nAb0qqRFi93rSEA0PWU9K4O-d2RRGm2fNkZg34qyHwHZ5s23wevDI5i
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-3Y3dtBRsvqNsD1hzTNRdWjJo61VxDTzLl
restore_session_key | BAZOiGXIj8NtxayHGrUPtOZSBxsuKg0ykj7jtsTdCuQKUb-Yd7_tPH5gfm2a0r
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  | AeL5WMy1W36EBiv0ea0BKhYxW0sV2Xlmrvzg0ucLUIKcg2JxlsU4zFnlBfIBW
restore_session_key | CWW8BbTi2bi9KWq-GdupA9fYok_B1anAGy5fuA8OcsDlPD_GHN01FRftqet0_
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_
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)
```

Parent topic: System Views

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 cancelled 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.

Column Name	Data Type	Description
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).
source_appliance_name	text	Name of the appliance where the database was backed up.

Example

```
premdb=# select * from sys.log_restore;
-[ RECORD 1 ]----------+-----------------------------------------------------------
session_key            | BSz3VTMYTqbeX6AR-QspATzuMelRcjI_WlkdaIL9EOjpm49witdmc20HCn
session_id             | 40984
client_hostname        | localhost
client_username        | yb100
user_name              | yb100
user_id                | 16417
database_name          | premdb_restored
database_id            | 16428
state                  | DONE
error_string           | [NULL]
start_time             | 2020-01-13 18:19:49.381-08
end_time               | 2020-01-13 18:20:07.967-08
elapsed_ms             | 18586
remaining_ms           | 0
received_bytes         | 8121140
```

```
received_bytes_per_sec | 436949.316689982
storage_bytes          | 8111423
storage_bytes_per_sec  | 436426.50382008
backup_point_id        | Jan13Full
chain_id               | January2020
source_database_name   | premdb
source_appliance_name  | localhost
```

Parent topic: System Views

sys.log_restore_legacy

This view records the history of restore (`ybrestore`) operations that are run with pre-Version 4.0 legacy tools. A unique row is defined by the transaction ID assigned to the restore operation.

See sys.log_backup for column descriptions.

Example

```
premdb=# select * from sys.log_restore;
-[ RECORD 1 ]---------+-----------------------------------------------------------
session_key           | CkXj39RYRVFhTNnHWzfzWxyG8VdTTXm5jIXDcgAzxZKjgJXAdnYYyGYaoJmp
state                 | DONE
error_string          | [NULL]
object_id             | 16437
user_id               | 16434
session_id            | 16703
object_name           | restored_premdb
user_name             | bobr
client_hostname       | localhost
client_username       | yb100
start_time            | 2018-12-02 14:33:59.352-07
end_time              | 2018-12-02 14:34:00.118-07
transaction_id        | 42153
restore_name          | dec02
directories           | [/home/ybdata/backups]
sent_bytes            | 0
sent_bytes_per_second | 0
nodata                | f
```

Parent topic: System Views

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`, `ybload`, `ybunload`, or `yb-smc`. 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.

Column Name	Data Type	Description
secure_details	varchar(256)	For secure connections, the 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_nar
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 |    durati
-----------------+-------------+---------+------------+------------------+--------
 yellowbrick_test |       16588 |   16590 |      17232 | ybsql            | 00:00:11
 yellowbrick_test |       16588 |   16590 |      18279 | ybsql            | 00:00:19
 yellowbrick_test |       16588 |   16590 |      19941 | ybsql            | 00:00:30
 yellowbrick_test |       16588 |   16590 |      23253 | ybsql            | 00:18:08
 premdb           |       26259 |   26261 |      23285 | ybsql            | 00:00:35
 premdb           |       26259 |   26261 |      23445 | ybsql            | 00:00:54
(6 rows)
```

Parent topic: System Views

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.
user_name	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.
elapsed_ms	bigint	Duration of the unload, in milliseconds.
sql_query	text	The SELECT statement that was run to unload data.

Column Name	Data Type	Description
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           | CDil7ZNCMkZDvlgSPiHZn7qR4Wv7KLkXM3DLpIIMYXd6arJT0L0OFBDKPBN
state                 | RUNNING
error_string          | [NULL]
database_id           | 16423
user_id               | 16434
session_id            | 16792
database_name         | premdb
user_name             | 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           | CDil7ZNCMkZDvlgSPiHZn7qR4Wv7KLkXM3DLpIIMYXd6arJT0L0OFBDKPBNS
state                 | COMPLETE
error_string          | [NULL]
database_id           | 16423
user_id               | 16434
session_id            | 16792
database_name         | premdb
user_name             | 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           | CDil7ZNCMkZDvlgSPiHZn7qR4Wv7KLkXM3DLpIIMYXd6arJT0L0OFBDKPBNS
state                 | DONE
error_string          | [NULL]
database_id           | 16423
user_id               | 16434
session_id            | 16792
database_name         | premdb
user_name             | 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

...
```

Parent topic: System Views

sys.log_warning

This view logs warning messages returned by the system. 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 d
-[ 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 d
-[ 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 d
-[ 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 d
-[ 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 d
```

Parent topic: System Views

sys.mount

This system view returns a list of NFS mount points that have been created for storing 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)
```

Parent topic: System Views

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.
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 varyin
       19121 |       16395 |      2200 | insert_hometeam | insert_hometeam(integer,
(2 rows)
```

Parent topic: System Views

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_tex
from sys.query where type='update' and username='bobr'
order by 1 desc;
...
                                                          Watch every 1s    Thu

 query_id | username | application_name |  type  |  state  |          qrytxt
----------+----------+------------------+--------+---------+-------------------------
  1141949 | bobr     | ybsql            | update | compile | update newmatchstats se
(1 row)

                                                          Watch every 1s    Thu

 query_id | username | application_name |  type  | state |          qrytxt
----------+----------+------------------+--------+-------+-------------------------
  1141949 | bobr     | ybsql            | update | run   | update newmatchstats set
(1 row)

                                                          Watch every 1s    Thu

 query_id | username | application_name |  type  | state |          qrytxt
----------+----------+------------------+--------+-------+-------------------------
  1141949 | bobr     | ybsql            | update | run   | update newmatchstats set
```

```
(1 row)
...
```

Parent topic: System Views

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, worker ID, and node ID. *All of the reported statistics are per plan node per worker.*

Column Name	Data Type	Description
query_id	bigint	Query execution ID
worker_id	text	Worker 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 worker nodes

Column Name	Data Type	Description
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
from sys.query_analyze order by worker_id, query_id, node_id;
 query_id |              worker_id               | node_id | memory_bytes | io_read_
----------+--------------------------------------+---------+--------------+---------
   25832 | 00000000-0000-0000-0000-000000000002 |       0 |      3145936 |
   25832 | 00000000-0000-0000-0000-000000000002 |       1 |     17301376 |
   25832 | 00000000-0000-0000-0000-000000000002 |       2 |       262144 |
   25832 | 00000000-0000-0000-0000-000000000002 |       3 |       262144 |
   25832 | 00000000-0000-0000-0000-000000000002 |       4 |        61440 |        13
   25832 | 00000000-0000-0000-0000-000000000002 |       5 |      1048576 |
   25832 | 00000000-0000-0000-0000-000000000002 |       7 |      1310864 |
   25832 | 00000000-0000-0000-0000-000000000002 |       8 |      1310720 |
   25832 | 00000000-0000-0000-0000-000000000002 |       9 |      2097152 |
   25832 | 00000000-0000-0000-0000-000000000002 |      10 |       524288 |
   25832 | 00000000-0000-0000-0000-000000000002 |      11 |       262144 |
   25832 | 00000000-0000-0000-0000-000000000002 |      12 |      1078272 |
   25832 | 00000000-0000-0000-0000-000000000002 |      13 |      1048576 |
   25832 | 00000000-0000-0000-0000-000000000002 |      15 |      1310828 |
   25832 | 00000000-0000-0000-0000-000000000002 |      16 |      1310720 |
...
```

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 worker 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
                     | 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
                     | 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
                     | 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
                     | distribute (newmatchstats.seasonid)
...
```

Parent topic: System Views

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)
...
                                                                                W

                 plan_id                    | node_id | index |      type       | v
--------------------------------------------+---------+-------+-----------------+
 v2i-xVAwlrav2rBiQmxF50CLB0fWdHyuoBvhOj7WMnU= |       0 |     0 | WRITE ROW STORE |
                                            |         |       |                 |
                                            |         |       |                 |
 v2i-xVAwlrav2rBiQmxF50CLB0fWdHyuoBvhOj7WMnU= |       1 |     1 | INSERT INTO     |
                                            |         |       |                 |
 v2i-xVAwlrav2rBiQmxF50CLB0fWdHyuoBvhOj7WMnU= |       3 |     2 | DISTRIBUTE HASH |
                                            |         |       |                 |
                                            |         |       |                 |
 v2i-xVAwlrav2rBiQmxF50CLB0fWdHyuoBvhOj7WMnU= |       2 |     3 | APPEND SCAN     |
                                            |         |       |                 |
 v2i-xVAwlrav2rBiQmxF50CLB0fWdHyuoBvhOj7WMnU= |       5 |     4 | TRANSPOSE       |
                                            |         |       |                 |
 v2i-xVAwlrav2rBiQmxF50CLB0fWdHyuoBvhOj7WMnU= |       4 |     5 | SCAN            |
                                            |         |       |                 |
                                            |         |       |                 |
 v2i-xVAwlrav2rBiQmxF50CLB0fWdHyuoBvhOj7WMnU= |       7 |     6 | SCAN ROW STORE  |
```

```
                                        |        |       |                |
                                        |        |       |                |
(7 rows)
...
```

Parent topic: System Views

sys.query_rule_event

This view captures information about WLM rule processing (as also displayed via the SMC). 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`
event	text	Rule processing events include `Rule processing starting`, `Rule processing complete`, `Setting...`, `Move to resource pool...`, `Rule is disabled`, and so on.

Examples

```
premdb=> select * from sys.query_rule_event;
 query_id |      rule_name      |         event_time         | event_type |
----------+---------------------+----------------------------+------------+--------
   30108 | rule                | 2017-04-11 15:17:15.584-07 | end        | Rule proc
   30108 | sys_defaultRowLimit | 2017-04-11 15:17:15.582-07 | set        | Setting M
   30108 | fastlane            | 2017-04-11 15:17:15.581-07 | set        | Setting F
   30108 | rule                | 2017-04-11 15:17:15.578-07 | begin      | Rule proc
...
```

```
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 |
----------+------------------------+---------------------+------------+------------
   33658 |  select * from match;  | rule                | end        | Rule processi
   33658 |  select * from match;  | sys_defaultRowLimit | set        | Setting Maxi
   33658 |  select * from match;  | fastlane            | set        | Setting Resou
   33658 |  select * from match;  | rule                | begin      | Rule processi
(4 rows)
```

The following example filters events in which a query was moved to another pool:

```
yellowbrick=# select * from sys.query_rule_event where event like 'Move%';;
 query_id |    rule_name    |         event_time         | event_type |           ev
----------+-----------------+----------------------------+------------+------------
 13450004 | system          | 2020-09-24 14:39:59.967-07 | move       | Move to reso
 13245785 | flex_moveToLong | 2020-09-24 12:35:18.543-07 | move       | Move to reso
(2 rows)
```

The following example finds events for a specific completion rule when it was not disabled or ignored:

```
yellowbrick=# select * from sys.query_rule_event where rule_name='completion_error'
 query_id |    rule_name     |         event_time         | event_type |
----------+------------------+----------------------------+------------+-----------
 13584534 | completion_error | 2020-09-24 15:59:05.644-07 | error      | Do not use
 13583194 | completion_error | 2020-09-24 15:58:50.191-07 | error      | Do not use
 13580925 | completion_error | 2020-09-24 15:57:36.721-07 | error      | Do not use
 13579296 | completion_error | 2020-09-24 15:57:13.002-07 | info       | Rule change
 13578087 | completion_error | 2020-09-24 15:57:08.935-07 | error      | Do not use
 13577467 | completion_error | 2020-09-24 15:56:27.999-07 | info       | Rule change
 13576070 | completion_error | 2020-09-24 15:56:13.623-07 | error      | Do not use
 13576029 | completion_error | 2020-09-24 15:56:07.99-07  | info       | Rule change
...
```

Parent topic: System Views

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	oid	Unique ID for the remote server.
name	text	User-defined name for the remote server object.
host	text	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 Manager Node Ports and Opening Network Ports for Clients.

For example:

```
premdb=# select * from sys.remote_server;
 remote_server_id |       name        |          host          | port | control_port
------------------+-------------------+------------------------+------+--------------
            16417 | yb100             | yb100.yellowbrick.io | 5432 |       [NULL] |
            16418 | br_yb_io          | 10.30.22.203         | 5432 |       [NULL] |
            16419 | yblocal_checkhost | localhost            | 5432 |       [NULL] |
(3 rows)
```

Parent topic: System Views

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.

Column Name	Data Type	Description
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         | t
reverse_replica                | [NULL]
```

```
yellowbrick=# select sr.replica_id, sr.name replica_name, sr.database_id, sd.name db
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)
```

Parent topic: System Views

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-tAr5Dav7M0QpUx5k2H25yEd2jMmWDo
restore_session_key | CmeN0iX2HD-t--YfAm9huHsjkraIYlpw9dZeZPT9-xFFSqwzswEsd8ZPlNXMA
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-HV
restore_session_key | DC4O8zCY6sFB1SSQILTUvZOfGMZFiOgj_XxbA6__gi9SM4ixM7xGViFp26XoxC
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-HS
restore_session_key | DH6janEMJMwUgCn3qtlS6030IGO_DdTefLrHin9BdbZHVdQSYL5aTF9Ki5U9DI
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  | AeL5WMy1W36EBiv0ea0BKhYxW0sV2Xlmrvzg0ucLUIKcg2JxlsU4zFnlBfIBWc
restore_session_key | CWW8BbTi2bi9KWq-GdupA9fYok_B1anAGy5fuA8OcsDlPD_GHN01FRftqet0_z
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-2YVbj
restore_session_key | AxtTYw1nAb0qqRFi93rSEA0PWU9K4O-d2RRGm2fNkZg34qyHwHZ5s23wevDI5i
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
----------------------+-------------+----------+-----------------------------+----
 premdb               | os_user     | LATIN9   |                             | f
 premdb_replica1_db   | yellowbrick | LATIN9   |                             | f
...
```

Parent topic: System Views

sys.restore

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 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 Feb 18 15:15:17 2020

-[ RECORD 1 ]----------+-----------------------------------------------------------
session_key            | TED9yW3bVpOtR-syMZcm_3K5smzR2mTqqbE16MOCP8xRSi_9EVFNPjottbz
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             | 2020-02-18 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        | Feb18Full
```

```
chain_id               | February2020
source_database_name   | premdb
source_appliance_name  | localhost

Watch every 5s  Tue Feb 18 15:15:22 2020

-[ RECORD 1 ]----------+-------------------------------------------------------------
session_key            | TED9yW3bVpOtR-syMZcm_3K5smzR2mTqqbE16MOCP8xRSi_9EVFNPjottbz
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             | 2020-02-18 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        | Feb18Full
chain_id               | February2020
source_database_name   | premdb
source_appliance_name  | localhost

Watch every 5s  Tue Feb 18 15:15:27 2020

-[ RECORD 1 ]----------+-------------------------------------------------------------
session_key            | TED9yW3bVpOtR-syMZcm_3K5smzR2mTqqbE16MOCP8xRSi_9EVFNPjottbz
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             | 2020-02-18 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        | Feb18Full
chain_id               | February2020
source_database_name   | premdb
source_appliance_name  | localhost

Watch every 5s  Tue Feb 18 15:15:32 2020
...
Watch every 5s  Tue Feb 18 15:15:58 2020

-[ RECORD 1 ]----------+-----------------------------------------------------------
session_key            | TED9yW3bVpOtR-syMZcm_3K5smzR2mTqqbE16MOCP8xRSi_9EVFNPjottbz
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             | 2020-02-18 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        | Feb18Full
chain_id               | February2020
source_database_name   | premdb
source_appliance_name  | localhost

Watch every 5s  Tue Feb 18 15:16:03 2020

-[ RECORD 1 ]----------+-----------------------------------------------------------
session_key            | TED9yW3bVpOtR-syMZcm_3K5smzR2mTqqbE16MOCP8xRSi_9EVFNPjottbz
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              | 2020-02-18 15:15:14-08
end_time                | 2020-02-18 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         | Feb18Full
chain_id                | February2020
source_database_name    | premdb
source_appliance_name   | localhost

Watch every 5s  Tue Feb 18 15:16:08 2020
```

Parent topic: System Views

sys.restore_legacy

This system view captures information about active restore (`ybrestore`) operations that are run with pre-Version 4.0 legacy tools.

Column Name	Data Type	Description
session_key	text	Unique session key generated for the restore operation.
state	text	Current state of the restore operation, such as `RUNNING` or `DONE` .
error_string	text	Error message reported for a failed restore.
object_id	bigint	Unique ID of the database that was restored.
user_id	bigint	Unique ID of the database user.
session_id	bigint	Unique ID of the `ybrestore` session.
object_name	text	Name of the restored database (this may be an alias that was specified in the `ybrestore` command).
user_name	text	Name of the database user, as passed in on the `ybrestore` command line (or set with `YBUSER`).
client_hostname	text	Client host name, if available.
client_username	text	Operating system user on the client system who is running the restore tool.
start_time	timestamptz	When the `ybrestore` command started running.
last_activity_time	timestamptz	When `ybrestore` activity was last logged.
transaction_id	bigint	Unique transaction ID for the restore.
backup_name	text	Name of the restored backup (as specified with `--name` in the `ybbackup` command).

Column Name	Data Type	Description
directories	text	Directory location where the backup is stored for this restore operation.
sent_bytes	bigint	Number of bytes transferred over the network.
sent_bytes_per_second	double precision	Number of bytes per second transferred over the network.
nodata	boolean	Whether this restore was from a "nodata" backup (metadata only).
tables_completed	bigint	Number of tables completely restored.
total_tables	bigint	Total number of tables to be restored.

Example

The following query reports the progress of a single restore operation.

```
premdb=# select * from sys.restore where start_time > '2018-12-04 13:25:00-07';
(0 rows)

premdb=# \watch 3
(0 rows)
...
Watch every 3s  Fri Dec  4 13:29:05 2018

-[ RECORD 1 ]---------+-----------------------------------------------------------
session_key           | Cy8_loaiYoBkniu8T8kp2bIeDytL56RedlPioDCpCXMl8-BU363zB6cTnAb
state                 | RUNNING
error_string          | [NULL]
object_id             | 16481
user_id               | 16399
session_id            | 16470
object_name           | newpremdb
user_name             | bobr
client_hostname       | localhost
client_username       | brumsby
```

```
start_time            | 2018-12-04 13:29:04-07
last_activity_time    | 2018-12-04 13:29:05-07
transaction_id        | 6688
backup_name           | dec04
directories           | [/home/brumsby/premdb_backups]
sent_bytes            | 2305
sent_bytes_per_second | 2305
nodata                | f
tables_completed      | 2
total_tables          | 6

...

Watch every 3s  Fri Dec  4 13:29:08 2018

-[ RECORD 1 ]---------+-----------------------------------------------------------
session_key           | Cy8_loaiYoBkniu8T8kp2bIeDytL56RedlPioDCpCXMl8-BU363zB6cTnAb
state                 | RUNNING
error_string          | [NULL]
object_id             | 16481
user_id               | 16399
session_id            | 16470
object_name           | newpremdb
user_name             | bobr
client_hostname       | localhost
client_username       | brumsby
start_time            | 2018-12-04 13:29:04-07
last_activity_time    | 2018-12-04 13:29:07-07
transaction_id        | 6688
backup_name           | dec04
directories           | [/home/brumsby/premdb_backups]
sent_bytes            | 267766
sent_bytes_per_second | 89255.3333333333
nodata                | f
tables_completed      | 5
total_tables          | 6

...

Watch every 3s  Fri Dec  4 13:29:53 2018

-[ RECORD 1 ]---------+-----------------------------------------------------------
session_key           | Cy8_loaiYoBkniu8T8kp2bIeDytL56RedlPioDCpCXMl8-BU363zB6cTnAb
state                 | DONE
error_string          | [NULL]
```

```
object_id             | 16481
user_id               | 16399
session_id            | 16470
object_name           | newpremdb
user_name             | bobr
client_hostname       | localhost
client_username       | brumsby
start_time            | 2018-12-04 13:29:04-07
last_activity_time    | 2018-12-04 13:29:35-07
transaction_id        | 6688
backup_name           | dec04
directories           | [/home/brumsby/premdb_backups]
sent_bytes            | 1246826127
sent_bytes_per_second | 40220197.6451613
nodata                | f
tables_completed      | 6
total_tables          | 6
```

Parent topic: System Views

sys.role

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. See Synchronizing Users and Groups.
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 |
```

```
---------+--------+-------------+-----------------+-----------+--------------------+
   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 does not have 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 | me
---------+------+-------------+-----------------+-----------+--------------------+--
(0 rows)
```

Parent topic: System Views

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.
name	varchar(256)	Name of the schema.
owner_id	bigint	Unique ID of the user who owns the schema. Join to `sys.user` to get the name of the user.
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.

For example:

```
yellowbrick=# select * from sys.schema order by 4,2,1;
 schema_id | database_id | owner_id |    name     |         creation_time          |
-----------+-------------+----------+-------------+--------------------------------+
     16552 |       16520 |    16007 | premdb_bobr | 2020-10-26 14:41:36.499661-07 |
     16553 |       16520 |    16007 | premdb_test | 2020-10-26 14:44:52.985893-07 |
      2200 |        4400 |       10 | public      | [NULL]                         |
      2200 |       16392 |       10 | public      | [NULL]                         |
      2200 |       16393 |       10 | public      | [NULL]                         |
      2200 |       16520 |       10 | public      | [NULL]                         |
      3300 |        4400 |       10 | sys         | [NULL]                         |
      3300 |       16392 |       10 | sys         | [NULL]                         |
      3300 |       16393 |       10 | sys         | [NULL]                         |
      3300 |       16520 |       10 | sys         | [NULL]                         |
(10 rows)
```

Parent topic: System Views

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.
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` reported in this column is calculated as follows: `last_value = ([NEXTVAL](../ybd_sqlref/nextval.md) (‘my_seq_name’) - [sys.worker\_id\(\)] (../ybd_sqlref/sys.worker_id.md)) / 1024` where `1024` is the maximum number of workers. For example, if `last_value` reports `49`: `49 = (51199 - 1023) / 1024` where `NEXTVAL` for the sequence returns `51199` and `sys.worker_id()` returns `1023`.
max_value	bigint	The maximum value that the sequence can generate.

Column Name	Data Type	Description
is_temp	boolean	Whether the sequence was created as a temporary sequence (`f` =persistent; `t` =temporary).
creation_time	timestamptz	When the sequence was created.

Example

```
premdb=# create sequence matchid start 50000;
CREATE SEQUENCE
premdb=# select * from sys.sequence;
 database_id | schema_id | sequence_id |    fullname    | start_value | last_value
-------------+-----------+-------------+----------------+-------------+------------
       16388 |      2200 |       16403 | public.matchid |          49 |         49 |
(1 row)

premdb=# select nextval('matchid');
 nextval
---------
   51199
(1 row)
```

Parent topic: System Views

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`, `ybload`, `ybunload`, or `yb-smc`. 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.

Column Name	Data Type	Description
secure_details	varchar(256)	For secure connections, the SSL version, cipher, and key size.

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, su.superuser, sd.databa
from sys.session ss, sys.user su, sys.database sd where ss.user_id=su.user_id and ss
and application_name='ybsql';
 session_id | user_id | username | superuser | database_id |      dbname
------------+---------+----------+-----------+-------------+------------------
      23281 |   16590 | ybd_test | f         |       16588 | yellowbrick_test
      23449 |   26261 | bobr     | t         |       26259 | premdb
(2 rows)
```

Return all active `ybsql` sessions:

```
yellowbrick_test=# select * from sys.session where state='active' and application_na
-[ RECORD 1 ]-----+-----------------------------------------------
session_id        | 40424
application_name  | ybsql
client_hostname   | [NULL]
client_ip_address | 10.30.22.203/32
database_id       | 16388
user_id           | 16007
start_time        | 2020-01-28 15:04:19.48118-08
process_id        | 22371
secure_connection | t
last_statement    | 2020-01-28 15:07:12.48244-08
state             | active
secure_details    | TLSv1.2/ECDHE-RSA-AES256-GCM-SHA384/256 bits
```

Parent topic: System Views

sys.storage

This system view returns current storage statistics per worker node.

Column Name	Data Type	Description
worker_id	uuid	Unique ID for the worker 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 worker; 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 worker.
replicated_bytes	bigint	Total number of bytes of replicated data on the worker.
random_bytes	bigint	Total bytes used for randomly distributed data on the worker.
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, distribute
             worker_id               |  total_bytes  |  used_bytes   |  free_bytes
-------------------------------------+---------------+---------------+------------
 00000000-0000-0000-0000-38b8ebd000b4 | 2048430964736 |  635417362432 | 141301360230
 00000000-0000-0000-0000-38b8ebd0024e | 4096828375040 | 1045096824832 | 305173155020
 00000000-0000-0000-0000-38b8ebd009bf | 4096828375040 | 1045094727680 | 305173364736
 00000000-0000-0000-0000-38b8ebd001a9 | 2048430964736 |  635432042496 | 141299892224
 00000000-0000-0000-0000-38b8ebd0086b | 4096828375040 | 1045119893504 | 305170848153
 00000000-0000-0000-0000-38b8ebd001f9 | 4096828375040 | 1045147156480 | 305168121856
 00000000-0000-0000-0000-38b8ebd009f1 | 4096828375040 | 1045084241920 | 305174413311
 00000000-0000-0000-0000-38b8ebd0023f | 2048430964736 |  635417362432 | 141301360230
 00000000-0000-0000-0000-38b8ebd00023 | 2048430964736 |  635417362432 | 141301360230
 00000000-0000-0000-0000-38b8ebd0082f | 4096828375040 | 1045113602048 | 305171477299
 00000000-0000-0000-0000-38b8ebd00163 | 2048430964736 |  635432042496 | 141299892224
 00000000-0000-0000-0000-38b8ebd009a6 | 4096828375040 | 1045080047616 | 305174832742
 00000000-0000-0000-0000-38b8ebd00087 | 4096828375040 | 1045096824832 | 305173155020
 00000000-0000-0000-0000-38b8ebd00866 | 4096828375040 | 1045088436224 | 305173993881
 00000000-0000-0000-0000-38b8ebd00235 | 4096828375040 | 1045098921984 | 305172945305
(15 rows)
```

Parent topic: System Views

sys.table

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	text	Distribution type: `replicated`, `hash`, `random`.
sort_key	text	Column name if sorted; otherwise `null`.
distribution_key	text	Column name if distributed by `hash`; othwerise `null`.
cluster_keys	text	Columns used as cluster keys, if any.
partition_keys	text	Columns used as partitioning columns, if any.
is_temp	boolean	Whether the table was created as a temporary table (`f`=persistent; `t`=temporary).
definition	text	*Not implemented*.

Column Name	Data Type	Description
is_auto_analyze	boolean	`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.
rowstore_full_action	"char"	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.

Column Name	Data Type	Description
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.

Examples

```
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 11:47:50.685
   16885 | team          | replicated   |                  | 2017-09-01 11:47:56.036
   16888 | hometeam      | replicated   |                  | 2017-09-01 11:44:19.424
   16891 | awayteam      | replicated   |                  | 2017-09-01 11:44:19.032
   16894 | match         | hash         | seasonid         | 2017-09-01 11:48:07.712
   16898 | newmatchstats | hash         | seasonid         | 2017-09-01 11:40:18.967
(6 rows)
```

```
premdb_match=# select name, rowstore_bytes, rowstore_row_count, rowstore_disk_usage,
 name  | rowstore_bytes | rowstore_row_count | rowstore_disk_usage | rowstore_size_
-------+----------------+--------------------+---------------------+---------------
 new   |            231 |                  3 |            33554432 |        5497558
 match |       33288310 |               6502 |            33554432 |            335
(2 rows)
```

Return information for a table named `season`:

```
premdb=# select * from sys.database where name = 'premdb';
-[ 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]
```

Parent topic: System Views

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 worker 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_
```

```
----------+---------------------+--------------------+---------------------+--------
   16998 |                   0 |                  0 |                   0 | f
   17011 |                   0 |                  0 |                   0 | f
   17017 |             1049932 |                  0 |                   0 | f
   16995 |                   0 |                  0 |                   0 | f
   17001 |                   0 |                  0 |                   0 | f
   16992 |                   0 |                  0 |                   0 | f
   17004 |                   0 |                  0 |                   0 | f
   17014 |                   0 |                  0 |                   0 | f
(8 rows)

premdb=# select * from sys.table_flush_summary;
 table_id | rows_pending_insert | size_pending_flush | rows_pending_flush | queued_
----------+---------------------+--------------------+---------------------+--------
   16998 |                   0 |                  0 |                   0 | f
   17011 |                   0 |                  0 |                   0 | f
   17017 |             1049932 |           71480152 |              100000 | t
   16995 |                   0 |                  0 |                   0 | f
   17001 |                   0 |                  0 |                   0 | f
   16992 |                   0 |                  0 |                   0 | f
   17004 |                   0 |                  0 |                   0 | f
   17014 |                   0 |                  0 |                   0 | f
(8 rows)

premdb=# select * from sys.table_flush_summary;
 table_id | rows_pending_insert | size_pending_flush | rows_pending_flush | queued_
----------+---------------------+--------------------+---------------------+--------
   16998 |                   0 |                  0 |                   0 | f
   17011 |                   0 |                  0 |                   0 | f
   17017 |                   0 |                  0 |                   0 | f
   16995 |                   0 |                  0 |                   0 | f
   17001 |                   0 |                  0 |                   0 | f
   16992 |                   0 |                  0 |                   0 | f
   17004 |                   0 |                  0 |                   0 | f
   17014 |                   0 |                  0 |                   0 | f
(8 rows)
```

Parent topic: System Views

sys.table_info

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 s
where (sti.table_id=sct.table_id) and (sct.schema_id=sch.schema_id) and (sct.databas
and sct.table_id>10000;
   name   | table_id | rows_rowstore | modified_rows_statistics |         last_analy
----------+----------+---------------+--------------------------+-------------------
 awayteam |    17560 |        [NULL] |                   [NULL] | 2020-10-27 12:35:0
 hometeam |    17556 |        [NULL] |                   [NULL] | 2020-10-27 12:35:0
 team     |    17552 |        [NULL] |                   [NULL] | 2020-10-27 13:10:4
 season   |    17548 |        [NULL] |                   [NULL] | 2020-10-27 13:10:4
 match    |    17564 |        [NULL] |                   [NULL] | 2020-10-27 15:14:2
(5 rows)
```

Parent topic: System Views

sys.table_partition

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 zeroes. 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 cou
group by name, stp.table_id;
          name            | table_id | max  | min  |          avg          |      st
--------------------------+----------+------+------+-----------------------+------
 partitioned_newmatchstats |    16895 | 9576 | 8664 | 8798.9596412556053812 | 324.19
(1 row)
```

Parent topic: System Views

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 worker 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 worker 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_
   sum
-----------
 600037902
(1 row)
```

```
yellowbrick_test=# select * from sys.table_storage where table_id=16606;
 table_id |              worker_id               | rows_columnstore | compressed_by
----------+--------------------------------------+------------------+--------------
   16606 | 00000000-0000-0000-0000-38b8ebd00023 |         40012122 |       169240160
   16606 | 00000000-0000-0000-0000-38b8ebd00087 |         40013865 |       169240160
   16606 | 00000000-0000-0000-0000-38b8ebd000b4 |         39998238 |       169030451
   16606 | 00000000-0000-0000-0000-38b8ebd00163 |         39990064 |       169240160
   16606 | 00000000-0000-0000-0000-38b8ebd001a9 |         40007412 |       169240160
   16606 | 00000000-0000-0000-0000-38b8ebd001f9 |         39983517 |       169030451
   16606 | 00000000-0000-0000-0000-38b8ebd00235 |         40008626 |       169240160
   16606 | 00000000-0000-0000-0000-38b8ebd0023f |         40006680 |       169240160
   16606 | 00000000-0000-0000-0000-38b8ebd0024e |         39975472 |       169240160
   16606 | 00000000-0000-0000-0000-38b8ebd0082f |         40015874 |       169240160
   16606 | 00000000-0000-0000-0000-38b8ebd00866 |         39993230 |       169030451
```

```
    16606 | 00000000-0000-0000-0000-38b8ebd0086b |         40008190 |       16944988
    16606 | 00000000-0000-0000-0000-38b8ebd009a6 |         40010891 |       16924016
    16606 | 00000000-0000-0000-0000-38b8ebd009bf |         40017426 |       16924016
    16606 | 00000000-0000-0000-0000-38b8ebd009f1 |         39996295 |       16924016
(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
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
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)
```

Parent topic: System Views

sys.unload

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_NZ1-6Md6QHlKZnSgaLxI
state                 | DONE
error_string          | [NULL]
database_id           | 16423
user_id               | 16434
session_id            | 16790
database_name         | premdb
user_name             | 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
```

Parent topic: System Views

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.
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.
memberof	name[]	Roles that the user is a member of.

Example

```
premdb=# select * from sys.user where name not like 'sys%' and user_id>16100;
 user_id |   name   | search_path | create_role | create_database | superuser | has_
---------+----------+-------------+-------------+-----------------+-----------+----
   16475 | backup2  | [NULL]      | f           | f               | f         | t
   16473 | bar      | [NULL]      | f           | f               | f         | t
   16387 | ybd_test | [NULL]      | f           | f               | f         | t
   16474 | backup1  | [NULL]      | f           | f               | f         | t
   16476 | restore1 | [NULL]      | f           | f               | f         | t
   16472 | bobr     | [NULL]      | t           | t               | t         | t
   16477 | restore2 | [NULL]      | f           | f               | f         | t
(7 rows)
```

Parent topic: System Views

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	text	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	text	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=
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
---------+-------------+-----------+-----------+----------+-----------------------+--
   16440 |       16402 |     16439 | teamview  |    16007 |  SELECT team.teamid,+| f
         |             |           |           |          |     team.htid,      +|
         |             |           |           |          |     team.atid,      +|
         |             |           |           |          |     team.name,      +|
         |             |           |           |          |     team.nickname,  +|
         |             |           |           |          |     team.city,      +|
         |             |           |           |          |     team.stadium,   +|
         |             |           |           |          |     team.capacity,  +|
         |             |           |           |          |     team.avg_att    +|
         |             |           |           |          |    FROM team;        |
(1 row)
```

Parent topic: System Views

sys.wlm_active_pool

This view returns the currently *active* WLM resource pools and identifies their profile and other attributes, including the number of pending queries for each prioritized queue.

Column Name	Data Type	Description
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 the SMC).
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 the SMC).
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.

Column Name	Data Type	Description
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.
running	bigint	Number of queries running on workers 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:

```
yellowbrick=# select * from sys.wlm_active_pool;
-[ RECORD 1 ]--------------+---------------
name                       | system
profile_name               | shortquerybias
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
running                    | 0
low_pending                | 0
normal_pending             | 0
high_pending               | 0
critical_pending           | 0
-[ RECORD 2 ]--------------+---------------
name                       | shortquerypool
profile_name               | shortquerybias
memory_requested           | remainder
memory_per_query_bytes     | 5722079232
```

```
temp_space_requested       | remainder
temp_space_per_query_bytes | 73345794048
max_concurrency            | 20
min_concurrency            | 1
queue_size                 | 100
running                    | 0
low_pending                | 0
normal_pending             | 0
high_pending               | 0
critical_pending           | 0
```

Parent topic: System Views

sys.wlm_active_profile

This view returns the current list of available WLM profiles, identifies the currently active profile, and names the default pool for each profile. For example, the `shortquerybias` profile is activated in this case:

```
yellowbrick=# select * from sys.wlm_active_profile;
           name             | active |       default_pool
----------------------------+--------+--------------------------
 maintenance                | f      | maintenance
 default                    | f      | large
 throttling_testing_profile | f      | throttling_resource_pool
 flex                       | f      | mix
 profile_one                | f      | profile_one_pool_a
 verification_profile       | f      | verification_pool_a
 shortquerybias             | t      | shortquerypool
(7 rows)
```

Note: This view always returns rows for the `default`, `flex`, and `maintenance` profiles.

Parent topic: System Views

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 |
----------------+-------------------+-----------+-------+---------+-----------+----
 shortquerybias | limit_concurrency | submit    |     1 | t       | f         | if
                |                   |           |       |         |           |
                |                   |           |       |         |           | }
 shortquerybias | log_premdb        | submit    |   100 | t       | f         |  if
(2 rows)
```

Parent topic: System Views

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 | superuse
----------------+-------------------+------------------+-------+---------+---------
 shortquerybias | fastlane          | compile          |   100 | t       | f
 shortquerybias | log_premdb        | submit           |   100 | t       | f
 shortquerybias | limit_concurrency | submit           |     1 | t       | f
                |                   |                  |       |         |
                |                   |                  |       |         |
 shortquerybias | assemble_rule     | assemble         |     1 | t       | f
```

```
                |                    |                   |       |         |
                |                    |                   |       |         |
 shortquerybias | restart_admin      | restart_for_user  |     1 | t       | f
                |                    |                   |       |         |
                |                    |                   |       |         |
(5 rows)
```

Parent topic: System Views

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
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 the SMC).
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 the SMC).
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.

Column Name	Data Type	Description
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.

Example

For example:

```
yellowbrick=# select * from sys.wlm_pending_pool where profile_name='shortquerybias
-[ RECORD 1 ]--------------+---------------
name                       | system
profile_name               | shortquerybias
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
-[ RECORD 2 ]--------------+---------------
name                       | shortquerypool
profile_name               | shortquerybias
memory_requested           | remainder
memory_per_query_bytes     | 5722079232
temp_space_requested       | remainder
temp_space_per_query_bytes | 73345794048
max_concurrency            | 20
min_concurrency            | 1
queue_size                 | 100
```

Parent topic: System Views

sys.worker

This view captures the current state and role of each worker node on the cluster.

Column Name	Data Type	Description
worker_id	uuid	UUID for the worker node.
chassis_id	integer	`0` for a single-chassis appliance; `0` or `1` for a dual-chassis appliance.
logical_id	integer	Logical ID number for the worker node.
ip_address	text	IP address for the worker node.
state	text	Current state, such as `IN_SERVICE` or `OFFLINE`. This value corresponds to the "Cluster status" for the blade, as returned by the ybcli blade status command.
role	text	Current role, such as `MEMBER` or `SPARE`. This value corresponds to the "Cluster role" for the blade, as returned by the ybcli blade status command.
state_updated	timestamptz	When the state of the worker changed.
drive_status	text(12)	Status of the worker node drives: `OK` (all drives online) or `DEGRADED` (one or more drives offline).

For example:

```
yellowbrick_test=# select * from sys.worker order by 2,3;
             worker_id               | chassis_id | logical_id |  ip_address   |   s
-------------------------------------+------------+------------+---------------+--
 00000000-0000-0000-0000-38b8ebd00c35 |          0 |          0 | 192.168.10.19 | IN
 00000000-0000-0000-0000-38b8ebd00c6c |          0 |          1 | 192.168.10.18 | IN
 00000000-0000-0000-0000-38b8ebd007fd |          0 |          2 | 192.168.10.11 | IN
 00000000-0000-0000-0000-38b8ebd00ca8 |          0 |          3 | 192.168.10.16 | IN
 00000000-0000-0000-0000-38b8ebd00c2b |          0 |          5 | 192.168.10.15 | IN
 00000000-0000-0000-0000-38b8ebd00c58 |          0 |          7 | 192.168.10.17 | IN
```

```
 00000000-0000-0000-0000-38b8ebd00c0d |          0 |          8 | 192.168.10.12 | IN
 00000000-0000-0000-0000-38b8ebd00c30 |          0 |          9 | 192.168.10.14 | IN
 00000000-0000-0000-0000-38b8ebd00c1c |          0 |         12 | 192.168.10.10 | IN
 00000000-0000-0000-0000-38b8ebd00bf9 |          0 |         13 | 192.168.10.13 | IN
 00000000-0000-0000-0000-38b8ebd00c53 |          0 |         20 | 192.168.10.20 | IN
 00000000-0000-0000-0000-38b8ebd00c99 |          1 |          4 | 192.168.20.20 | IN
 00000000-0000-0000-0000-38b8ebd00c3f |          1 |          6 | 192.168.20.17 | IN
 00000000-0000-0000-0000-38b8ebd00c26 |          1 |         10 | 192.168.20.11 | IN
 00000000-0000-0000-0000-38b8ebd00c76 |          1 |         11 | 192.168.20.19 | IN
 00000000-0000-0000-0000-38b8ebd00c7b |          1 |         14 | 192.168.20.15 | IN
 00000000-0000-0000-0000-38b8ebd00c62 |          1 |         15 | 192.168.20.12 | IN
 00000000-0000-0000-0000-38b8ebd00c17 |          1 |         16 | 192.168.20.14 | IN
 00000000-0000-0000-0000-38b8ebd00c08 |          1 |         17 | 192.168.20.18 | IN
 00000000-0000-0000-0000-38b8ebd00c49 |          1 |         18 | 192.168.20.13 | IN
 00000000-0000-0000-0000-38b8ebd00cad |          1 |         19 | 192.168.20.10 | IN
 00000000-0000-0000-0000-38b8ebd00c21 |          1 |         21 | 192.168.20.16 | IN
(22 rows)
```

Parent topic: System Views

Loading and Unloading Data

This section covers different ways in which you can bulk load data into Yellowbrick tables and export (unload) data from those tables.

In This Section

- Bulk Loading Tables
- Trickle Loading Data via JDBC
- Loading Tables with Spark
- Unloading Data

Parent topic: Yellowbrick Documentation

Bulk Loading Tables

The Yellowbrick bulk loader (`ybload`) is a Java-based bulk data loader that you invoke from a client system (not from a node within a Yellowbrick cluster). 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 the worker 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 appliance so that your client and server versions correspond. The client tools are backward-compatible but not always forward-compatible.

In This Section

- Running a Bulk Load
- Bulk Load Examples
- Loading Tables from Object Storage
- ybload Command

Parent topic: Loading and Unloading Data

Running a Bulk Load

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*. For example, the default `yellowbrick` user is 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). 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.

  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.

  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.

In This Section

- Using the Dry Run Option
- Setting the Commit Interval
- Saving Load Options to a File
- Loading Generated Key Values
- Specifying NULL Behavior for Different Columns
- Deletes, Updates, and Upserts
- Handling Duplicate Rows
- Monitoring Load Operations

- Analyzing Rejected Rows
- Resuming a Partial Load
- Loading Data Exported from SQL Server

Parent topic: Bulk Loading Tables

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/mat
Password for user bobr:
17:03:07.065 [ INFO] ABOUT CLIENT:
   app.cli_args         = --dryrun -d premdb -t match --format csv --username bobr
   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, mixe
   jvm.options          = -Xmx16g, -Xms1g, -ea, -Dapp.name=ybload, -Dapp.pid=2823,
   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: LAT
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
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 860
```

```
17:03:08.477 [ INFO] READ:305.8KB(224.0KB/s). ROWS G/B: 8606/0( 6.16K/s). WRITE:285
17:03:08.477 [ INFO] SUCCESSFUL BULK LOAD (--dryrun): Loaded 8606 good rows in   0:0
```

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
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,-
...
```

Parent topic: Running a Bulk Load

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.

Parent topic: Running a Bulk Load

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 ...
```

Parent topic: Running a Bulk Load

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)
```

Parent topic: Running a Bulk Load

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_nu
--format csv --date-field-options '{"nullmarker": "----/--/--"}' --per-field-option
```

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      |     sta
--------+------+------+-------------------------+----------+---------------+-------
    46 |   47 |   96 | Wimbledon               | Dons     | London        | Selhurst
    47 |   48 |   97 | Wolverhampton Wanderers | Wolves   | Wolverhampton | Molineux
    48 |   49 |   98 | [NULL]                  | [NULL]   | [NULL]        | [NULL]
    49 |   50 |   99 | [NULL]                  | [NULL]   | [NULL]        | [NULL]
    50 |   51 |  100 | [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.)

Parent topic: Running a Bulk Load

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.

Parent topic: Running a Bulk Load

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`

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.
>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
>1 and `--duplicate-handler order by	random`	1

Number of matching rows in source file	Number of matching rows in target table	Behavior
>1 and `--duplicate-handler order by	random`	>1

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."

Number of matching rows in source file	Number of matching rows in target table	Behavior
>1 and `--duplicate-handler order by	random`	0
>1 and `--duplicate-handler order by	random`	1
>1 and `--duplicate-handler order by	random`	>1

DELETE Loads

The `--duplicate-handler` option is ignored for `delete` loads.

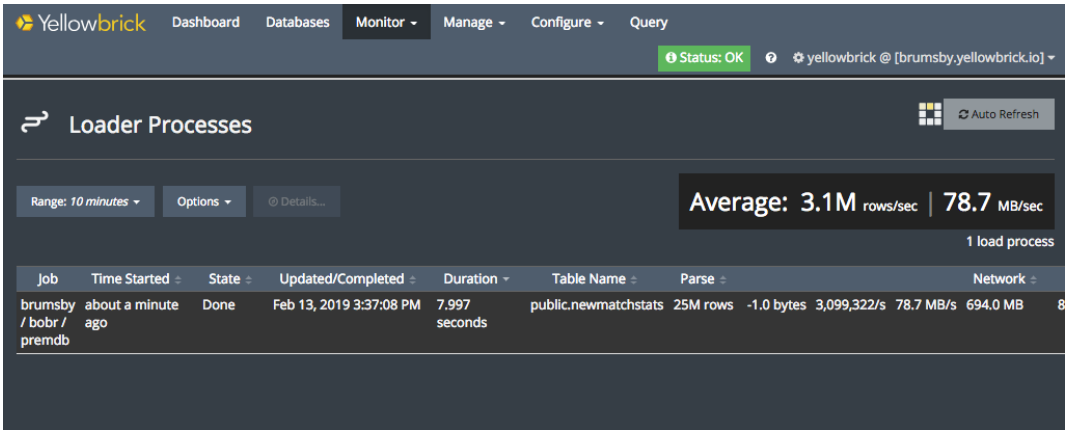
Number of matching rows in source file	Number of matching rows in target table	Behavior
0	0, 1, or >1	No deletes.
1	0	No deletes.
1 or >1	1 or >1	All matching target rows are deleted.
>1	0	No deletes.
>1	1 or >1	All matching target rows are deleted.

Parent topic: Running a Bulk Load

Monitoring Load Operations

You can monitor active bulk load operations in the System Management Console (SMC). After logging in, go to **Monitor** > **Loader Processes**. Load progress is reported periodically, with a brief summary when the load is complete. By default, rejected rows do not stop the load. They are logged to a file (and its location is reported to the SMC).

For example:



Alternatively, you can 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/n
Password for user bobr:
...
13:30:44.841 [ INFO] Flushing last 6818022 rows (of 24785280 total) in transaction #
READ:600.1MB(94.58MB/s). ROWS G/B: 24785280/0( 3.73M/s). WRITE:661.8MB(104.3MB/s).
13:30:46.535 [ INFO] Committed transaction #1 after a total of 693987840 bytes and 1
13:30:46.560 [ INFO] READ:600.1MB(77.65MB/s). ROWS G/B: 24785280/0( 3.06M/s). WRITE
13:30:46.561 [ INFO] SUCCESSFUL BULK LOAD: Loaded 24785280 good rows in   0:00:07 (
```

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, s
from sys.log_load sll join sys.log_query slq on sll.transaction_first=slq.transacti
where sll.username='bobr' and sll.table_name='newmatchstats'
order by 4 desc;
  table_name   | username |          end_time          | transaction_first | inserte
---------------+----------+----------------------------+-------------------+-------
 newmatchstats | bobr     | 2020-10-30 13:30:46.579-07 |             96045 |      24
(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 5.0.0-32710
```

Parent topic: Running a Bulk Load

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

#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"
"delim":"auto"}]}
#.....................................................................<--------
 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"
"delim":"auto"}]}
#.....................................................................<--------
 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"
#                                     ..................<--------- Doesn't match any
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"
#                                     ..................<--------- Doesn't match any
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 allow
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:{}
#
2|PGL08LJI|Vladimir|Humphrey|Murfreesboro|SK|Suspendisse.tristique@nonnisiAenean.edu

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

Parent topic: Running a Bulk Load

Resuming a Partial 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
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

2018-01-17 17:02:59.989 [ WARN] <main>  BEWARE: Additional bad rows were written to
2018-01-17 17:02:59.989 [ WARN] <main>          When fixing rows in the bad row file
2018-01-17 17:02:59.990 [ WARN] <main>          "----- successful commit after 3 bad
```

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
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
2018-01-17 17:03:06.950 [ WARN] <main>          When fixing rows in the bad row file
2018-01-17 17:03:06.950 [ WARN] <main>          "----- successful commit after 3 bad
```

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
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
   2) ybload <original options>  \
   /data/tests/big3.csv

2018-01-17 17:03:12.119 [ WARN] <main>  BEWARE: Additional bad rows were written to
2018-01-17 17:03:12.119 [ WARN] <main>          When fixing rows in the bad row file
2018-01-17 17:03:12.119 [ WARN] <main>          "----- successful commit after 8 bad
```

Parent topic: Running a Bulk Load

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 32
```

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
```

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.

Parent topic: Running a Bulk Load

Bulk Load Examples

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 S3 Object Storage Examples and Azure Blob Storage Examples.

In This Section

- Bulk Load Insert Examples
- Delete, Update, and Upsert Examples
- Examples with Duplicate Rows

Parent topic: Bulk Loading Tables

Bulk Load Insert Examples

The following examples show how to use various `ybload` options in the context of a default `INSERT` load. See also 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/
   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, mixe
   jvm.options          = -Xmx16g, -Xms1g, -ea, -Dapp.name=ybload, -Dapp.pid=19168,
   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: LATI
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
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
18:18:34.275 [ INFO] READ:305.8KB(193.0KB/s). ROWS G/B: 8606/0( 5.30K/s). WRITE:285
18:18:34.276 [ INFO] SUCCESSFUL BULK LOAD: Loaded 8606 good rows in   0:00:01 (READ
```

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 /ho
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: LATI
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.20
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
18:39:26.452 [ INFO] READ:305.8KB(194.7KB/s). ROWS G/B: 8606/0( 5.35K/s). WRITE:285
18:39:26.453 [ INFO] SUCCESSFUL BULK LOAD: Loaded 8606 good rows in   0:00:01 (READ
```

Load the same table from multiple CSV files

```
$ ybload -d premdb --username bobr -t match -W --format csv /home/bobr/premdata/mat
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.ba
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 258
18:50:40.607 [ INFO] READ:917.4KB(570.9KB/s). ROWS G/B: 25818/0(15.69K/s). WRITE:85
18:50:40.607 [ INFO] SUCCESSFUL BULK LOAD: Loaded 25818 good rows in   0:00:01 (REA
```

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_
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 --quie
   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, mixe
   jvm.options          = -Xmx16g, -Xms1g, -ea, -Dapp.name=ybload, -Dapp.pid=12595,
   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
```

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 loa
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 parsi
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 parsi
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 parsi
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 parsi
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 '/'
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 '|' --nullma
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 versus 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
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
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 '|' 
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
...
12:19:02.492 [ INFO] Bulk Loading /home/premdata/awayteam.csv into TABLE **"public"
...
```

Parent topic: Bulk Load Examples

Delete, Update, and Upsert 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 -
...
20:34:57.159 [ INFO] Flushing last 16 rows (of 16 total) in transaction #1 for mini
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
20:34:57.514 [ INFO] Committed transaction #1 after a total of 544 bytes and 16 good
20:34:57.531 [ INFO] READ:576.0 B(278.5 B/s). ROWS G/B: 16/0( 7.74 /s). WRITE:544.0
20:34:57.533 [ INFO] SUCCESSFUL BULK LOAD: Read 16 good rows from the source file(s)
```

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, inse
from sys.log_load where username='bobr' and table_name='minimatch' and error_string
order by end_time desc;
                          session_key                           | table_name | user
----------------------------------------------------------------+------------+----
 BrJx6HcnNQkNOGxorNSPD76G-br5LQwqtgcM5HPDCclCej4OOLAaqwzcd9wF1dnK | minimatch  | bob
 ...
```

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. In other words,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)
```

Parent topic: Bulk Load Examples

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.

Parent topic: Bulk Load Examples

Loading Tables from Object Storage

This section describes how to use `ybload` to bulk load tables from supported object storage systems, including Azure Blob storage and AWS S3. Other S3-compatible object stores are also supported.

See also the documentation for the Yellowbrick Test Drive.

In This Section

- Loading from Amazon S3
- Loading from S3-Compatible Object Storage
- Loading from Azure Blob Storage
- ybload Object Storage Options

Parent topic: Bulk Loading Tables

Loading from Amazon S3

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/mat
```

Note: The bad rows file (and a log file, if specified) must be written to the local file system. Specifying these files in an object storage location, such as an S3 bucket, is not supported.

In This Section

- S3 Authentication Methods
- S3 Object Storage URIs
- S3 Object Storage Examples

Parent topic: Loading Tables from Object Storage

S3 Authentication Methods

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:
- ybload 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; Reque
```

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`)

Parent topic: Loading from Amazon S3

S3 Object Storage URIs

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.

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.

Parent topic: Loading from Amazon S3

S3 Object Storage Examples

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-r
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: LATI
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
```

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-r
--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: LATI
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-r
--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: ********
...
```

Parent topic: Loading from Amazon S3

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.

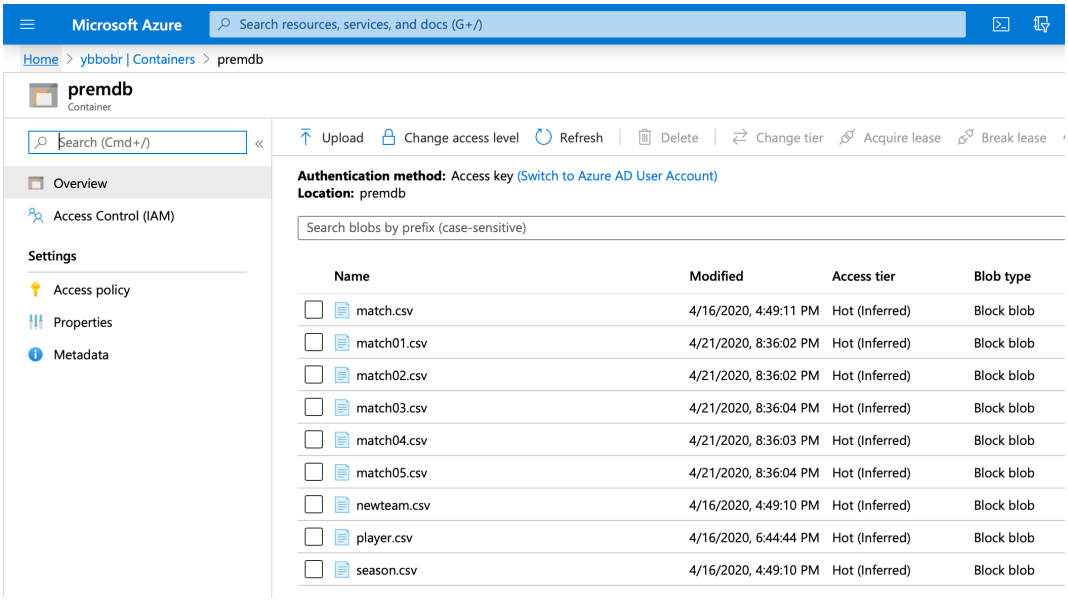
You can only access one object store per `ybload` operation. For example, you cannot load data from multiple S3 endpoints or storage accounts in a single command.

Parent topic: Loading Tables from Object Storage

Loading from Azure Blob Storage

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, as long as 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 (and the log file, if specified)
- 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 ybload Object Storage Options and Azure Blob Storage URIs.

In This Section

- Azure Authentication Methods
- Azure Blob Storage URIs
- Azure Blob Storage Examples

Parent topic: Loading Tables from Object Storage

Azure Authentication Methods

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 is disabled by default for `ybload`. 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

Parent topic: Loading from Azure Blob Storage

Azure Blob Storage URIs

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`

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.

Parent topic: Loading from Azure Blob Storage

Azure Blob Storage Examples

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-r
--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"
   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, mix
   jvm.options          = "-Xmx16g, -Xms1g, -Dapp.name=ybload, -Dapp.pid=23460, -Da
   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: LATI
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 m
READ:305.8KB(88.83KB/s). ROWS G/B: 8606/0( 2.44K/s). WRITE:285.7KB(83.01KB/s).  TIME
18:50:01.446 [ INFO] Committed transaction #1 after a total of 292604 bytes and 8606
18:50:01.464 [ INFO] READ:305.8KB(66.00KB/s). ROWS G/B: 8606/0( 1.81K/s). WRITE:285
18:50:01.465 [ INFO] SUCCESSFUL BULK LOAD: Loaded 8606 good rows in   0:00:04 (READ
```

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-r
--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: LATI
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
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-r
--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-r
--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"
   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, mix
   jvm.options          = "-Xmx16g, -Xms1g, -Dapp.name=ybload, -Dapp.pid=6255, -Dap
   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-r
--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 an
[
  {
    "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-
--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/
$ export AZURE_CLIENT_ID=“************************************"
$ export AZURE_CLIENT_SECRET=“************************************”
$ export AZURE_TENANT_ID=“************************************”
$ az role assignment create --role “Storage Blob Data Reader” --assignee $AZURE_CLIE
$ az login --service-principal --username $AZURE_CLIENT_ID --password $AZURE_CLIENT_
...
```

At this point, you can run a `ybload` command using implicit authentication via the established service principal identity and credential.

Parent topic: Loading from Azure Blob Storage

ybload Object Storage Options

The following options are used to specify connectivity, credentials, and configuration settings for object storage systems, including Azure Blob storage and S3 object stores.

Note: The usage and requirements for these options depends on the object store provider.

--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 Blob storage, 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 `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.

--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, `ybload` requires a region value to 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.

--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 Blob storage.

--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 Blob storage, 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-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 Blob storage, this credential is a storage account access key or a generated SAS token.

--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: `{}`.

Java Properties File

If you choose to specify object storage options in a properties file, drop the `--object-store-` prefix from the `ybload` option names. Optionally, you can add a `yb.file.` prefix to the entries in the file.

AWS S3

For example:

```
$ more ybbobrS3.properties

yb.file.region = us-east-2
yb.file.identity = ABCD1EF2GHIJKLMN3OPQ
yb.file.credential = ****************************************
```

This properties file is distinct from the `~/.aws/*` configuration files that the S3 SDK consults.

Azure Blob storage

endpoint = https://mystorageaccount.blob.core.windows.net identity = mystorageaccount credential =

myStorageAccountKey

```
For example:
```

$ more ybbobrAz.properties

yb.file.endpoint = https://ybbobr.blob.core.windows.net yb.file.identity = ybbobr yb.file.credential =

****

```
## S3 Pass-Through Options {#section_h5t_25k_llb .section}

The following pass-through options are supported for S3 clients. You can specify the

accelerateModeEnabled
:   Defaults to `false`.

checksumValidationEnabled
:   Defaults to `true`.

chunkedEncodingEnabled
:   Defaults to `true`.

dualStackEnabled
:   Defaults to `false`.

**Parent topic:**[Loading Tables from Object Storage](../bulk_loading/ybload_from_ob
```

ybload Command

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

---

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.

```
me@mydesk.io:~/ybtools/bin$ ./ybload --help
ybload version 2.0.0-10349
ybload loads source files into an existing table in a Yellowbrick Database.
...
```

Setting Up a Database Connection

To run the bulk loader, you have to start a database session on the server where the destination database and table reside. This session requires connection information that you can provide either as `ybload` options on the command line or as current values for environment variables:

ybload Options	Environment Variable	Description	Example

---

ybload Options	Environment Variable	Description	Example

`-d` or `--dbname`	`YBDATABASE`	Destination database name. Default: `yellowbrick` See also SQL Identifiers.	`--dbname premdb`
			
			`export YBDATABASE=premdb`

`-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`

ybload Options	Environment Variable	Description	Example

`-U` or `--username`	`YBUSER`	Database login username. Defaults to the current OS user on the client system. **Note:** Do not run bulk loads as a `superuser`. The user must have `INSERT` privileges on the target table.	`-U bobr`
			
			`export YBUSER=bobr`

`-W` or `--password`	`YBPASSWORD`	Interactive prompt for the database user's password. No default.	`--password`
			
			`export YBPASSWORD=********`

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 an appliance 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 comes with a large number of options that define ways to process the load, log results, and parse incoming data correctly. See ybload Options for details. Common examples include setting null markers and defining

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.

In This Section

- ybload Options
- Setting the --format Option
- NULL and Empty Markers
- Character Escape Sequences
- ybload Field Options
- ybload Date Formats
- ybload Time Formats
- ybload Timestamp Formats
- ybload Advanced Processing Options
- Escaping Quotes in Windows Clients

Parent topic: Bulk Loading Tables

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.
- (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 load. See Saving Load Options to a File.

--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: 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.

--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.

--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.

See also Secure Connections for Java-based ybtools.

--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 worker 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`.

--dbname, -d

Name of the destination database. See Setting up a Database Connection.

--decimal-field-options

See ybload 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 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 the --format Option.

--disable-trust, -k

Disable SSL/TLS trust when using secured communications. Trust is enabled by default. See also Enabling and 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 appliance.

--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

Specify the formatting style of the incoming data (how the source files were formatted by the export or unload tool that produced them). See also Setting the --format Option. In particular, 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`

--help

Return basic usage information for the `ybload` command and its options.

--help-advanced

Return more advanced usage information for the `ybload` command and its options.

--host

Host name. See Setting up a Database Connection.

--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.

--initial-connection-timeout NUMBER

Number of seconds to wait for initial connections to the database. The default is `120`. This timeout option ensures that `ybload` 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 turn off this option and allow an unlimited wait time, set this option to `0`.

--integer-field-options

See ybload Field Options.

--ip-field-options

See ybload Field 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.

--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.

--logfile STRING

Specify the name and location of a log file for the load operation. 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.

--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-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.

--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 abort 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 ybload Object Storage Options. These options apply to loads from 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 and can be used in combination with the `--on-extra-field` and `--on-missing-field` options.

--password

Prompt for the `ybload` user's password. See Setting up a Database Connection. The user who runs the load must have INSERT permissions on the table (but does not have to own the table).

--per-field-options {JSON Object}

Specify parsing options for individual fields. See ybload Field Options.

--port

Port number. See Setting up a Database Connection.

--quiet

Do not write any output to console. This option is suitable for `cron` invocations of the loader. If `--quiet` is specified, you must also specify `--logfile`.

--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.

--secured

Use SSL/TLS to secure all communications. The default is not secured. See also Enabling and Verifying SSL/TLS Encryption.

--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).

--smallint-field-options

See ybload Field Options.

--source-compression 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`). This option applies to loads from all supported source types and overrides other compression detection methods.

--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).

--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).

--username

Database username. See Setting up a Database Connection.

--uuid-field-options

See ybload Field Options.

--version

Display the version of `ybload` you are running (as part of `ybtools`). This option is not intended to be combined with other options. For example:

```
$ ybload --version
ybload version 1.2.2-5563
```

--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.

Parent topic: ybload Command

Setting the --format Option

The `--format` option is a mechanism for specifying the expected format of `ybload` source files (flat files), given some assumptions about the standard behavior of the export program that produced those files. Specifying the `--format` option is always recommended, but the setting you choose must be based on knowledge of the data and the export program that produced the data. 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 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
`--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.

Parent topic: ybload Command

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.

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
`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`	

Source Data for Three Adjacent Fields	Data Type of Second Field	ybload Marker Settings	Second Field Parsed As?	Notes
`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`	

Parent topic: ybload Command

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.

Parent topic: ybload Command

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: `'{}'`

- `CHAR` and `VARCHAR`: `--char-field-options '{"nullmarker": STRING, "emptymarker": STRING, "onZeroChar": REMOVE | ERROR, "convertAsciiControl": BOOLEAN, "convertCEscape": BOOLEAN``}'`

Default: `'{}'`

- `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: `'{}'`

- `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"
```

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}}
```

Parent topic: ybload Command

ybload Date Formats

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

Date Mask	Examples
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
```

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.

Parent topic: ybload Command

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`

Time Format	Examples
`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.

Parent topic: ybload Command

ybload Timestamp Formats

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)

`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"
```

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.

Parent topic: ybload Command

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 lots of TIMESTAMP fields will parse slower than a file with lots of 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 as long as 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 on the fly 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 versus 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 worker 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.

Parent topic: ybload Command

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

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 quote in front of all the double quote characters.

Parent topic: ybload Command

Trickle Loading Data via JDBC

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`).

In This Section

- Performance Factors
- Performance Tuning
- Example of a Trickle Load Java Program

Parent topic: Loading and Unloading Data

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 event 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.

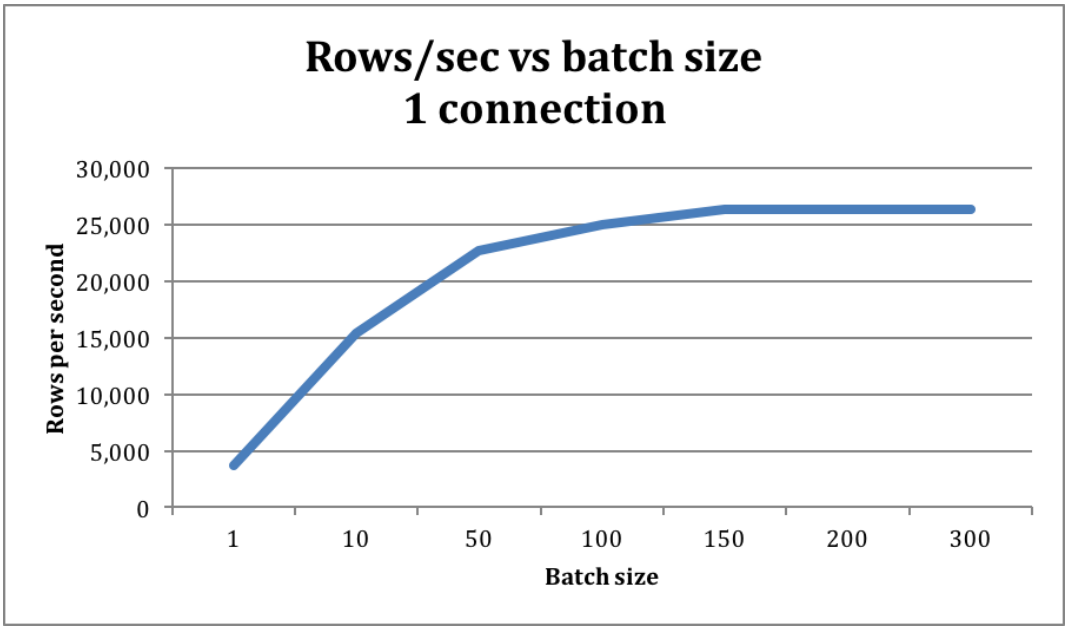
Parent topic: Trickle Loading Data via JDBC

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

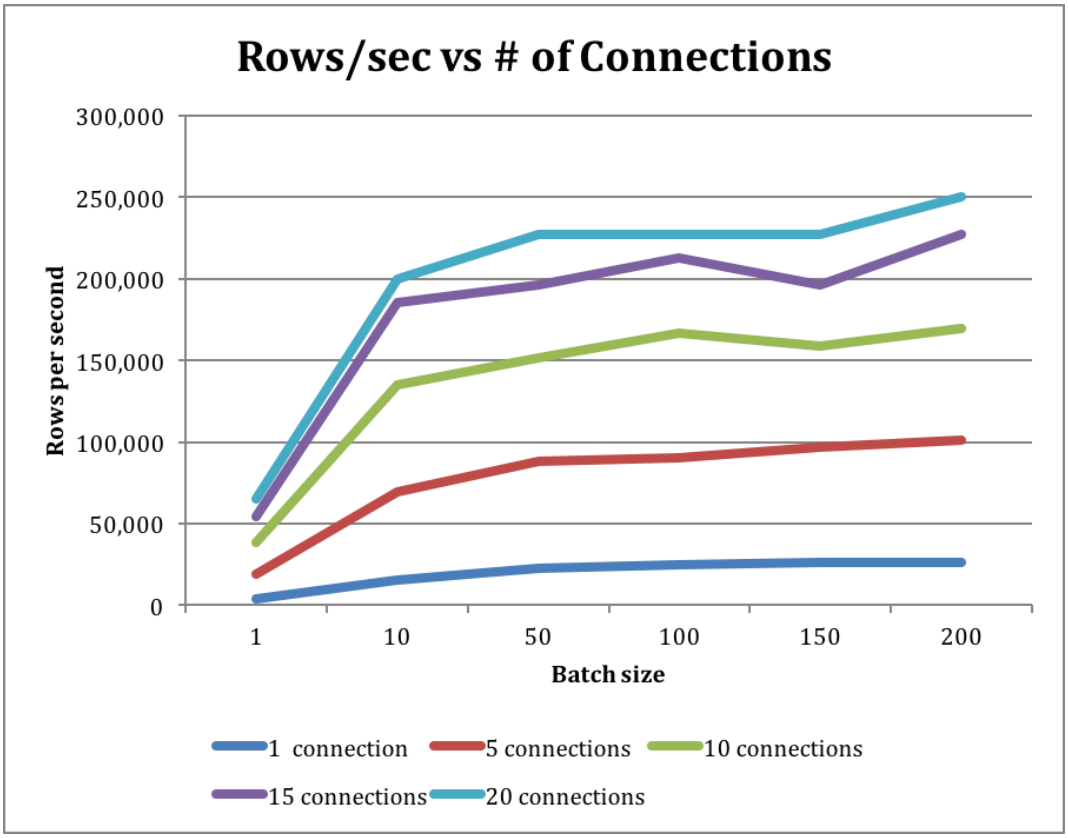
The following graph shows how batch size affects performance when data is inserted using a single connection:



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.

Parent topic: Trickle Loading Data via JDBC

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.

```
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 SQLExcepti
        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 [option
            System.err.println("Where: options can be:");
            System.err.printf("       [loaders=NN] # of loaders (default: %d)\n", opt
            System.err.printf("       [batchsize=NN] size of batches (default: %d)\n
            System.err.printf("       [transactionsize=NN] size of transactions (defa
            System.err.printf("       [insertions=NN] # of insertions (default: %d r
            System.err.printf("       [driver=postgresql|pgsql] driver (default: %s)
            System.err.printf("       [table=tablename] driver (default: %s)\n", opti
            System.err.printf("       [table-per-load=true|false] unique table per l
            System.err.printf("       [drop-table=true|false] drop table before load
```

```
            System.err.printf("       [unlogged=true|false] create table UNLOGGED (de
            System.err.println("  or any jdbc property for the jdbc driver");
            System.err.println();
            System.err.println("Example: Load yb11 5432 yellowbrick yellowbrick yello
            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 connec
            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
    }

    private void run() throws Throwable {
```

```
        System.out.printf("Connecting to database %s with parameters %s\n", mUrl, mP
        System.out.printf("Using %d loader threads with batch size %d, transaction s

        // 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 ExecutorC
            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)
                }

                // 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.SECO
                            if (task == null) {
                                report();
                            } else {
                                Object loader = task.get();
                                System.out.printf("\nReceived completion for 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.toM
            }

        } finally {
            executor.shutdown();
        }

        System.out.println("Done");
    }
}

private void prepareTable(Connection connection, String tableName) throws SQLExc

    // 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 VARC
    }
}

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", mOpti
            result.put("table", tableName);

            // Unique table for the loader?
            if (mOptions.mTablePerLoad) {
                prepareTable(connection, tableName);
            }

            // Create insertion statement.
            try (PreparedStatement preparedStatement = connection.prepareStatement(St

                // 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.curren
                        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
        }
    }

    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
}

private Connection connect() throws SQLException {
    return this.mDriver.connect(mUrl, mProperties);
}
}
```

Parent topic: Trickle Loading Data via JDBC

Loading Tables with Spark

This section describes how to bulk load data from source files that `ybload` cannot read directly, such as Avro and Parquet data. 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.

In This Section

- Setting Up the ybrelay Service
- Setting up and Running a Spark Job
- ybrelay Options
- Spark and ybrelay Glossary

Parent topic: Loading and Unloading Data

Setting Up the ybrelay Service

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.

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.

In This Section

- Running the ybrelay-init Script
- TLS Support for ybrelay and Spark
- Creating a Java Keystore
- Starting the ybrelay Service Manually

Parent topic: Loading Tables with Spark

Running the ybrelay-init Script

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
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: disa
   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/jre1.8.0_91/bin/java -Dvisualvm.display.name=YB:Relay -X
/opt/ybtools/etc:/opt/ybtools/lib/relay-service-3.2.0-SNAPSHOT.jar:/opt/ybtools/lib/

Nov 14 18:31:23 rhel7.localdomain systemd[1]: Started Yellowbrick Data Load Relay Se
Nov 14 18:31:23 rhel7.localdomain ybrelay[3320]: Warning; java is not in your PATH.
/opt/java/jre1.8.0_91/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/s
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: disa
   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.

Parent topic: Setting Up the ybrelay Service

TLS Support for ybrelay and Spark

Important:

TLS support for ybrelay and Spark is a tech preview feature in Version 5.1.

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.

Parent topic: Setting Up the ybrelay Service

Creating a Java Keystore

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 appliances 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 trust store.

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
```

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

Parent topic: Setting Up the ybrelay Service

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/jre1.8.0_91”
```

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
```

```
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: disa
   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/jre1.8.0_91/bin/java -Dvisualvm.display.name=YB:Relay ->
```

```
/opt/ybtools/etc:/opt/ybtools/lib/relay-service-3.2.0-SNAPSHOT.jar:/opt/ybtools/lib/

Nov 14 18:40:33 rhel7.localdomain systemd[1]: Started Yellowbrick Data Load Relay Se
Nov 14 18:40:33 rhel7.localdomain ybrelay[3196]: Warning; java is not in your PATH.
/opt/java/jre1.8.0_91/bin.
```

Parent topic: Setting Up the ybrelay Service

Setting up and Running a Spark Job

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)
```

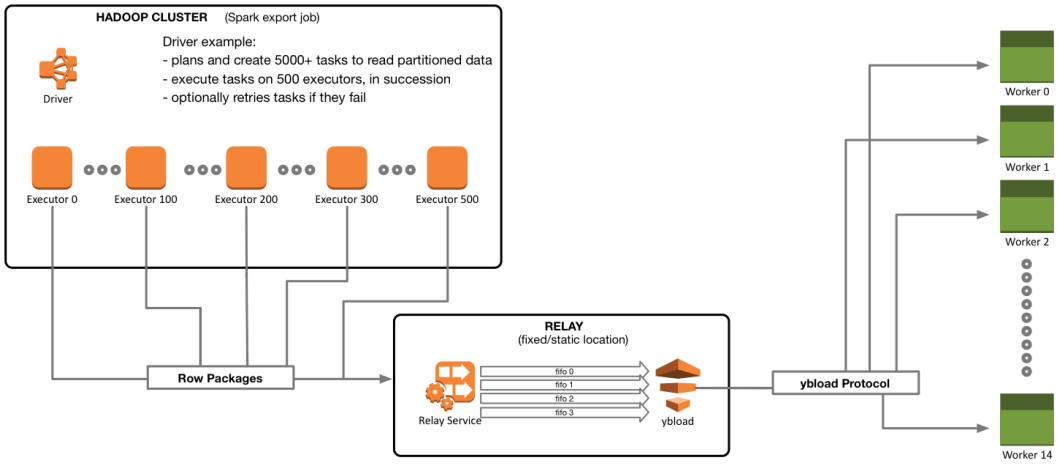
The driver executes first:

- *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
```

In This Section

- Spark Application Options
- Spark Format for Reading Schemas
- Error Tolerance
- Examples of Spark Jobs
- Output for a Spark Job

Parent topic: Loading Tables with Spark

Spark Application Options

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 Enabling and 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 appliance.

--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
--post-sql "update job_log set finish_time = current_timestamp where transacti
```

--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 Enabling and 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.

Parent topic: Setting up and Running a Spark Job

Spark Format for Reading Schemas

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.

Parent topic: Setting up and Running a Spark Job

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.

Parent topic: Setting up and Running a Spark Job

Examples of Spark Jobs

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"
}]
```

Parent topic: Setting up and Running a Spark Job

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-1564598030511
"-t" "people2"
"--field-names" "playerid,birthyear,birthmonth,birthday,birthcountry,birthstate,birt
"--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/java-1.8.0-openjdk-1.8.0.212.b04-0.el7_6.x8
   java.version         = "1.8.0_212"
   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/201
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" sta
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 encod
2019-07-31 11:33:56.461 [DEBUG] <pool-1-thread-5>  BgTask "Determining actual source
```

```
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" start
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
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
2019-07-31 11:33:56.726 [DEBUG] <ybload db init>  getTableSchema(input): "agoade_de
2019-07-31 11:33:56.726 [ INFO] <Choosing apx source encoding>  Assuming source enc
2019-07-31 11:33:56.726 [DEBUG] <Choosing apx source encoding>  BgTask "Choosing ap
2019-07-31 11:33:56.748 [DEBUG] <ybload db init>  getTableSchema(normalized): "agoa
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" st
2019-07-31 11:33:56.926 [DEBUG] <pool-1-thread-16>  BgTask "Wait for Sequencers" st
2019-07-31 11:33:58.753 [DEBUG] <PreReader #2>  Starting pre-reading of /var/lib/yb
2019-07-31 11:33:58.754 [DEBUG] <PreReader #1>  Starting pre-reading of /var/lib/yb
2019-07-31 11:33:58.754 [DEBUG] <PreReader #0>  Starting pre-reading of /var/lib/yb
2019-07-31 11:33:58.754 [DEBUG] <PreReader #3>  Starting pre-reading of /var/lib/yb
2019-07-31 11:34:00.591 [DEBUG] <Determining actual source encoding>  BgTask "Deter
2019-07-31 11:34:00.591 [DEBUG] <TranscodeContext>  BgTask "TranscodeContext" exite
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 separat
   --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"."
   "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 = CWL72YSdnTskgHuL2
2019-07-31 11:34:01.331 [DEBUG] <BulkSession init>  Sending BulkService.init(#colum
```

```
2019-07-31 11:34:01.596 [DEBUG] <BulkSession init>  Received BulkService.init() resp
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
2019-07-31 11:34:01.617 [DEBUG] <main>  Running with 1.2GB of memory. Expandable to
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" sta
2019-07-31 11:34:02.044 [DEBUG] <BgTask idle thread>  BgTask "segment reader #0" sta
2019-07-31 11:34:02.044 [DEBUG] <BgTask idle thread>  BgTask "segment reader #2" sta
2019-07-31 11:34:02.044 [DEBUG] <BgTask idle thread>  BgTask "segment reader #3" sta
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) -- Openi
2019-07-31 11:34:02.955 [DEBUG] <sender 1.03>  Worker(/10.34.205.10:31273) -- Openi
2019-07-31 11:34:02.955 [DEBUG] <sender 1.04>  Worker(/10.34.205.10:31274) -- Openi
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) -- Openi
2019-07-31 11:34:02.955 [DEBUG] <sender 1.00>  Worker(/10.34.205.10:31270) -- Openi
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) -- Openi
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) -- Openi
2019-07-31 11:34:02.961 [DEBUG] <sender 1.08>  Worker(/10.34.205.10:31278) -- Openi
2019-07-31 11:34:02.960 [DEBUG] <sender 1.09>  Worker(/10.34.205.10:31279) -- Openi
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) -- Openi
2019-07-31 11:34:02.959 [DEBUG] <sender 1.11>  Worker(/10.34.205.10:31281) -- Openi
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) -- Openi
2019-07-31 11:34:02.963 [DEBUG] <sender 1.14>  Worker(/10.34.205.10:31284) -- Openi
2019-07-31 11:34:02.963 [DEBUG] <sender 1.06>  Worker(/10.34.205.10:31276) -- Openi
2019-07-31 11:34:02.965 [DEBUG] <sender 1.12>  Worker(/10.34.205.10:31282) -- Openi
2019-07-31 11:34:02.969 [DEBUG] <sender 1.04>  Worker(/10.34.205.10:31274) -- Opene
2019-07-31 11:34:02.969 [DEBUG] <sender 1.01>  Worker(/10.34.205.10:31271) -- Opene
2019-07-31 11:34:02.969 [DEBUG] <sender 1.09>  Worker(/10.34.205.10:31279) -- Opene
```

```
2019-07-31 11:34:02.969 [DEBUG] <sender 1.08>  Worker(/10.34.205.10:31278) -- Opened
2019-07-31 11:34:02.969 [DEBUG] <sender 1.10>  Worker(/10.34.205.10:31280) -- Opened
2019-07-31 11:34:02.969 [DEBUG] <sender 1.02>  Worker(/10.34.205.10:31272) -- Opened
2019-07-31 11:34:02.969 [DEBUG] <sender 1.03>  Worker(/10.34.205.10:31273) -- Opened
2019-07-31 11:34:02.969 [DEBUG] <sender 1.06>  Worker(/10.34.205.10:31276) -- Opened
2019-07-31 11:34:02.969 [DEBUG] <sender 1.14>  Worker(/10.34.205.10:31284) -- Opened
2019-07-31 11:34:02.969 [DEBUG] <sender 1.12>  Worker(/10.34.205.10:31282) -- Opened
2019-07-31 11:34:02.969 [DEBUG] <sender 1.05>  Worker(/10.34.205.10:31275) -- Opened
2019-07-31 11:34:02.969 [DEBUG] <sender 1.00>  Worker(/10.34.205.10:31270) -- Opened
2019-07-31 11:34:02.969 [DEBUG] <sender 1.11>  Worker(/10.34.205.10:31281) -- Opened
2019-07-31 11:34:02.969 [DEBUG] <sender 1.07>  Worker(/10.34.205.10:31277) -- Opened
2019-07-31 11:34:02.969 [DEBUG] <sender 1.13>  Worker(/10.34.205.10:31283) -- Opened
2019-07-31 11:34:02.972 [DEBUG] <sender 1.03>  Worker(/10.34.205.10:31273) -- Sent H
2019-07-31 11:34:02.972 [DEBUG] <sender 1.09>  Worker(/10.34.205.10:31279) -- Sent H
2019-07-31 11:34:02.972 [DEBUG] <sender 1.00>  Worker(/10.34.205.10:31270) -- Sent H
2019-07-31 11:34:02.972 [DEBUG] <sender 1.12>  Worker(/10.34.205.10:31282) -- Sent H
2019-07-31 11:34:02.972 [DEBUG] <sender 1.07>  Worker(/10.34.205.10:31277) -- Sent H
2019-07-31 11:34:02.972 [DEBUG] <sender 1.06>  Worker(/10.34.205.10:31276) -- Sent H
2019-07-31 11:34:02.972 [DEBUG] <sender 1.01>  Worker(/10.34.205.10:31271) -- Sent H
2019-07-31 11:34:02.972 [DEBUG] <sender 1.11>  Worker(/10.34.205.10:31281) -- Sent H
2019-07-31 11:34:02.972 [DEBUG] <sender 1.02>  Worker(/10.34.205.10:31272) -- Sent H
2019-07-31 11:34:02.972 [DEBUG] <sender 1.04>  Worker(/10.34.205.10:31274) -- Sent H
2019-07-31 11:34:02.972 [DEBUG] <sender 1.14>  Worker(/10.34.205.10:31284) -- Sent H
2019-07-31 11:34:02.972 [DEBUG] <sender 1.13>  Worker(/10.34.205.10:31283) -- Sent H
2019-07-31 11:34:02.972 [DEBUG] <sender 1.05>  Worker(/10.34.205.10:31275) -- Sent H
2019-07-31 11:34:02.972 [DEBUG] <sender 1.08>  Worker(/10.34.205.10:31278) -- Sent H
2019-07-31 11:34:02.972 [DEBUG] <sender 1.10>  Worker(/10.34.205.10:31280) -- Sent H
2019-07-31 11:34:03.896 [ INFO] <PreReader #3>  Skipping empty source '/var/lib/ybre
2019-07-31 11:34:03.896 [ INFO] <PreReader #0>  Skipping empty source '/var/lib/ybre
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 sequencingQueu
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 sequencingQueu
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 I
2019-07-31 11:34:04.237 [DEBUG] <PreReader #1>  Setting noMoreData on sequencingQueu
2019-07-31 11:34:04.237 [DEBUG] <segment reader #0>  Segment3.0: starting 2 bundle I
2019-07-31 11:34:04.237 [DEBUG] <PreReader #2>  Setting noMoreData on sequencingQueu
```

```
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. Cal
2019-07-31 11:34:04.238 [DEBUG] <Wait for PreReaders>  Setting noMoreData on postRe
2019-07-31 11:34:04.239 [DEBUG] <Wait for PreReaders>  BgTask "Wait for PreReaders"
2019-07-31 11:34:04.239 [DEBUG] <BgTask idle thread>  BgTask "segment parser #1.0" s
2019-07-31 11:34:04.239 [DEBUG] <segment reader #2>  BgTask "segment reader #2" exi
2019-07-31 11:34:04.239 [DEBUG] <BgTask idle thread>  BgTask "segment parser #1.1" s
2019-07-31 11:34:04.239 [DEBUG] <BgTask idle thread>  BgTask "segment parser #0.1" s
2019-07-31 11:34:04.239 [DEBUG] <segment reader #3>  BgTask "segment reader #3" exi
2019-07-31 11:34:04.239 [DEBUG] <BgTask idle thread>  BgTask "segment parser #0.0" s
2019-07-31 11:34:04.311 [DEBUG] <segment reader #0>  Segment3.0: finalizing audit w
2019-07-31 11:34:04.311 [DEBUG] <segment reader #0>  Setting noMoreData on slurpedB
2019-07-31 11:34:04.376 [DEBUG] <segment parser #0.1>  BgTask "segment parser #0.1"
2019-07-31 11:34:04.391 [DEBUG] <segment parser #0.0>  BgTask "segment parser #0.0"
2019-07-31 11:34:04.391 [DEBUG] <segment reader #0>  Segment3.0: all bundle lanes e
2019-07-31 11:34:04.391 [DEBUG] <segment reader #0>  Segment3.0: mustAdoptNext=null
2019-07-31 11:34:04.391 [DEBUG] <segment reader #0>  BgTask "segment reader #0" exi
2019-07-31 11:34:04.391 [DEBUG] <Sequencer #2>  Segment3.0: sequenced. Submitting t
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
2019-07-31 11:34:04.435 [DEBUG] <segment reader #1>  Setting noMoreData on slurpedB
2019-07-31 11:34:04.451 [DEBUG] <segment parser #1.0>  BgTask "segment parser #1.0"
2019-07-31 11:34:04.475 [DEBUG] <segment parser #1.1>  BgTask "segment parser #1.1"
2019-07-31 11:34:04.475 [DEBUG] <segment reader #1>  Segment0.0: all bundle lanes e
2019-07-31 11:34:04.476 [DEBUG] <segment reader #1>  Segment0.0: mustAdoptNext=null
2019-07-31 11:34:04.476 [DEBUG] <Sequencer #1>  Segment0.0: sequenced. Submitting t
2019-07-31 11:34:04.476 [DEBUG] <segment reader #1>  BgTask "segment reader #1" exi
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. Cal
2019-07-31 11:34:04.477 [DEBUG] <Wait for Sequencers>  Setting noMoreData on preFin
2019-07-31 11:34:04.477 [DEBUG] <Wait for Sequencers>  BgTask "Wait for Sequencers"
2019-07-31 11:34:04.477 [ INFO] <main>  Flushing last 15981 rows (of 19617 total) f
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 cl
2019-07-31 11:34:04.481 [DEBUG] <sender 1.12>  Declaring success (maybe partial) be
2019-07-31 11:34:04.484 [DEBUG] <sender 1.06>  Worker(/10.34.205.10:31276) -- Sent
2019-07-31 11:34:04.484 [DEBUG] <sender 1.03>  Worker(/10.34.205.10:31273) -- Sent
2019-07-31 11:34:04.484 [DEBUG] <sender 1.13>  Worker(/10.34.205.10:31283) -- Sent
2019-07-31 11:34:04.484 [DEBUG] <sender 1.04>  Worker(/10.34.205.10:31274) -- Sent
2019-07-31 11:34:04.485 [DEBUG] <sender 1.04>  Worker(/10.34.205.10:31274) -- Flush
2019-07-31 11:34:04.484 [DEBUG] <sender 1.05>  Worker(/10.34.205.10:31275) -- Sent
2019-07-31 11:34:04.484 [DEBUG] <sender 1.10>  Worker(/10.34.205.10:31280) -- Sent
```

```
2019-07-31 11:34:04.484 [DEBUG] <sender 1.08>  Worker(/10.34.205.10:31278) -- Sent E
2019-07-31 11:34:04.485 [DEBUG] <sender 1.08>  Worker(/10.34.205.10:31278) -- Flushi
2019-07-31 11:34:04.484 [DEBUG] <sender 1.09>  Worker(/10.34.205.10:31279) -- Sent E
2019-07-31 11:34:04.484 [DEBUG] <sender 1.14>  Worker(/10.34.205.10:31284) -- Sent E
2019-07-31 11:34:04.484 [DEBUG] <sender 1.01>  Worker(/10.34.205.10:31271) -- Sent E
2019-07-31 11:34:04.486 [DEBUG] <sender 1.01>  Worker(/10.34.205.10:31271) -- Flushi
2019-07-31 11:34:04.484 [DEBUG] <sender 1.07>  Worker(/10.34.205.10:31277) -- Sent E
2019-07-31 11:34:04.484 [DEBUG] <sender 1.12>  Worker(/10.34.205.10:31282) -- Sent E
2019-07-31 11:34:04.484 [DEBUG] <sender 1.11>  Worker(/10.34.205.10:31281) -- Sent E
2019-07-31 11:34:04.484 [DEBUG] <sender 1.02>  Worker(/10.34.205.10:31272) -- Sent E
2019-07-31 11:34:04.486 [DEBUG] <sender 1.02>  Worker(/10.34.205.10:31272) -- Flushi
2019-07-31 11:34:04.484 [DEBUG] <sender 1.00>  Worker(/10.34.205.10:31270) -- Sent E
2019-07-31 11:34:04.486 [DEBUG] <sender 1.11>  Worker(/10.34.205.10:31281) -- Flushi
2019-07-31 11:34:04.486 [DEBUG] <sender 1.12>  Worker(/10.34.205.10:31282) -- Flushi
2019-07-31 11:34:04.486 [DEBUG] <sender 1.07>  Worker(/10.34.205.10:31277) -- Flushi
2019-07-31 11:34:04.485 [DEBUG] <sender 1.14>  Worker(/10.34.205.10:31284) -- Flushi
2019-07-31 11:34:04.485 [DEBUG] <sender 1.09>  Worker(/10.34.205.10:31279) -- Flushi
2019-07-31 11:34:04.485 [DEBUG] <sender 1.10>  Worker(/10.34.205.10:31280) -- Flushi
2019-07-31 11:34:04.485 [DEBUG] <sender 1.05>  Worker(/10.34.205.10:31275) -- Flushi
2019-07-31 11:34:04.484 [DEBUG] <sender 1.13>  Worker(/10.34.205.10:31283) -- Flushi
2019-07-31 11:34:04.484 [DEBUG] <sender 1.03>  Worker(/10.34.205.10:31273) -- Flushi
2019-07-31 11:34:04.484 [DEBUG] <sender 1.06>  Worker(/10.34.205.10:31276) -- Flushi
2019-07-31 11:34:04.486 [DEBUG] <sender 1.00>  Worker(/10.34.205.10:31270) -- Flushi
2019-07-31 11:34:04.489 [DEBUG] <sender 1.01>  Worker(/10.34.205.10:31271) -- Flushe
2019-07-31 11:34:04.489 [DEBUG] <sender 1.05>  Worker(/10.34.205.10:31275) -- Flushe
2019-07-31 11:34:04.489 [DEBUG] <sender 1.03>  Worker(/10.34.205.10:31273) -- Flushe
2019-07-31 11:34:04.489 [DEBUG] <sender 1.14>  Worker(/10.34.205.10:31284) -- Flushe
2019-07-31 11:34:04.489 [DEBUG] <sender 1.13>  Worker(/10.34.205.10:31283) -- Flushe
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) -- Flushe
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) -- Flushe
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) -- Flushe
2019-07-31 11:34:04.489 [DEBUG] <sender 1.04>  Worker(/10.34.205.10:31274) -- Flushe
2019-07-31 11:34:04.490 [DEBUG] <sender 1.00>  Worker(/10.34.205.10:31270) -- Flushe
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) -- Flushe
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) -- Flushe
2019-07-31 11:34:04.489 [DEBUG] <sender 1.08>  Worker(/10.34.205.10:31278) -- Flushe
2019-07-31 11:34:04.489 [DEBUG] <sender 1.09>  Worker(/10.34.205.10:31279) -- Flushe
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) -- Flushe
```

```
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 fo
2019-07-31 11:34:06.640 [DEBUG] <main>  Received BulkService.commitTransaction() res
2019-07-31 11:34:06.641 [ INFO] <main>  Committed transaction #1 after a total of 34
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(
2019-07-31 11:34:06.691 [ INFO] <main>  SUCCESSFUL BULK LOAD: Loaded 19617 good rows
2019-07-31 11:34:06.691 [DEBUG] <main>  SessionNotifier.close() calling ntsStopAndRe
2019-07-31 11:34:06.697 [DEBUG] <main>  Sending BulkService.done(success=true, clien
2019-07-31 11:34:06.705 [DEBUG] <main>  Received BulkService.done() response for pen
2019-07-31 11:34:06.709 [DEBUG] <main>  All top-level threads have stopped
```

Parent topic: Setting up and Running a Spark Job

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-8-oracle/jre"
   java.version         = "1.8.0_101"
   jvm.memory           = "491.00 MB (max=7.11 GB)"
   jvm.name_and_version = "Java HotSpot(TM) 64-Bit Server VM (build 25.101-b13, mix
   jvm.options          = "-Xms512m, -Xmx8g, -ea, -Dapp.name=ybrelay, -Dapp.pid=114
   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 Enabling and 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 appliance.

-?, --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 Enabling and 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 key store 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 in the `spark-submit` command.

--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
```

Parent topic: Loading Tables with Spark

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.

Parent topic: Loading Tables with Spark

Unloading Data

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.

Download and install the unload client (as part of the ybtools package) from the SMC 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 appliance so that your client and server versions correspond. The client tools are backward-compatible but not always forward-compatible.

In This Section

- Setting Up an Unload
- Saving Unload Options to a File
- Unloading Data to an S3 Bucket
- ybunload Command

Parent topic: Loading and Unloading Data

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 format of the files to be unloaded.

- **Destination**: A local file system where the client user has write permissions or an AWS S3 bucket where the client user has credentials to upload files.
- **Format**: CSV (comma-delimited by default) or text (tab-delimited by default) Compressed GZIP (`.gz`) output files are supported.

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 -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. For details, see ybunload Options or the online help text.
- `table`: the target table for the unload. Alternatively, you can unload the results of a query, using the `-s` (select) option instead of `-t`.
- `destination`: a local file system path or an S3 bucket. You can also unload to `stdout`. You can specify a prefix for the output files so that they are named consistently and uniquely for a given unload. For more details, see ybunload Options 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

Parent topic: Unloading Data

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 1.2.2-5653
16:24:16.211 [ INFO] COMMAND LINE = @/home/ybdata/myoptions.txt -W --truncate-existi
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 writ
```

Parent topic: Unloading Data

Unloading Data to an S3 Bucket

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; Re
```

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_unloa
```

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.
```

Note: The log file, if specified, 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.

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 abort the upload. For example:

```
$ ybunload -d premdb -t newmatchstats --username bobr -W -o s3://yb-tmp/premdb/prem
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 -
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_1wLK8RKYPDg
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.RGhm
...
[1]+  Stopped                 ybunload -d premdb -t newmatchstats --username bobr -
$ ybunload -d premdb --username bobr -W -o s3://yb-tmp/premdb/premdb_unloads --canc
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/pre
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.
$
```

Parent topic: Unloading Data

ybunload Command

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 or S3)
- 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
```

In This Section

- ybunload Options
- ybunload Output Files
- ybunload Examples

Parent topic: Unloading Data

ybunload Options

This section describes the options you can use in a `ybunload` command.

@file

Specify a file that includes a set of options and values to use for the unload. See Saving Unload Options to a File. You can use this option and the `--select-file` option in the same `ybunload` command.

--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 Enabling and Verifying SSL/TLS Encryption.

--cancel-hung-uploads

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 -
```

Warning: Any other uploads to the specified bucket may be cancelled 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.

--compress, -c NONE | GZIP_FAST | GZIP | GZIP_MORE | GZIP_BEST | GZIP_STREAM_FAST | GZIP_STREAM | GZIP_STREAM_MORE | GZIP_STREAM_BEST

The default is `NONE`. The other 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.

--dbname, -d

Name of the source database. The default is `yellowbrick`. See also Setting up a Database Connection.

--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-trust, -k

Disable SSL/TLS trust when using secured communications. Trust is enabled by default. See also Enabling and 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 appliance.

--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

Select the output format to use: `text` (tab-delimited by default) or `csv` (comma-delimited by default). The default format is `csv` and creates files with a `.csv` extension. Text format creates files with a `.txt` extension.

-?, --help

Return basic usage information for the `ybunload` command and its options.

--help-advanced

Return more advanced usage information for the `ybunload` command and its options.

-h, --host

Host name. See Setting up a Database Connection.

--initial-connection-timeout NUMBER

Specify the number of seconds to wait for initial connections to the database. The minimum value is 0, which means wait forever. The default is `120`.

--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`).

--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 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 for the unload operation. 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.

--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 alre
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.

--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 worker. 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 worker, 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.

-p, --port

Port number. See Setting up a Database Connection.

--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 worker, and a single worker may provide multiple streams.
- `partnumber_` : An incrementing number starting from `1` for each stream.
- `.extension` : file type, such as `.csv` or `.gz` .

-q, --quiet

Do not write any output to console. This option is suitable for `cron` invocations. If `--quiet` is specified, you must also specify `--logfile` .

--quote UNICODE_CHARACTER

Specify the character to use for quoting.

--quote-escape UNICODE_CHARACTER

Specify the character to use for escaping quotes.

--secured

Use SSL/TLS to secure all communications. The default is not secured. See also Enabling and Verifying SSL/TLS Encryption.

--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`).

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.

-U, --username

Database username. See Setting up a Database Connection.

--version

Display the version of `ybunload` you are running (as part of `ybtools`). This option is not intended to be combined with other options. For example:

```
$ ybunload --version
ybunload version 1.2.2-5653
```

-W, --password

Database user's password. See Setting up a Database Connection.

Parent topic: ybunload Command

ybunload Output Files

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 worker, and a single worker 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 unzip them. If necessary, you can use the `gunzip` command with the `-c` option to unzip 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 worker. Each worker unloads a separate data stream.
- The plan that is generated for the unload query and how many worker 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 versus uncompressed files will differ, but the number is the same. Block mode compression (or no compression) results in one file per worker node that has data at the top of the plan tree. If a worker 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 worker 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-worker queries: Only one worker executes the plan, so only as many files as there are cores on that worker 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 worker 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 worker, 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 worker 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-blade appliance 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.

Parent topic: ybunload Command

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 te
   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
   jvm.options          = -Xms512m, -Xmx6g, -XX:+UseG1GC, -Dapp.name=ybunload, -Dapp
   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\_uni
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.4rbHB3NL7ZLXtdC
12:56:39.357 [ INFO] state: FINALIZING - (Open Sockets:   0 - Open Files   0 Other C
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 writ
```

Unload the results of a query to a 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\_u
"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
```

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
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
```

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 -
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
```

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
```

Parent topic: ybunload Command

Workload Management

This section describes how to make the best use of system resources by managing workloads.

In This Section

- Sharing System Resources
- How Queries Are Executed
- How WLM Works
- A Simple WLM Example

Parent topic: Yellowbrick Documentation

Sharing System Resources

Like any other database system, a Yellowbrick appliance has finite resources, but these 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 superuser 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 the System Management Console or by using SQL commands.

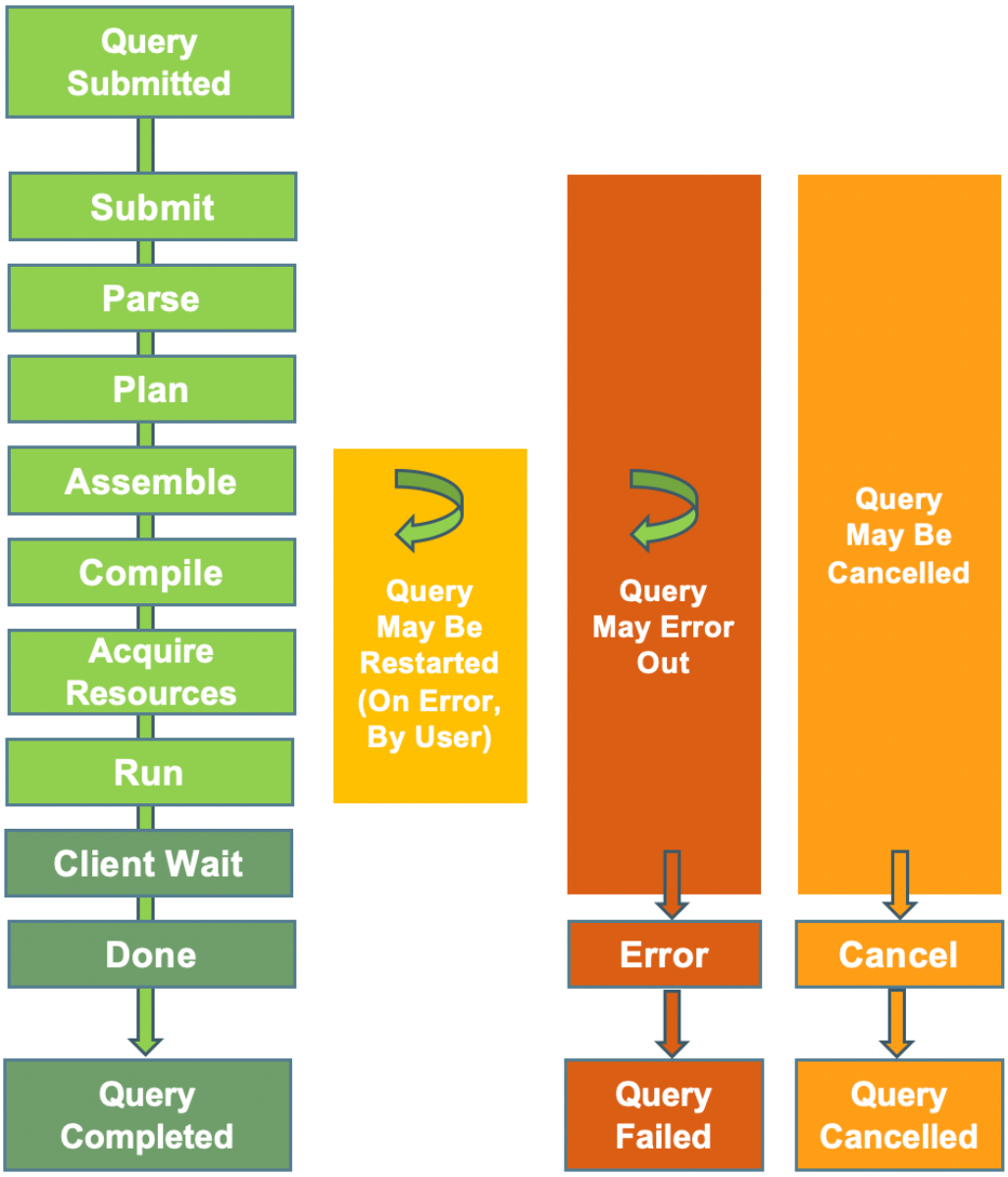
Parent topic: Workload Management

How Queries Are Executed

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 on the manager node, while it is being prepared for execution, then it starts running (executing) on the worker nodes.



The diagram also identifies when queries can be cancelled or restarted. Once submitted, a query runs to completion, is cancelled, 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 cancelled, it cannot be restarted.)

A query has the following states:

Submit

The query is received by the front-end database on the manager node 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. You cannot apply rules to queries in this state. 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`.

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 worker 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 cancelled 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.

A number of 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.

Parent topic: Workload Management

How WLM Works

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 (and applies to all databases that you create on a single system). 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. For example, 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.

For more details, see Creating WLM Profiles.

Resource Pools

Every profile is associated with a number of *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.

All profiles inherit a fixed `system` pool that you cannot edit or remove. The presence of this pool guarantees that background system operations and SMC 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.
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.

In This Section

- Creating WLM Profiles
- Creating WLM Resource Pools

- Creating WLM Rules
- Throttling Concurrency
- Moving Queries
- Restarting Queries
- Using Query Tags
- Exporting and Importing Profiles
- Monitoring WLM Activity

Parent topic: Workload Management

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 the SMC or by using a SQL command:

- Go to **Manage** > **Workloads** > **Create Profile** in the SMC.

In the SMC, you can create a new blank profile, which contains the `system` pools only, or you can create a profile as a copy of another profile, which contains copies of all of the resource pools in the base profile (copied pools are named `profile: pool` in the SMC).

- Use the SQL command CREATE WLM PROFILE.

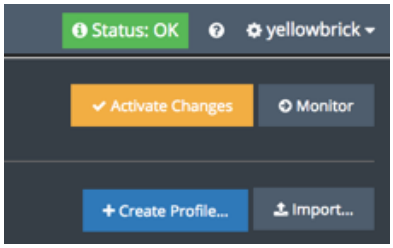
When you create any new profile, it contains the `system` resource pools by default.

To see a list of profiles that are currently in the system, either query the sys.wlm_active_profile view or use the `ybsql \dwp` command.

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 Changes** button in the SMC, which becomes available on the **Profiles/Resource Pools** screen when any changes are 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.

If you are activating a new or different profile, you can also use the ALTER WLM PROFILE command.

Note: When you activate a profile, the currently active profile is deactivated. There is no specific option or command that deactivates a profile.

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 the Yellowbrick database is installed, it contains a `flex` profile, a `default` profile, and a `maintenance` profile. The `default` profile remains active until you activate another profile. You cannot modify or remove the `default` 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.

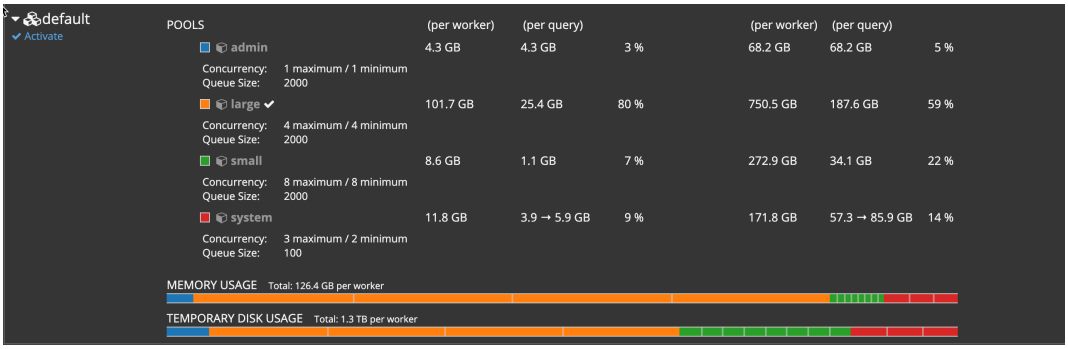
---

default Profile

This profile contains four resource pools, which you can monitor when the profile is active. Like the profile itself, these pools cannot be changed. The actual memory allocation varies by system. The following examples show actual memory and temporary space allocation on a 124GB, single-chassis, 15-blade system.

Name of Pool	Memory %	Slots (max/min)	Queue Size
`admin`	3%	1/1	2000
`large` (default pool)	80%	4/4	2000
`small`	7%	8/8	2000

Name of Pool	Memory %	Slots (max/min)	Queue Size
`system`	9%	3/2	100

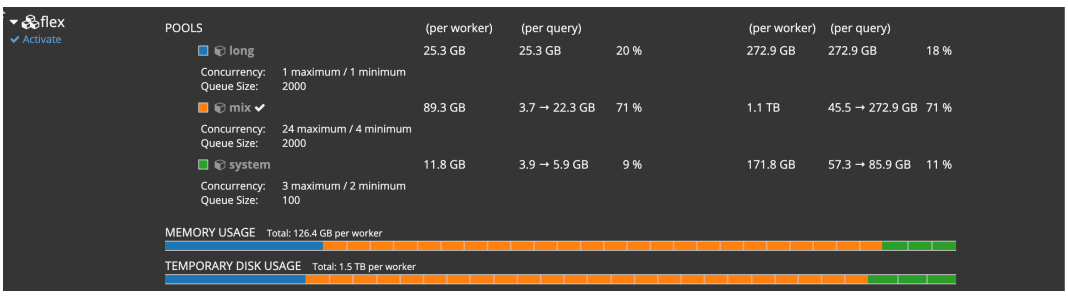


---

flex Profile

This profile contains three resource pools, which you can monitor when the profile is active. Like the profile itself, these pools cannot be changed. This pool is a template that you can use as an example of a flexible profile with a flex pool called `mix`.

Name of Pool	Memory %	Slots (max/min)	Queue Size
`long`	20%	1/1	2000
`mix` (default pool)	71%	24/4	2000
`system`	9%	3/2	100

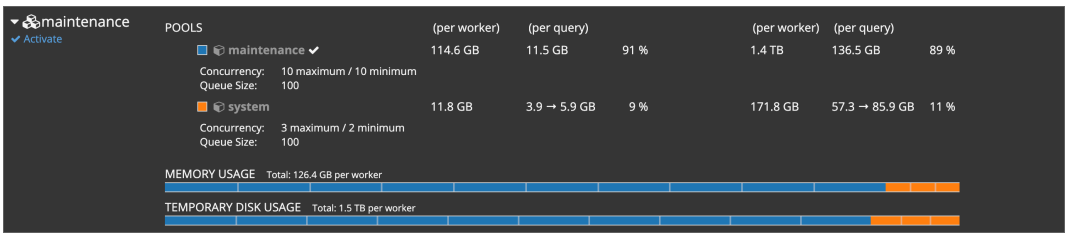


maintenance Profile

The `maintenance` profile contains only two resource pools, which can be monitored when the profile is active. Like the profile itself, these pools cannot be changed.

Name of Pool	Memory %	Slots (max/min)	Queue Size
`maintenance` (default pool)	91%	10/10	100
`system`	9%	3/2	100

The following example shows actual memory and temporary space allocation on a dual-chassis, 30-blade system:



WLM Maintenance Mode

Administrators can put the Yellowbrick system in *maintenance mode* by running the following command:

```
alter wlm profile "maintenance" activate;
```

All connected, non-superuser sessions 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` fatal error message. Any superuser can turn maintenance mode off by switching to another profile. For example:

```
alter wlm profile "default" activate;
```

Parent topic: How WLM Works

Creating WLM Resource Pools

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 the following resource pools:

- A `system` pool, which is created by default; you cannot remove or modify this pool.
- A user-defined pool marked as the `default` pool

You can add any number of user-defined pools to a profile, as long as you mark one of them as the `default` pool.

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 cancelled

To create a resource pool:

- Go to **Manage** > **Workloads** > **Create Resource Pool** in the SMC. **Create Resource Pool** is an action that you can select below each user-defined profile.
- Alternatively, use the SQL command CREATE WLM RESOURCE POOL.

To see a list of resource pools that are currently in the system, go to **Manage** > **Workloads** in the SMC, query the sys.wlm_pending_pool view, or use the `ybsql \dwrp` command, as shown here:

```
yellowbrick=# \dwrp
                                                                List of WLM resourc
   Name     | Max Concurrency | Min Concurrency | Queue size | Wait limit | Row limi
------------+-----------------+-----------------+------------+------------+-------
 admin      |               2 |                 |       2000 |            |
 large      |               4 |                 |       2000 |            |
 long       |               1 |                 |       2000 |            |
 maintenance |              10 |                 |        100 |            |
 mix        |              24 |               4 |       2000 |            |
 small      |               8 |                 |       2000 |            |
 system     |               3 |               2 |        100 |            |
(7 rows)
```

See also Creating WLM Profiles.

In This Section

- Allocating Query Memory
- Example: Memory Allocation Among Resource Pools
- Examples of Flex Pools

Parent topic: How WLM Works

Allocating Query Memory

The total memory available on each active worker node of the cluster is divided among active WLM resource pools and a flexible number of query slots within those pools. (If you are not sure how much memory is on your system, start an SMC session and go to **Monitor** > **Memory**.) 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: 1GB for pools with spilling enabled and 512MB for pools with spilling disabled.
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).

See Example: Memory Allocation Among Resource Pools to see how these policies are applied, and how adding pools to a profile changes the existing memory allocation. 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 worker node. For example, assume you have a 15-node cluster with 59GB 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 14.7GB 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.

By default, 20% of raw system capacity is reserved for spilling. You can change this percentage in the SMC by going to **Configure** > **Settings** > **Storage**. (Note that changing the spill space cancels all running queries and reconfigures the database. The database will be unavailable during this operation.)

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 SMC will adjust the memory configuration 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, use the **Query Details** screen in the SMC. You can see where queries spill (and how much data they spill) because the plan nodes are marked with a warning. You can also query the sys.log_query view, which has three spilling-related columns:

- `io_spill_write_bytes`: data written to temporary space because of spilling.

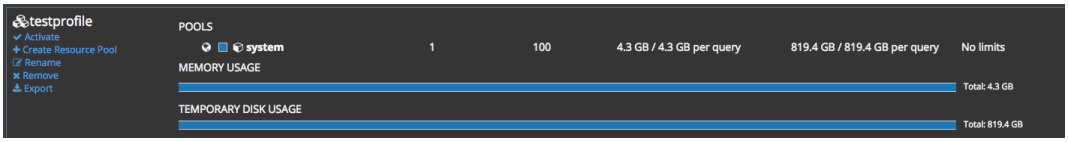
- `io_spill_read_bytes`: data read back from the spill area. This value could be less than what was originally written/spilled: for example, if there is a LIMIT clause on the query. (However, if it routinely occurs that `io_spill_read_bytes` is less than `io_spill_write_bytes`, you may need to rewrite 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 worker.

Parent topic: Creating WLM Resource Pools

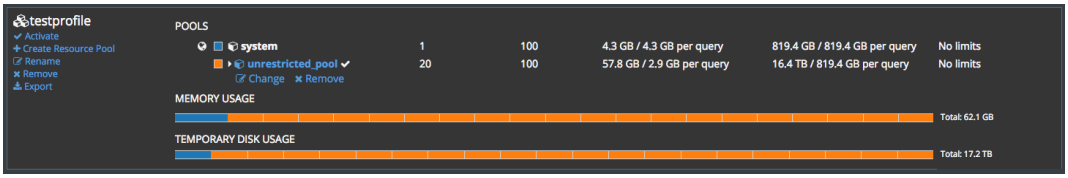
Example: Memory Allocation Among Resource Pools

The following images demonstrate memory allocation per worker node on a 15-node appliance. Each worker node has approximately 63GB of memory available (about 1TB total memory for the system). A new WLM profile, `testprofile`, is created, then associated with some resource pools that have different memory requirements.

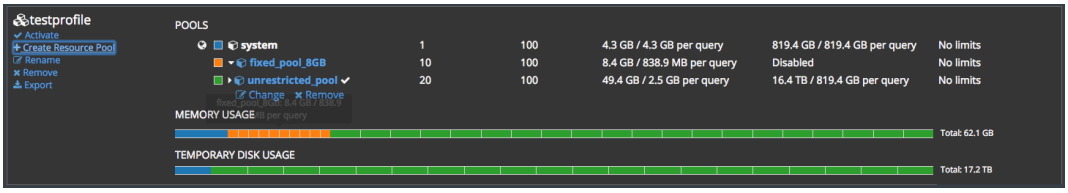
When `testprofile` is created, it contains a default `system` pool with 4.3GB of memory (a fixed amount).



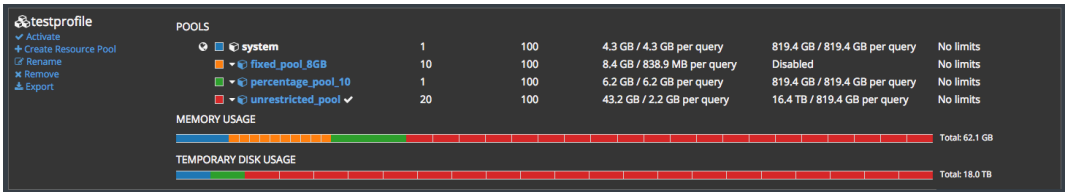
A second resource pool named `unrestricted_pool`, with 20 slots, is added. Because this pool does not restrict memory, it accrues all of the available memory (57.8GB). 2.9GB is available per slot.



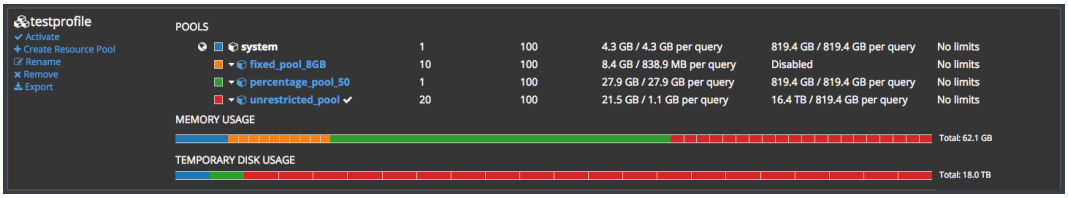
A third pool, `fixed_pool_8GB`, is added, with 10 slots and spilling disabled. It claims 8.4GB of memory, and because spilling is disabled, it is allowed to allocate less than 1GB to each slot. Note that `unrestricted_pool` now drops to 49.4GB to accommodate the new fixed-memory pool.



A fourth pool, percentage_pool_10, is added, with 1 slot and its percentage set to 10. It accrues 10% of the memory on the node, approximately 6.2GB. (Note that the percentage is the percentage of the total memory, not the percentage of the remaining memory.) In turn, `unrestricted_pool` drops again, now to 43.2GB.



In the final step, the percentage pool is modified to 50% (`percentage_pool_50`). Note that this change makes a big difference to its own memory allocation, which goes up to 27.9GB. However, its allocation cannot go any higher because `unrestricted_pool` , with 20 slots and spilling enables, must maintain a minimum allocation of 1.1GB per slot, and the two fixed-memory pools must maintain their fixed allocations. If you were to change the percentage pool to 75%, or even 100%, the allocations shown here would stay the same.



Parent topic: Creating WLM Resource Pools

Examples of Flex 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. A resource pool 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. On the other hand, 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.)

See CREATE WLM RESOURCE POOL for examples of the syntax for flex pools. Alternatively, you can create resource pools in the SMC.

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

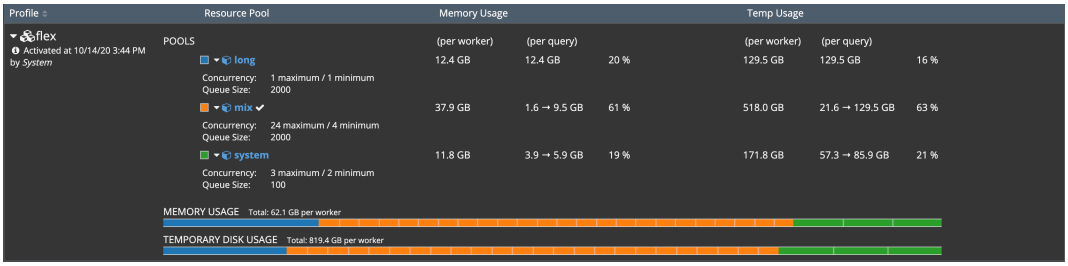
In this high-concurrency example, the system has `80GB` available for queries in a pool defined with concurrency of `32/8`:

- 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).

Default Flex Profile

On a system under heavy concurrency with the `flex` profile activated, each query that runs in the `mix` pool automatically gets 1/24 of the resources (max/min concurrency = 24/4).



In the example shown here, memory per query/per worker typically flexes between 1.6GB and 9.5GB (1/24 of the available memory up to ¼ of the available memory).

Although minimum and maximum concurrency settings are automatically applied by the system, queries are not necessarily restricted by the minimum setting. In this 24/4 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 12 slots are open, take half the memory, and allow up to 12 more queries to start in the pool.

The following rule 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;
}
```

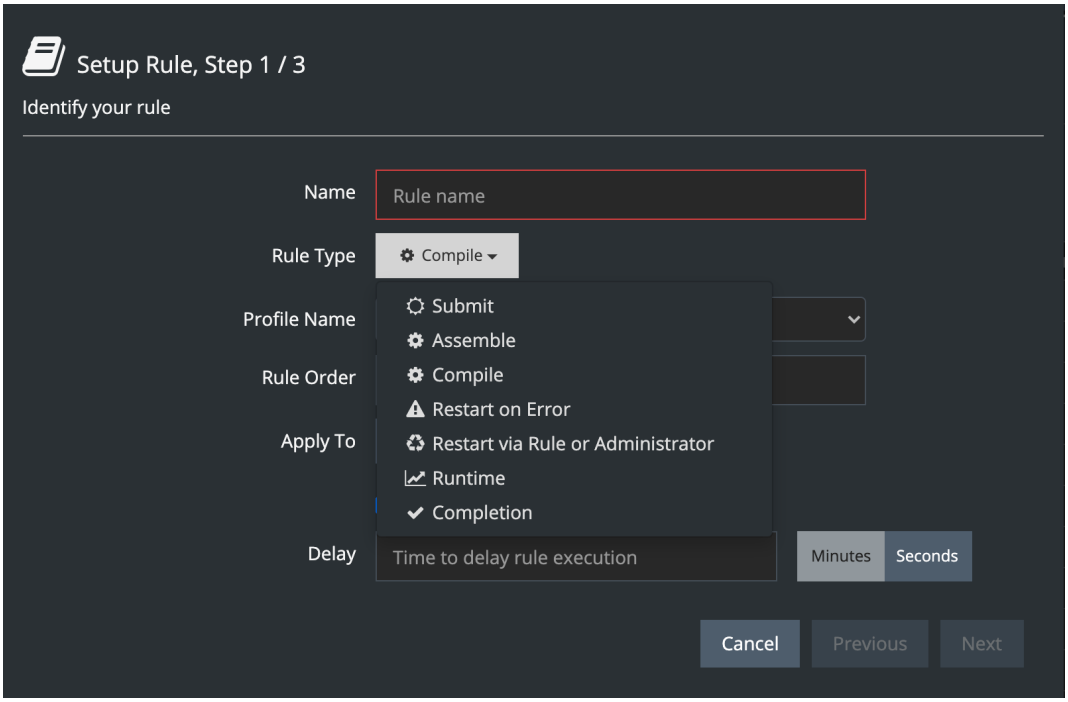
Parent topic: Creating WLM Resource Pools

Creating WLM Rules

Given knowledge of the workloads on your system, 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 abort the query.

To create a rule, use one of the following methods:

- Go to **Manage** > **Workloads** > **Rules** > **Create Rule** in the SMC. You can use assisted mode or advanced mode.



In assisted mode, 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 the SMC).

Note: Choosing the *rule type* is a critical step. The type affects when the rule is evaluated during the query lifecyle and determines what information is available to the rule, and in turn what kinds of actions can be taken. For details, see Rule Types Rule Examples.

To see a list of rules that are currently in the system, either 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

Several WLM rules are predefined in the `default` and `maintenance` profiles, and a number of global system rules belong to all profiles. The database applies system rules as needed in conjunction with the rules you define. You can remove or modify certain system rules. To see if a rule is modifiable or removable, double-click the rule name in the main **Rules** screen.

System rules are named `default_*`, `maintenance_*`, or `global_*`. For example, by default the `global_defaultRowLimit` rule prevents queries run by non-superusers 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 by changing the definition of the rule action. For example:



After changing rules, you have to activate the changes, as discussed in Creating WLM Profiles.

In This Section

- Rule Types
- Rule Conditions
- Rule Actions
- Rule Processing

- Rule Examples
- Quick Rules in the SMC
- JavaScript Properties for WLM Rules
- Recoverable Error Codes

Parent topic: How WLM Works

Rule Types

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.
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.

Rule Type	Description
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`.

Rule Type	Description
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. A number of 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.

Parent topic: Creating WLM Rules

Rule Conditions

Rules consist of one or more conditions. When satisfied, these conditions trigger rule actions. You can define different kinds of conditions in the SMC 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 Worker 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 (regex) pattern matching. Escaping is applied only when items are selected from a drop-down list (for quick rules). Regular expression validation is applied when entries are typed; warnings are displayed.	SQL matches `sys\.const`; SQL matches `sum\(substr`

Filter Operator	Description	Examples
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 `smc`. 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** > **Utilization** > **Average CPU %** in the SMC or the sys.log_query view to find this value for past executions of a query.	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 workers and vice versa). This number does not include data being returned to the client.Use **Query Details** > **Statistics** > **Network I/O Data** in the SMC to find this value for past executions of a query.	Runtime, Completion	Positive whole numbers
Bytes Read	Use **Query Details** > **Statistics** > **I/O Data Read** in the SMC or io_read_bytes in sys.log_query to find this value for past executions of a query.	Runtime, Completion	Positive whole numbers
Bytes Written	Use **Query Details** > **Statistics** > **I/O Data Written** in the SMC or io_write_bytes in sys.log_query to find this value for past executions of a query.	Runtime, Completion	Positive whole numbers

Filter Condition	Description	Rule Types	Accepted Values
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. Cost for a past run of a query is recorded in the **Query Details** > **Query** tab.	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.

Filter Condition	Description	Rule Types	Accepted Values
Error Message	Error message text.	Restart, Runtime, Completion	Any string
Execution Duration (in ms)	Number of milliseconds the query spent during execution on the worker 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** > **Memory Used Maximum** in the SMC 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** > **Memory Used Estimate** in the SMC or memory_estimated_bytes in sys.log_query to find this value for past executions of a query.	Compile, Restart, Runtime, Completion	Any number
Memory Estimate Confidence	Estimated degree of confidence in the memory estimate value.	Compile, Restart, Runtime, Completion	Unknown, None, Low, High

Filter Condition	Description	Rule Types	Accepted Values
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`

Filter Condition	Description	Rule Types	Accepted Values
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 worker” queries.	Compile, Runtime, Completion	`Is True`, `Not True`
Temporary Bytes Read	Number of bytes read from disk because spilling occurs during the query. Use **Query Details** > **Statistics** in the SMC or io_spill_read_bytes in 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. Use **Query Details** > **Statistics** in the SMC or io_spill_write_bytes in 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** > **Total Time** in the SMC or total_ms in sys.log_query to find this value for past executions of a 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** > **Memory Used Total** in the SMC or memory_total_bytes in sys.log_query to find this value for past executions of a query.	Runtime, Completion	Any number

Filter Condition	Description	Rule Types	Accepted Values
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.

Parent topic: Creating WLM Rules

Rule Actions

The following table lists the specific rule actions that you can trigger based on one or more conditions (as shown in the SMC). 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

Action	Description	Rule Types	Accepted Values
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. See System Alerts.	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

Action	Description	Rule Types	Accepted Values
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

Action	Description	Rule Types	Accepted Values
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
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 worker 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.

Parent topic: Creating WLM Rules

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. In the SMC, go to **Manage** > **Workloads** > **Active Configuration** and take a close look at the list under **Active Rules**.

To see how specific rules are applied to logged and active queries, go to **Manage** > **Workloads** > **Rule Processing**. This screen presents detailed information about the application of each rule to each query, including when rule processing begins and ends. Use the **Filter** field to narrow the scope of the entries. The Rule Examples section shows some examples of rule processing in the SMC for different types of rules.

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. To monitor and find runtime errors like this, go to **Rule Processing**, enter `error` in the **Filter** field, and look for `error` under **Type**.

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.

Parent topic: Creating WLM Rules

Rule Examples

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. Keep in mind 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';
```

In This Section

- Submit Rule Examples
- Assemble Rule Examples
- Compile Rule Examples
- Runtime Rule Examples
- Completion Rule Examples
- Restart Rule Examples
- Examples of JavaScript for Rules

Parent topic: Creating WLM Rules

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 abort 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).

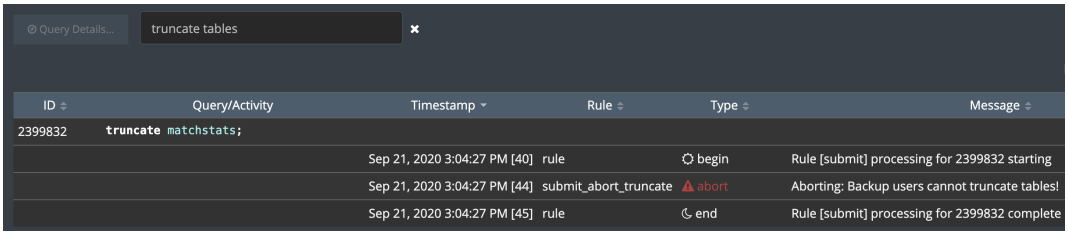
The rule logic is defined as follows:

```
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, c
You are now connected to database "premdb" as user "backup1".
premdb=> truncate matchstats;
ERROR:  canceling statement due to user request
```

In the SMC Rule Processing screen, the administrator sees:



Note that the search field is used here to isolate the `TRUNCATE` command that caused the rule to be applied.

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. The administrator will see the `throttle` rule in effect:



Parent topic: Rule Examples

Assemble 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 is run by a non-superuser and 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 t
CONTEXT:  select count(*) from match;
premdb=> select count(*) from matchstats;
ERROR:  WLM: Denied by administrator: Flex profile: ybsql users cannot query match t
CONTEXT:  select count(*) from matchstats;
premdb=> select count(*) from newmatchstats;
ERROR:  WLM: Denied by administrator: Flex profile: ybsql users cannot query match t
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. In the SMC Rule Processing screen, the administrator sees:

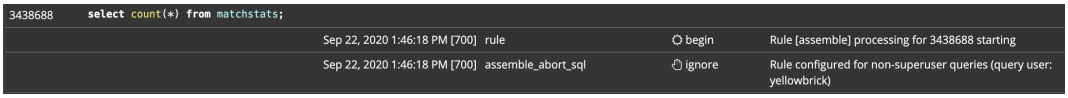


3425999	select count(*) from match;				
		Sep 22, 2020 1:30:57 PM [967]	rule	begin	Rule [assemble] processing for 3425999 starting
		Sep 22, 2020 1:30:57 PM [968]	assemble_abort_sql	abort	Aborting: Flex profile: ybsql users cannot query match tables!
		Sep 22, 2020 1:30:57 PM [970]	rule	end	Rule [assemble] processing for 3425999 complete
3425999	select count(*) from match;				
		Sep 22, 2020 1:30:57 PM [964]	rule	begin	Rule [submit] processing for 3425999 starting
		Sep 22, 2020 1:30:57 PM [964]	submit_abort_truncate	info	Rule changed no query settings
		Sep 22, 2020 1:30:57 PM [964]	submit_throttle_dml	info	Rule changed no query settings
		Sep 22, 2020 1:30:57 PM [964]	rule	end	Rule [submit] processing for 3425999 complete
n/a	Rules reloaded: 'flex'	Sep 22, 2020 1:30:49 PM [652]			

However, note that a superuser *can* run this type of query:

```
premdb=> \c premdb yellowbrick
Password for user yellowbrick:
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, c
You are now connected to database "premdb" as user "yellowbrick".
premdb=# select count(*) from matchstats;
   count
-----------
 147937140
(1 row)
```

In this case the administrator sees that the rule was ignored:

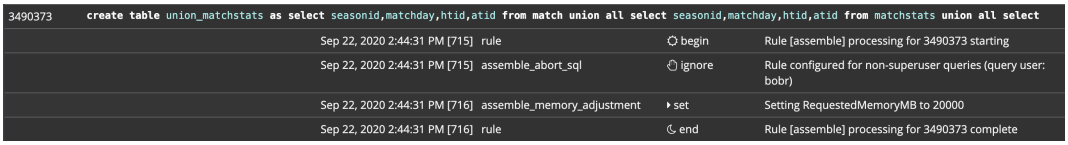


Adjust Memory for a User

This example makes sure that a large amount of memory is available to `CTAS` statements run by superuser `bobr`:

```
Rule Preview:
if (w.memoryEstimate < 20000 &&
   w.user === 'bobr' &&
   w.type === 'ctas') {
    w.requestedMemoryMB = 20000;
}
```

When `bobr` runs a `CTAS` query, the administrator sees:



The `sys.log_query` view records requested versus estimated memory use:

```
yellowbrick=# select query_id, state, username, substr(query_text,1,33), memory_esti
from sys.log_query
where query_id=3490373;
 query_id | state | username |              substr               | memory_estimated_
----------+-------+----------+-----------------------------------+------------------
  3490373 | done  | bobr     | create table union_matchstats as  |            36175
(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.

Parent topic: Rule Examples

Compile Rule Examples

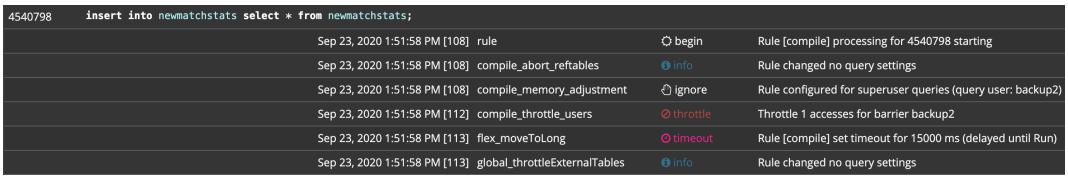
The following examples show rules that are evaluated as queries enter the `compile` state.

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:



4540798	insert into newmatchstats select * from newmatchstats;			
	Sep 23, 2020 1:51:58 PM [108]	rule	begin	Rule [compile] processing for 4540798 starting
	Sep 23, 2020 1:51:58 PM [108]	compile_abort_reftables	info	Rule changed no query settings
	Sep 23, 2020 1:51:58 PM [108]	compile_memory_adjustment	ignore	Rule configured for superuser queries (query user: backup2)
	Sep 23, 2020 1:51:58 PM [112]	compile_throttle_users	throttle	Throttle 1 accesses for barrier backup2
	Sep 23, 2020 1:51:58 PM [113]	flex_moveToLong	timeout	Rule [compile] set timeout for 15000 ms (delayed until Run)
	Sep 23, 2020 1:51:58 PM [113]	global_throttleExternalTables	info	Rule changed no query settings

Note the following message: `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 and a query is moved, the administrator sees:

3639753	insert into newmatchstats select * from matchstats;			
		Sep 22, 2020 5:35:38 PM [243]	flex_moveToLong	timeout	Rule [compile] invoked timeout after 15000 ms
		Sep 22, 2020 5:35:38 PM [243]	flex_moveToLong	info	Moving long-running query to resource pool "long": 3639753
		Sep 22, 2020 5:35:38 PM [243]	flex_moveToLong	move	Move to resource pool long
3639753	insert into newmatchstats select * from matchstats;			
		Sep 22, 2020 5:35:22 PM [387]	rule	begin	Rule [compile] processing for 3639753 starting
		Sep 22, 2020 5:35:22 PM [387]	compile_abort_reftables	info	Rule changed no query settings
		Sep 22, 2020 5:35:22 PM [387]	compile_memory_adjustment	ignore	Rule configured for superuser queries (query user: backup1)
		Sep 22, 2020 5:35:22 PM [387]	flex_moveToLong	timeout	Rule [compile] set timeout for 15000 ms (delayed until Run)
		Sep 22, 2020 5:35:22 PM [388]	global_throttleExternalTables	info	Rule changed no query settings

Note that the timeout is set, then applied when the query runs longer than 15000 ms. The following system view query confirms what happened:

```
premdb=# select pool_id, run_ms from sys.log_query where query_id=3639753;
 pool_id |  run_ms
---------+-----------
 long    | 97083.613
(1 row)
```

Parent topic: Rule Examples

Runtime 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.

When this rule is applied, the administrator sees:

4781170	`insert into matchstats select * from newmatchstats;`			
	Sep 23, 2020 6:18:16 PM [368]	rule	begin	Rule [runtime] processing for 4781170 starting
	Sep 23, 2020 6:18:16 PM [368]	runtime_log_info_msg	ignore	Rule configured for non-superuser queries (query user: yellowbrick)
	Sep 23, 2020 6:18:16 PM [368]	runtime_no_stats_for_premdb	disabled	Rule is disabled
	Sep 23, 2020 6:18:16 PM [368]	runtime_set_critical_priority	set	Setting Priority to critical
	Sep 23, 2020 6:18:16 PM [368]	rule	end	Rule [runtime] processing for 4781170 complete
4781170	`insert into matchstats select * from newmatchstats;`			
	Sep 23, 2020 6:18:11 PM [367]	rule	begin	Rule [runtime] processing for 4781170 starting
	Sep 23, 2020 6:18:11 PM [367]	runtime_log_info_msg	ignore	Rule configured for non-superuser queries (query user: yellowbrick)
	Sep 23, 2020 6:18:11 PM [367]	runtime_no_stats_for_premdb	disabled	Rule is disabled
	Sep 23, 2020 6:18:11 PM [367]	runtime_set_critical_priority	set	Setting Priority to critical
	Sep 23, 2020 6:18:11 PM [367]	rule	end	Rule [runtime] processing for 4781170 complete

---

Log an INFO Message for Large DELETE Queries

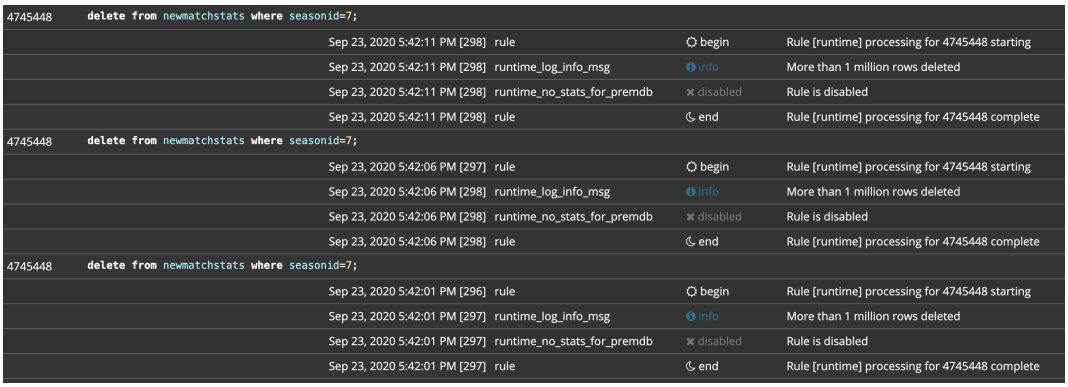
The following runtime rule, `runtime_log_info_msg`, logs an `INFO` message when more than 1 million rows are deleted from any table:

```
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
```

In the SMC the administrator sees:



4745448	`delete from newmatchstats where seasonid=7;`			
	Sep 23, 2020 5:42:11 PM [298]	rule	begin	Rule [runtime] processing for 4745448 starting
	Sep 23, 2020 5:42:11 PM [298]	runtime_log_info_msg	info	More than 1 million rows deleted
	Sep 23, 2020 5:42:11 PM [298]	runtime_no_stats_for_premdb	disabled	Rule is disabled
	Sep 23, 2020 5:42:11 PM [298]	rule	end	Rule [runtime] processing for 4745448 complete
4745448	`delete from newmatchstats where seasonid=7;`			
	Sep 23, 2020 5:42:06 PM [297]	rule	begin	Rule [runtime] processing for 4745448 starting
	Sep 23, 2020 5:42:06 PM [298]	runtime_log_info_msg	info	More than 1 million rows deleted
	Sep 23, 2020 5:42:06 PM [298]	runtime_no_stats_for_premdb	disabled	Rule is disabled
	Sep 23, 2020 5:42:06 PM [298]	rule	end	Rule [runtime] processing for 4745448 complete
4745448	`delete from newmatchstats where seasonid=7;`			
	Sep 23, 2020 5:42:01 PM [296]	rule	begin	Rule [runtime] processing for 4745448 starting
	Sep 23, 2020 5:42:01 PM [297]	runtime_log_info_msg	info	More than 1 million rows deleted
	Sep 23, 2020 5:42:01 PM [297]	runtime_no_stats_for_premdb	disabled	Rule is disabled
	Sep 23, 2020 5:42:01 PM [297]	rule	end	Rule [runtime] processing for 4745448 complete

Note that runtime rules are processed at 5-second intervals. The `INFO` message is displayed as expected:

```
More than 1 million rows deleted
```

Do Not Store Stats for a Database

The following runtime rule, `runtime_no_stats_for_premdb`, turns of 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!');
}
```

Parent topic: Rule Examples

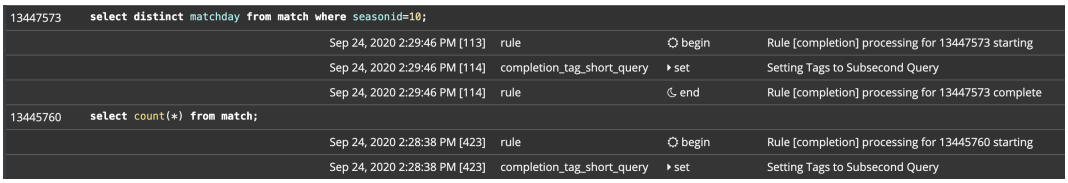
Completion Rule Examples

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-superusers complete in less than 1 second.

For every subsecond query that runs, the administrator sees:



Log a Warning Message for the yellowbrick User

This completion rule, `completion_warn_yellowbrick_user`, tracks all of the queries run by the `yellowbrick` account and logs a warning message.

```
Rule Preview:
if (w.user === 'yellowbrick') {
    log.warn('Running as yellowbrick!');
}
```

For every query run by the `yellowbrick` account, the administrator sees:



Parent topic: Rule Examples

Restart Rule Examples

The following examples describe rules that are evaluated when queries are restarted because of an error.

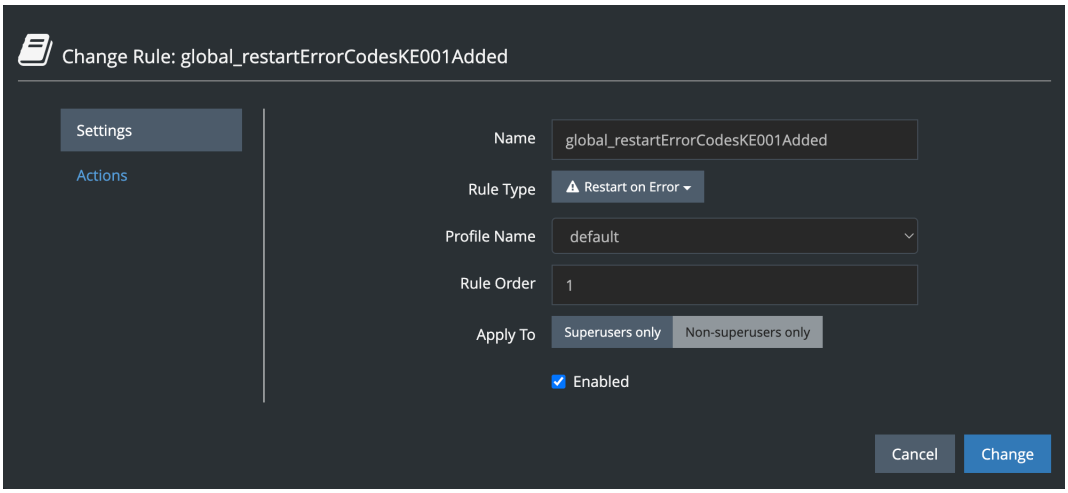
Modified Restart on Error System Rule

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.)

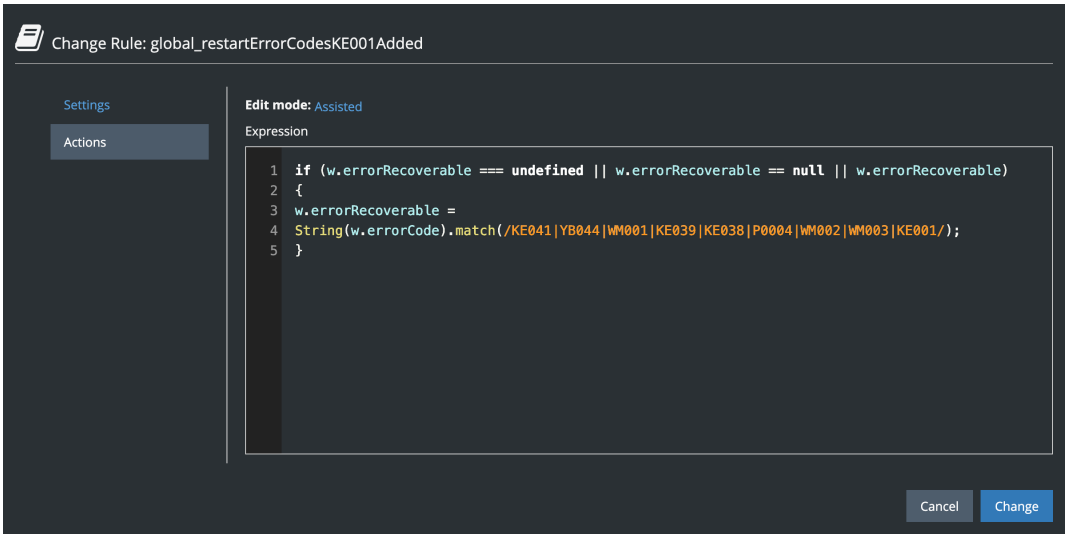
Start by looking at the default definition of `global_restartErrorCodes`:

```
yellowbrick=# select * from sys.wlm_active_rule where rule_name ='global_restartErr
-[ 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 = String(w.errorCode).match(/KE041|YB044|WM001|KE
             | }
             |
```

You can use the SMC or a SQL command to create a new rule with a different name that copies most of this rule definition. For example, use the same main settings:



Now use the same rule definition, but add another error code from the recoverable list (`KE001` in this example):



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.errorRecover
```

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 resou
  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 +
  }
}
```

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.

Parent topic: Rule Examples

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 the SMC or with a SQL command. You can use Advanced mode in the SMC 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`, abort 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 and the **Manage** > **Query Alerts** view in the SMC. 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
   w.execId, (w.stats.maxMemory / (1024*1024*1024)).toFixed(1));
}
```

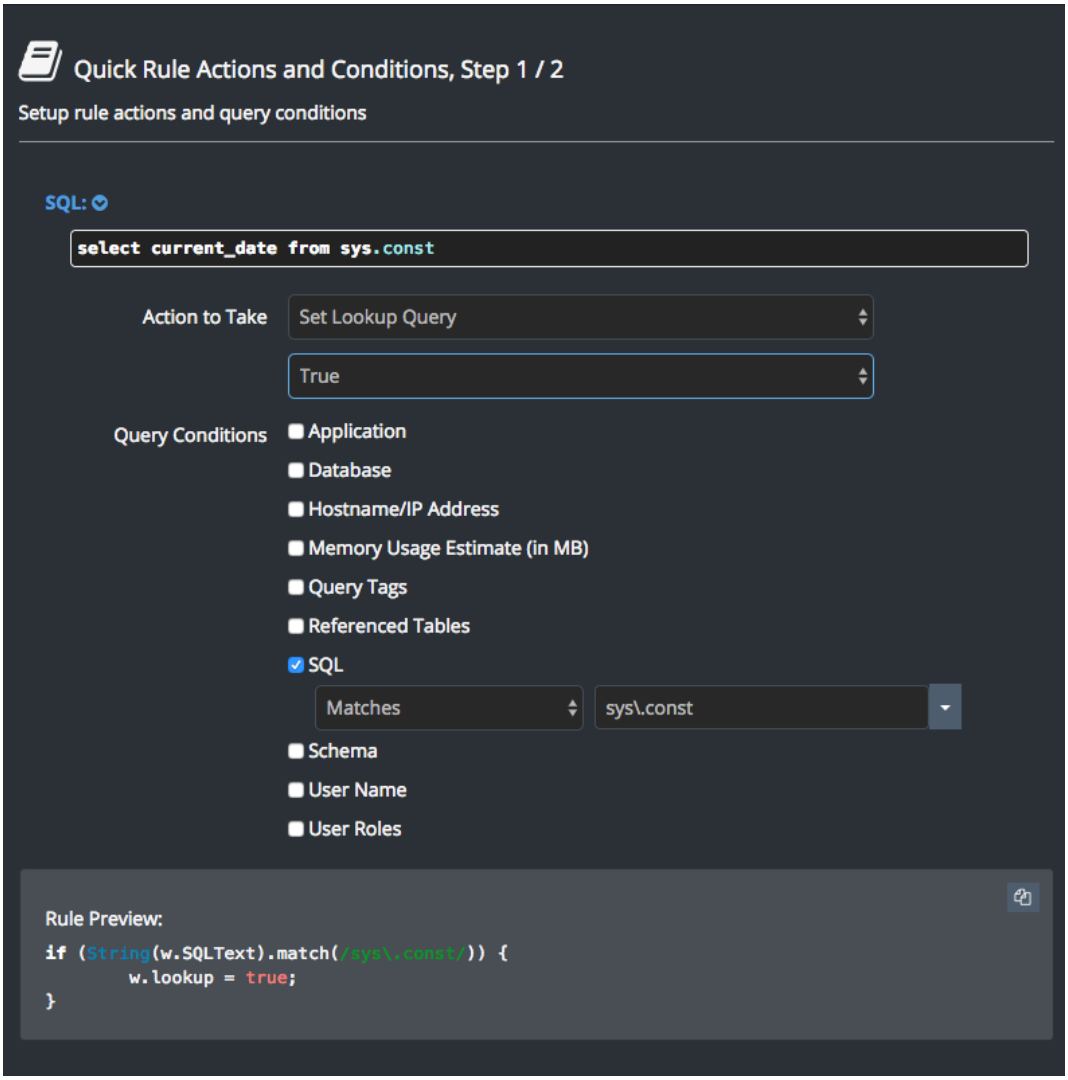
This rule examines the current statistics for the query, and if it has used more than 2GB (`2*1024*1024*1024`), and its priority is not already `high`, its priority is set to `high` and an error is logged. (The decision to log an error is arbitrary; a warning might be more appropriate here.)

Parent topic: Rule Examples

Quick Rules in the SMC

On the **Query Details** screen in the SMC, you can use the **Quick Rule** option to create a rule based on some attribute of the query itself.

1. Go to **Monitor** > **Query Performance**.
2. Double-click the query you are interested in to open the **Query Details** screen.
3. Click **Quick Rule**.
4. Complete the rule definition, making use of the **SQL** drop-down if you want to use the query text as a reference point. For example:



Parent topic: Creating WLM Rules

JavaScript Properties for 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 the SMC. (Some properties do not exist as choices in the SMC.)

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
`w.abort` or `w.abortReason`	Abort Query action (cancel query and return a reason message)
w.allowSQB	Set Short Query Bias action (or a condition)
w.application	Application condition
`w.authenticatedUsername`	Condition on the user account that originates the session. If `SET ROLE` is used, this value does not change.
w.command	Command condition
w.cost	Cost condition
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).

Name	Description
w.database	Database condition
w.duration or `w.elapsedTime`	Total Duration condition. (`w.duration` and `w.elapsedTime` are synonyms.)
w.errorCode	Error Code condition
w.errorMessage	Error Message condition
`w.errorRecoverable`	Recoverable Error condition; whether the query can be restarted on error (defined as part of the default error recovery policy).
`w.execId`	Execution ID condition (where execution ID is the same value as a `query_id` in `sys.query`)
w.executionDuration	Execution Duration condition
`w.explainAnalyze`	`EXPLAIN ANALYZE` condition (whether the query is an `EXPLAIN ANALYZE` command)
`w.externalScan`	External table scan condition
`w.externalWrite`	External table write condition
`log.debug` , `log.error` , `log.info` , `log.warn`	Action to log messages at different levels

Name	Description
w.lookup	Set Lookup Query action; also a condition
w.maximumExecTimeLimit	Set Maximum Execution Time Limit action
w.maximumRowLimit	Set Maximum Row Limit action
w.maximumWaitLimit	Set Maximum Wait Limit action
w.memoryEstimate	Memory Estimate condition (value in MB, but equivalent to `memory_estimated_bytes` in `sys.query`)
w.memoryEstimateConfidence	Memory Estimate Confidence condition
`w.memoryRequiredMB`	Memory required condition (in MB but maps to `memory_required_bytes` in `sys.query`)
w.movable	Movable query condition
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).
w.namespace	Schema condition

Name	Description
`w.numRestartError`	Number of restarts on error (`num_error` in `sys.query`)
`w.numRestartUser`	Number of restarts by user (`num_restart` in `sys.query`)
w.priority	Priority condition or action
w.referencedTables	Referenced Tables condition
w.requestedMemoryMB	Set Requested Memory action (in MB)
w.requestedMemoryPercent	Set Requested Memory action (as a percentage)
w.requestedSpillMB	Set Requested Temp Space action (in MB)
w.requestedSpillPercent	Set Requested Temp Space action (as a percentage)
w.resourcePool	Resource Pool condition or action. See also Setting and Matching Resource Pools in WLM Rules.
w.restartable	Restartable query condition

Name	Description
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).
w.roles	User Roles condition
`w.sessionId`	Session ID condition
`w.setTimeout(function () ..., timeoutMs)`	Timeout condition (specifies a `timeoutMs` delay on the body of the rule)
w.singleWorker	Single Worker query condition
w.SQLText	SQL condition
`w.stats.AvgCpu`	Average CPU across cluster
w.stats.bytesNetwork	Bytes Network condition
w.stats.bytesRead	Bytes Read condition
`w.stats.BytesReadClient`	Bytes Read by Client condition
`w.stats.BytesWriteClient`	Bytes Sent by Client condition
w.stats.bytesReadSpill	Temporary Bytes Read condition
w.stats.bytesWrite	Bytes Written condition
w.stats.bytesWriteSpill	Temporary Bytes Written condition

Name	Description
`w.stats.cpu`	CPU percent usage on longest worker (`cpu_percent` in `sys.query`)
`w.stats.longestWorker`	Longest worker condition
`w.stats.maxCpu`	CPU percent usage of max worker (`cpu_percent_max` in `sys.query`)
`w.stats.memory`	Memory condition (on longest worker, `memory_bytes` in `sys.query`)
w.stats.maxMemory	Max Memory condition
w.stats.rowsDeleted	Rows Deleted condition
w.stats.rowsInserted	Rows Inserted condition
w.stats.rowsReturned	Rows Returned condition
w.statsStored	Store Execution Statistics action and condition
`w.superUser`	Superuser condition (`superuser` column in `sys.user` and `sys.role`)
w.tags	Set Query Tags action or condition
`w.transactionId`	Transaction ID condition (`transaction_id` in `sys.query`)

Name	Description
w.type	Type condition (type of query)
w.user	User Name condition
`w.userId`	User ID condition
`w.workstation`	Workstation condition, where this value corresponds to `client_ip_address` in the `sys.log_session` and `sys.log_session` views.
`wlm.assignMaximumResources`	Action to assign the maximum available resources in a pool. See Default Error Recovery.
`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.
wlm.throttle(max, barrierName)	Limit Concurrent Queries action (maximum number, name of "barrier," such as a user, role, or application)

Parent topic: Creating WLM Rules

Recoverable Error Codes

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. 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 spil
KE031 FILESYSTEMSPILLFULL                : File system space reserved for spilling
KE032 SPILLLIMITEXCEEDED                 : Attempt to use spill space above limit
KE036 TCPSOCKETTIMEOUT                   : TCP socket timeout
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 Pari
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 shou
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 read
YB008 NOSUCHEXECUTIONID                 : ExecutionID not found
YB009 NOSUCHNODEID                      : NodeID not found
YB010 INVALIDREGISTRATION               : Invalid worker registration.
YB044 RecoverableGeneric
YU002 NOSUCHWORKERID                    : invalid worker id
```

Parent topic: Creating WLM Rules

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.

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.

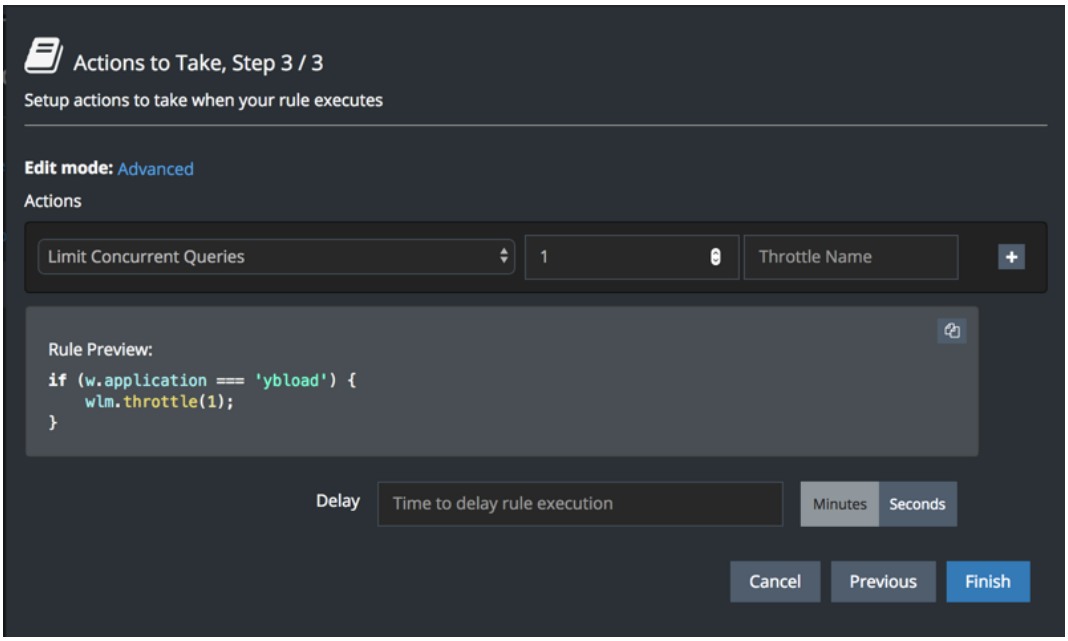
```
if (w.type === 'insert') {
    wlm.throttle(1, w.user);
```

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 the SMC, 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); }
```

In other words, if 5 developers belong to this role, maximum concurrency for the pool when this rule is applied is 5.

Parent topic: How WLM Works

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 the SMC 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 worker 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.

Parent topic: How WLM Works

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`)

Within these types, a query cannot be restarted if it references external tables.

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 resou
  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 +
    }
  }
```

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.

Parent topic: How WLM Works

Using Query Tags

You can define labels, known as *query tags*, to track query activity. You can create WLM rules that define conditions based on the presence of tags, and you can log query tags as actions for rules. 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. 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='bobr 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.

Parent topic: How WLM Works

Exporting and Importing Profiles

You can use the SMC to export a WLM profile to a JSON file. Go to **Manage** > **Workloads**, then click **Export** below the profile.

In turn, you can import a profile into the SMC and activate it. You cannot import a profile that has the same name as an existing profile.

Note: In general, manually editing exported profiles is not recommended. You may introduce characters that cause a subsequent import to fail.

This example is for a simple profile that was exported before any rules were defined:

```
{
 "default_pool": "shortquerypool",
 "name": "shortquerybias",
 "active": true,
 "pools": [
  {
   "maximum_row_limit": 500000,
   "slots": 20,
   "queue_size": 100,
   "profile": "shortquerybias",
   "name": "shortquerypool",
   "next_memory_queue": null,
   "maximum_wait_limit": null,
   "maximum_exec_time_limit": 1,
   "max_spill_pct": null,
   "requested_memory": null,
   "next_exec_time_limit_queue": null
  }
 ],
 "rules": []
}
```

You cannot export the `default` and `maintenance` profiles.

Parent topic: How WLM Works

Monitoring WLM Activity

You can use the SMC to monitor workload management behavior in real time.

- To see resource utilization statistics for specific resource pools, go to **Monitor** > **Resource Pools** or **Monitor** > **Execution Timeline**.
- To see how specific rules are applied to logged and active queries, go to **Manage** > **Workloads** > **Rule Processing**. See also Rule Examples and A Simple WLM Example.

Parent topic: How WLM Works

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 a specific query is run. You can use the SMC or SQL commands to complete the same setup.

In This Section

- WLM Example: Using the SMC
- WLM Example: Using SQL

Parent topic: Workload Management

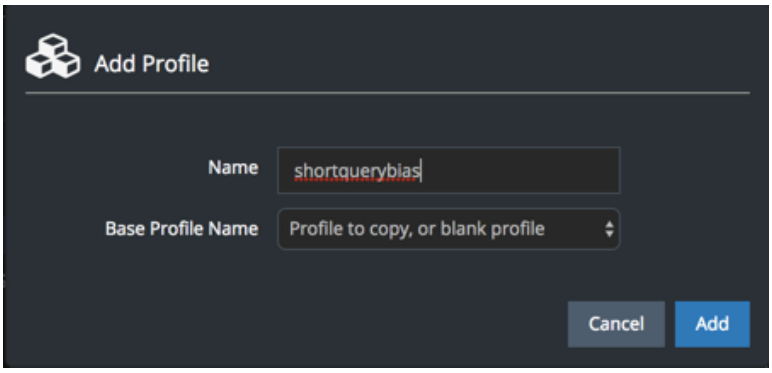
WLM Example: Using the SMC

All of the setup steps in the following example can be completed in the SMC (as shown) or by using SQL commands.

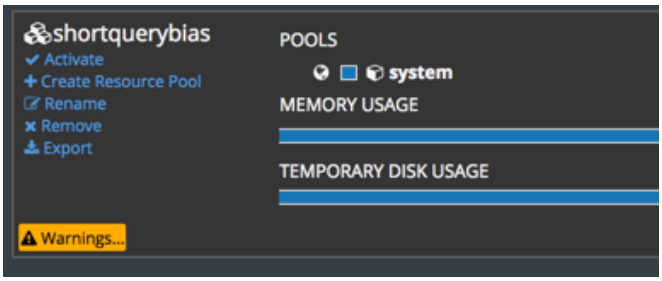
To set up a simple example of workload management:

1. Log into the SMC and go to **Manage** > **Workloads** > **Create Profile**.

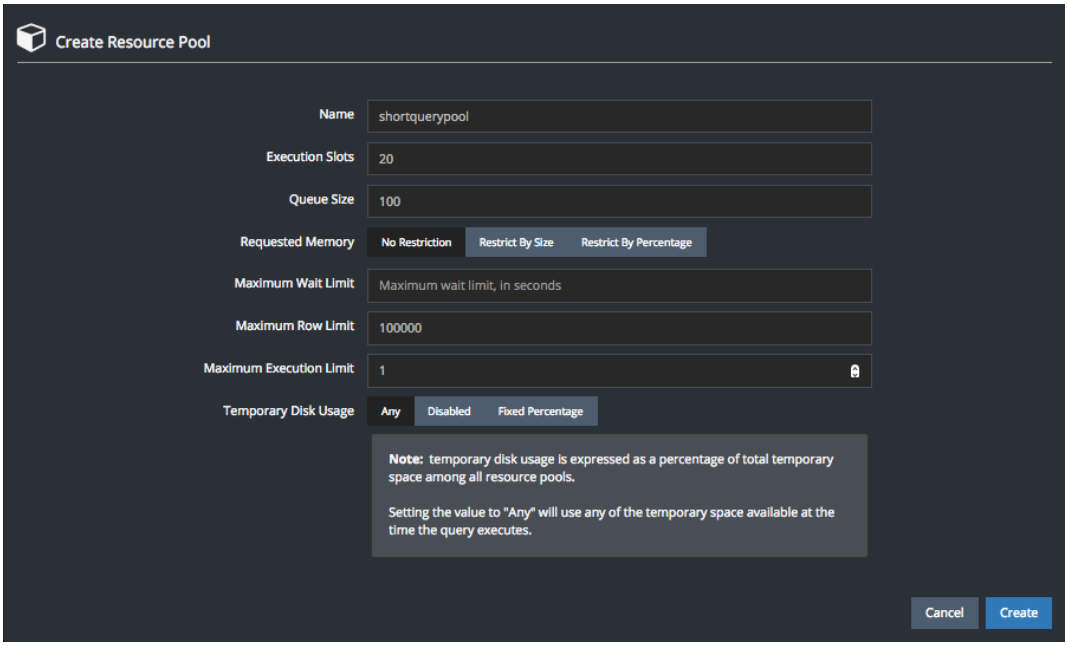
2. Create a profile and give it a meaningful name, such as `shortquerybias`.



By default, this profile contains a single resource pool: `system`.

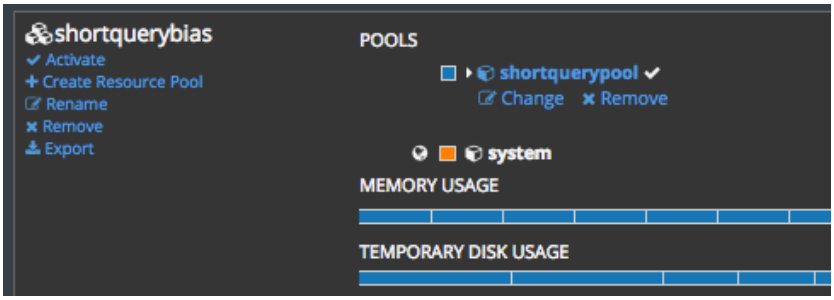


3. Click **Create Resource Pool** under the `shortquerybias` profile. Name the new pool `shortquerypool`, and give it 20 slots, a queue size of 100, a maximum row limit of 100000, and a maximum execution time of 1.



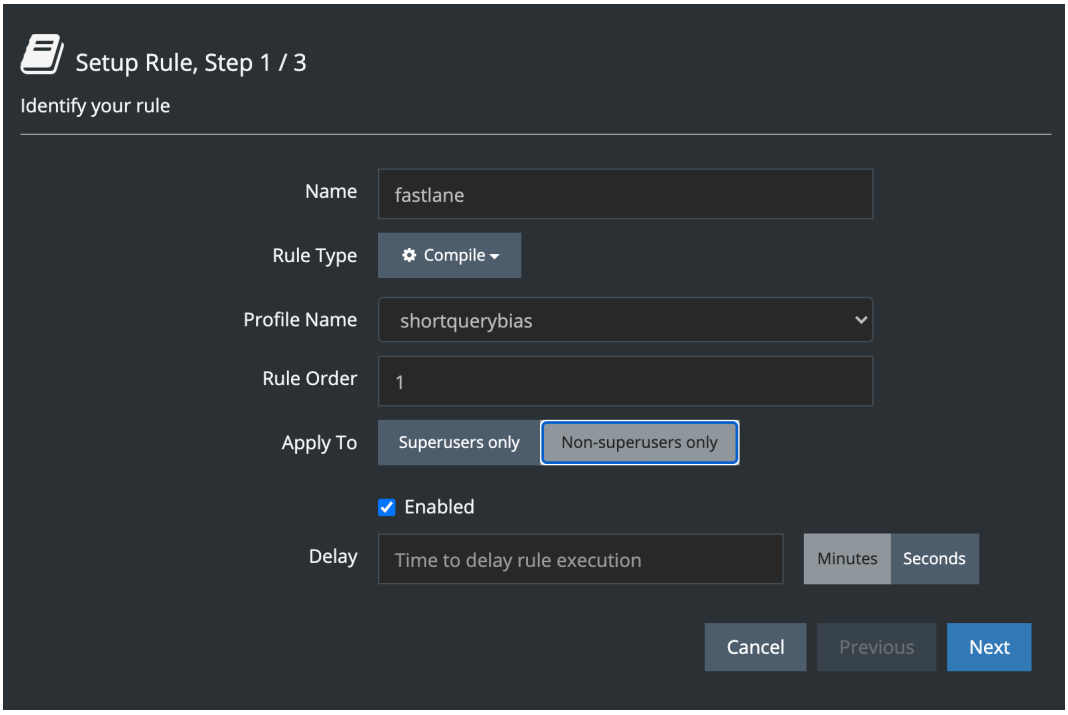
4. Mark the new resource pool as the default pool for this profile. Under `shortquerypool`, select **Mark as Default Pool**.

A check mark will appear next to the resource pool name.

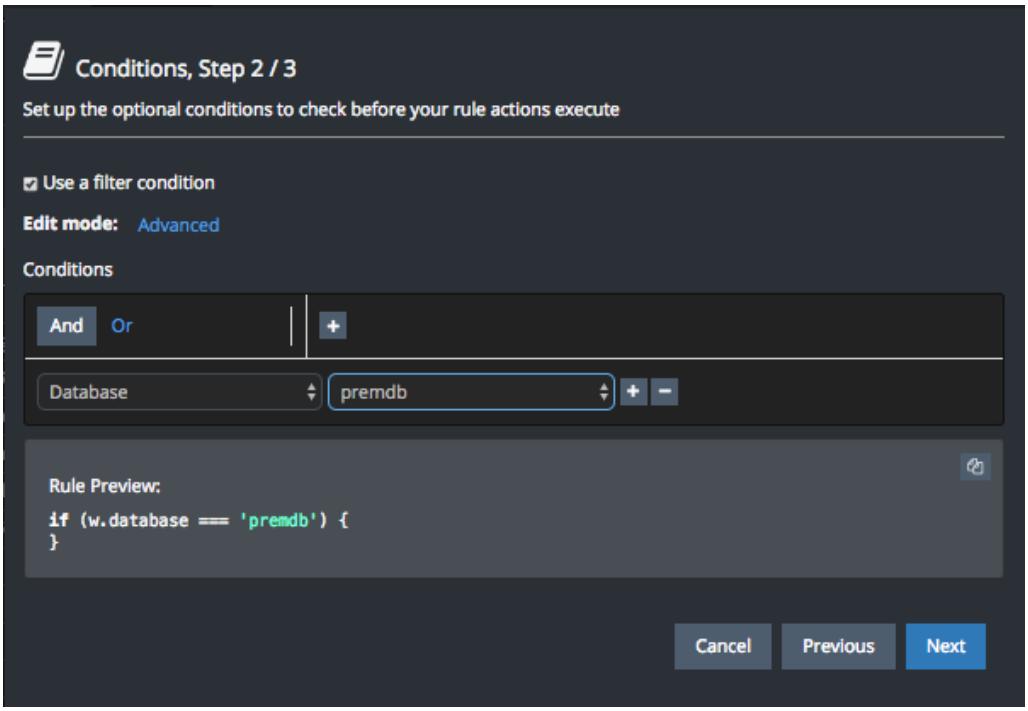


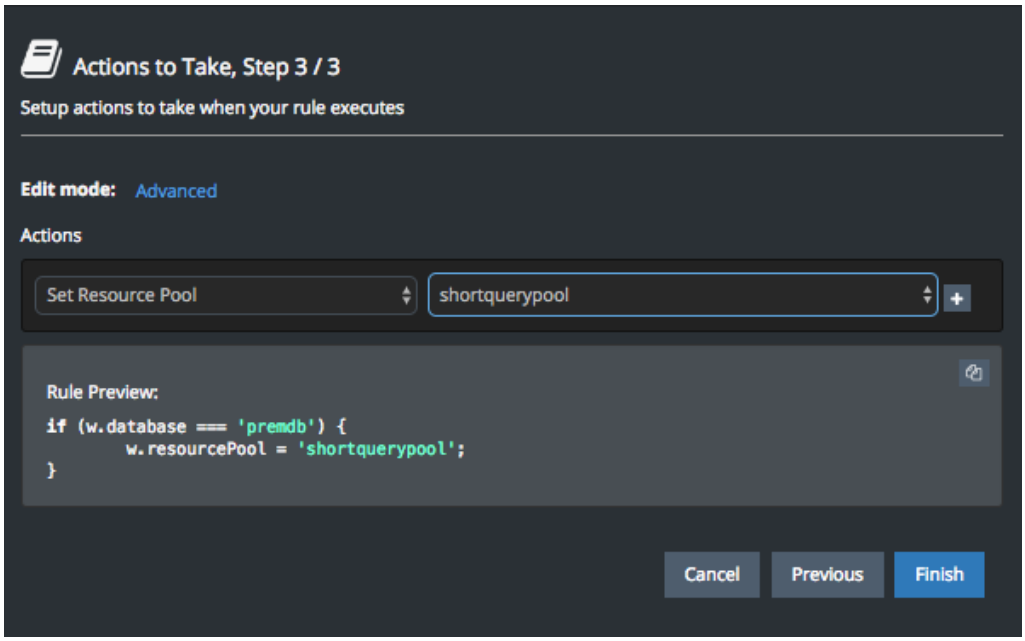
5. Click **Rules** > **Create Rule**.

6. Create a `Compile` type rule and name it `fastlane`.

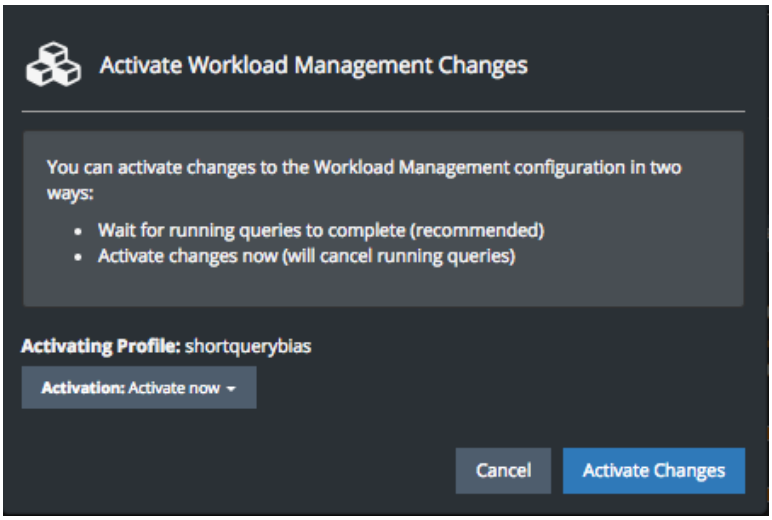


7. Require all queries against the `premdb` database to run in `shortquerypool`.

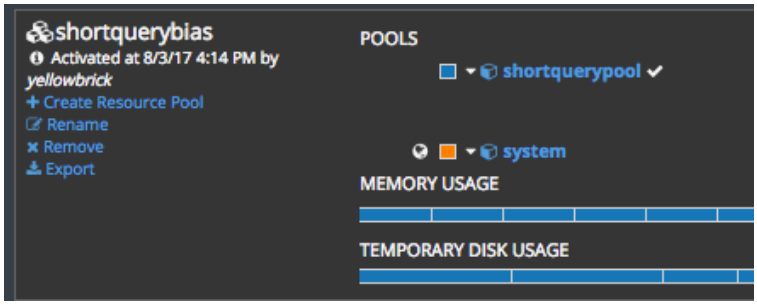




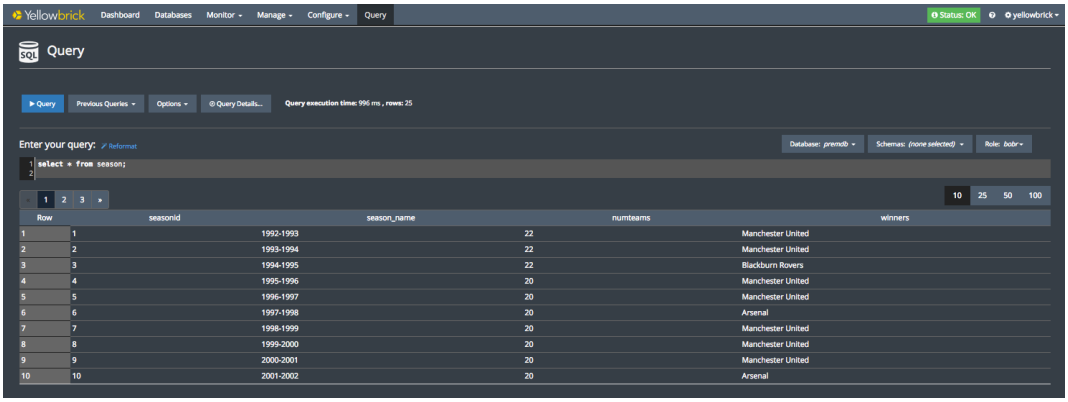
8. Go back to **Profiles/Resource Pools**, click `shortquerybias`, and create a new active configuration.



You will see an activation message under the profile name.

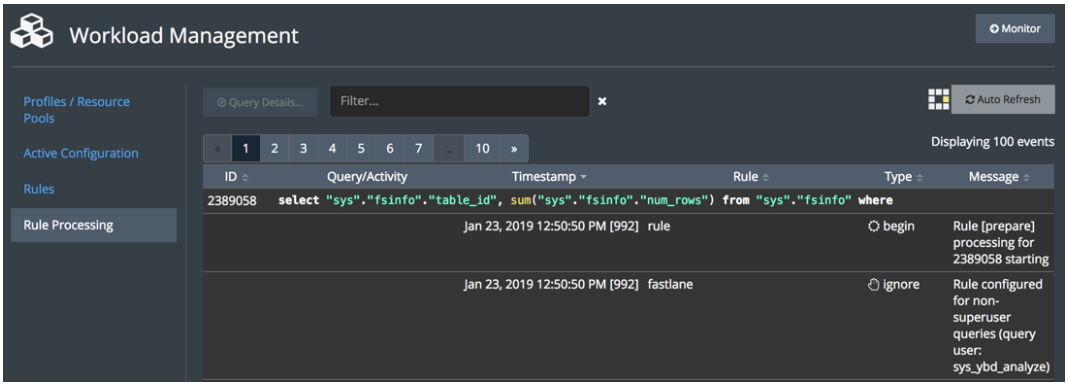


9. Go to the **Query** tab and run a query *as a non-superuser* with `select` privileges on the tables in the `premdb` database. For example:



Note: When admin users and superusers (such as the `yellowbrick` user) submit requests, they all go to the `admin` resource pool.

10. Monitor WLM activity. Go to **Manage** > **Workloads** > **Rule Processing** to see which rules and pools were processed for the query.



Parent topic: A Simple WLM Example

WLM Example: Using SQL

All of the setup steps in the following example can be completed by using SQL commands (as shown) or in the SMC.

To set up a simple example of workload management:

1. Create a profile and give it a meaningful name, such as `shortquerybias`.

```
premdb=# create wlm profile shortquerybias;
CREATE WLM PROFILE
```

By default, this profile contains a single resource pool: `system`.

2. Create a resource pool named `shortquerypool` and set `shortquerybias` as its profile.

Give this pool a max/min concurrency of 20/1, a long queue, a maximum row limit, and a maximum execution time.

```
premdb=# create wlm resource pool shortquerypool
(min_concurrency 1, max_concurrency 20, queue_size 100, maximum_row_limit 100000, ma
profile shortquerybias);
CREATE WLM RESOURCE POOL
```

3. Mark the new resource pool as the default pool for this profile.

```
premdb=# alter wlm profile shortquerybias (default_pool shortquerypool);
ALTER WLM PROFILE
```

4. Create a rule and name it `fastlane`. Require all queries against the `premdb` database to run in `shortquerypool`.

```
premdb=# create wlm rule fastlane
(javascript $$ if (w.database === 'premdb') { w.resourcePool = 'shortquerypool';}$$,
enabled true, profile shortquerybias);
CREATE WLM RULE
```

By default, rules are created as `Compile` type rules. See Rule Types.

5. Create a new active configuration by activating the `shortquerybias` profile.

```
premdb=# alter wlm profile shortquerybias activate;
ALTER WLM PROFILE
```

6. Start a `ybsql` session *as a non-superuser* with `select` privileges on the tables in the `premdb` database. For example:

```
premdb=# create user sqb with password 'sqb';
CREATE ROLE
premdb=# grant select on match, team, season, awayteam, hometeam to sqb;
GRANT
premdb=# \c premdb sqb
Password for user sqb:
You are now connected to database "premdb" as user "sqb".
premdb=>
```

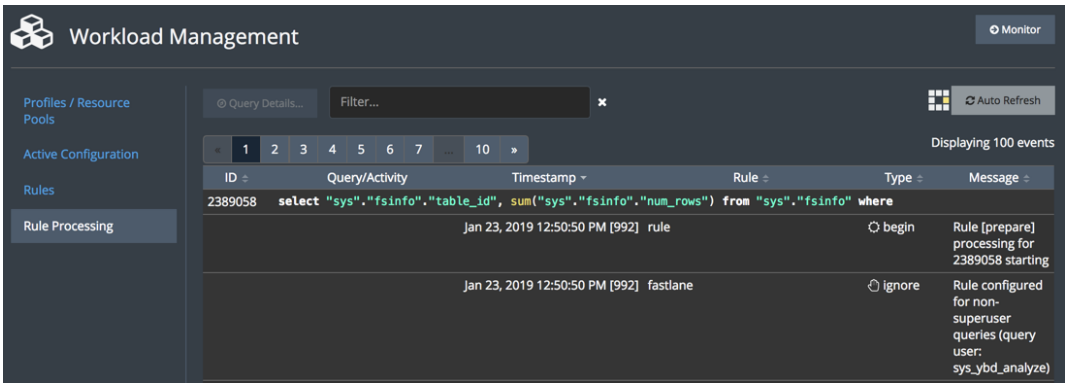
Note: When admin users and superusers (such as the `yellowbrick` user) submit requests, they all go to the `admin` resource pool.

7. Run a simple query that you expect to run in `shortquerypool` because the connected database is `premdb`. For example:

```
premdb=> select * from season;
 seasonid | season_name | numteams |      winners
----------+-------------+----------+-------------------
        1 | 1992-1993   |       22 | Manchester United
        2 | 1993-1994   |       22 | Manchester United
        3 | 1994-1995   |       22 | Blackburn Rovers
...
```

8. Monitor WLM activity in the SMC. Use the **Manage** > **Workloads** > **Rule Processing** feature to see which rules and pools were processed for the query.

Note that `shortquerypool` was selected and the `fastlane` rule was applied, but the superuser system rules were not.



Tip: You can navigate to equivalent information by going to **Monitor** > **Query Performance**, then double-clicking a specific query, then selecting **Rule**.

Parent topic: A Simple WLM Example

Backup and Restore

This section explains how to back up and restore Yellowbrick databases by using Java-based tools that you invoke from a client system (`ybbackup` and `ybrestore`). You can also manage backups with the `ybbackupctl` tool.

In This Section

- Overview
- ybbackup Commands
- ybrestore Commands
- ybbackupctl Commands
- Monitoring Backup and Restore Operations
- Decrypting Columns in a Restored Database
- HOT_STANDBY and READONLY Modes

Parent topic: Yellowbrick Documentation

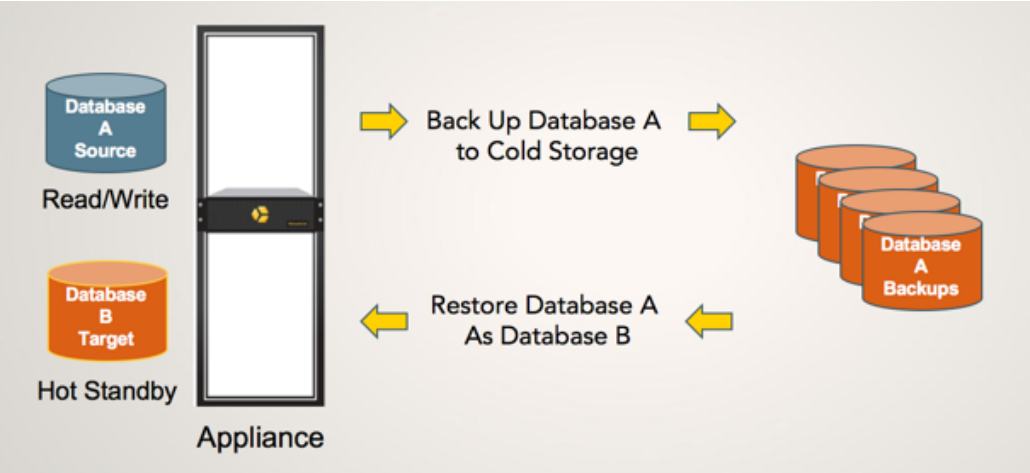
Overview

This section introduces concepts, terms, and behavior that you need to be familiar with before attempting to back up and restore Yellowbrick databases.

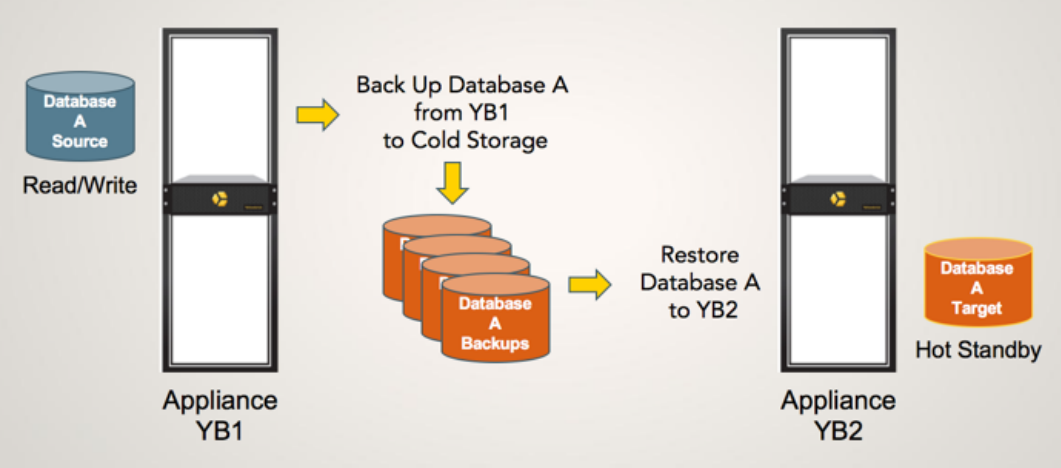
Use Cases

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.

You can back up and restore a database to the same cluster where the original database was created:



Alternatively, you can restore a database to a completely separate Yellowbrick system.



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. You can use either backup and restore operations or replication to "clone" databases.

Database replication is a related but separate feature, which uses database backups (or backup snapshots) to maintain database replicas on a specific target system. The fundamental differences between backup/restore operations and replication are as follows:

- Replication requires no intermediate storage system for backups. Data is streamed directly from the source to the target.
- Replication typically implies the use of a second Yellowbrick cluster that is used as the target system. However, local "loopback" replication is also supported and can be used to clone databases on the same system.

For details, see Database Replication.

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.

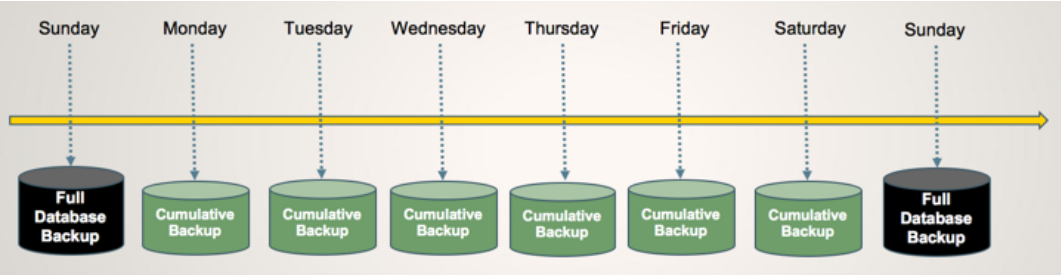
Backup Types

Yellowbrick supports backups at three different levels:

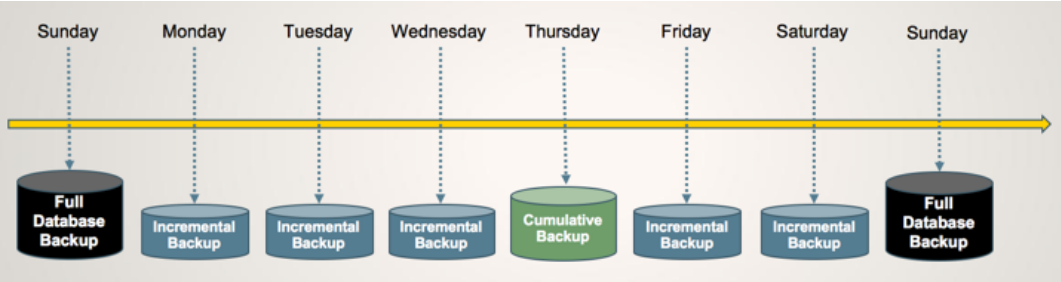
- **Full**: backs up the entire database from scratch.
- **Cumulative**: backs up all changes to the database since the last full or cumulative backup.
- **Incremental**: backs up all changes to the database since the last full, cumulative, or incremental backup.

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, such as WLM profiles, external table files, and NFS mounts for external tables, are not backed up. Table statistics are not backed up; they are recalculated and re-created when a database is restored.

Encryption keys for SQL access, as created with the `CREATE KEY` command, are also not backed up.

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.

Snapshots are also used as the basis for database replication.

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; 6GB per worker 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 6GB minimum, backup and restore operations may run out of memory. This recommendation applies to regular backup and restore operations and database replication.

In This Section

- Backup Strategies
- Restore Operations
- Backup and Restore Clients
- Backup and Restore Glossary

Parent topic: Backup and Restore

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 the event 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

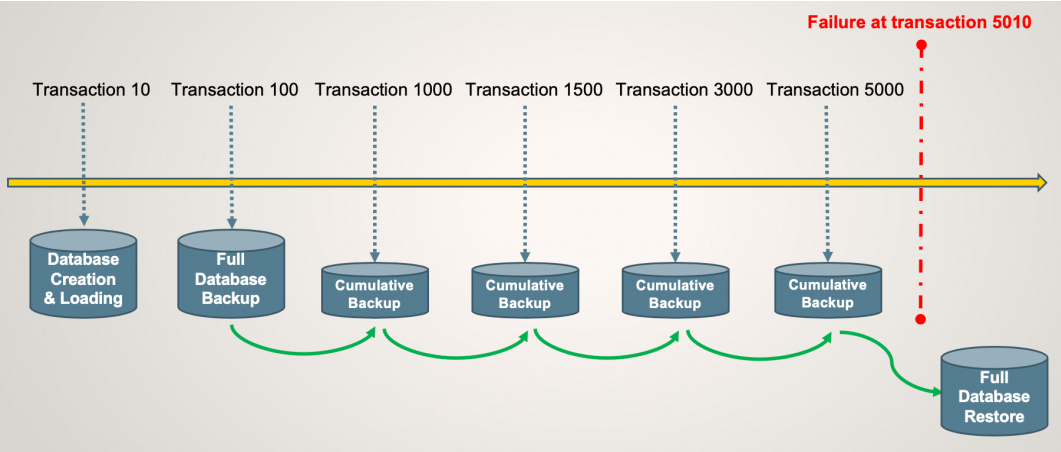
Assume that a database has the following backups:

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.

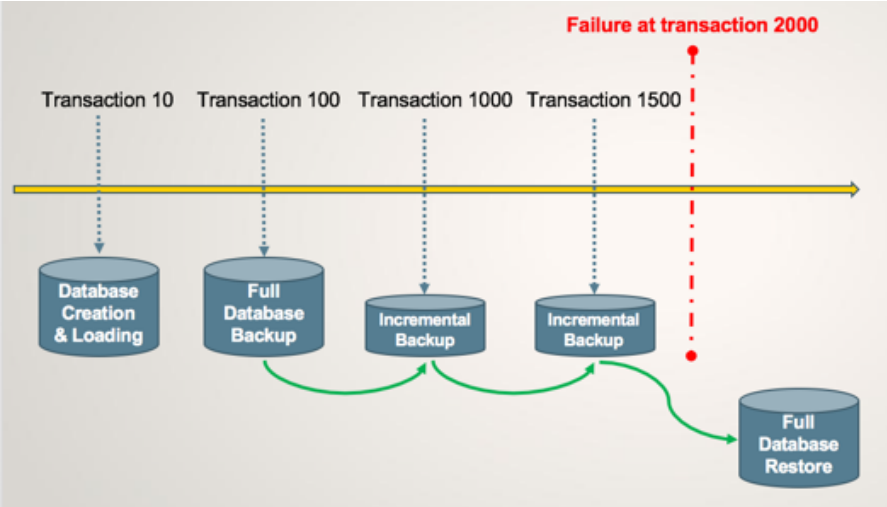
In the following example, the restore path has multiple steps because a failure occurs after a full backup and a series of cumulative backups. Any writes that occurred between transactions `5000` and `5010` would be lost, but the database could be fully recovered to the latest backup point, using the 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.



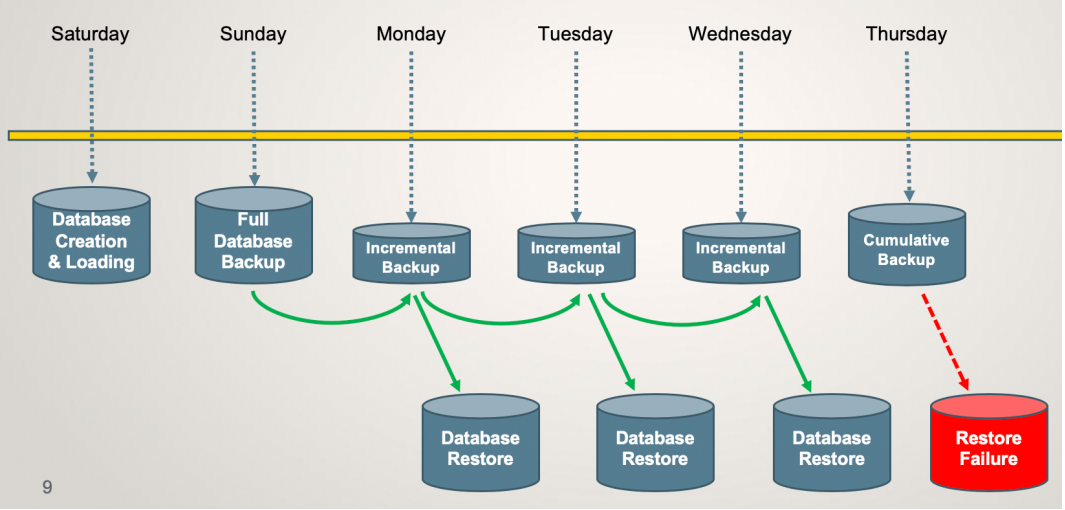
Because only full and cumulative backups are used in this case, the database can be restored from any one of the cumulative backups without compromising subsequent backups in the chain. For example, you could restore to the backup at transaction `1500`, then later restore to the backup at transaction `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.

Parent topic: Overview

Restore Operations

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 synch 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 into the database until the restore is complete.

Parent topic: Overview

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 4.0.0-566

Backup one database and associated metadata on a Yellowbrick appliance to the speci

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

`-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`

Command-Line Options	Environment Variable	Description	Example

`-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=********`

Parent topic: Overview

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 in remote data centers. (*Not supported in 4.0 Beta.*)

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

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.

Parent topic: Overview

ybbackup Commands

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

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
```

Only local or locally mounted file systems are supported. Relative paths are allowed.

The same rules 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.

In This Section

- ybbackup Options
- ybbackup Examples

Parent topic: Backup and Restore

ybbackup Options

This section contains detailed descriptions of the `ybbackup` command-line options. See also ybbackup Examples and 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.

--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 Enabling SSL/TLS Encryption.

-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.

-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.

--disable-trust, -k

Disable SSL/TLS trust when using secured communications. Trust is enabled by default. See also Enabling 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 appliance.

--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.

-?, --help, --help-advanced

Return help text for all of the `ybbackup` options.

-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.

--initial-connection-timeout NUMBER_MIN_0

Specify the number of seconds to wait for initial connections to the database. Specify `0` to wait indefinitely. Default: `120`.

--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 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 for the backup operation. 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 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.

--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.

-p, --port

Specify the database server port. Alternatively, set this value with the `YBPORT` environment variable.

Default: `5432`

-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`.

--secured

Use SSL/TLS to secure all communications. The default is not secured. See also Enabling SSL/TLS Encryption.

-U, --username

Specify the database user running the backup. Alternatively, set this value with the `YBUSER` environment variable.

--version

Return the software version for the backup tools.

-W, --password

Request an interactive 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.

Parent topic: ybbackup Commands

ybbackup Examples

This section contains some simple examples of backup commands, using different options.

Run a 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` (must be a superuser).
`-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 directory (no option name required).

Note: The host is not set explicitly in this case. It is set with the `YBHOST` environment variable.

Here is the command line and the full output of the backup operation:

```
~$ ybbackup -W --username yb100 -d premdb --full --name Oct23Full --chain October201
Password for user yb100:
15:16:16.003 [ INFO] ABOUT CLIENT:
   app.cli_args         = "-W" "--username" "yb100" "-d" "premdb" "--full" "--name"
   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, mixe
    jvm.options          = "-Xms512m, -Xmx4g, -XX:+UseG1GC, -Dapp.name=ybbackup2, -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/prem
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

ybbackup_data | 100.0% |       Read: 950.68 MiB / 950.68 MiB       |    Write: 950.9
```

Run an 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
Password for user yb100:
15:58:33.395 [ INFO] ABOUT CLIENT:
...
15:58:33.396 [ INFO] Running backup using output locations: [file:///home/yb100/prem
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

ybbackup_data | 100.0% |       Read: 951.68 MiB / 951.68 MiB       |    Write: 951.9
```

Run a backup to 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 --chai
/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.

Run a 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
```

Parent topic: ybbackup Commands

ybrestore Commands

Each `ybrestore` command defines:

- Database connection options
- 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 appliance. 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.

In This Section

- ybrestore Options
- ybrestore Examples
- Running Incremental Restores
- Restoring Users and Roles

Parent topic: Backup and Restore

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.

To see all of these options in the online help, run the `ybrestore --help` and `ybrestore --help-advanced` commands.

--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 Enabling SSL/TLS Encryption.

-c STRING, --chain STRING

Name the backup chain to use for the restore. See ybbackup Options for information about backup chains.

-d, --dbname STRING

Name the database to restore. The named database must not already exist on the target system.

--disable-trust, -k

Disable SSL/TLS trust when using secured communications. Trust is enabled by default. See also Enabling 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 appliance.

-?, --help, --help-advanced

Return help text for all of the `ybrestore` options.

-h, --host

The database server host. 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.

--initial-connection-timeout NUMBER_MIN_0

Specify the number of seconds to wait for initial connections to the database. Specify `0` to wait indefinitely. Default: `120`.

--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 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 for the restore operation. 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 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.

-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`

-p, --port

The database server port. Alternatively, set this value with the `YBPORT` environment variable. Default: `5432`

--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.

-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`.

--secured

Use SSL/TLS to secure all communications. The default is not secured. See also Enabling SSL/TLS Encryption.

--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.*

-U, --username

Specify the database user running the restore. Alternatively, set this value with the `YBUSER` environment variable.

--version

Return the software version for the backup tools.

-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.

Parent topic: ybrestore Commands

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 must be a valid backup name for the database you want to restore: `October15Inc3` in this case.
`--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.)

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 Oct
Password for user yb100:
17:44:56.066 [ INFO] ABOUT CLIENT:
   app.cli_args         = "-W" "--username" "yb100" "-d" "premdb_restored" "--point"
   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, mixe
   jvm.options          = "-Xms512m, -Xmx4g, -XX:+UseG1GC, -Dapp.name=ybrestore2, -D
   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/prem
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: 21

ybrestore_data | 100.0% |       Read: 284.73 KiB / 284.73 KiB       |    Write: 284
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: 6

ybrestore_data | 100.0% |         Read: 7.98 MiB / 7.98 MiB         |     Write: 7.9
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: 1
```

```
ybrestore_data | 100.0% |       Read: 225.88 MiB / 225.88 MiB       |    Write: 225
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: 1

ybrestore_data | 100.0% |        Read: 34.64 MiB / 34.64 MiB        |    Write: 34.6
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 Decembe
--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 October15I
Password for user yb100:
16:49:11.963 [ INFO] ABOUT CLIENT:
...
```

```
16:49:11.965 [ INFO] Running restore using input locations: [file:/home/brumsby/pre
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 'Octobe
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 Octobe
/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/p
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 Oct
Password for user yb100:
18:45:18.752 [ INFO] ABOUT CLIENT:
...
18:45:18.754 [ INFO] Running restore using input locations: [file:/home/brumsby/pre
```

```
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 datal
```

Parent topic: ybrestore Commands

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 appliance or with a different name on the same source appliance. 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.

Parent topic: ybrestore Commands

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_appliance_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.

Parent topic: ybrestore Commands

ybbackupctl Commands

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.

In This Section

- ybbackupctl Options
- ybbackupctl Examples

Parent topic: Backup and Restore

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 Storage Locations for Backups.

-c STRING, --chain STRING

The backup chain to use. Default: `default`.

-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.

-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, -p FULL | INVALID

Purge either obsolete full backups or invalid backups from the backup set. 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). You must specify `FULL` or `INVALID` for the purge command. You also need to specify the following options:

- `--chain` (if not using the `default` chain)
- `--uuid` to identify the database

See also -D, --delete.

--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`.

Parent topic: ybbackupctl Commands

ybbackupctl Examples

The following examples show how to use the `ybbackupctl` tool to list, delete, purge, or copy backups.

Example: 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, mixe
   jvm.options          = "-Xms512m, -Xmx4g, -XX:+UseG1GC, -Dapp.name=ybbackupctl, -
   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
Database: premdb (057af4c6-b817-4181-0b51-0f7e65df3df0)
Chain: October2019
   Tables                                                 size    deleted ro
   (6 total)                                              (GB)         (tota
     public.hometeam                                      0.00
     public.awayteam                                      0.00
     public.season                                        0.00
     public.match                                         0.04
     public.team                                          0.00
     public.newmatchstats                                 0.24

Restore points:

       type                          snapshot_name            backup_timestamp
       FULL                          October15Full 2019-10-15T22:16:23.108024Z
 INCREMENTAL                          October15Inc1 2019-10-15T22:20:42.999975Z
 INCREMENTAL                          October15Inc2 2019-10-15T22:55:10.787697Z
 INCREMENTAL                          October15Inc3 2019-10-15T23:02:15.944328Z
```

Example: 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-b7e926db03e
21:43:16.743 [ INFO] ABOUT CLIENT:
   app.cli_args         = "--delete" "--point" "Feb16Inc" "--uuid" "fed0f424-94c8-3
   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, mix
   jvm.options          = "-Xms512m, -Xmx8g, -XX:+UseG1GC, -Dapp.name=ybbackupctl,
   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/yb
21:43:17.989 [ INFO] Removing backup:[Feb16Inc] from database:[premdb/fed0f424-94c8
```

Example: 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 --
21:47:59.726 [ INFO] ABOUT CLIENT:
...
21:47:59.728 [ INFO] Running backup control using input locations: [file:///home/yb
21:48:01.347 [ INFO] Removing backup [Feb14Full/FULL] from backup chain: [February2
```

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 0b
17:47:31.597 [ INFO] ABOUT CLIENT:
   app.cli_args         = "--purge" "INVALID" "/home/yb100/premdb_backups" "--chain
   app.name_and_version = "Yellowbrick Backup and Restore version 4.0.0-913"
...
Removing backup [Sept9_2019_IncAgain/INCREMENTAL] from backup chain: [Sept2019] tak
```

Example: 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/CopyPremdbFeb
20:59:47.391 [ INFO] ABOUT CLIENT:
   app.cli_args         = "--copy-to" "/home/yb100/CopyPremdbFeb18Full" "--point" "
   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, mix
   jvm.options          = "-Xms512m, -Xmx8g, -XX:+UseG1GC, -Dapp.name=ybbackupctl,
   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/yb10
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  me
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). In other words, 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 Fe
20:59:47.391 [ INFO] ABOUT CLIENT:
...
```

Example: 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, mix
   jvm.options          = "-Xms512m, -Xmx8g, -XX:+UseG1GC, -Dapp.name=ybbackupctl,
   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/yb
22:45:48.546 [ INFO] Found bundle. Locations stored in the bundle=[file:///home/yb1
22:45:48.875 [ INFO] Verifying files for [Database=premdb] [Backup chain=February20
22:45:48.875 [ INFO] Verifying backup file: [relative path=/0f/be/5d/2f/bc/0fbe5d2f
22:45:48.876 [ WARN] [File=/0f/be/5d/2f/bc/0fbe5d2f-bcc2-4ab0-a804-83d739f81ff6] was
22:45:48.888 [ WARN] [File=/0f/be/5d/2f/bc/0fbe5d2f-bcc2-4ab0-a804-83d739f81ff6] was
22:45:48.896 [ INFO] Verifying backup file: [relative path=/4d/4d/af/05/a8/4d4daf05
22:45:48.896 [ WARN] [File=/4d/4d/af/05/a8/4d4daf05-a837-47a9-8228-9d6aa8702014] was
22:45:48.901 [ WARN] [File=/4d/4d/af/05/a8/4d4daf05-a837-47a9-8228-9d6aa8702014] was
22:45:48.903 [ INFO] Verifying backup file: [relative path=/66/cb/96/f9/aa/66cb96f9
...
```

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/yb
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=February202
22:41:32.905 [ INFO] Verifying backup file: [relative path=/78/fc/a3/8b/d2/78fca38b-
22:41:32.905 [ INFO] [File=/78/fc/a3/8b/d2/78fca38b-d2e3-4e86-81cf-2bb10a800f45] was
...
```

Parent topic: ybbackupctl Commands

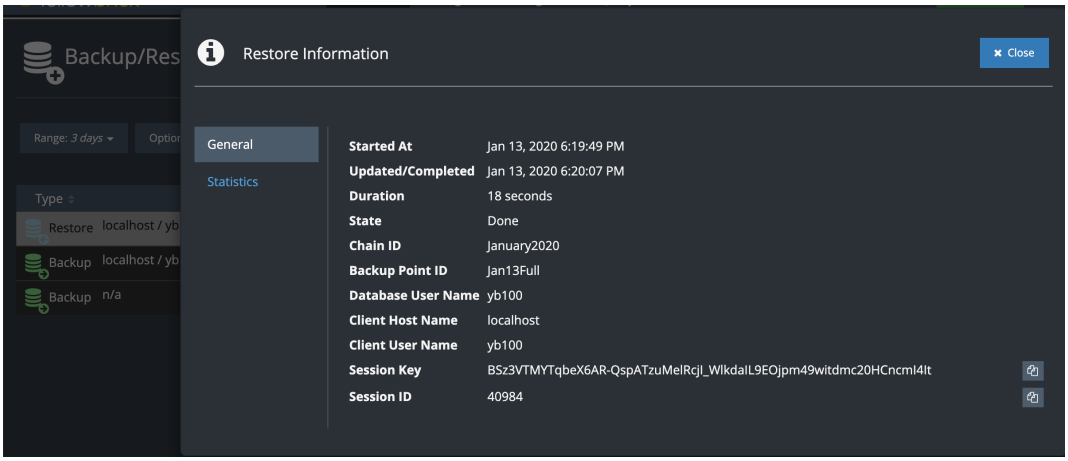
Monitoring Backup and Restore Operations

You can monitor backup and restore operations by running system view queries or by looking at the reports in the SMC.

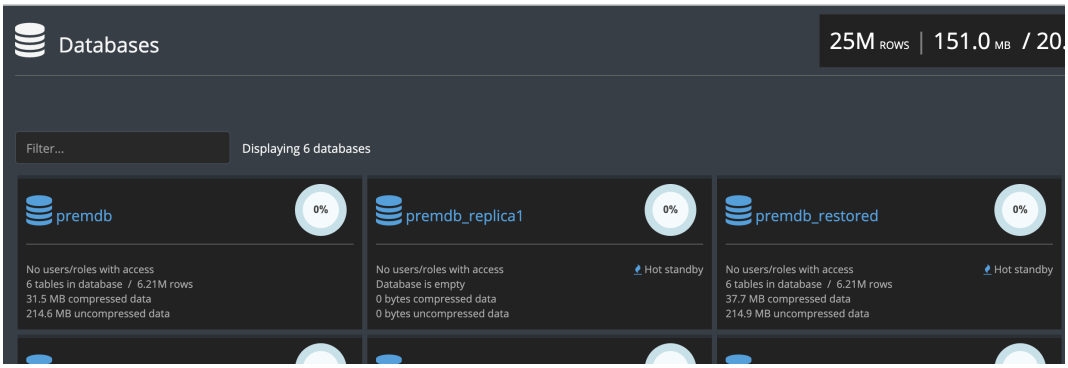
SMC Monitoring

Go to **Monitor** > **Backup/Restore Processes** in the SMC. For example:

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.

Backup and restore operations that use the pre-Version 4.0 legacy tools cannot be monitored in the SMC. Use the legacy system views for this purpose:

- sys.backup_legacy
- sys.log_backup_legacy
- sys.restore_legacy
- sys.log_restore_legacy

Parent topic: Backup and Restore

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 into 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 into 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, c
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
...
```

Parent topic: Backup and Restore

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
```

As long as 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
CREATE EXTERNAL TABLE	No	Yes	No
GRANT, REVOKE on objects	Yes	No	No
GRANT, REVOKE on roles, users	Yes	Yes	Yes

Operation	HOT_STANDBY State	READONLY State	Both States
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.

Parent topic: Backup and Restore

Database Replication

Asynchronous database replication is the process of maintaining a replica of a physical Yellowbrick database on a remote Yellowbrick appliance. 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 the event 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 single-chassis Yellowbrick appliance to a 3-chassis appliance.
- Adding reporting horsepower: for example, replication supports two appliances 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 on the manager node. The changes are automatically streamed to the database replica in the target system at regular intervals.

The following sections explain how to set up appliances for replication, create database replicas, and start the replication of specific databases.

In This Section

- Overview
- Setting Up Replication
- Monitoring Replication
- Managing Replication
- Replication SQL Commands

- Replication Glossary

Parent topic: Yellowbrick Documentation

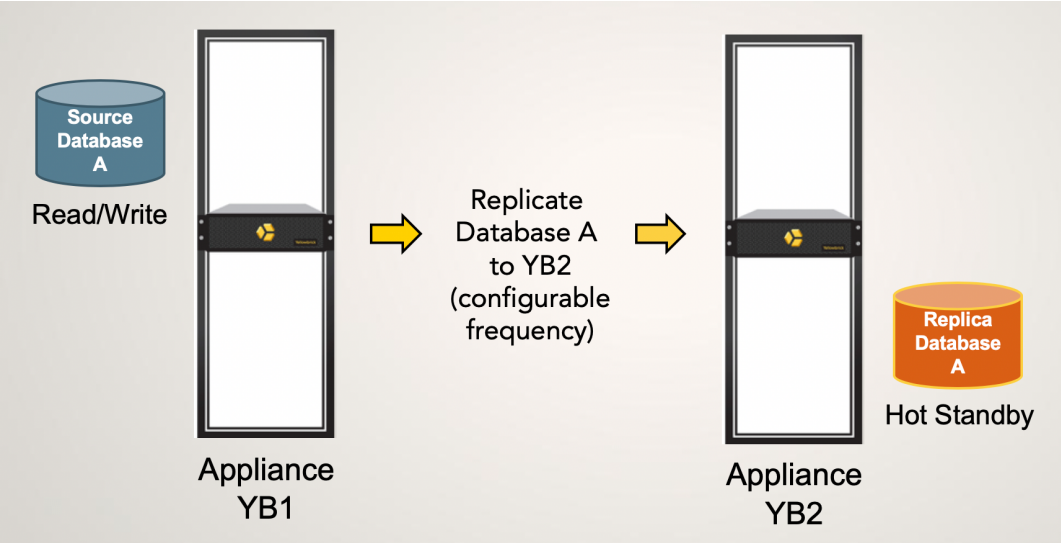
Overview

This section introduces concepts, terms, and behavior that you need to be familiar with before attempting to replicate Yellowbrick databases.

System Requirements

Note the following requirements for setting up and using database replication:

- You will need two Yellowbrick Data appliances to set up replication. However, you can also set up "loopback replication" into an aliased database on the same system where the source database resides.
- You can replicate any number of databases from one appliance 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 appliance 1 to database B on appliance 2, but you cannot in turn replicate database B on appliance 2 to database C on appliance 3.
- Source and target systems need not have the same topology. For example, you can replicate a database from a dual-chassis appliance to a single-chassis appliance or vice versa.



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. LDAP synchronization is supported from source to target and vice versa.

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 system. 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 appliance 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 either in the SMC or 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.

Parent topic: Database Replication

Setting Up Replication

This section explains how to set up replication between two Yellowbrick appliances: a source system and a target system. The specific commands that you need to run are listed alphabetically as part of the main SQL Commands section.

To set up replication on two Yellowbrick appliances:

1. Make sure that the appliances 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 appliance. This appliance will contain 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. Log into the SMC, then go to **Monitor** > **Replication Processes**. Select the replica you created and check that its status is `Running`.

8. Monitor replication cycles by using the SMC or running queries against the sys.replica_status and sys.log_replica_status views.

You can see replication status in the SMC via a tabular at-a-glance view for all current replicas or a detailed view per replica. The second view contains a replication performance history. See Monitoring Replication.

In This Section

- Setting Up Loopback Replication
- Configuring SSL Trust
- Seeding a Replica

Parent topic: Database Replication

Setting Up Loopback 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 appliance 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 tru
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
------------------+-------------------+------------------------+------+--------------
            16419 | yb100local        | localhost              | 5432 |       [NULL]
(1 row)
```

If you are using the default self-signed certificate from the appliance 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. Log into the SMC, then go to **Monitor** > **Replication Processes**. Select the replica you created and check that its status is `Running`.

6. Monitor replication cycles by using the SMC or running queries against the sys.replica_status `and` sys.log_replica_status views.

You can see replication status in the SMC via a tabular at-a-glance view for all current replicas or a detailed view per replica. The second view contains a replication performance history. See Monitoring Replication.

Parent topic: Setting Up Replication

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 for the appliance using a well-known Certificate Authority (CA).
- Import the self-signed certificate from the appliance into the client trust store configuration. The recommended procedure is to install an SSL certificate for the system, which 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 SMC 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 into 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 the SMC 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" -L repl_src.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_src.pem -text -noout
Certificate:
    Data:
```

```
        Version: 3 (0x2)
        Serial Number: 10443218342509856973 (0x90edc3c08124c8cd)
...
```

2. **On the source system**, log into 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'# ZTgtNDNlYi1hZDUwLWRiYjk2NDU3ZjVhYTEfMB0GA1UEAwwWYnJ1bXNieS55ZWxs
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=# 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-
            | subject= /C=US/ST=CA/O=Yellowbrick Data, Inc./OU=b5853632-1de8-43eb
            | 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 trust store.

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 trust store of the service.

1. **On the source system**, log into 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" -L 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 into 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 (via **Configure** > **Settings** > **SSL** > **Import Certificates** in the SMC).
4. Establish trust for the new certificates by following the procedure in the previous section.
5. Resume replication. (See Pausing and Resuming Replication.)

Parent topic: Setting Up Replication

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 appliances. 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 appliances 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 with Version 4.0 backup and restore for some time and you have one or more other appliances available for replication purposes. Note that 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 February20
```

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 Febr
```

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 appliance. *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_poi
-------------+--------------+--------+------------------------+--------------------
       20309 | February2020 | {}     | Feb13Full              | [NULL]
(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 in the usual way via the SMC or by querying the system views. Check that the initial replication cycles only replicate changes made *after* the target database was restored.

Parent topic: Setting Up Replication

Monitoring Replication

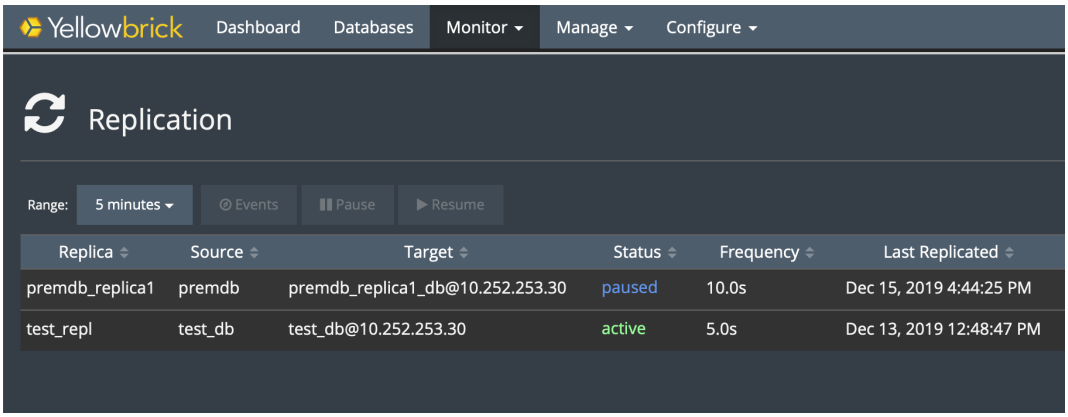
You can monitor replication by running system view queries or using the Monitor screens in the SMC. You can also monitor replication operations from a WLM perspective.

SMC Monitoring

On the source system, go to **Monitor** > **Replication Processes** in the SMC.

Note: The SMC on the target system does not display any information about replication processes.

For example:



You can filter replication events, as well as pause and resume replication. Too see a full history of replication on a specific replica, select a replica from the list. For example:

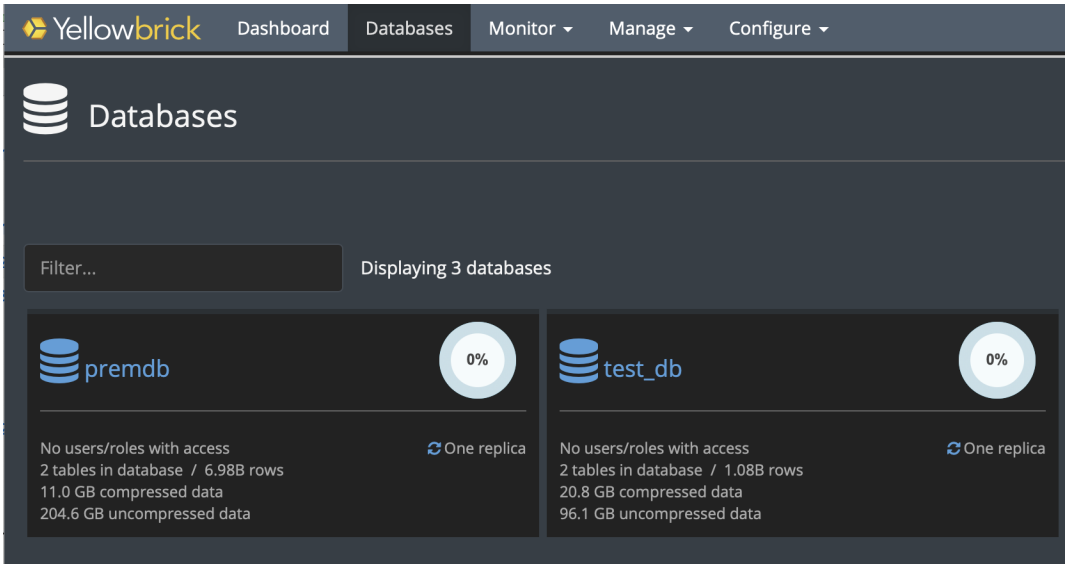
Range: *5 minutes* 140 total events

« 1 2 3 4 5 6 »

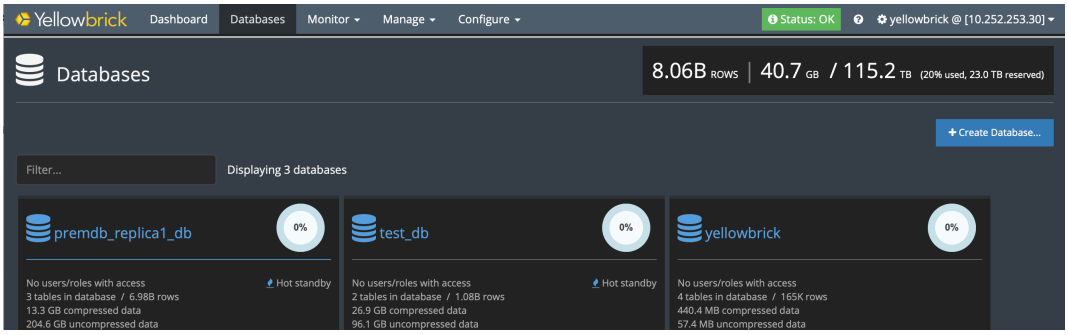
Replica	Start Time	End Time	Elapsed Time	State	Bytes Written
premdb_replica1	Dec 15, 2019 8:14:33 AM	Dec 15, 2019 8:14:40 AM	6 seconds	DONE	0 bytes
premdb_replica1	Dec 15, 2019 8:14:26 AM	Dec 15, 2019 8:14:33 AM	6 seconds	DONE	0 bytes
premdb_replica1	Dec 15, 2019 8:14:19 AM	Dec 15, 2019 8:14:26 AM	6 seconds	DONE	0 bytes
premdb_replica1	Dec 15, 2019 8:13:33 AM	Dec 15, 2019 8:14:19 AM	46 seconds	DONE	17.8 GB
premdb_replica1	Dec 15, 2019 8:13:06 AM	Dec 15, 2019 8:13:33 AM	26 seconds	DONE	8.9 GB
premdb_replica1	Dec 15, 2019 8:12:56 AM	Dec 15, 2019 8:13:03 AM	6 seconds	DONE	0 bytes
premdb_replica1	Dec 15, 2019 8:12:46 AM	Dec 15, 2019 8:12:53 AM	6 seconds	DONE	0 bytes
premdb_replica1	Dec 15, 2019 8:12:38 AM	Dec 15, 2019 8:12:44 AM	6 seconds	DONE	0 bytes
premdb_replica1	Dec 15, 2019 8:12:32 AM	Dec 15, 2019 8:12:38 AM	6 seconds	DONE	0 bytes
premdb_replica1	Dec 15, 2019 8:12:26 AM	Dec 15, 2019 8:12:32 AM	6 seconds	DONE	0 bytes
premdb_replica1	Dec 15, 2019 8:12:09 AM	Dec 15, 2019 8:12:26 AM	17 seconds	DONE	4.5 GB
premdb_replica1	Dec 15, 2019 8:11:56 AM	Dec 15, 2019 8:12:09 AM	12 seconds	DONE	2.2 GB
premdb_replica1	Dec 15, 2019 8:11:46 AM	Dec 15, 2019 8:11:56 AM	9 seconds	DONE	955.3 MB
premdb_replica1	Dec 15, 2019 8:11:36 AM	Dec 15, 2019 8:11:42 AM	5 seconds	DONE	0 bytes

Note that the elapsed time of a replication cycle fluctuates based on the amount of data that needs to be written to the replica database.

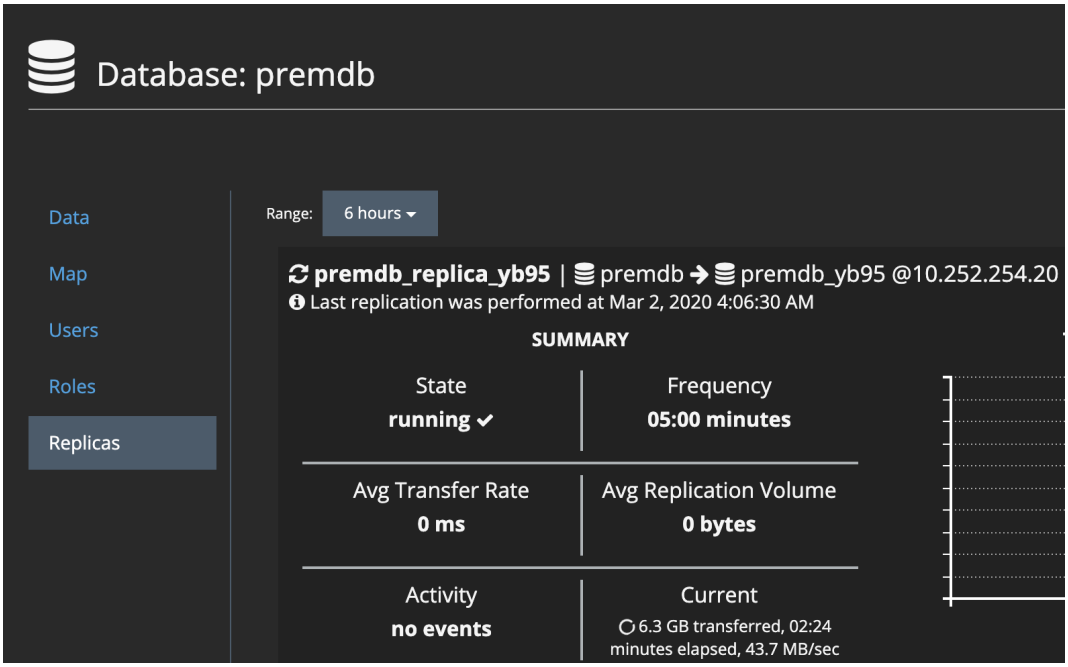
The Databases view in the SMC also shows which databases have replicas. For example:



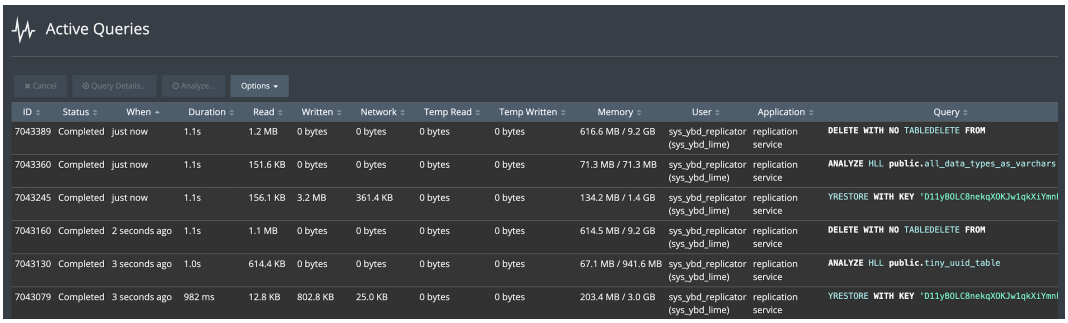
On the target system, this view shows that active replica databases are in hot-standby mode. For example:



If replication is in progress, you can see the current state of a replication cycle. Select a database that has a replica, then go to the **Replicas** tab. Note the statistics for the current cycle in the lower-right corner. For example:



To monitor replication at the table level while replication is in progress (or just completed), go to **Monitor** > **Active Queries**, then look for `YRESTORE WITH KEY` queries for the `replication service` application. For example:



By looking at the full query text, you will be able to see how much data is being written to specific tables. Hover the mouse over the **Written** value to display the number of rows. For complete information about a given operation, double-click the row and look at the **Query Details** tabs.

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'
```

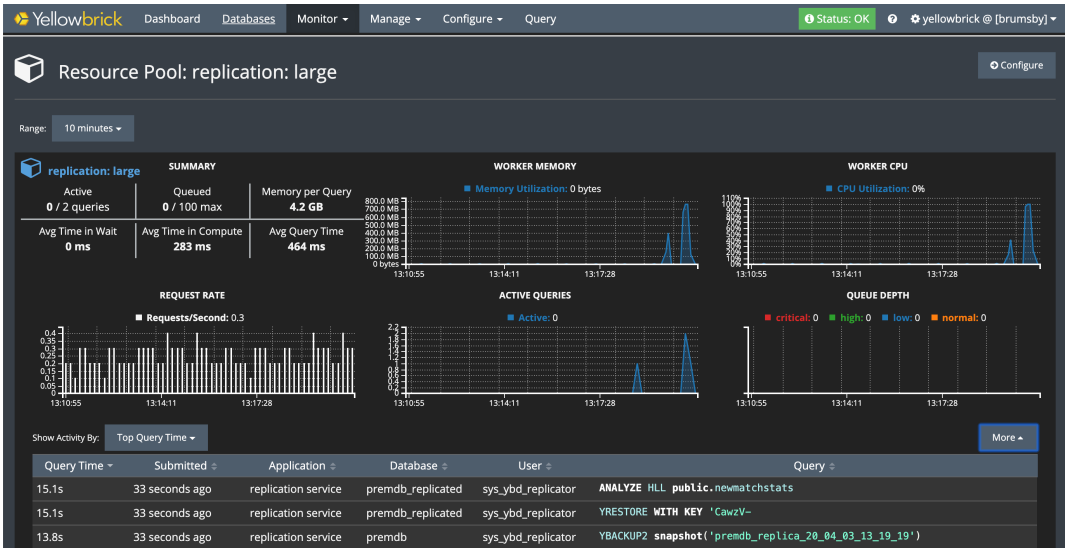
See the view descriptions for examples.

WLM Monitoring for Replication Operations

Via the SMC, you can monitor workload management (WLM) activity on both systems while replication is in progress. Go to **Monitor** > **Resource Pools**, then select the resource pool that is being used for replication.

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.

The following example shows the monitor for a loopback replication test. Replication is running in a user-defined WLM profile called `replication`, using the `large` resource pool. Note the application name and user name.



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)
```

Parent topic: Database Replication

Managing Replication

This section covers some tasks that you may need to perform to make changes to your replication setup.

In This Section

- Pausing and Resuming Replication
- Dropping a Replica
- Replication Failover and Failback
- Rolling Back a Replicated Database

Parent topic: Database Replication

Pausing and Resuming 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.

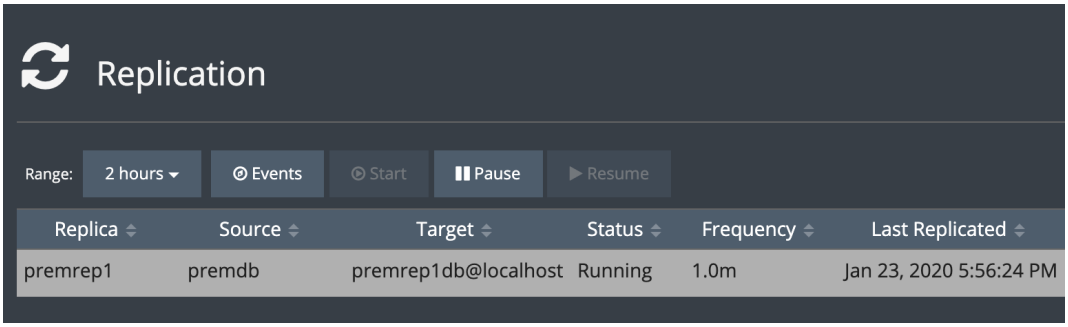
```
Total time (in seconds) = (compressed data size MB) / MIN(3200, Max Network Bandwidt
```

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 a lot 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 cluster 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 a lot of 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 or by going to **Monitor** > **Replication Processes** in the SMC. For example:



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.

Parent topic: Managing Replication

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.

1. On the source system, pause replication for a specific replica. You can do this in the SMC or 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.

Parent topic: Managing Replication

Replication Failover and Failback

This section contains procedures for failing over and failing back (in the context of using database replication). The example in this section is based on the following configuration of appliances, databases, and replicas:

Appliance	Remote Server	Source Database	Replica	Replica Database	Rever
ybwest (primary)	ybwest_svr	premdb	premdb_replica	premdb_replicated	N/A
ybeast (secondary)	ybeast_svr	premdb_replicated (after failover)	N/A	premdb (after failover)	revers

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 appliance. The primary appliance is `ybwest` and the secondary is `ybeast`. The `ybwest` system has a database called `premdb` that replicates data to a database called `premdb_replicated` on `ybeast`.

The following commands were run on the primary appliance, `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 appliance, `ybeast`.

Prepare for a Possible Failover Event on the Secondary System

To be ready in the event 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 into 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.

Fail Over: Promote the Secondary System

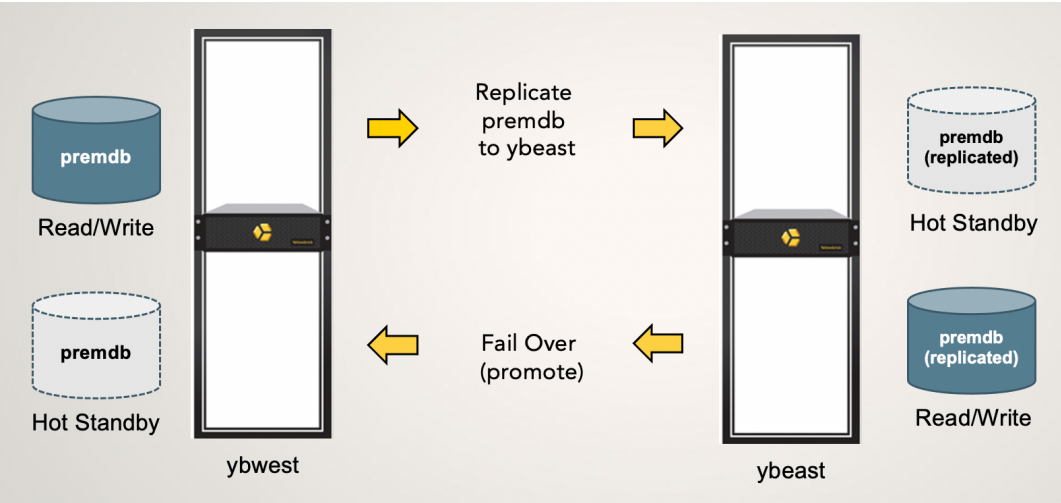
Assume that the `ybwest` appliance cannot function as the primary system for some reason, and that replication from `ybwest` to `ybeast` needs to be paused or has stopped running. You can now reverse the roles of the two systems and fail over to the secondary system, `ybeast`.

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 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` appliance 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

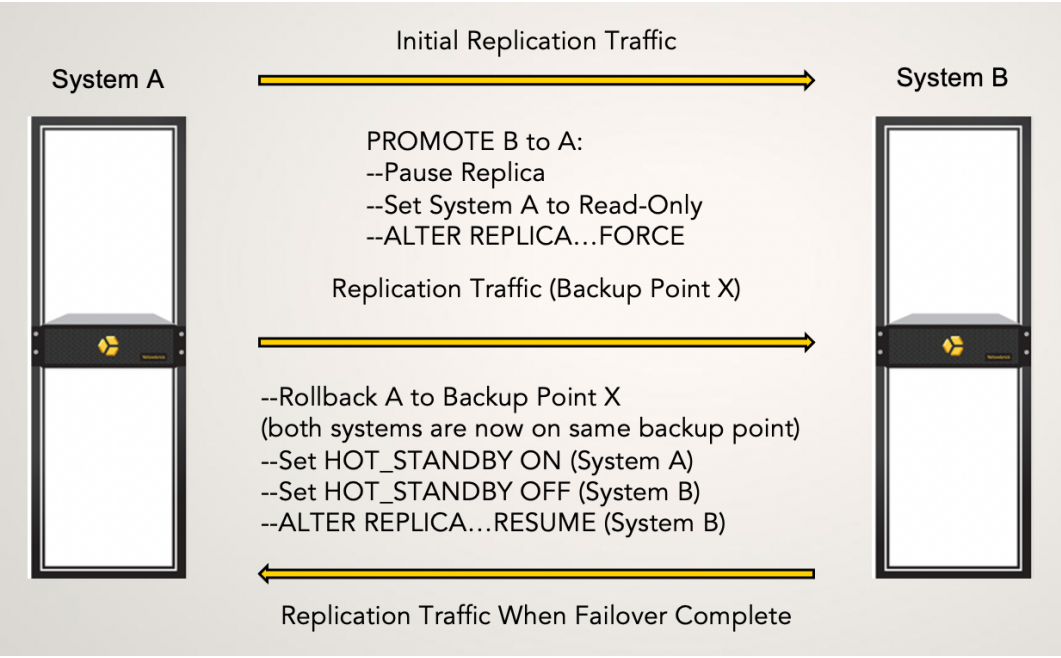
The `PROMOTE` command operates in two modes:

```
ALTER DATABASE name ALTER REPLICA name PROMOTE
or
ALTER DATABASE name ALTER REPLICA name PROMOTE WITH DATA LOSS
```

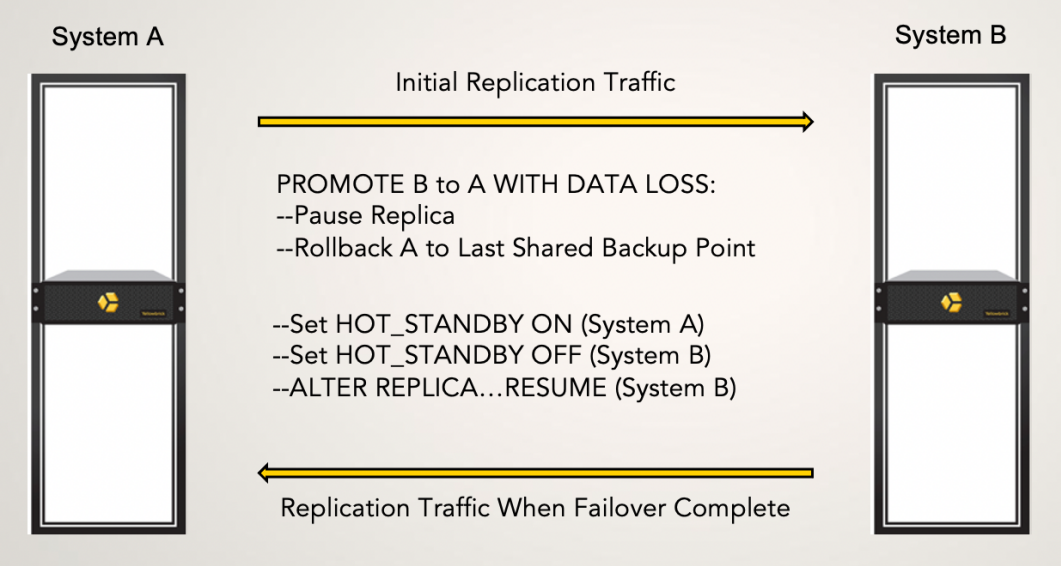
If you use the `PROMOTE` option (without data loss, as described in the previous section), the command tries to force replication from the primary to the secondary and rolls back the source database to a known good rollback point for both databases. If the database on the primary system cannot be reached 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 a number of 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.

Parent topic: Managing Replication

Rolling Back a Replicated Database

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.

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      |     olde
-------------+----------------+--------+----------------------------------+--------
       16393 | premdb_replica | {}     | premdb_replica_20_02_10_17_56_08 | premdb_
(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
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 synch. 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 whe
 count
-------
 34200
(1 row)
```

Source system:

```
premdb=# select count(*) from premdb.public.newmatchstats where seasonid=21;
 count
-------
 34200
(1 row)
```

Parent topic: Managing Replication

Replication SQL Commands

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 appliances used to replicate databases:

- IMPORT SSL TRUST
- SHOW SSL TRUST
- SHOW SSL SYSTEM
- SHOW SSL CA
- SHOW SSL IDENTITY
- REVOKE SSL TRUST

Parent topic: Database Replication

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 appliance. 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 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 appliance 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.

Parent topic: Database Replication

SQL Reference

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`.

In This Section

- SQL Data Types
- SQL Commands
- SQL Functions and Operators
- SQL Reserved Words
- SQL Quick Reference

Parent topic: Yellowbrick Documentation

SQL Data Types

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
UUID		128-bit generated value (a sequence of lowercase or uppercase hexadecimal digits)
VARCHAR	`CHARACTER VARYING`	Variable-length character string
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

Data Type	Aliases	Description
IPV4		IP address (4-byte)
IPV6		IP address (16-byte)
MACADDR		MAC address (6-byte)
MACADDR8		MAC address (8-byte, EUI-64 format)

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.

In This Section

- BIGINT
- BOOLEAN
- CHAR
- DATE
- DECIMAL
- DOUBLE PRECISION

- INTEGER
- IPV4
- IPV6
- MACADDR
- MACADDR8
- REAL
- SMALLINT
- TIME
- TIMESTAMP
- TIMESTAMP WITH TIME ZONE
- UUID
- VARCHAR
- Trailing Blanks in Character Data
- Data Type Casting
- SQL String Constants

Parent topic: SQL Reference

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 zeroes and whitespace characters are allowed. Trailing whitespace characters are also allowed.

Parent topic: SQL Data Types

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`.

Parent topic: SQL Data Types

CHAR

The CHAR data type is *conceptually* a fixed-length character string that is padded with trailing blanks.

Note: 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 representation of the column data.

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.

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.

Parent topic: SQL Data Types

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)
```

Parent topic: SQL Data Types

DECIMAL

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 zeroes 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.

Column Specification	Allowed Range
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 thar
```

In This Section

- Calculations with DECIMAL Values
- Examples of DECIMAL Calculations

Parent topic: SQL Data Types

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

Operation	Resulting Precision (p)	Resulting Scale (s)
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.

Parent topic: DECIMAL

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+123456789123456789123456789.123456789 from dec385;
ERROR:  numeric value out of range
```

This query will run if you manually cast the result:

```
premdb=# select c1+123456789123456789123456789.123456789::decimal(38,5) from dec38
               ?column?
-----------------------------------------
  955567901234467901234467901234467.67899
 100001234567880112334546779001213 4.23457
(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)
```

Parent topic: DECIMAL

DOUBLE PRECISION

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 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 zeroes 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` versus `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>
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)
```

Parent topic: SQL Data Types

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 zeroes and whitespace characters are allowed. Trailing whitespace characters are also allowed.

Parent topic: SQL Data Types

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.

Parent topic: SQL Data Types

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.

Parent topic: SQL Data Types

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.

Parent topic: SQL Data Types

MACADDR8

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.

Parent topic: SQL Data Types

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 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 zeroes 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` 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` versus `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>
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)
```

Parent topic: SQL Data Types

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 zeroes and whitespace characters are allowed. Trailing whitespace characters are also allowed.

Parent topic: SQL Data Types

TIME

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 values are 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)
```

Parent topic: SQL Data Types

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 timestamps are 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`).

Inserted Value	Stored Value	Notes
`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.

Parent topic: SQL Data Types

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 values are 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`).

Inserted Value	Stored Value	Notes
`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.

Parent topic: SQL Data Types

UUID

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-f04442a2-9000
             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 fill in the unknown digits with zeroes. For example:

```
premdb=# select * from uuidtest where c1 >= '653dfdba-0000-0000-0000-000000000000';
                c1
--------------------------------------
 653dfdba-f044-42a2-9000-199945f61dce
(1 row)
```

Parent topic: SQL Data Types

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.

```
{ 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:

```
premdb=# create table var(c1 varchar, c2 varchar(max), c3 varchar(any), c4 varchar(1
CREATE TABLE
premdb=# \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.

Parent topic: SQL Data Types

Trailing Blanks in Character Data

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 called, 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;
 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
 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 f
 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;
 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;
 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_nam
 seasonid | season_name  |      winners      | seasonid | season_name |      winners
----------+--------------+-------------------+----------+-------------+-------------
       20 | 2011-2012    | Manchester City   |       20 | 2011-2012   | Manchester Ci
       21 | 2012-2013    | Manchester United |       21 | 2012-2013   | Manchester Un
       22 | 2013-2014    | Manchester City   |       22 | 2013-2014   | Manchester Ci
       23 | 2014-2015    | Chelsea           |       23 | 2014-2015   | Chelsea
       24 | 2015-2016    | Leicester City    |       24 | 2015-2016   | Leicester Cit
(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.

Parent topic: SQL Data Types

Data Type Casting

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.

In This Section

- Explicit Casting
- Implicit Casting
- Implicit Casting Examples

Parent topic: SQL Data 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`.

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)
```

Parent topic: Data Type Casting

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
`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`

From Data Type	To Data Type(s)
`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`
`MACADDR8`	`VARCHAR`, `CHAR`
`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.

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.

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 ad
```

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.

- `TIMESTAMPTZ`
- `TIMESTAMP`
- `DATE`
- `INTERVAL`
- `TIME`
- `DOUBLE PRECISION` (`FLOAT8`)
- `REAL` (`FLOAT4`)
- `DECIMAL` (`NUMERIC`)
- `BIGINT`
- `INT`
- `SMALLINT`
- `BOOLEAN`
- `UUID`
- `IPV6`
- `IPV4`
- `MACADDR8`
- `MACADDR`
- `VARCHAR`
- `CHAR`
- `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`.

Expression	Rule Details
`NVL(<bool>, <expr0>)`	Return the type of `<expr0>`.
`NVL2(<bool>, <expr0>, <expr1>)`	Return the highest type of `<expr0>` and `<expr1>`.

`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.

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 w
   ->  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
                                                           distribute on (assign.b:
 ...
```

Parent topic: Data Type Casting

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=5
     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 chartim
                                                          distribute on (chartime.c
  3            100     57.00Mi H         0.00       all   SCAN chartime
                                                          (chartime.c2)
                                                          distribute on (chartime.c
...
```

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    |
----------+---------------------+------+------+---------+---------+----------+-----
       21 | 2012-12-22 00:00:00 |   44 |   78 | 2-1     | 1-1     | 00:00:00 | 2012-1
       21 | 2012-12-22 00:00:00 |   45 |   67 | 1-2     | 1-0     | 00:00:00 | 2012-1
       21 | 2012-12-22 00:00:00 |   46 |   51 | 0-1     | 0-0     | 00:00:00 | 2012-1
...
```

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.matchda
 13        8812544       all   |-SCAN match
                               |   match.matchday = bloom(2) AND scan_constraints: m
  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, varda
                                                          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
                                                              QUERY PLAN
-----------------------------------------------------------------------------------------
 id   rows_planned   mem_planned   mem_actual   workers   node
  1             3     32.98Mi H         0.00    single   SELECT
                                                          (team.teamid::VARCHAR(6)
                                                          distribute single
  2             3      4.00Mi H         0.00    single   LIMIT $4
                                                          (team.teamid::VARCHAR(6)
                                                          distribute single
  4           100     57.00Mi H         0.00    single   SCAN team
                                                          (team.teamid, team.name,
                                                          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
--------+------+------+-------------------+------------+------------+---------------
    25 |   26 |   75 | Manchester United | Red Devils | Manchester | Old Trafford |
(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(htsco
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 ty
```

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::IPV
                                                            distribute on (NULL)
   3              1    272.00Mi H          0.00       all   DISTRIBUTE ON HASH(NULL)
                                                            (NULL, ipv4_ipv6.c1::IPV
                                                            distribute on (NULL)
...
```

Parent topic: Data Type Casting

SQL 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)
```

In This Section

- C-style Escape Sequences in String Constants
- Unicode Escape Sequences in String Constants

- Escape Sequences for Non-Printable Characters

Parent topic: SQL Data Types

C-style Escape Sequences in String Constants

A string constant that has an `E` or `e` directly before the opening single quote 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
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

Backslash Escape Sequence	Interpretation
`\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 quote 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)
```

Parent topic: SQL String Constants

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 quote, a double quote, or a whitespace character.

To include the escape character in the string literally, type it twice.

Parent topic: SQL String Constants

Escape Sequences for Non-Printable Characters

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)***	Abbr
0	\0 *	\000	\x00	\u0000	\U000000	U+0000	NUL
1		\001	\x01	\u0001	\U000001	U+0001	SOH
2		\002	\x02	\u0002	\U000002	U+0002	STX
3		\003	\x03	\u0003	\U000003	U+0003	ETX
4		\004	\x04	\u0004	\U000004	U+0004	EOT
5		\005	\x05	\u0005	\U000005	U+0005	ENQ
6		\006	\x06	\u0006	\U000006	U+0006	ACK
7		\007	\x07	\u0007	\U000007	U+0007	BEL
8	\b	\010	\x08	\u0008	\U000008	U+0008	BS
9	\t	\011	\x09	\u0009	\U000009	U+0009	HT
10	\n	\012	\x0A	\u000A	\U00000A	U+000A	LF
11		\013	\x0B	\u000B	\U00000B	U+000B	VT
12	\f	\014	\x0C	\u000C	\U00000C	U+000C	FF
13	\r & \m	\015	\x0D	\u000D	\U00000D	U+000D	CR

Dec	Backslash Char	Backslash Octal	Hex	Backslash SQL Unicode (4) **	Backslash SQL Unicode (6) **	ybload Unicode (2,4,6,8)***	Abbr
14		\016	\x0E	\u000E	\U00000E	U+000E	SO
15		\017	\x0F	\u000F	\U00000F	U+000F	SI
16		\020	\x10	\u0010	\U000010	U+0010	DLE
17		\021	\x11	\u0011	\U000011	U+0011	DC1
18		\022	\x12	\u0012	\U000012	U+0012	DC2
19		\023	\x13	\u0013	\U000013	U+0013	DC3
20		\024	\x14	\u0014	\U000014	U+0014	DC4
21		\025	\x15	\u0015	\U000015	U+0015	NAK
22		\026	\x16	\u0016	\U000016	U+0016	SYN
23		\027	\x17	\u0017	\U000017	U+0017	ETB
24		\030	\x18	\u0018	\U000018	U+0018	CAN
25		\031	\x19	\u0019	\U000019	U+0019	EM
26		\032	\x1A	\u001A	\U00001A	U+001A	SUB
27		\033	\x1B	\u001B	\U00001B	U+001B	ESC

Dec	Backslash Char	Backslash Octal	Hex	Backslash SQL Unicode (4) **	Backslash SQL Unicode (6) **	ybload Unicode (2,4,6,8)***	Abbr
28		\034	\x1C	\u001C	\U00001C	U+001C	FS
29		\035	\x1D	\u001D	\U00001D	U+001D	GS
30	\rs*	\036	\x1E	\u001E	\U00001E	U+001E	RS
31	\us*	\037	\x1F	\u001F	\U00001F	U+001F	US
127		\177	\x7F	\u007F	\U00007F	U+007F	DEL

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.

Parent topic: SQL String Constants

SQL Commands

Yellowbrick supports the SQL commands that are described in this section. For background information, see Database Administration.

In This Section

- SQL Identifiers
- ABORT
- ALTER DATABASE
- ALTER DATABASE ADD REPLICA
- ALTER DATABASE ALTER REPLICA
- ALTER DATABASE DROP REPLICA
- ALTER DEFAULT PRIVILEGES
- ALTER PROCEDURE
- ALTER REMOTE SERVER
- ALTER ROLE
- ALTER SCHEMA
- ALTER SEQUENCE
- ALTER SYSTEM
- ALTER TABLE
- ALTER USER
- ALTER VIEW
- ALTER WLM PROFILE
- ALTER WLM RESOURCE POOL

- ALTER WLM RULE
- BEGIN
- CALL
- CANCEL
- CLOSE
- COMMENT ON
- COMMIT
- COPY
- CREATE DATABASE
- CREATE EXTERNAL MOUNT
- CREATE EXTERNAL TABLE
- CREATE KEY
- CREATE PROCEDURE
- CREATE REMOTE SERVER
- CREATE ROLE
- CREATE SEQUENCE
- CREATE SCHEMA
- CREATE TABLE
- 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 DATABASE
- DROP EXTERNAL MOUNT
- 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
- GRANT
- HELP
- IMPORT SSL TRUST
- INSERT
- LOCK SEQUENCE
- MOVE cursor
- MOVE query
- PREPARE
- REASSIGN OWNED BY
- RELEASE SAVEPOINT
- RESET
- RESTART query
- REVOKE
- REVOKE SSL TRUST
- ROLLBACK
- ROLLBACK DATABASE
- ROLLBACK TO SAVEPOINT
- SAVEPOINT
- SELECT
- SELECT INTO
- SET

- SET ROLE
- SET SESSION AUTHORIZATION
- SHOW
- SHOW SSL CA
- SHOW SSL IDENTITY
- SHOW SSL SYSTEM
- SHOW SSL TRUST
- START TRANSACTION
- TABLE
- TRUNCATE
- UPDATE

Parent topic: SQL Reference

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)
```

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..."
...
```

Parent topic: SQL Commands

ABORT

Roll back the current transaction.

```
ABORT [ WORK | TRANSACTION ]
```

`WORK` and `TRANSACTION` are both optional keywords; they have no effect.

See also ROLLBACK.

Parent topic: SQL Commands

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 | SECONI
```

See also ALTER DATABASE ADD REPLICA, ALTER DATABASE ALTER REPLICA, and ALTER DATABASE DROP REPLICA.

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. See HOT_STANDBY and READONLY Modes.

`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.

See HOT_STANDBY and READONLY Modes.

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
```

Parent topic: SQL Commands

ALTER DATABASE ADD REPLICA

Create a replica for a source database. You must be a superuser to run this command.

```
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 would like 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
```

Parent topic: SQL Commands

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 `NEW 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 DATA
...
ERROR: Failed to promote replica
DETAIL: Unable to promote replica “repldb2_failover” Reason: ... ROLLBACK DATA
```

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 RE
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.

Parent topic: SQL Commands

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 curre
```

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
```

Parent topic: SQL Commands

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. Keep in mind 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

For example, grant all privileges to `bobr` on new tables and views that the current user creates:

```
premdb=# alter default privileges grant all on tables to bobr;
ALTER DEFAULT PRIVILEGES
```

Grant `INSERT` privileges to `bobr` on all new tables created by `yellowbrick`:

```
premdb=# alter default privileges for role yellowbrick grant insert on tables to bo
ALTER DEFAULT PRIVILEGES
premdb=# create table newtable(c1 int);
CREATE TABLE
premdb=# \c premdb bobr
Password for user bobr:
You are now connected to database "premdb" as user "bobr".
premdb=> select * from newtable;
ERROR:  permission denied for relation newtable
premdb=> insert into newtable values(1);
INSERT 0 1
```

You can use the `ybsql \ddp` command to display default access privileges.

```
premdb=> \ddp
                 Default access privileges
   Owner    | Schema | Type  |        Access privileges
------------+--------+-------+---------------------------------
 yellowbrick |        | table | yellowbrick=arwdDxt/yellowbrick+
            |        |       | bobr=a/yellowbrick
(1 row)
```

In this case, `bobr` has INSERT privilege (`bobr=a`) on tables owned by `yellowbrick`.

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 a superuser and owner of the database `newdb`. The users `dbuser2` and `dbuser3` belong to a role named `dbrole`.

```
premdb=# create user dbuser1 superuser login password 'dbuser1';
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
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).

Parent topic: SQL Commands

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)
```

Parent topic: SQL Commands

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.

Parent topic: SQL Commands

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). For example, you can alter the password for the `yellowbrick` user but nothing else. 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 clear 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.

Examples

```
premdb=# alter role henry connection limit 10;
```

```
premdb=# alter role all set search_path to 'premdb';
```

Parent topic: SQL Commands

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 prembd set max_size='1GB';
ALTER SCHEMA
```

Parent topic: SQL Commands

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
```

Parent topic: SQL Commands

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 superuser to run this command, and 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 (via ybcli commands).

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 `yellowbrick` or other 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
ALTER SYSTEM
```

After a reload (or a server restart, though a restart is not needed in this specific case), user `bobr` and all other non-superusers 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, c
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 superuser:

```
premdb=# alter system reset idle_session_timeout;
WARNING:  Will be effective after the next server configuration reload, or after the
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.

Parent topic: SQL Commands

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 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 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 a number of 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)
```

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
```

Parent topic: SQL Commands

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.

Parent topic: SQL Commands

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
```

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.

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) |
```

Parent topic: SQL Commands

ALTER WLM PROFILE

Make a workload management (WLM) profile active or inactive or rename the profile.

```
ALTER WLM PROFILE name
( DEFAULT_POOL name ) |
ACTIVATE [ seconds ] [ WITH CANCEL | WITHOUT CANCEL ] |
RENAME TO name
```

Only superusers 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 as a number of seconds. If you activate immediately, any running queries are cancelled. If you specify an interval, active queries have some time to complete before the new profile is activated.

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 cancelling 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.

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.

Parent topic: SQL Commands

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 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
```

Parent topic: SQL Commands

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 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
```

Parent topic: SQL Commands

BEGIN

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
```

Parent topic: SQL Commands

CALL

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.

Parent topic: SQL Commands

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 the SMC. See also Canceling Queries.

Parent topic: SQL Commands

CLOSE

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
```

Parent topic: SQL Commands

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 sea
COMMENT
```

Parent topic: SQL Commands

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
```

Parent topic: SQL Commands

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
...
```

Parent topic: SQL Commands

CREATE DATABASE

Create a new physical database.

You can create a maximum of 1,000 databases per appliance. (The `yellowbrick` database is included in this number.)

```
CREATE DATABASE name
   [ [ WITH ] [ OWNER [=] user_name ]
          [ ENCODING [=] encoding ]
          [ 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 the System Management Console (SMC) to create databases. When you create a database in the SMC, 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.

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. For more details, see HOT_STANDBY and READONLY Modes and Database Replication.

Note: You can run queries that reference tables in different databases. See Cross-Database Queries.

Examples

For example:

```
premdb=# create database testdb encoding=latin9;
CREATE DATABASE
```

```
premdb=# create database mydb allow_connections=false;
CREATE DATABASE
```

```
premdb=# create database premdb connection limit 100;
CREATE DATABASE
```

```
premdb=# create database shakespeare with owner=ybd encoding=LATIN1;
ERROR:  Only LATIN9 and UTF8 database encodings are supported
```

```
yellowbrick=# create database premdb_hot with hot_standby on;
CREATE DATABASE
```

Parent topic: SQL Commands

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;
```

Parent topic: SQL Commands

CREATE EXTERNAL TABLE

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.

In This Section

- USING Options
- External Table Examples

Parent topic: SQL Commands

USING Options

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 v data: `1` or `0`, `T` c `FALSE`. This option i `ybload` automatical

Option name	Accepted values	Default value	Data type	Description
`compress`	`true, false, TYPE`	Auto-detected (based on file suffix)	Boolean	Define the compressio specified without a valu applied for `SELECT` - If this option is not inc is auto-detected based - You can explicitly spe type values differ for `[--source-compres (../bulk_loading/ compression)` and `(../unloading/ybu`
`datedelim`	`CHARACTER`	Auto-detected	String	Specify the delimiter fo `ybload --date-fi`
`datestyle`	`FORMAT`	`YMD`	String	Specify the format for of the supported date s accepted. Only one da
`delim` or `delimiter`	`SPECIAL_CHARACTER`	Auto-detected	String	Specify the field delimi delimiter option in `yb`
`encoding`	`ENCODING_NAME`	Server encoding	String	Equivalent to `ybloa (../bulk_loading/` option does not apply t
`escapechar`	`SPECIAL_CHARACTER`	Depends on the `format` option	String	Use this character as t char option in `ybloa`

Option name	Accepted values	Default value	Data type	Description
`fillrecord`	`true` , `false`	`false` (`ERROR`)	Boolean	Define the behavior wh `null` value or returr `ybload [--on-mis` `(../bulk_loading/` `SUPPLYNULL` . This op `TABLE` operations.
`format`	`csv` , `text` , `bcp`	`csv`	String	Define the format of th `BCP` (see the `yblo` `(../bulk_loading/` details). If this option is is specified without a v returns an error. This c `TABLE` operations.
`ignorezero`	`true` , `false`	`false` (`ERROR`)	Boolean	Remove null bytes (`0` appear within strings in `true` is equivalent to `(../bulk_loading/` `REMOVE` . This option operations.
`includeheader`	`true` , `false`	`false`	Boolean	`TRUE` : Write a single `FALSE` : Do not inclu option applies to `CRE`

Option name	Accepted values	Default value	Data type	Description
`logdir`	`PATH`	Same location as external files	String	Set a path to a writable point (see CREATE EX `logdir '/qumulo/external` If this option is not spe to the same directory a only NFS mount for the
`maxerrors`	`NUMBER`	0	Integer	Set the maximum num `ybload [--max-ba (../bulk_loading/ NUMBER`. **Note:** `max` the load will fail on the This option does not a operations.
`notruncate`	`true`, `false`	`false`	Boolean	Preserve or overwrite file exists with the spec error. `FALSE` is the d overwritten. This optio operations only.
`nullvalue`	`STRING`	`NULL`	String	Specify the string that to the --nullmarker opti

Option name	Accepted values	Default value	Data type	Description
`quotedvalue`	`yes`, `no`, `single`, `double`, `SPECIAL_CHARACTER`	"	String	Use this character as t Equivalent to the --quo `ybload`. **Note:** The quote (`'`). The value (`"`), which is the sam specified.
`recorddelim`	`SPECIAL_CHARACTER`	Auto-detected	String	Define the record delin source files. Equivalen `(../bulk_loading/` does not apply to `CRE`
`skiprows`	`NUMBER`	0	Integer	Skip the specified num external file. This optio header lines. This opti `TABLE` operations.
`timedelim`	`CHARACTER`	:	String	Specify the delimiter cl value. The default is th specified in combinatio `ybload --time-fi options`. (See ybloa
`timestyle`	`'24HOUR'`, `'12HOUR'`	`'24HOUR'`	String	Specify the time forma is `'24HOUR'`. This o tables; `ybload` auto
`truncstring`	`true`, `false`	`false`	Boolean	Truncate strings that a return an error. Equiva `(../bulk_loading/` `true` is equivalent t `error`. This option operations.

Option name	Accepted values	Default value	Data type	Description
`y2base`	`YEAR` (`0-9999`)	No default	integer	Equivalent to the `ybl` `(../bulk_loading/` option does not apply t

Parent topic: CREATE EXTERNAL TABLE

External Table Examples

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, d
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, mixe
   jvm.options          = "-Xms512m, -Xmx6g, -XX:+UseG1GC, -Dlogfile=/home/brumsby/y
app.pid=21892, -Dapp.repo=/home/brumsby/ybd/external/build/lib, -Dapp.home=/home/bru
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_d
2019-06-05 13:58:50.756 [ INFO] <pool-6-thread-1>  state: RUNNING - Network BW: 5.5
2019-06-05 13:58:51.147 [ INFO] <unload-4>  Network I/O Complete. Waiting on file I
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 B
```

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, mix
   jvm.options          = "-Xms512m, -Xmx6g, -XX:+UseG1GC, -Dlogfile=/home/brumsby/
app.pid=21892, -Dapp.repo=/home/brumsby/ybd/external/build/lib, -Dapp.home=/home/br
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
2019-06-05 13:59:10.459 [ INFO] <LFSourceContext>  Gathering metadata on input file
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 = h7TVdFVK4c0fehPzOk8R4fQUPd9
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 separa
   --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/e
```

```
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/bru
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
2019-06-05 13:59:11.152 [ INFO] <sender 1.00>  SUCCESSFUL BULK LOAD: Loaded 774540 r
```

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, del
nms join
external '/ext_db/ext_season_new' (seasonid int) using(format csv, logdir '/ext_db/l
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.

Parent topic: CREATE EXTERNAL TABLE

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]. 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 must create encryption keys in the same database as the tables that will reference them. Do not use cross-database references to encryption keys in your `CREATE TABLE` statements. If you do this, you will lose those references when the database is backed up and restored (or replicated).

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-e
```

Parent topic: SQL Commands

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 proc
 public | sp2              | integer          | val integer            | stored proc
 public | sp_team_table    | SETOF team       |                        | stored proc
 ...
```

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 invoke 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

Procedure	Returns	Invoked with CALL	Invoked with SELECT
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' );
...
```

Parent topic: SQL Commands

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 Manager Node Ports and 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', nohostna
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 specifi
```

Parent topic: SQL Commands

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
    | [ ENCRYPTED | UNENCRYPTED ] 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 (in other words, 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).

ENCRYPTED | UNENCRYPTED 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.

You can store the password `ENCRYPTED` in the system catalogs. The default encryption behavior depends on the setting of the `password_encryption` configuration parameter.) If the password string is already MD5-encrypted, it is stored encrypted as is, regardless of the `ENCRYPTED` or `UNENCRYPTED` option.

CAUTION:

If you create and update unencrypted passwords with CREATE ROLE and ALTER ROLE commands, passwords are transmitted in clear text and may appear in log messages. A more secure alternative is the `ybsql \password` command.

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
encrypted 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 o
-------------+--------------------------------------------------------------+--------
 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
```

Parent topic: SQL Commands

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 worker 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.

Parent topic: SQL Commands

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 superusers, 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
...
```

Parent topic: SQL Commands

CREATE TABLE

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 | RANDO

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 of thumb, 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={ DETERMINISTI
```

`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`

In This Section

- CREATE TABLE Examples
- Distribution Options
- Partitioning Options
- Sorted and Clustered Tables
- Disabling Join Elimination for LEFT Joins

Parent topic: SQL Commands

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
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
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, fi
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=RANI
weekly_wages varchar(100) encrypted with(COLUMN_ENCRYPTION_KEY=playerkey,ENCRYPTION_
avg_mins_per_match double precision,
matches_played real,
cob varchar(100) encrypted with(COLUMN_ENCRYPTION_KEY=playerkey,ENCRYPTION_TYPE=RANI
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     | weel
----------+--------+----------+-----------+-----------+----------+------------+----
        2 |     41 |       27 | Harry     | Kane      | F        | 07-28-1993 | 25000
        3 |     41 |       27 | Harry     | Winks     | M        | 02-02-1996 | 40000
...
```

If you want users to see clear 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.p
 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.

Parent topic: CREATE TABLE

Distribution Options

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 worker 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 var
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 varch
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.

Parent topic: CREATE TABLE

Partitioning Options

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 a number of 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' ea
```

```
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 a number of 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

Data Type	Range Partitioning Supported	Hash Partitioning Supported
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 resul
```

- 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
distribute on (seasonid)
partition by (
range (matchday between date '8/1/1992' and date '5/31/2017' each interval '1 year',
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 interva
   "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
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 betwe
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)
```

Parent topic: CREATE TABLE

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(
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(
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.

Parent topic: CREATE TABLE

Disabling Join Elimination for LEFT Joins

Join elimination is a query optimization that can occur during planning when primary and foreign keys exist on the tables. The query plan may eliminate redundant joins when these key relationships have been declared. However, because Yellowbrick does not *enforce* primary key and foreign key constraints, your tables may have key relationships that are not valid. If this is true, the elimination of joins between those tables may cause queries to return wrong results.

To prevent this problem from occurring specifically for `LEFT OUTER` joins, you can set the `enable_join_elimination` configuration parameter to `OFF`. This parameter defaults to `ON`. Note that changing this parameter to `OFF` does not influence the planning of other joins, such as `INNER` joins.

For example, consider the primary and foreign key constraints for the following tables:

```
yellowbrick=# \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)
Columns:
   seasonid PRIMARY KEY
Foreign-key constraints:
   "season_fk" FOREIGN KEY (seasonid) REFERENCES season(seasonid)
Primary-key constraints:
   "match_pk" PRIMARY KEY (seasonid)

yellowbrick=# \d season
             Table "public.season"
   Column    |         Type          | Modifiers
-------------+-----------------------+-----------
 seasonid    | smallint              |
 season_name | character(9)          |
 numteams    | smallint              |
```

```
 winners     | character varying(30) |

Distribution: Hash (seasonid)
Columns:
   seasonid UNIQUE
Unique constraints:
   "season_seasonid_key" UNIQUE (seasonid)
Referenced by:
   TABLE "match" CONSTRAINT "season_fk" FOREIGN KEY (seasonid) REFERENCES season(sea
```

By default, the following `LEFT OUTER` join of these tables eliminates the join from the query plan:

```
yellowbrick=# show enable_join_elimination;
 enable_join_elimination
-------------------------
 on
(1 row)
yellowbrick=# explain select m.* from match m left join season s on m.seasonid=s.sea
                  QUERY PLAN
-----------------------------------------------------
 id   rows_planned   workers   node
  1           8606       all   SELECT
  3           8606       all   SCAN match AS m
 Database: yellowbrick
 Version: 5.2.9-20211208203120
 Hostname: yb007-mgr1.mvw.yellowbrick.io

(7 rows)
```

When you set `enable_join_elimination` to `OFF`, the plan changes:

```
yellowbrick=# set enable_join_elimination to off;
SET
yellowbrick=# explain select m.* from match m left join season s on m.seasonid=s.sea
                                    QUERY PLAN
-------------------------------------------------------------------------------------
 id   rows_planned   workers   node
  1           8606       all   SELECT
  9           8606       all   LEFT OUTER HASH JOIN ON (s.seasonid = m.seasonid)
 12           8606       all   |-SCAN match AS m
  2             25       all   |-BUILD
```

```
  5            25       all     SCAN season AS s
 Database: yellowbrick
 Version: 5.2.9-20211208203120
 Hostname: yb007-mgr1.mvw.yellowbrick.io

(10 rows)
```

Here is another example with the same two tables:

```
yellowbrick=# show enable_join_elimination;
 enable_join_elimination
-------------------------
 on
(1 row)
yellowbrick=# explain select distinct(m.seasonid) from match m left join season s o
                     QUERY PLAN
------------------------------------------------------------
 id   rows_planned   workers   node
  1           8606       all   SELECT
  3           8606       all   GROUP BY (m.seasonid)
  6           8606       all   SCAN match AS m
 Database: yellowbrick
 Version: 5.2.9-20211208203120
 Hostname: yb06-mgr1.slc.yellowbrick.io

(8 rows)

yellowbrick=# set enable_join_elimination to off;
SET
yellowbrick=# explain select distinct(m.seasonid) from match m left join season s o
                                    QUERY PLAN
-------------------------------------------------------------------------------------
 id   rows_planned   workers   node
  1           8606       all   SELECT
  3           8606       all   GROUP BY (m.seasonid)
 11           8606       all   LEFT OUTER HASH JOIN ON (s.seasonid = m.seasonid)
 14           8606       all   |-SCAN match AS m
  4             25       all   |-BUILD
  7             25       all     SCAN season AS s
 Database: yellowbrick
 Version: 5.2.9-20211208203120
 Hostname: yb06-mgr1.slc.yellowbrick.io
```

```
(11 rows)
```

Parent topic: CREATE TABLE

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. Tables have a limit of 2,000 user-defined columns. 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) f
ERROR:  column "varchar" specified more than once in "casts"
premdb=# create table casts as select cast(1 as varchar) cast1, cast(2 as varc
SELECT 1
```

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
```

Parent topic: SQL Commands

CREATE USER

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.

Parent topic: SQL Commands

CREATE VIEW

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` as long

as the columns that originally existed in the table also exist in the new instance of the table and have the same data types.

Restriction: You cannot create views by selecting from external tables.

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
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,f
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, fts
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.

Parent topic: SQL Commands

CREATE WLM PROFILE

Create a workload management (WLM) profile.

```
CREATE WLM PROFILE name [ ( DEFAULT_POOL name ) ]
```

Only superusers can create, alter, and drop WLM profiles. Alternatively, you can use the SMC to create and manage profiles.

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.

Parent topic: SQL Commands

CREATE WLM RESOURCE POOL

Create a workload management (WLM) resource pool.

```
CREATE WLM RESOURCE POOL name (PROFILE name [ options ])
```

Only superusers 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` in the SMC, 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 worker, 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_
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', pro
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', pro
CREATE WLM RESOURCE POOL
```

Parent topic: SQL Commands

CREATE WLM RULE

Create a workload management (WLM) rule.

```
CREATE WLM RULE name [ ( PROFILE name [ options ] ) ]
```

Only superusers 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`, `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
```

Parent topic: SQL Commands

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
--------+------+------+----------------+----------+-----------+---------------------
    22 |   23 |   72 | Leicester City | Foxes    | Leicester | King Power Stadium |
(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
```

Parent topic: SQL Commands

DECLARE

Create a cursor within a transaction block.

```
DECLARE name [ INSENSITIVE ] [ NO SCROLL | SCROLL ] CURSOR  [ { WITH | WITHOUT } HO
```

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)
...
```

Parent topic: SQL Commands

DELETE

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. You cannot use `DELETE` commands on tables in hot-standby databases; see HOT_STANDBY and READONLY Modes.

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 sea
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)
```

Parent topic: SQL Commands

DESCRIBE

Describe the definition of a table, view, sequence, procedure, or encryption key. 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 ]
```

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(season
 )
 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)
```

Parent topic: SQL Commands

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.

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
-------------+-------------+------------------+-------------------------------------
       16395 |      175440 | 0:175440:175440: | Aug21Full
       16395 |      177767 | 0:177767:177767: | Aug21Incr
       16395 |      177989 | 0:177989:177989: | Aug21IncrNew
       16395 |      175439 | 0:3:3:           | ybd_cf861c56-0cf7-19a8-f459-338203621
(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_
-------------+------------+--------+------------------------+-----------------------
(0 rows)
```

```
premdb=# select * from sys.backup_snapshot;
 database_id | backup_txid | backup_txsnap | snapshot_name | creation_time
-------------+-------------+---------------+---------------+---------------
(0 rows)
```

Parent topic: SQL Commands

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
```

Parent topic: SQL Commands

DROP EXTERNAL MOUNT

Drop an NFS mount 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
```

Parent topic: SQL Commands

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, they will receive the following error message:

```
premdb=# drop key yb100key cascade;
ERROR:  must be owner of key yb100key
```

Parent topic: SQL Commands

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.

Parent topic: SQL Commands

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.

Parent topic: SQL Commands

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
```

Parent topic: SQL Commands

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
```

Parent topic: SQL Commands

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
```

Parent topic: SQL Commands

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
```

Parent topic: SQL Commands

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"
```

Parent topic: SQL Commands

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.

Parent topic: SQL Commands

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"
```

Parent topic: SQL Commands

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 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.

Parent topic: SQL Commands

DROP WLM RESOURCE POOL

Drop a workload management (WLM) resource pool.

```
DROP WLM RESOURCE POOL [ IF EXISTS ] name
```

Only superusers 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.

Parent topic: SQL Commands

DROP WLM RULE

Drop a workload management (WLM) rule.

```
DROP WLM RULE [ IF EXISTS ] name
```

Only superusers 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.

Parent topic: SQL Commands

END

Commit the current transaction block.

```
END [ WORK | TRANSACTION ]
```

`WORK` and `TRANSACTION` are both optional keywords; they have no effect. See also COMMIT.

Parent topic: SQL Commands

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',
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)
```

Parent topic: SQL Commands

EXPLAIN

Return the text version of the query plan for either a specific `SELECT` query or a statement that runs a query. You can specify the query either with a complete SQL statement or with the ID of a query that is currently running.

Use the SMC to see a graphical version of the plan.

Syntax

```
EXPLAIN ( [ ANALYZE ] [ , VERBOSE ] [ , STABLE ] [ , TYPE ] [ , NODE_OUTPUT ] [ , DI
```

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. See EXPLAIN ANALYZE for a DELETE.

Setting the `enable_query_trace` configuration parameter to `ON` prior to running `EXPLAIN ANALYZE` on queries enables full query statistics to be sent from the worker nodes, containing data from each node and each thread. The `enable_query_trace` parameter defaults to `OFF`.

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 query 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, which you can select 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_fro
 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_fro
 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   SELEC
                                                                               (awa
                                                                               dist
  3            50            50   0.00     57.00Mi H       1.00Mi    single   SCAN
                                                                               (awa
                                                                               dist
                                                                               read
 50 rows returned
 Memory planned (fixed): 71.00MiB, Memory planned (variable): 128.00MiB, High confi
 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 =
                                                     read_efficiency=99.6%, rows_fr
 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:
 33820 rows returned
(7 rows)
```

Add the TYPE option to the previous query

```
premdb=# explain (analyze, stable, type) select * from newmatchstats where seasonid=
                                                                                QUERY
-------------------------------------------------------------------------------------
 rows_actual   workers   node
      33820       all   SELECT
                          (newmatchstats.seasonid INT2, newmatchstats.matchday DATE,
                          distribute on (newmatchstats.seasonid)
      33820       all   SCAN newmatchstats
                          (newmatchstats.seasonid INT2, newmatchstats.matchday DATE,
                          distribute on (newmatchstats.seasonid)
                          (newmatchstats.seasonid::INT4 = $0) AND scan_constraints:
 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 interva
   "htid"       RANGE (BETWEEN 2 AND 51 EACH 1, IS NULL, OUTSIDE RANGE)

(15 rows)
```

```
premdb=# explain (distribution, partitions) select matchday, count(*) from newmatchs
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 (`4656199847`) was previously selected from the `sys.query` view, then used in the EXPLAIN command.

```
yellowbrick=# explain 4656199847;
                                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------
 id   rows_planned   rows_actual   skew   workers   node
  2           8606    1361961688   0.00       all   INSERT INTO match
  5           8606    1361961688   0.00       all   SCAN match
                                                     read_efficiency=61.8%, rows_fr
 1361961688 rows inserted
 Read: 2.84GiB, Distributed: 0.00B, Write: 6.36GiB
 Database: premdb
 Version: 5.2.21-221028094805
 Hostname: yb007-mgr0.sjc.yellowbrick.io
 query_id: 4656199847
 Lookup cache hit: true
 Code cache hit: true
 Current execution time: 35485.303 ms
(13 rows)
```

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)

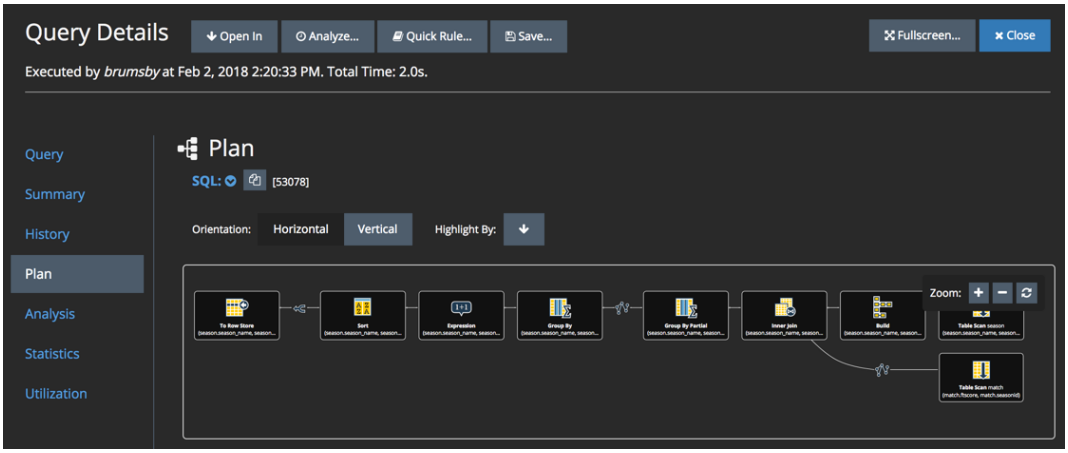
premdb=# explain analyze delete from season where winners='Arsenal';
                                                        QUERY PLAN
```

```
 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_co
                                                     read_efficiency=0.0%, rows_fro
 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)
```

SMC Graphical EXPLAIN

Tip: You may also want to use the visual EXPLAIN feature in the System Management Console (SMC). After logging in, go to **Monitor**, then **Active Queries** or **Query Performance**, double-click the query you want to explain, and click **Plan**. For example:



Parent topic: SQL Commands

FETCH

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. In other words, 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
```

Parent topic: SQL Commands

GRANT

Grant privileges on database objects to specific roles and grant membership in roles.

You can grant privileges at the system level (`ON SYSTEM`) and on the following objects:

- Databases (`ON DATABASE`)
- Schemas (`ON SCHEMA`)
- Tables (`ON TABLE`)
- Columns in tables (`ON TABLE`)
- Views (`ON TABLE`)
- Sequences (`ON SEQUENCE`)
- Stored procedures (`ON PROCEDURE`)
- Encryption keys (`ON KEY`)
- Users and roles (`ON ROLE`)

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 |
    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, except . 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. On the other hand, 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.

In This Section

- ON DATABASE
- ON KEY
- ON PROCEDURE
- ON ROLE
- ON SCHEMA
- ON SEQUENCE
- ON SYSTEM
- ON TABLE
- ROLE

- Abbreviations for ACLs

Parent topic: SQL Commands

ON DATABASE

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 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.pu
                                                                        QUERY PLAN
-----------------------------------------------------------------------------------------
 id   rows_planned   workers   node
  0            150    single   SELECT
  2            150    single   GROUP BY (union.[36], union.[37], union.[38], 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
--------+------+------+-------------------+------------+------------+-----------------
    25 |   26 |   75 | Manchester United | Red Devils | Manchester | Old Trafford
     1 |    2 |   51 | Arsenal           | Gunners    | London     | Emirates Stadiu
    24 |   25 |   74 | Manchester City   | Citizens   | Manchester | Etihad Stadium
(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 t
 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 t
                                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.

Parent topic: GRANT

ON KEY

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
```

Parent topic: GRANT

ON PROCEDURE

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
```

Parent topic: GRANT

ON ROLE

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
```

Parent topic: GRANT

ON SCHEMA

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.

Parent topic: GRANT

ON SEQUENCE

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
```

Parent topic: GRANT

ON SYSTEM

Syntax

```
GRANT { { CREATE ROLE | ALTER ANY ROLE | DROP ANY ROLE | VIEW ROLE |
     CREATE DATABASE | ALTER ANY DATABASE | DROP ANY DATABASE | BACKUP ANY DATABASE
     TRACE QUERY | VIEW QUERY TEXT | EXPLAIN QUERY |
     CONTROL ANY SESSION | CONTROL LDAP }
     [, ...] | ALL [ PRIVILEGES ] }
     ON SYSTEM
     TO { role_specification } [, ...] [ WITH GRANT OPTION ]
```

Parameters

ON SYSTEM

Grant privileges system-wide, not on specific databases or objects.

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 in the system (except `yellowbrick` and other system databases).

BACKUP ANY DATABASE

Privilege to back up any database in the system (using `ybbackup`).

RESTORE ANY DATABASE

Privilege to restore any database (in the system (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 LDAP

Privilege to run the function sys.ldap_sync function, which synchronizes LDAP users and roles. 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)
```

See also HAS_SYSTEM_PRIVILEGE.

Examples

Grant system-wide 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 on the system to two users:

```
premdb=# grant backup any database, restore any database on system to restore1, res
GRANT
```

Grant `CONTROL LDAP` to user `control1`:

```
premdb=# grant control ldap on system to control1;
GRANT
premdb=# \c premdb control1
Password for user control1:
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, c
You are now connected to database "premdb" as user "control1".
premdb=> select sys.ldap_sync();
               ldap_sync
-----------------------------------------
 LDAP authorization data was synchronized+

(1 row)
```

Parent topic: GRANT

ON TABLE

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
```

Parent topic: GRANT

ROLE

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 o
-------------+----------------------------------------------------------------+--------
 ...
```

```
allqa       | Create DB, Cannot login                                    | {}
newhire     | Cannot login                                               | {allqa}
...
```

Parent topic: GRANT

Abbreviations for ACLs

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
a (append)		BULK LOAD		INSERT	INSERT
r (read)	VIEW ROLE			SELECT	SELECT
w (write)		ALTER ANY DATABASE	ALTER ANY SCHEMA	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				

Abbreviation	ON SYSTEM	ON DATABASE	SCHEMAS	TABLES/VIEWS	COLUMNS
U	CONTROL ANY SESSION		USAGE		
C	CREATE ROLE	CREATE	CREATE		
T		CREATE TEMPORARY			
c		CONNECT			
e					
E					
h					
b	CREATE DATABASE				
p	EXPLAIN QUERY	EXPLAIN QUERY			
q	VIEW QUERY TEXT	VIEW QUERY TEXT			

Abbreviation	ON SYSTEM	ON DATABASE	SCHEMAS	TABLES/VIEWS	COLUMNS
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			

Parent topic: GRANT

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 [, ...] [ C
(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.

Parent topic: SQL Commands

IMPORT SSL TRUST

Import the remote SSL system configuration into the source system.

```
IMPORT SSL TRUST FROM 'certificate'
```

SSL trust is required for database replication; see Configuring SSL Trust.

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
```

Parent topic: SQL Commands

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. You cannot use `INSERT` commands on tables in hot-standby databases; see HOT_STANDBY and READONLY Modes.

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. See EXPLAIN ANALYZE for a DELETE.

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 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
```

Select data from an external table and insert it into a regular database table. You do not have to declare the column list for the file in this case; it is optional. The `USING` clause is required. See External Tables and USING Options.

```
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.

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.

Parent topic: SQL Commands

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
```

Parent topic: SQL Commands

MOVE cursor

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   | capaci
--------+------+------+-------------------+----------+--------+------------+-------
    12 |   13 |   62 | Charlton Athletic | Addicks  | London | The Valley |    2711
(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.

Parent topic: SQL Commands

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,spi
MOVE
```

```
premdb=# move 13450004 to wlm resource pool long with(priority critical);
MOVE
```

Parent topic: SQL Commands

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=#
```

Parent topic: SQL Commands

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)
```

Parent topic: SQL Commands

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)
```

Parent topic: SQL Commands

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.

Parent topic: SQL 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
```

Parent topic: SQL Commands

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
     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
```

Parent topic: SQL Commands

REVOKE SSL TRUST

Revoke the SSL trust configuration on the system.

```
REVOKE { 'hash' | ALL } FROM SSL TRUST
```

SSL trust is required for database replication; see Configuring SSL Trust.

'hash'

Revoke an SSL certificate from the system trust store. Use the SHOW SSL TRUST command to display certificates and determine the hash value.

ALL

Revoke all SSL certificates from the system trust store.

For example:

```
premdb=# revoke all from ssl trust;
REVOKE SSL TRUST
```

```
premdb=# revoke '2de2f78e.0' from ssl trust;
REVOKE SSL TRUST
```

Parent topic: SQL Commands

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
```

Parent topic: SQL Commands

ROLLBACK DATABASE

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
...
ERROR: Failed to promote replica
DETAIL: Unable to promote replica “repldb2_failover” Reason: ... ROLLBACK DATABASE
```

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
ROLLBACK DATABASE TO SNAPSHOT
```

Parent topic: SQL Commands

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)
```

Parent topic: SQL Commands

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)
```

Parent topic: SQL Commands

SELECT

Write a query that returns data from tables and views.

Syntax

```
[ WITH name as (subquery) [, ...] ]
SELECT [ ALL | DISTINCT expression [ [ AS ] output_name ] [, ...] ]
   [ * | expression [ [ AS ] output_name ] [, ...] ]
   [ FROM table_reference [, ...] ]
   [ WHERE condition ]
   [ GROUP BY 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
   [ LIMIT { count | ALL } ]
   [ OFFSET start [ ROW | ROWS ] ]
```

In This Section

- WITH Clause
- SELECT List
- FROM Clause
- WHERE Clause
- GROUP BY Clause
- HAVING Clause
- WINDOW Clause
- ORDER BY Clause
- LIMIT Clause

- OFFSET Clause
- UNION, INTERSECT, EXCEPT
- Subqueries

Parent topic: SQL Commands

WITH Clause

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 |
----------+---------------------+------+------+---------+---------+------+---------
       1 | 1992-08-01 00:00:00 |    2 |   52 | 0-1     | -       |   52 | Aston Vil
       1 | 1992-08-01 00:00:00 |    2 |   55 | 0-1     | -       |   55 | Blackburn
       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 C
       1 | 1992-08-01 00:00:00 |    2 |   65 | 3-0     | -       |   65 | Crystal Pa
...
```

Parent topic: SELECT

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.

*

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 left-most 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)
```

Parent topic: SELECT

FROM Clause

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 [ EXTERNAL ]
table_reference [ * ] [ [ AS ] table_alias [ ( column_alias [, ...] ) ] ]
[ join_type table_reference ON join_condition | USING ( join_column [, ...] ) ]
[, ...]
```

Parameters

EXTERNAL

Specify an external table. External table references are of the following form:

```
'/my_external_tables/ext_table100.csv' (c1 int, c2 char(10)) using (format tex
```

External table references consist of three parts:

- Path to an external source file, relative to an NFS mount point (see CREATE EXTERNAL MOUNT)
- Column list, naming each column and its data type
- `USING` clause, naming one or more options for selecting from the file, including the format of the data

See also External Tables and CREATE EXTERNAL TABLE.

Note: You can also select from a symbolic link to a file. For example, assume you have a symbolic link like this:

```
~/ext_db$ ll
...
ext_team_link.csv -> /home/yb100/ext_db/ext_team.csv
```

A query can select from either the symbolic link, `ext_team_link.csv`, or the source file, `ext_team.csv`.

table_reference

Name a persistent or temporary table, a view, a table derived from a subquery, a `SETOF` stored procedure, or an external table.

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 can combine references to regular tables and external tables in the same query.

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      |      stadiu
--------+------+------+-------------------+------------+---------------+-----------
    20 |   21 |   70 | Ipswich Town      | Blues      | Ipswich       | Portman Roac
    21 |   22 |   71 | Leeds United      | Whites     | Leeds         | Elland Road
    22 |   23 |   72 | Leicester City    | Foxes      | Leicester     | King Power S
    23 |   24 |   73 | Liverpool         | Reds       | Liverpool     | Anfield
    24 |   25 |   74 | Manchester City   | Citizens   | Manchester    | Etihad Stadi
    25 |   26 |   75 | Manchester United | Red Devils | Manchester    | Old Traffor
    26 |   27 |   76 | Middlesbrough     | Boro       | Middlesbrough | Riverside St
    27 |   28 |   77 | Newcastle United  | Magpies    | Newcastle     | St. James Pa
    28 |   29 |   78 | Norwich City      | Canaries   | Norwich       | Carrow Road
    29 |   30 |   79 | Nottingham Forest | Forest     | Nottingham    | City Ground
(10 rows)
```

Here is the same query with `JOIN ON` syntax:

```
premdb=# select teamid, team.htid, atid, team.name, nickname, city, stadium, capacit
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
...
```

The following query joins three instances of the `season` table: the first one is external (alias `ext_s`), the second one is in a remote database (alias `yps`), and the third is in the current database (alias `pps`):

```
premdb=# select ext_s.seasonid, ext_s.season_name, ext_s.winners from
external 'ext_db/season.csv'
  (seasonid int, season_name char(9), numteams int, winners varchar(40))
  using (format csv)
ext_s,
yellowbrick.public.season yps,
premdb.public.season pps
where ext_s.seasonid=yps.seasonid
  and yps.seasonid=pps.seasonid;
 seasonid | season_name |      winners
----------+-------------+-------------------
        1 | 1992-1993   | Manchester United
        2 | 1993-1994   | Manchester United
        3 | 1994-1995   | Blackburn Rovers
        4 | 1995-1996   | Manchester United
        5 | 1996-1997   | Manchester United
        6 | 1997-1998   | Arsenal
        7 | 1998-1999   | Manchester United
        8 | 1999-2000   | Manchester United
        9 | 2000-2001   | Manchester United
       10 | 2001-2002   | Arsenal
       11 | 2002-2003   | Manchester United
       12 | 2003-2004   | Arsenal
       13 | 2004-2005   | Chelsea
       14 | 2005-2006   | Chelsea
       15 | 2006-2007   | Manchester United
```

```
        16 | 2007-2008   | Manchester United
        17 | 2008-2009   | Manchester United
        18 | 2009-2010   | Chelsea
        19 | 2010-2011   | Manchester United
        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   |
(25 rows)
```

Parent topic: SELECT

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)
```

Parent topic: SELECT

GROUP BY Clause

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.

```
[ GROUP BY 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.

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.

Parent topic: SELECT

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)
```

Parent topic: SELECT

WINDOW Clause

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
...
```

Parent topic: SELECT

ORDER BY Clause

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 co
```

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
```

Parent topic: SELECT

LIMIT Clause

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)
```

Parent topic: SELECT

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)
```

Parent topic: SELECT

UNION, INTERSECT, EXCEPT

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)
```

Parent topic: SELECT

Subqueries

In general, Yellowbrick supports subqueries in the standard way.

Subquery expressions with ALL are not supported.

In This Section

- Subquery Examples
- Correlated Subqueries

Parent topic: SELECT

Subquery Examples

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)
```

Parent topic: Subqueries

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 awaytea
```

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
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
```

Parent topic: Subqueries

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 [, ...] ]
   [ HAVING condition [, ...] ]
   [ WINDOW window_name AS ( window_definition ) [, ...] ]
   [ { UNION | INTERSECT | { EXCEPT | MINUS } [ ALL | DISTINCT ] select ]
   [ ORDER BY expression [ ASC | DESC ] [ NULLS { FIRST | LAST } ] [, ...] [ COLLATE
   [ 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
```

Parent topic: SQL Commands

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 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.

Parent topic: SQL Commands

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)
```

Parent topic: SQL Commands

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 superuser. 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 by 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
```

Parent topic: SQL Commands

SHOW

Show the current value of a configuration parameter or show the 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
-------------------
 5.2.4
(1 row)
```

`SHOW ALL` includes a description of each parameter.

```
yellowbrick=# show all;
               name                  |     setting     |
-------------------------------------+-----------------+-----------------------------
 application_name                    | ybsql           | Sets the application name t
 client_encoding                     | UTF8            | Sets the client's characte
 DateStyle                           | ISO, MDY        | Sets the display format fo
 extra_float_digits                  | 0               | Sets the number of digits d
 idle_in_transaction_session_timeout | 0               | Sets the maximum allowed du
 idle_session_timeout                | 0               | Sets the maximum allowed se
 lc_collate                          | C               | Shows the collation order l
 lc_ctype                            | C               | Shows the character classif
 lc_time                             | C               | Sets the locale for formatt
 max_index_keys                      | 32              | Shows the maximum number of
 search_path                         | "$user", public | Sets the schema search orde
 server_encoding                     | LATIN9          | Sets the server (database)
```

```
 statement_timeout                | 0              | Sets the maximum allowed du
 TimeZone                         | UTC            | Sets the time zone for disp
 transaction_isolation            | read committed | Sets the current transactio
 yb_last_execid                   | 3204465        | Shows the execution-id of t
 yb_server_version                | 5.2.13         | Shows the Yellowbrick Datab
 yb_server_version_num            | 50213          | Shows the Yellowbrick Datab
 ybd_analyze_after_loads          | on             | Enable or disable immediate
 ybd_analyze_after_writes         | on             | Enable or disable immediate
 ybd_query_tags                   |                | Sets the optional query tag
(21 rows)
```

Note that 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 appliance.

The following parameters cannot be changed (by superusers or regular users):

- `lc_collate`
- `lc_ctype`
- `lc_time`
- `server_encoding`
- `yb_last_execid`
- `yb_server_version`
- `yb_server_version_num`

Parent topic: SQL Commands

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
 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-----
(34 rows)
```

Parent topic: SQL Commands

SHOW SSL IDENTITY

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
 BAYTAlVTMQswCQYDVQQIDAJDQTEfMB0GA1UECgwWWWVsbG93YnJpY2sgRGF0YSwg
 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)
```

Parent topic: SQL Commands

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)
```

Parent topic: SQL Commands

SHOW SSL TRUST

Validate the results of the IMPORT SSL TRUST command.

```
SHOW SSL TRUST
```

SSL trust is required for database replication. See Configuring SSL Trust for details.

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
            | subject= /C=US/ST=CA/O=Yellowbrick Data, Inc./OU=a993ad4e-446d-4e66-90a
            | 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-----
        |
```

Parent topic: SQL Commands

START TRANSACTION

Start a transaction block.

```
START TRANSACTION
```

This command is identical to BEGIN. See the description of `BEGIN` for details.

Parent topic: SQL Commands

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
...
```

Parent topic: SQL Commands

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
```

Parent topic: SQL Commands

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. You cannot use `UPDATE` commands on tables in hot-standby databases; see HOT_STANDBY and READONLY Modes.

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. See EXPLAIN ANALYZE for a DELETE.

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 d
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
;
```

Parent topic: SQL Commands

SQL Functions and Operators

This section describes supported SQL functions and operators in several main categories, including a section on SQL conditions.

Note: User-defined functions (UDFs) are not supported.

In This Section

- Supported Functions and Return Types
- SQL Conditions
- String Functions
- Pattern Matching
- Datetime Functions
- Mathematical Functions
- Aggregate Functions
- Window Functions
- Conditional Expressions
- Formatting Functions
- Type-Safe Casting Functions
- JSON Functions
- CONVERT (SQL Server Migration Function)
- GETBIT Function
- NEXTVAL Function
- System Functions
- Network Address Functions

Parent topic: SQL Reference

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

Function	Return Type	Category
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
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

Function	Return Type	Category
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
DLE_DST	INTEGER	String
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

Function	Return Type	Category
GETDATABASEENCODING()	NAME	System
GREATEST and LEAST	Depends on input type	Conditional Expression
GROUP_CONCAT	VARCHAR	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

Function	Return Type	Category
IIF	Depends on input type	Conditional Expression
INITCAP	VARCHAR	String
INSTR	INTEGER	String
IPV4_HOSTMASK, IPV6_HOSTMASK	IPV4, IPV6	Network Address
INT2, INT4, INT8	INT2, INT4, INT8	Formatting
IPV4_INET_MERGE, IPV6_INET_MERGE	VARCHAR	Network Address
IPV4_GET_INET, IPV6_GET_INET	IPV4, IPV6	Network Address
IPV4_GET_MASKBITS, IPV6_GET_MASKBITS	INTEGER	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
JUSTIFY_INTERVAL	INTERVAL	Datetime
LAG	Same as input type	Window

Function	Return Type	Category
LAST_DAY	DATE	Datetime
LAST_VALUE	Same as input type	Window
LE_DST	INTEGER	String
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

Function	Return Type	Category
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
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

Function	Return Type	Category
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	VARCHAR	String
REPLACE	VARCHAR	String
REVERSE	VARCHAR	String
RIGHT	VARCHAR	String
ROUND	Depends on input type	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

Function	Return Type	Category
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
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

Function	Return Type	Category
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

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

Parent topic: SQL Functions and Operators

SQL Conditions

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.

In This Section

- Logical Operators
- Comparison Operators
- ANY
- BETWEEN
- EXISTS
- IN

Parent topic: SQL Functions and Operators

Logical Operators

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='Manches
 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='Manche
 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
```

Parent topic: SQL Conditions

Comparison Operators

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)

Operator	Definition
`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
--------+------+------+-------------------+------------+------------+--------------
      1 |    2 |   51 | Arsenal           | Gunners    | London     | Emirates Stadi
     24 |   25 |   74 | Manchester City   | Citizens   | Manchester | Etihad Stadium
     25 |   26 |   75 | Manchester United | Red Devils | Manchester | Old Trafford
(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)
```

Parent topic: SQL Conditions

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);
```

Parent topic: SQL Conditions

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      |     sta
--------+------+------+-------------------------+----------+---------------+-------
    40 |   41 |   90 | Swindon Town            | Robins   | Swindon       | County
    41 |   42 |   91 | Tottenham Hotspur       | Spurs    | London        | White Ha
```

```
     42 |   43 |   92 | Watford                | Hornets  | Watford       | Vicarage
     43 |   44 |   93 | West Bromwich Albion   | Baggies  | Birmingham    | The Haw
     44 |   45 |   94 | West Ham United        | Hammers  | London        | Upton Pa
     45 |   46 |   95 | Wigan Athletic         | Latics   | Wigan         | DW Stadi
     46 |   47 |   96 | Wimbledon              | Dons     | London        | Selhurst
     47 |   48 |   97 | Wolverhampton Wanderers | Wolves   | Wolverhampton | Molineux
     48 |   49 |   98 |                        |          |               |
     49 |   50 |   99 |                        |          |               |
     50 |   51 |  100 |                        |          |               |
(11 rows)
```

Parent topic: SQL Conditions

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)
```

Parent topic: SQL Conditions

IN

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) l
 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);
```

Parent topic: SQL Conditions

String Functions

This section describes functions that operate on character strings.

In This Section

- ASCII
- BIT_LENGTH
- BTRIM
- CHAR_LENGTH
- CHR
- CONCAT, ||
- CONCAT_WS
- DECRYPT (deprecated)
- DECRYPT_KS
- DLE_DST
- ENCODE
- ENCRYPT (deprecated)
- ENCRYPT_KS
- HASH
- HASH4
- HASH8
- HMAC
- HMAC_KS

- INITCAP
- INSTR
- LE_DST
- LEFT
- LENGTH
- LOWER
- LPAD
- LTRIM, RTRIM
- MD5
- NYSIIS
- OCTET_LENGTH
- POSITION
- REPEAT
- REPLACE
- REVERSE
- RIGHT
- RPAD
- SPLIT_PART
- STRPOS
- SUBSTR
- SUBSTRING
- TO_ASCII
- TO_HEX

- TRANSLATE
- TRIM
- UPPER

Parent topic: SQL Functions and Operators

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
--------+------+------+---------------------+------------+--------+-----------------
     12 |   13 |   62 | Charlton Athletic   | Addicks    | London | The Valley
     18 |   19 |   68 | Fulham              | Cottagers  | London | Craven Cottage
     46 |   47 |   96 | Wimbledon           | Dons       | London | Selhurst Park
     15 |   16 |   65 | Crystal Palace      | Eagles     | London | Selhurst Park
      1 |    2 |   51 | Arsenal             | Gunners    | London | Emirates Stadium
     44 |   45 |   94 | West Ham United     | Hammers    | London | Upton Park
     32 |   33 |   82 | Queens Park Rangers | Hoops      | London | Loftus Road
     13 |   14 |   63 | Chelsea             | Pensioners | London | Stamford Bridge
     41 |   42 |   91 | Tottenham Hotspur   | Spurs      | London | White Hart Lane
(9 rows)
```

Parent topic: String Functions

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)
```

Parent topic: String Functions

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
...
```

Parent topic: String Functions

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='Londor
 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)
```

Parent topic: String Functions

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
...
```

Parent topic: String Functions

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.

CONCAT and `||` are different in the following ways:

- CONCAT ignores `NULL` values, and the `||` operator does not. For `||`, if either argument is `NULL`, the result is `NULL`.
- CONCAT performs implicit type casting to `CHAR`, and the `||` operator does not.
- CONCAT has a limit of 100 strings that can be concatenated.

The following example concatenates four strings (two VARCHAR columns 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
...
```

Parent topic: String Functions

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
...
```

Parent topic: String Functions

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
--------+------+------+------------------+-------------+------------+--------------
      1 |    2 |   51 | Arsenal          | A8jstaVuQ0  | London     | Emirates Stadi
      2 |    3 |   52 | Aston Villa      | RuCsrql{7K2 | Birmingham | Villa Park
      3 |    4 |   53 | Barnsley         | PuztyyE     | Barnsley   | Oakwell Stadiu
```

```
      4 |    5 |   54 | Birmingham City  | FiSmyyE     | Birmingham | St. Andrew's
      5 |    6 |   55 | Blackburn Rovers | VWimyuEu    | Blackburn  | Ewood Park
(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 o
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)
```

Parent topic: String Functions

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 en
 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)
```

Parent topic: String Functions

DLE_DST

Tip: This function was added in **Version 5.2.26** and is not available in earlier 5.2.x versions.

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*.

This function returns an integer.

Performance of these functions may be optimized by setting two configuration parameters.

See also LE_DST.

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=
       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.

Parent topic: String Functions

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 orde
                    encode
-----------------------------------------------
 so9Ir6BcSA219q3jbWx5Y2ugrXsM/YG39F3hOdp5J+g=
 99oaUrPWoGE6DuZUw6Ar2QDMFaCsUtatXk24TGi5I4A=
 W8Apz44L4MrcAm4Bz8iCCbunokywHuvtQAzSnTI9+z8=
(3 rows)
```

Parent topic: String Functions

ENCRYPT (deprecated)

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
 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
 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(
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
...
```

Parent topic: String Functions

ENCRYPT_KS

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.

In This Section

- ENCRYPT_KS Examples
- Encryption and Decryption Algorithms

Parent topic: String Functions

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 l
 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 teami
 nickname | encrypt_ks
----------+-------------
 Gunners  | 5pOQhIRyw1
 Villains | K3vQf2hwd5E
 Tykes    | M38RWAA
 Blues    | 0NfSWAA
 Rovers   | GRPSWEAy
(5 rows)
```

Parent topic: ENCRYPT_KS

Encryption and Decryption Algorithms

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`

Function Parameter	Algorithm	Output Length (Encryption)	Output Length (Decryption)
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 cipher-text by reverse-engineering the bit-stream used to protect the plain-text string.

Parent topic: ENCRYPT_KS

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)
```

Parent topic: String Functions

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)
```

Parent topic: String Functions

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 o
 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)
```

Parent topic: String Functions

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
-------------------------------------------------------------------------------------------
 5a724fdd70367f56a2d290f6dbbd729782821846d7e4cfa7c99606b16446fc76b8c3012fe6bfc6ae43
 c23caf7fd7029b3a5d14ecfd38947b87a8a0dbc51bbfaa10c72dd5138f97b201aa171b425f24999a1a
 16a5c1e96d75b3f3d99df35530ba13877cd29accae35799f3c6a1b27796798574230855 53bb41ff904
(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)
```

Parent topic: String Functions

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
------------------------------------------------------------------------------------------------
 c000eb5a52cbb8452ab36e84856a5d4fe9d29d12c841eed6cc19f4824a5a1f56c058e78b15ea2736b0
 d3e3b3e354b8b6eb3c927476f33c22c0aba497e34d972a4643bf0fec3eb012dc707178ff71c4bc0d87
 86617d840fd40ebd627790d24495cce28ec06851d3255e4517680ac89fab6284f93e02a8a376a2fc2d
(3 rows)
```

Parent topic: String Functions

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)
```

Parent topic: String Functions

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.

Parent topic: String Functions

LE_DST

Tip: This function was added in **Version 5.2.26** and is not available in earlier 5.2.x versions.

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.

See also DLE_DST.

Configuration

The behavior of the `LE_DST` and `DLE_DST` functions is configurable, using the following parameters:

- `max_levenshtein_string_size` : Maximum size (in bytes) of the two input arguments to the function. The default is `255` .

Because these functions can be very CPU and memory intensive, you may have to wait for results when long strings are used as inputs. You may be able to improve performance by setting an upper bound on their length.

- `max_levenshtein_distance` : Maximum distance for which you expect an accurate answer. The default 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, just whether or not it is bigger than 5. 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.

The ability to check that a distance between two strings is no greater than a known value is a common use case; setting this parameter may optimize performance in those situations.

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.

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.co
 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' guc
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' guc
```

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.

Parent topic: String Functions

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.

Parent topic: String Functions

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='Londo
 length | char_length
--------+-------------
      7 |           7
      7 |           7
     10 |          10
      6 |           6
      9 |           9
      5 |           5
      5 |           5
      7 |           7
      4 |           4
(9 rows)
```

Parent topic: String Functions

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
...
```

Parent topic: String Functions

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 zeroes 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
...
```

Parent topic: String Functions

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
     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 b
     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)
```

Parent topic: String Functions

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
...
```

Parent topic: String Functions

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     | weeki
----------+--------+----------+-----------+----------+----------+------------+-----
        6 |     25 |       27 | Alexis    | Sanchez  | F        | 1988-12-19 |
       13 |     23 |       27 | Mo        | Salah    | F        | 1992-06-15 |
(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(
 teamid | htid | atid |     name     | nickname |  city   |       stadium        | c
--------+------+------+--------------+----------+---------+----------------------+-
    11 |   12 |   61 | Cardiff City | Clarets  | Cardiff | Cardiff City Stadium |
(1 row)
```

Parent topic: String Functions

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)
```

Parent topic: String Functions

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
...
```

Parent topic: String Functions

REPEAT

Return a character string that is repeated *n* times.

```
REPEAT(string, number)
```

The second argument must be a literal number value. You cannot specify a column name. A negative number is interpreted as zero.

For example, return the `season_name` repeated three times:

```
premdb=# SELECT REPEAT(season_name,3) FROM season ORDER BY 1;
          repeat
-----------------------------
 1992-19931992-19931992-1993
 1993-19941993-19941993-1994
 1994-19951994-19951994-1995
...
```

Parent topic: String Functions

REPLACE

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)
```

Parent topic: String Functions

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)
```

Parent topic: String Functions

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.

Parent topic: String Functions

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 zeroes 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
...
```

Parent topic: String Functions

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,'-
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)
```

Parent topic: String Functions

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)
```

Parent topic: String Functions

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) awaygoal
FROM match WHERE seasonid=2 AND htid=2 AND atid=52;
 ftscore | homegoals | awaygoals
---------+-----------+-----------
 1-2     | 1         | 2
(1 row)
```

Parent topic: String Functions

SUBSTRING

Return an exact set of characters from a string. See also SUBSTR and SUBSTRING (Pattern Match).

Note: `SUBSTRING` and `SUBSTR` are not synonymous. `SUBSTRING` returns a `VARCHAR` data type, but `SUBSTR` returns the data type of its input string.

```
SUBSTRING(string [FROM start] [FOR count])
```

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
...
```

Parent topic: String Functions

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
```

Parent topic: String Functions

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)
```

Parent topic: String Functions

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 zeroes 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
...
```

Parent topic: String Functions

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 li
  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 seasc
 season_name |   rtrim
-------------+-----------
 1992-1993   | 1992-
 1993-1994   | 1993-
 1994-1995   | 1994-
 1995-1996   | 1995-
 1996-1997   | 1996-
...
```

Parent topic: String Functions

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
...
```

Parent topic: String Functions

Pattern Matching

This section covers pattern matching operators and functions, including supported regular expression (regex) operators and syntax.

In This Section

- SQL Operators and Pattern Matching Functions
- Regular Expression Details

Parent topic: SQL Functions and Operators

SQL Operators and Pattern Matching Functions

Yellowbrick supports a subset of the POSIX-style regular expressions (regex).

Within a Yellowbrick query, a pattern may be used in the following different ways:

- Preceded by one of the regex 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.

In This Section

- SQL Regex Operators
- REGEXP_REPLACE
- REGEXP_INSTR
- REGEXP_EXTRACT
- REGEXP_LIKE
- LIKE
- SIMILAR TO
- SIMILAR TO: Best Practices
- SUBSTRING (Pattern Match)
- Simple Regex Examples

Parent topic: Pattern Matching

SQL Regex Operators

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).

Parent topic: SQL Operators and Pattern Matching Functions

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)
 season_name | regexp_replace
-------------+----------------
 1992-1993   | year:1992-1993
 1993-1994   | year:1993-1994
 1994-1995   | year:1994-1995
(3 rows)
```

Parent topic: SQL Operators and Pattern Matching Functions

REGEXP_INSTR

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
 season_name | regexp_instr
-------------+--------------
 1992-1993   |            5
 1993-1994   |            5
 1994-1995   |            5
(3 rows)
```

Parent topic: SQL Operators and Pattern Matching Functions

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 wh
 season_name | regexp_extract
-------------+----------------
 1992-1993   | 1992-1993
 1993-1994   | 1993-1994
 1994-1995   | 1994-1995
(3 rows)
```

Parent topic: SQL Operators and Pattern Matching Functions

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)
```

Parent topic: SQL Operators and Pattern Matching Functions

LIKE

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
```

Parent topic: SQL Operators and Pattern Matching Functions

SIMILAR TO

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 regex notation, but the results are likely to differ from regex 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 regex-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.

Parent topic: SQL Operators and Pattern Matching Functions

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
--------+------+------+-------------------+------------+------------+--------------
    25 |   26 |   75 | Manchester United | Red Devils | Manchester | Old Trafford |
(1 row)
premdb=# select * from team
where name ~ '(Manchester United|Man Utd|Man United|Man U)';
 teamid | htid | atid |       name        |  nickname  |    city    |   stadium
--------+------+------+-------------------+------------+------------+--------------
    25 |   26 |   75 | Manchester United | Red Devils | Manchester | Old Trafford |
(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, keep in mind 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%|OB%|OS%|BU%|''|OY%|OZ%|VT%)'
```

as follows:

```
c1 SIMILAR TO '(12|13|14|15|16|17|OB|OS|BU|OY|OZ|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]|O[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).

Parent topic: SQL Operators and Pattern Matching Functions

SUBSTRING (Pattern Match)

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)
```

Parent topic: SQL Operators and Pattern Matching Functions

Simple Regex 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)
```

Parent topic: SQL Operators and Pattern Matching Functions

Regular Expression Details

Yellowbrick supports a subset of the POSIX-style regular expressions (regex), 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.

In This Section

- Character Classes and Escaped Special Characters
- Quantifiers
- Anchors
- Grouping Constructs
- Alternation Constructs
- Back References
- Flags

Parent topic: Pattern Matching

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

Short Form	Long Form	Description	Matches
	[[: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
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

Parent topic: Regular Expression Details

Quantifiers

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
--------+------+------+----------+----------+----------+-----------------+----------
      3 |    4 |   53 | Barnsley | Tykes    | Barnsley | Oakwell Stadium |    23009
(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      |       sta
--------+------+------+----------------------+-----------+---------------+---------
      6 |    7 |   56 | Blackpool            | Seasiders | Blackpool     | Bloomfiel
     26 |   27 |   76 | Middlesbrough        | Boro      | Middlesbrough | Riverside
     35 |   36 |   85 | Sheffield Wednesday  | Owls      | Sheffield     | Hillsboro
     37 |   38 |   87 | Stoke City           | Potters   | Stoke         | Britannia
     43 |   44 |   93 | West Bromwich Albion | Baggies   | Birmingham    | The Hawth
(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      |       sta
--------+------+------+---------------------+-----------+---------------+----------
      6 |    7 |   56 | Blackpool           | Seasiders | Blackpool     | Bloomfield
     26 |   27 |   76 | Middlesbrough       | Boro      | Middlesbrough | Riverside
     35 |   36 |   85 | Sheffield Wednesday | Owls      | Sheffield     | Hillsboroug
     37 |   38 |   87 | Stoke City          | Potters   | Stoke         | Britannia
(4 rows)
```

```
premdb=> select * from team where city ~ 'po*';
 teamid | htid | atid |          name           | nickname  |     city      |      s
--------+------+------+-------------------------+-----------+---------------+------
     6 |    7 |   56 | Blackpool               | Seasiders | Blackpool     | Bloomfi
    17 |   18 |   67 | Everton                 | Toffees   | Liverpool     | Goodisc
    20 |   21 |   70 | Ipswich Town            | Blues     | Ipswich       | Portman
    23 |   24 |   73 | Liverpool               | Reds      | Liverpool     | Anfield
    36 |   37 |   86 | Southampton             | Saints    | Southampton   | St. Mar
    47 |   48 |   97 | Wolverhampton Wanderers | Wolves    | Wolverhampton | Molineu
(6 rows)
```

```
premdb=> select * from team where city ~ 'po+';
 teamid | htid | atid |   name    | nickname  |   city    |     stadium     | capaci
--------+------+------+-----------+-----------+-----------+-----------------+------
     6 |    7 |   56 | Blackpool | Seasiders | Blackpool | Bloomfield Road |    173
    17 |   18 |   67 | Everton   | Toffees   | Liverpool | Goodison Park   |    402
    23 |   24 |   73 | Liverpool | Reds      | Liverpool | Anfield         |    447
(3 rows)
```

```
premdb=> select * from team where city ~ 'po{2,}';
 teamid | htid | atid |   name    | nickname  |   city    |     stadium     | capaci
--------+------+------+-----------+-----------+-----------+-----------------+------
     6 |    7 |   56 | Blackpool | Seasiders | Blackpool | Bloomfield Road |    173
    17 |   18 |   67 | Everton   | Toffees   | Liverpool | Goodison Park   |    402
    23 |   24 |   73 | Liverpool | Reds      | Liverpool | Anfield         |    447
(3 rows)
```

Parent topic: Regular Expression Details

Anchors

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
--------+------+------+-------------------+----------+------------+-----------------
      2 |    3 |   52 | Aston Villa       | Villains | Birmingham | Villa Park
     21 |   22 |   71 | Leeds United      | Whites   | Leeds      | Elland Road
     34 |   35 |   84 | Sheffield United  | Blades   | Sheffield  | Bramall Lane
     41 |   42 |   91 | Tottenham Hotspur | Spurs    | London     | White Hart Lane
(4 rows)
```

```
premdb=> select * from team where stadium ~ '\y[Ll]a';
 teamid | htid | atid |       name        | nickname |   city    |     stadium
--------+------+------+-------------------+----------+-----------+-----------------
```

```
     34 |   35 |   84 | Sheffield United  | Blades   | Sheffield | Bramall Lane    |
     41 |   42 |   91 | Tottenham Hotspur | Spurs    | London    | White Hart Lane |
(2 rows)
```

```
premdb=> select * from team where stadium ~ '\Y[Ll]a';
 teamid | htid | atid |     name     | nickname |    city    |   stadium    | capaci
--------+------+------+--------------+----------+------------+--------------+-------
      2 |    3 |   52 | Aston Villa  | Villains | Birmingham | Villa Park   |    4278
     21 |   22 |   71 | Leeds United | Whites   | Leeds      | Elland Road  |    3946
(2 rows)
```

Parent topic: Regular Expression Details

Grouping Constructs

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
 regexp_replace
-----------------
 Hello World ->
(1 row)

premdb=# select REGEXP_REPLACE('Hello World','(\w+) (?:\w+)','\& -> \1') from sys.c
   regexp_replace
----------------------
 Hello World -> Hello
(1 row)

premdb=# select REGEXP_REPLACE('Hello World','(?:\w+) (\w+)','\& -> \1') from sys.c
```

```
    regexp_replace
----------------------
 Hello World -> World
(1 row)
```

Parent topic: Regular Expression Details

Alternation Constructs

Yellowbrick supports the following constructs:

Operator	Description
|	OR operator **Note:** Either side of an OR operator (|) may contain an empty string.

Note: The Yellowbrick regex 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.

Parent topic: Regular Expression Details

Back References

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]') fr
   name     | regexp_replace
------------+-----------------
 Aston Villa | [Aston] [Villa]
 Blackpool   | Blackpool
 Liverpool   | Liverpool
(3 rows)
```

Parent topic: Regular Expression Details

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)
```

Parent topic: Regular Expression Details

Datetime Functions

This section covers functions that work with datetime data types and literals.

In This Section

- ADD_MONTHS
- AGE
- AT TIME ZONE
- CURRENT_DATE
- CURRENT_TIMESTAMP
- CURRENT_UTC_TIMESTAMP()
- DATEADD
- DATEDIFF
- DATENAME
- DATE_PART
- DATE_TRUNC
- DAY
- DAYS_BETWEEN
- EOMONTH
- EXTRACT
- GETDATE
- JUSTIFY_DAYS, _HOURS, _INTERVAL
- LAST_DAY

- LOCALTIME
- LOCALTIMESTAMP
- MAKE_DATE
- MAKE_INTERVAL
- MAKE_TIME
- MAKE_TIMESTAMP
- MAKE_TIMESTAMPTZ
- MONTH
- MONTHS_BETWEEN
- NEXT_DAY
- NOW()
- OVERLAPS
- TIMEOFDAY()
- TIMEZONE
- YEAR
- Supported Dateparts

Parent topic: SQL Functions and Operators

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=6
     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.

Parent topic: Datetime Functions

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)
```

Parent topic: Datetime Functions

AT TIME ZONE

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)
```

Parent topic: Datetime Functions

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)
```

Parent topic: Datetime Functions

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 zeroes 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.

Parent topic: Datetime Functions

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)
```

Parent topic: Datetime Functions

DATEADD

Add a specified interval of time to a date or timestamp value.

```
DATEADD( datepart, interval, { date | timestamp } )
```

datepart

See Supported Dateparts.

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. On the other hand, 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 '201
     timestamp      |     plus300secs
--------------------+---------------------
 2017-11-21 03:31:00 | 2017-11-21 03:36:00
(1 row)
```

Parent topic: Datetime Functions

DATEDIFF

Given a datepart, return the difference between two dates or timestamps.

```
DATEDIFF ( datepart, { date | timestamp | timestamptz }, { date | timestamp | timest
```

See Supported Dateparts.

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.

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 12:3
 ?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
 datediff
----------
        2
(1 row)
```

Parent topic: Datetime Functions

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) FRO
     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)
```

Parent topic: Datetime Functions

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 Dateparts. 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)
```

Parent topic: Datetime Functions

DATE_TRUNC

Truncate a datetime expression to the first hour, day, or month (for example), for that expression.

```
DATE_TRUNC('datepart', expression)
```

See Supported Dateparts. 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', '202
      date_trunc       |    date_trunc_plus1
-----------------------+------------------------
 2020-10-12 00:00:00-07 | 2020-10-13 00:00:00-07
(1 row)
```

Parent topic: Datetime Functions

DAY

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.

Parent topic: Datetime Functions

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
     matchday       | days_between
---------------------+--------------
 2010-03-06 00:00:00 |         3055
(1 row)

premdb=# select matchday, days_between(matchday, current_timestamp) from match where
     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 hour
     matchday       | days_between
---------------------+--------------
 1998-03-11 00:00:00 |         7433
(1 row)
```

Parent topic: Datetime Functions

EOMONTH

`EOMONTH` is a synonym for LAST_DAY.

Parent topic: Datetime Functions

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 Dateparts.

For example, 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)
```

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)
```

Parent topic: Datetime Functions

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)
```

Parent topic: Datetime Functions

JUSTIFY_DAYS, _HOURS, _INTERVAL

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_
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)
```

Parent topic: Datetime Functions

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)
```

Parent topic: Datetime Functions

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 zeroes 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)
```

Parent topic: Datetime Functions

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)
```

Parent topic: Datetime Functions

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)
```

Parent topic: Datetime Functions

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)
```

Parent topic: Datetime Functions

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)
```

Parent topic: Datetime Functions

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)
```

Parent topic: Datetime Functions

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)
```

Parent topic: Datetime Functions

MONTH

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.

Parent topic: Datetime Functions

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 mat
   num_months     |      matchday
------------------+---------------------
 -97.3225806451613 | 2010-03-06 00:00:00
(1 row)

premdb=# select months_between(current_date,matchday) num_months, matchday from mat
   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
 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.

Parent topic: Datetime Functions

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 a number of 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`

Parent topic: Datetime Functions

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)
```

Parent topic: Datetime Functions

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 number 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)
```

Parent topic: Datetime Functions

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)
```

Parent topic: Datetime Functions

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)
```

Parent topic: Datetime Functions

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.

Parent topic: Datetime Functions

Supported Dateparts

You can use the following dateparts as arguments to the following functions:

- DATE_PART
- DATE_TRUNC
- EXTRACT
- DATEADD
- DATENAME

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	
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	

Datepart	Description	Notes
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	
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	

Dateparts can be specified in uppercase or lowercase.

Parent topic: Datetime Functions

Mathematical Functions

This section covers mathematical functions and operators.

In This Section

- Mathematical Operators
- ABS
- ACOS
- ASIN
- CBRT
- CEIL
- COS
- DEGREES
- DIV
- ERF
- EXP
- FLOOR
- LN
- LOG
- MOD
- POWER
- PROBNORM
- RADIANS

- RANDOM()
- ROUND
- SIGN
- SIN
- SQRT
- TRUNC

Parent topic: SQL Functions and Operators

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.

Parent topic: Mathematical Functions

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
...
```

Parent topic: Mathematical Functions

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)
```

Parent topic: Mathematical Functions

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)
```

Parent topic: Mathematical Functions

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
...
```

Parent topic: Mathematical Functions

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
...
```

Parent topic: Mathematical Functions

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
...
```

Parent topic: Mathematical Functions

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
...
```

Parent topic: Mathematical Functions

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)
```

Parent topic: Mathematical Functions

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)
```

Parent topic: Mathematical Functions

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
```

Parent topic: Mathematical Functions

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 B
 seasonid |            avg             | floorgoals
----------+----------------------------+------------
       15 | 0.9973684210526315789473 6 |          0
       14 | 1.0236842105263157894736 8 |          1
       13 | 1.0657894736842105263157 8 |          1
        9 | 1.0657894736842105263157 8 |          1
        7 | 1.0684210526315789473684 2 |          1
...
```

Parent topic: Mathematical Functions

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
...
```

Parent topic: Mathematical Functions

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 | 5.90554345147192101283114722145180057 6
(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)
```

Parent topic: Mathematical Functions

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 de
         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
 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)
```

Parent topic: Mathematical Functions

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.

Parent topic: Mathematical Functions

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.

Parent topic: Mathematical Functions

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)
```

Parent topic: Mathematical Functions

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 | hts
----------------------+----------+---------------------+------+------+---------+---
 0.000131850332641704 |       16 | 2008-03-16 00:00:00 |   46 |   57 | 1-0     | 1-0
 0.000151282328515424 |        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)
```

Parent topic: Mathematical Functions

ROUND

Round a number (up or down) to the nearest integer. Optionally, specify a decimal precision for the rounding.

```
ROUND(number [, places])
```

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 of precision.

The first argument can be any numeric data type. However, if both arguments are used, the first argument cannot be a floating-point data type. If you want to round a floating-point number to a specific decimal precision, you must explicitly cast the number to a decimal data type. For example:

```
premdb=# create table floats (f1 float8);
CREATE TABLE
premdb=# insert into floats values(1234567890.12334567890);
INSERT 0 1
premdb=# select round(f1,3) from floats;
ERROR:  function round(double precision, integer) does not exist
LINE 2: select round(f1,3) from floats;
               ^
HINT:  No function matches the given name and argument types. You might need to add
premdb=# select round(f1::decimal(19,5),3) from floats;
    round
-----------------
 1234567890.123
(1 row)
```

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 ORDE
 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 8 |      1.074
        4 | 1.0736842105263157894736 8 |      1.074
        3 | 1.0779220779220779220779 2 |      1.078
       17 | 1.0789473684210526315789 4 |      1.079
        1 | 1.0800865800865800865800 8 |      1.080
        5 | 1.0815789473684210526315 7 |      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
...
```

Parent topic: Mathematical Functions

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
...
```

Parent topic: Mathematical Functions

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
...
```

Parent topic: Mathematical Functions

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
...
```

Parent topic: Mathematical Functions

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 ORDE
 seasonid |            avg             | truncgoals
----------+----------------------------+------------
       15 | 0.9973684210526315789473 |      0.997
       14 | 1.0236842105263157894736 |      1.024
       13 | 1.0657894736842105263157 |      1.066
        9 | 1.0657894736842105263157 |      1.066
        7 | 1.0684210526315789473684 |      1.068
...
```

Parent topic: Mathematical Functions

Aggregate Functions

This section describes standard SQL aggregate functions. For details about aggregate window functions (or *windowed aggregates*), see Window Functions.

In This Section

- AVG
- COUNT
- COUNT(DISTINCT)
- GROUP_CONCAT
- LISTAGG
- MAX
- MEDIAN()
- MIN
- PERCENTILE_CONT
- PERCENTILE_DISC
- STDDEV_SAMP and STDDEV_POP
- STRING_AGG
- SUM
- VAR_SAMP and VAR_POP

Parent topic: SQL Functions and Operators

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`

Parent topic: Aggregate Functions

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.

Parent topic: Aggregate Functions

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).

Parent topic: Aggregate Functions

GROUP_CONCAT

Concatenate a set of character strings and insert a delimiter between each string. See STRING_AGG for details.

Parent topic: Aggregate Functions

LISTAGG

Concatenate a set of character strings and insert a delimiter between each string. See STRING_AGG for details.

Parent topic: Aggregate Functions

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_m
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).

Parent topic: Aggregate Functions

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)
```

Parent topic: Aggregate Functions

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).

Parent topic: Aggregate Functions

PERCENTILE_CONT

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)
```

Parent topic: Aggregate Functions

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)
```

Parent topic: Aggregate Functions

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) s
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_p
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`).

Parent topic: Aggregate Functions

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-
    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-
    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-
    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-
   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
```

```
    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
    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
    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
    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
    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
    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
    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
    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
    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
    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
    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
    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
    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
    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
    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
(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
    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
    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
   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
   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
   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
   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
   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
   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
   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
   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
   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
    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)
```

Parent topic: Aggregate Functions

SUM

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)`

Parent topic: Aggregate Functions

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_at
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`).

Parent topic: Aggregate Functions

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.

In This Section

- Syntax for Window Functions
- 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)
- VAR_SAMP and VAR_POP (window functions)

Parent topic: SQL Functions and Operators

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 a number of rows that precede or follow the current row.

In `ROWS` mode, `CURRENT ROW` simply means the current row.

Parent topic: Window Functions

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
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)
```

Parent topic: Window Functions

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_de
```

`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 co
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
 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)
```

Parent topic: Window Functions

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)
```

Parent topic: Window Functions

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)
```

Parent topic: Window Functions

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 betwe
from team where city in ('London','Birmingham','Manchester');
 teamid |         name         |    city    |     stadium      | capacity | first_va
--------+----------------------+------------+------------------+----------+--------
     2 | Aston Villa          | Birmingham | Villa Park       |    42785 |       427
     4 | Birmingham City      | Birmingham | St. Andrew's     |    30016 |       427
    43 | West Bromwich Albion | Birmingham | The Hawthorns    |    27000 |       300
     1 | Arsenal              | London     | Emirates Stadium |    60260 |       602
    12 | Charlton Athletic    | London     | The Valley       |    27111 |       602
    13 | Chelsea              | London     | Stamford Bridge  |    41663 |       271
    15 | Crystal Palace       | London     | Selhurst Park    |   [NULL] |       410
    18 | Fulham               | London     | Craven Cottage   |   [NULL] |      [NUL
    32 | Queens Park Rangers  | London     | Loftus Road      |   [NULL] |       361
    41 | Tottenham Hotspur    | London     | White Hart Lane  |    36284 |       362
    44 | West Ham United      | London     | Upton Park       |    35016 |       362
    46 | Wimbledon            | London     | Selhurst Park    |   [NULL] |       350
    24 | Manchester City      | Manchester | Etihad Stadium   |    55097 |       550
    25 | Manchester United    | Manchester | Old Trafford     |    75635 |       550
(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 betw
from team where city in ('London','Birmingham','Manchester');
 teamid |         name         |    city    |     stadium      | capacity | first_va
--------+----------------------+------------+------------------+----------+--------
      2 | Aston Villa          | Birmingham | Villa Park       |    42785 |       427
      4 | Birmingham City      | Birmingham | St. Andrew's     |    30016 |       427
     43 | West Bromwich Albion | Birmingham | The Hawthorns    |    27000 |       300
      1 | Arsenal              | London     | Emirates Stadium |    60260 |       602
     12 | Charlton Athletic    | London     | The Valley       |    27111 |       602
     13 | Chelsea              | London     | Stamford Bridge  |    41663 |       271
     15 | Crystal Palace       | London     | Selhurst Park    |   [NULL] |       410
     18 | Fulham               | London     | Craven Cottage   |   [NULL] |      [NUL
     32 | Queens Park Rangers  | London     | Loftus Road      |   [NULL] |      [NUL
     41 | Tottenham Hotspur    | London     | White Hart Lane  |    36284 |      [NUL
     44 | West Ham United      | London     | Upton Park       |    35016 |       362
     46 | Wimbledon            | London     | Selhurst Park    |   [NULL] |       350
     24 | Manchester City      | Manchester | Etihad Stadium   |    55097 |       550
     25 | Manchester United    | Manchester | Old Trafford     |    75635 |       550
(14 rows)
```

Parent topic: Window Functions

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)
```

Parent topic: Window Functions

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)
```

Parent topic: Window Functions

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_de
```

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 ci
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)
```

Parent topic: Window Functions

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)
```

Parent topic: Window Functions

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)
```

Parent topic: Window Functions

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)
```

Parent topic: Window Functions

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
```

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)
```

Parent topic: Window Functions

PERCENTILE_DISC (window function)

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
```

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)
```

Parent topic: Window Functions

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;
```

Parent topic: Window Functions

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)
```

Parent topic: Window Functions

STDDEV_SAMP and STDDEV_POP (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 preced
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`).

Parent topic: Window Functions

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
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)
```

Parent topic: Window Functions

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
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`).

Parent topic: Window Functions

Conditional Expressions

This section describes special SQL expressions that evaluate conditions.

In This Section

- CASE
- COALESCE
- DECODE
- GREATEST and LEAST
- IIF
- ISNULL
- NULLIF
- NVL
- NVL2

Parent topic: SQL Functions and Operators

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)
```

Parent topic: Conditional Expressions

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...
```

Parent topic: Conditional Expressions

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)
```

Parent topic: Conditional Expressions

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`.

Parent topic: Conditional Expressions

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',
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',
CASE WHEN avg_att > 38.000 is true THEN 'Above 2018-19 season average' ELSE 'Below 2
FROM team ORDER BY teamid LIMIT 5;
      name       |     stadium      |      average_attendance      |      average_at
-----------------+------------------+------------------------------+--------------
 Arsenal         | Emirates Stadium | Above 2018-19 season average | Above 2018-19
 Aston Villa     | Villa Park       | Below 2018-19 season average | Below 2018-19
 Barnsley        | Oakwell Stadium  | Below 2018-19 season average | Below 2018-19
 Birmingham City | St. Andrew's     | Below 2018-19 season average | Below 2018-19
 Blackburn Rovers | Ewood Park      | Below 2018-19 season average | Below 2018-19
(5 rows)
```

Parent topic: Conditional Expressions

ISNULL

ISNULL and NVL are equivalent conditional expressions. See NVL.

Parent topic: Conditional Expressions

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
...
```

Parent topic: Conditional Expressions

NVL

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.

Parent topic: Conditional Expressions

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 des
 teamid |     city
--------+---------------
     50 | No City
     49 | No City
     48 | No City
     47 | Wolverhampton
     46 | London
     45 | Wigan
     44 | London
...
```

Parent topic: Conditional Expressions

Formatting Functions

This section covers functions that are used to coerce values to specific data types and formats.

In This Section

- FLOAT4, FLOAT8
- INT2, INT4, INT8
- TO_CHAR
- TO_DATE
- TO_NUMBER
- TO_TIMESTAMP
- Formats for Datetime Values
- Formats for Numeric Values

Parent topic: SQL Functions and Operators

FLOAT4, FLOAT8

Coerce numeric expressions to floating-point numbers. See Data Type Casting.

Parent topic: Formatting Functions

INT2, INT4, INT8

Coerce numeric expressions to integers. See Data Type Casting.

Parent topic: Formatting Functions

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 [,
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 whe
     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
        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

Pattern	Description
`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 zeroes 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
`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

Pattern	Description
`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

Parent topic: Formatting Functions

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)
```

Parent topic: Formatting Functions

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 d
 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
 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 seaso
 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.

Parent topic: Formatting Functions

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 s
ERROR:  Invalid datetime format YYYY MM DD HH MI SS SS
DETAIL:   [reason => Invalid datetime format string. Reason: Duplicate token SS fou
```

Parent topic: Formatting Functions

Formats for Datetime Values

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

Format	Description
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)
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 zeroes and padding blanks)

Format	Description
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 zeroes that appear in the format for microseconds. For example:

```
premdb=# select to_timestamp('20180228 11 59 59 000999 AM', 'YYYYMMDD HH12 MI SS US
       to_timestamp
----------------------------
```

```
 2018-02-28 11:59:59.999-08
(1 row)
```

Parent topic: Formatting Functions

Formats for Numeric Values

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.

Format	Example	Description
`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`).

Parent topic: Formatting Functions

Type-Safe Casting Functions

The following SQL functions are supported for compatibility with applications developed in 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, run the following commands:

```
yellowbrick=# alter system set enable_safe_compat_functions_with_nz to true;
WARNING:  Will be effective after the next server configuration reload, or after the
ALTER SYSTEM
yellowbrick=# select pg_reload_conf();
 pg_reload_conf
----------------
 t
(1 row)
```

In This Section

- SAFE_TO_DATE
- SAFE_TO_DOUBLE
- SAFE_TO_TIMESTAMP

Parent topic: SQL Functions and Operators

SAFE_TO_DATE

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)
```

Parent topic: Type-Safe Casting Functions

SAFE_TO_DOUBLE

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)
```

Parent topic: Type-Safe Casting Functions

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)
```

Parent topic: Type-Safe Casting Functions

JSON Functions

This section covers functions that operate on JSON strings and compose objects and arrays.

In This Section

- JSON_ARRAY_STR
- JSON_INGEST
- JSON_LOOKUP
- JSON_OBJECT_STR

Parent topic: SQL Functions and Operators

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, ftsc
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"]
...
```

Parent topic: JSON Functions

JSON_INGEST

Tip: This function was added in **Version 5.2.23** and is not available in earlier 5.2.x versions.

Convert a JSON input string into a Document Object Model (DOM) representation of the string that you can query with the JSON_LOOKUP function.

```
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:

```
yellowbrick=# create table json_movies (jsonstr varchar(1000));
CREATE TABLE
yellowbrick=# insert into json_movies values('{"title":"After Dark in Central Park",
INSERT 0 1
yellowbrick=# select * from json_movies;
                                jsonstr
-------------------------------------------------------------------------
 {"title":"After Dark in Central Park","year":1900,"cast":[],"genres":[]}
(1 row)
```

Now create a table that can store the JSON data alongside the results of the `JSON_INGEST` function:

```
yellowbrick=# create table json_movies_ingested as select jsonstr, json_ingest(jsons
SELECT 1
yellowbrick=# \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:

```
yellowbrick=# select json_lookup(json_ingest,'$.title','jpath') from json_movies_ing
       json_lookup
---------------------------
 After Dark in Central Park
(1 row)
yellowbrick=# select json_lookup(json_ingest,'$.year','jpath') from json_movies_inge
 json_lookup
-------------
 1900
(1 row)
```

Parent topic: JSON Functions

JSON_LOOKUP

Tip: This function was added in **Version 5.2.23** and is not available in earlier 5.2.x versions.

Return the key value for a JSON object.

```
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 st
```

- 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:

```
premdb=# 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:

```
premdb=# create table seasons_ingest as select "25 Seasons", json_ingest("25 Seasons
SELECT 25
premdb=# \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:

```
premdb=# select json_lookup(json_ingest,'$.Season Years','jpath') from seasons_inges
 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:

```
premdb=# 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:

```
premdb=# 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:

```
yellowbrick=# create table json_gloss(entry varchar(1000);
yellowbrick(#
yellowbrick=# create table json_gloss(entry varchar(1000));
CREATE TABLE
yellowbrick=# insert into json_gloss(entry) values('{
yellowbrick'#     "glossary": {
```

```
yellowbrick'#         "title": "example glossary",
yellowbrick'#         "GlossDiv": {
yellowbrick'#             "title": "S",
yellowbrick'#             "GlossList": {
yellowbrick'#                 "GlossEntry": {
yellowbrick'#                     "ID": "SGML",
yellowbrick'#                     "SortAs": "SGML",
yellowbrick'#                     "GlossTerm": "Standard Generalized Markup Language
yellowbrick'#                     "Acronym": "SGML",
yellowbrick'#                     "Abbrev": "ISO 8879:1986",
yellowbrick'#                     "GlossDef": {
yellowbrick'#                         "para": "A meta-markup language, used to creat
yellowbrick'#                         "GlossSeeAlso": [
yellowbrick'#                             "GML",
yellowbrick'#                             "XML"
yellowbrick'#                         ]
yellowbrick'#                     },
yellowbrick'#                     "GlossSee": "markup"
yellowbrick'#                 }
yellowbrick'#             }
yellowbrick'#         }
yellowbrick'#     }
yellowbrick'# }');
INSERT 0 1
```

Now create a table that holds both the original data and the output for the `JSON_INGEST` function:

```
yellowbrick=# create table json_gloss_ingested as select entry, json_ingest(entry) f
SELECT 1
```

Now run queries on the `json_gloss_ingested` table:

```
yellowbrick=# select json_lookup(json_ingest,'$.glossary.title','jpath') from json_g
   json_lookup
------------------
 example glossary
(1 row)
yellowbrick=# select json_lookup(json_ingest,'$.glossary.GlossDiv.GlossList.GlossEnt
 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 `''`:

```
premdb=# select * from scalar_json;
    col1
---------------
 "2018-2019"
 "Twenty-five"
 "25"
(3 rows)

premdb=# select json_lookup(col1,'','jpointer') from scalar_json;
 json_lookup
-------------
 2018-2019
 Twenty-five
 25
(3 rows)
```

Parent topic: JSON Functions

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_na
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, 'nu
                                        seasontable
-------------------------------------------------------------------------------
 {"seasonid" : 1, "season_name" : "1992-1993", "numteams" : 22, "winners" : "Manches
 {"seasonid" : 2, "season_name" : "1993-1994", "numteams" : 22, "winners" : "Manches
 {"seasonid" : 3, "season_name" : "1994-1995", "numteams" : 22, "winners" : "Blackbu
 {"seasonid" : 4, "season_name" : "1995-1996", "numteams" : 20, "winners" : "Manches
 {"seasonid" : 5, "season_name" : "1996-1997", "numteams" : 20, "winners" : "Manches
 {"seasonid" : 6, "season_name" : "1997-1998", "numteams" : 20, "winners" : "Arsenal
 {"seasonid" : 7, "season_name" : "1998-1999", "numteams" : 20, "winners" : "Manches
 {"seasonid" : 8, "season_name" : "1999-2000", "numteams" : 20, "winners" : "Manches
 {"seasonid" : 9, "season_name" : "2000-2001", "numteams" : 20, "winners" : "Manches
 {"seasonid" : 10, "season_name" : "2001-2002", "numteams" : 20, "winners" : "Arsena
 {"seasonid" : 11, "season_name" : "2002-2003", "numteams" : 20, "winners" : "Manche
 {"seasonid" : 12, "season_name" : "2003-2004", "numteams" : 20, "winners" : "Arsena
 {"seasonid" : 13, "season_name" : "2004-2005", "numteams" : 20, "winners" : "Chelse
 {"seasonid" : 14, "season_name" : "2005-2006", "numteams" : 20, "winners" : "Chelse
 {"seasonid" : 15, "season_name" : "2006-2007", "numteams" : 20, "winners" : "Manche
 {"seasonid" : 16, "season_name" : "2007-2008", "numteams" : 20, "winners" : "Manche
 {"seasonid" : 17, "season_name" : "2008-2009", "numteams" : 20, "winners" : "Manche
 {"seasonid" : 18, "season_name" : "2009-2010", "numteams" : 20, "winners" : "Chelse
 {"seasonid" : 19, "season_name" : "2010-2011", "numteams" : 20, "winners" : "Manche
 {"seasonid" : 20, "season_name" : "2011-2012", "numteams" : 20, "winners" : "Manche
 {"seasonid" : 21, "season_name" : "2012-2013", "numteams" : 20, "winners" : "Manche
 {"seasonid" : 22, "season_name" : "2013-2014", "numteams" : 20, "winners" : "Manche
 {"seasonid" : 23, "season_name" : "2014-2015", "numteams" : 20, "winners" : "Chelse
 {"seasonid" : 24, "season_name" : "2015-2016", "numteams" : 20, "winners" : "Leices
 {"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
ERROR:  The key for this json object is invalid / null / not a string
DETAIL:   [param=>7]
```

Parent topic: JSON Functions

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

Style	Pattern	Notes
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	
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

Style	Pattern	Notes
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.

From	To	Rewrite
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`
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', '/',
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)
```

Parent topic: SQL Functions and Operators

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)
```

Parent topic: SQL Functions and Operators

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`.

Parent topic: SQL Functions and Operators

System Functions

This section covers functions that return information about the system or generate data.

In This Section

- CURRENT_DATABASE()
- CURRENT_QUERY()
- CURRENT_SCHEMA
- CURRENT_SCHEMAS()
- CURRENT_SETTING
- CURRENT_USER
- GETDATABASEENCODING()
- HAS_DATABASE_PRIVILEGE
- HAS_FUNCTION_PRIVILEGE
- HAS_KEY_PRIVILEGE
- HAS_ROLE_PRIVILEGE
- HAS_SCHEMA_PRIVILEGE
- HAS_SYSTEM_PRIVILEGE
- HAS_TABLE_PRIVILEGE
- PG_ADVISORY_LOCK
- PG_ADVISORY_UNLOCK
- PG_HAS_ROLE
- PG_RELOAD_CONF()

- PG_TRY_ADVISORY_LOCK
- SESSION_USER
- sys.execution_id()
- sys.filesys_number_from_rowid
- sys.gen_random_uuid()
- sys.inode_number_from_rowid
- sys.ldap_sync
- sys.oldest_rollback_point_id_in_replica
- sys.row_number_from_rowid
- sys.worker_id()
- sys.worker_id_from_rowid
- sys.worker_uuid
- VERSION()
- yb_terminate_session

Parent topic: SQL Functions and Operators

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)
```

Parent topic: System Functions

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)
```

Parent topic: System Functions

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

The `search_path` configuration parameter defines the visibility of schemas to users.

```
yellowbrick=# set search_path to bobr,sys;
SET

yellowbrick=# show search_path;
 search_path
-------------
bobr, sys
(1 row)
```

Alternatively, you can use a `set schema` command with single quotes around the schema name to change the current schema.

```
yellowbrick=# set schema 'bobr';
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.

The `search_path` parameter additionally allows you to control which schemas do not require schema qualification when users access tables, views, and other objects.

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.

$user

The first schema on the search path is the current schema, which is the default location for creating new objects. When running the `show` command, the first element specifies that a schema with the same name as the current user is to be searched. The element is ignored if the schema does not exist. `“$user”` is the default placeholder that resolves to the current user’s name schema. The second element of the show command return refers to the `public` schema.

```
yellowbrick=# show search_path;
search_path
-----------------
"$user", public
(1 row)
```

The result of this `show` command means:

- The user exists with the ability to create tables.
- The user's named schema exists, and the user owns it.
- The user schema is listed first in the user's search path, so new tables are automatically created and searched there first.

Parent topic: System Functions

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.

Parent topic: System Functions

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.

Parent topic: System Functions

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`.

Parent topic: System Functions

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, c
You are now connected to database "premdb_utf8" as user "yellowbrick".
premdb_utf8=# select getdatabaseencoding();
 getdatabaseencoding
---------------------
 UTF8
(1 row)
```

See also CREATE DATABASE.

Parent topic: System Functions

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)
```

Parent topic: System Functions

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)
```

Parent topic: System Functions

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)
```

Parent topic: System Functions

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)
```

Parent topic: System Functions

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)
```

Parent topic: System Functions

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)
```

Parent topic: System Functions

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)
```

Parent topic: System Functions

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.

Parent topic: System Functions

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.

Parent topic: System Functions

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 (in other words, 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)
```

Parent topic: System Functions

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, such as `idle_session_timeout`.

See also ALTER SYSTEM and SHOW.

Note: This reload function is not supported in queries with table references in the `FROM` clause.

Parent topic: System Functions

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.

Parent topic: System Functions

SESSION_USER

Return the user who initiated the session.

```
SESSION_USER
```

```
premdb=# \c premdb yellowbrick
You are now connected to database "premdb" as user "yellowbrick".
premdb=# select session_user;
 session_user
--------------
 yellowbrick
(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.

Parent topic: System Functions

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)
```

Parent topic: System Functions

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 li
     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)
```

Parent topic: System Functions

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.

Parent topic: System Functions

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
 inode_number_from_rowid
-------------------------
                    4097
                    8193
                   19457
                   19458
                   46082
                   68609
                  130049
                  146433
                  149505
                  196609
                  250881
(11 rows)
```

Parent topic: System Functions

sys.ldap_sync

Synchronize LDAP users and groups.

```
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).

Parent topic: System Functions

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.

```
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]
```

Parent topic: System Functions

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 worker, 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)
```

Parent topic: System Functions

sys.worker_id()

Return the logical ID of the worker 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 worker ID, depending on which worker 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 worker that *processed* the row (not the worker 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)
```

Parent topic: System Functions

sys.worker_id_from_rowid

Return the worker 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 worker 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)
```

Parent topic: System Functions

sys.worker_uuid

Given a logical worker ID, return the UUID of that worker if it was involved in the query where the function is used. If the worker 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)
```

Parent topic: System Functions

VERSION()

Return the version of the Yellowbrick software that is installed. See also Checking the Server Version.

```
VERSION()
```

For example:

```
premdb=# select version();
                version
----------------------------------------
 Yellowbrick Database version 1.2.0-4309
(1 row)
```

Parent topic: System Functions

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
------------+------------------+-----------------+-------------------+-------------
      18120 | ybsql            | [NULL]          | [NULL]            |        4400 |
      18126 | ybsql            | [NULL]          | [NULL]            |       16388 |
(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.

Parent topic: System Functions

Network Address Functions

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

In This Section

- CONTAINS
- HOSTMASK
- INET_MERGE
- IPV4_GET_INET, IPV6_GET_INET
- IPV4_GET_MASKBITS, IPV6_GET_MASKBITS

- IPV4_NETMASK, IPV6_NETMASK
- MACADDR8_SET7BIT
- TEXT
- TRUNC

Parent topic: SQL Functions and Operators

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
 contains
----------
 t
(1 row)

premdb=# select contains(ipv6 '684D:1111:222:3333:4444:5555:6:88',128,ipv6 '684D:111
 contains
----------
 f
(1 row)
```

Parent topic: Network Address Functions

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
               hostmask
----------------------------------------
 0000:00b8:85a3:0000:0000:8a2e:0370:7334
(1 row)
```

Parent topic: Network Address Functions

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
   inet_merge
----------------
 192.168.0.0/22
(1 row)
```

```
premdb=# select ipv6_inet_merge('2001:0db8:85a3:0000:0000:8a2e:0370:0000', 128, '200
             ipv6_inet_merge
--------------------------------------------
 2001:0db8:85a3:0000:0000:8a2e:0370:0000/113
(1 row)
```

Parent topic: Network Address Functions

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)
```

Parent topic: Network Address Functions

IPV4_GET_MASKBITS, IPV6_GET_MASKBITS

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 sy
 ipv6_get_maskbits
-------------------
                 0
(1 row)
```

Parent topic: Network Address Functions

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)
```

Parent topic: Network Address Functions

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)
```

Parent topic: Network Address Functions

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.c
                   text
-----------------------------------------------
 2001:0db8:85a3:0000:0000:8a2e:0370:0000/128
(1 row)
```

Parent topic: Network Address Functions

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)
```

Parent topic: Network Address Functions

SQL Reserved Words

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.

```
ALL
 ANALYSE
 ANALYZE
 AND
 ANY
 ARRAY
 AS
 ASC
 ASYMMETRIC
 AUTHORIZATION
 AUTOANALYZE
 BINARY
 BOTH
 CASE
 CAST
 CHECK
 COLLATE
 COLLATION
 COLUMN
 CONCURRENTLY
 CONSTRAINT
 CREATE
 CROSS
 CURRENT_CATALOG
 CURRENT_DATE
 CURRENT_ROLE
 CURRENT_SCHEMA
 CURRENT_TIME
 CURRENT_TIMESTAMP
 CURRENT_USER
 DEFAULT
 DEFERRABLE
 DESC
 DISTINCT
 DO
 ELSE
 END
 EXCEPT
```

```
FALSE
FETCH
FOR
FOREIGN
FREEZE
FROM
FULL
GRANT
GROUP
HAVING
HLL
IGNORE
ILIKE
IN
INITIALLY
INNER
INTERSECT
INTO
IS
ISNULL
JOIN
LATERAL
LEADING
LEFT
LIKE
LIMIT
LOCALTIME
LOCALTIMESTAMP
MINUS
NATURAL
NOT
NOTNULL
NULL
OFFSET
ON
ONLY
OR
ORDER
OUTER
OVERLAPS
PARTITION
PLACING
PRIMARY
REFERENCES
RESPECT
```

```
RETURNING
RIGHT
ROWID
SAMPLED
SELECT
SESSION_USER
SIMILAR
SOME
SYMMETRIC
TABLE
TABLESAMPLE
THEN
TO
TRAILING
TRUE
UNION
UNIQUE
USER
USING
VARIADIC
VERBOSE
WHEN
WHERE
WINDOW
WITH
```

Parent topic: SQL Reference

Error Codes

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 readabl
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
YB016    X    InvalidBulkDataLength     The bulk load data stream was invalid (bad l
YB017    X    InvalidBulkDataCrc        The bulk load data stream was invalid (bad C
YB018    X    ExternalConnectTimeout    External client establishing connection time
YB019    X    JSONInvalidKey            The key for this json object is invalid / nu
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 exce
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 c
WL002    E    ERRCODE_WORKLOAD_WAIT_TIME_EXCEEDED                           wait ti
WL003    E    ERRCODE_WORKLOAD_EXECUTION_TIME_EXCEEDED                      executi
WL004    E    ERRCODE_WORKLOAD_ROWCOUNT_EXCEEDED                            row cou
```

```
WL005    E    ERRCODE_WORKLOAD_MEMORY_EXCEEDED                          memory
WL006    E    ERRCODE_WORKLOAD_OTEHR_ERROR                              WLM oth
WL007    E    ERRCODE_WORKLOAD_SHUTTING_DOWN                            WLM shu
WL008    E    ERRCODE_WORKLOAD_COMPILE_ERROR                            compile
WL009    E    ERRCODE_WORKLOAD_CONFIG_ERROR                             WLM con
WL009    E    ERRCODE_WORKLOAD_CANNOT_MOVE_QUERY                        cannot
WL010    E    ERRCODE_WORKLOAD_CANNOT_RESTART_QUERY                     cannot
WL011    E    ERRCODE_WORKLOAD_PRIOR_QUERY_NOT_COMPLETED                prior q
WL012    E    ERRCODE_WORKLOAD_QUERY_CANNOT_THROTTLE                    query w
WL013    E    ERRCODE_WORKLOAD_COMPILE_SHUTDOWN                         query c
WL014    E    ERRCODE_WORKLOAD_RULE_ABORT                               WLM rul
WL015    E    ERRCODE_WORKLOAD_RESOURCE_POOL_FULL                       WLM res
WL016    E    ERRCODE_WORKLOAD_RESTART_CLIENT_WAIT                      query c
WL017    E    ERRCODE_WORKLOAD_CANCEL_CLIENT_WAIT                       query c

# WM Error codes from the worker manager
WM001    E   ERRCODE_WORKER_OFFLINE                                     worker
WM002    E   ERRCODE_SYSTEM_NOT_READY                                   system
WM003    E   ERRCODE_CANCEL_FOR_RESTART                                 Query r
WM004    E   ERRCODE_MISMATCHED_WORKER_ID                               mismat
WM005    E   ERRCODE_WORKER_CONFIG_ERROR                                worker

# 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 hand
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 ba
BR003   E   ERRCODE_BAR2_NO_VALID_SNAPSHOT_IN_TARGET        Target does not contai
BR004   E   ERRCODE_BAR2_TARGET_MODIFIED                    Target has been modifi
BR005   E   ERRCODE_BAR2_NOTHING_TO_RESTORE                 Target does not contai
BR006   E   ERRCODE_BAR2_CROSS_DB_VIEW                      Cannot restore cross-DB
BR007   E   ERRCODE_BAR2_RESTORE_TO_NON_EMPTY_DB            Cannot restore to targe
BR008   E   ERRCODE_BAR2_NO_VALID_SNAPSHOT_IN_BACKUP_SET    Cannot locate snapshot
BR011   E   ERRCODE_BAR2_INVALID_LOCATION                   Cannot find DB metadata

#
# 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 n
KUSEG    E   Segv                                           Segfault
KU001    XS  InvalidConfigAttributeIndex                     Invalid c
KU002    XS  InvalidConfigTypeName                           Invalid c
KU003    XS  IOValidation                                    Malformed
```

```
KU004    XS  InvalidIORequest                                  Malforme
KU005    XS  InvalidNVMeIORequest                              Malforme
KU006    XS  FBDDriverFailure                                  FakeBloc
KU007    XS  NVMeDriverFailure                                 NMVe dri
KU008    XS  UnexpectedTCPSocketError                          Unexpect
KU009    X   ScheduleFailure                                   Failed t
KU010    XS  IBDriverFailure                                   IB drive
KU011    X   FileSystemMissingPrefix                          Could no
KU012    X   FileSystemNonUniquePrefix                        Too many
KU013        FILESYSTEMOUTOFINODE                              Out of i
KU016        FPGADRIVERFAILURE                                 FPGA dri
KU017        BOUNCEBUFFERDRIVERFAILURE                         Bounce b
KUSEG        SEGV                                              Segfault
KUUUU    E   UnexpectedException                               Generic
KUXXX    XS  FileSystemOutOfINode                              Out of i
```

Kernel Expected Exceptions

```
Base: ExpectedException
KE001    E   MissingFile                                       File not
KE002    E   OutOfMemory                                       Out of m
KE003    X   ShuttingDown                                      Shutting
KE004    X   QueueFull                                         Queue fu
KE005    X   InvalidTopology                                   Invalid
KE006    XS  InvalidConfig                                     Invalid
KE007    E   MissingID                                         Missing
KE008    E   UnknownConfigAttribute                            Unkown c
KE009    E   ConfigParsingError                                Config p
KE010    X   FileRead                                          File rea
KE011    X   FileWrite                                         File wri
KE012    X   OutOfRange                                        Out of r
KE013    E   ArithmeticError                                   Arithmet
KE014    X   SystemBusy                                        System b
KE015    X   FileSystemSpaceFull                              File sys
KE016    X   FileSystemCorrupt                                 File sys
KE017    X   FileSystemMissingFile                            Missing
KE018    X   FileSystemMetadataFull                           Metadata
KE019    X   FileSystemOutOfBounds                            File sys
KE020    X   FileSystemBusy                                   File sys
KE021    X   FileSystemDuplicateName                         File sys
KE022    XS  TCPSocketError                                    TCP sock
KE023    XS  RPCProtocolError                                  RPC prot
```

```
KE024    X   BadArguments                                          Bad argu
KE025    X   FileSystemUnsupportedDeviceSize                       Unsuppor
KE026    X   MissingShard                                          Missing
KE027    X   MissingKRI                                            Missing
KE028    X   UnreconstructableFile                                 File is
KE029    X   SpillLimitsExceeded                                   WLM assi
KE030    X   FileNotVisibleInGC                                    Given fi
KE031    X   FileSystemSpillFull                                   File sys
KE032    X   SpillLimitExceeded                                    Attempt
KE033    x   BadTimeZone                                           Bad Time
KE036        TCPSOCKETTIMEOUT                                      TCP sock
KE038        RPCCHANNELCLOSED                                      RPC chan
KE039        RPCCHANNELBROKEN                                      RPC chan
```

Class 00 - Successful Completion

```
00000    S   ERRCODE_SUCCESSFUL_COMPLETION                         successf
```

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_
01008    W   ERRCODE_WARNING_IMPLICIT_ZERO_BIT_PADDING             implicit_
01003    W   ERRCODE_WARNING_NULL_VALUE_ELIMINATED_IN_SET_FUNCTION  null_valu
01007    W   ERRCODE_WARNING_PRIVILEGE_NOT_GRANTED                 privileg
01006    W   ERRCODE_WARNING_PRIVILEGE_NOT_REVOKED                 privileg
01004    W   ERRCODE_WARNING_STRING_DATA_RIGHT_TRUNCATION          string_da
01P01    W   ERRCODE_WARNING_DEPRECATED_FEATURE                    deprecat
```

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_additi
```

Class 03 - SQL Statement Not Yet Complete

```
03000    E   ERRCODE_SQL_STATEMENT_NOT_YET_COMPLETE                    sql_state
```

Class 08 - Connection Exception

```
08000    E   ERRCODE_CONNECTION_EXCEPTION                             connecti
08003    E   ERRCODE_CONNECTION_DOES_NOT_EXIST                        connecti
08006    E   ERRCODE_CONNECTION_FAILURE                               connecti
08001    E   ERRCODE_SQLCLIENT_UNABLE_TO_ESTABLISH_SQLCONNECTION      sqlclient
08004    E   ERRCODE_SQLSERVER_REJECTED_ESTABLISHMENT_OF_SQLCONNECTION sqlserver
08007    E   ERRCODE_TRANSACTION_RESOLUTION_UNKNOWN                   transacti
08P01    E   ERRCODE_PROTOCOL_VIOLATION                               protocol_
08P02    E   ERRCODE_INVALID_EXECID                                   invalid_e
```

Class 09 - Triggered Action Exception

```
09000    E   ERRCODE_TRIGGERED_ACTION_EXCEPTION                       triggered
```

Class 0A - Feature Not Supported

```
0A000    E   ERRCODE_FEATURE_NOT_SUPPORTED                           feature_n
```

Class 0B - Invalid Transaction Initiation

```
0B000    E   ERRCODE_INVALID_TRANSACTION_INITIATION                  invalid_t
```

Class 0F - Locator Exception

```
0F000    E   ERRCODE_LOCATOR_EXCEPTION                                locator_e
0F001    E   ERRCODE_L_E_INVALID_SPECIFICATION                        invalid_
```

Class 0L - Invalid Grantor

```
0L000    E   ERRCODE_INVALID_GRANTOR                                  invalid_g
0LP01    E   ERRCODE_INVALID_GRANT_OPERATION                          invalid_g
```

Class 0P - Invalid Role Specification

```
0P000    E   ERRCODE_INVALID_ROLE_SPECIFICATION                       invalid_r
```

Class 0Z - Diagnostics Exception

```
0Z000    E   ERRCODE_DIAGNOSTICS_EXCEPTION                            diagnosti
0Z002    E   ERRCODE_STACKED_DIAGNOSTICS_ACCESSED_WITHOUT_ACTIVE_HANDLER   stacked_d
```

Class 20 - Case Not Found

```
20000    E   ERRCODE_CASE_NOT_FOUND                                   case_not_
```

Class 21 - Cardinality Violation

```
# this means something returned the wrong number of rows
21000    E   ERRCODE_CARDINALITY_VIOLATION                                more_tha
21001    E   ERRCODE_CARDINALITY_DUPLICATE                                data_con
```

Class 22 - Data Exception

```
22000    E   ERRCODE_DATA_EXCEPTION                                       data_exc
2202E    E   ERRCODE_ARRAY_ELEMENT_ERROR
2202E    E   ERRCODE_ARRAY_SUBSCRIPT_ERROR                               array_su
22021    E   ERRCODE_CHARACTER_NOT_IN_REPERTOIRE                          characte
22008    E   ERRCODE_DATETIME_FIELD_OVERFLOW                             datetime_
22031    E   ERRCODE_DATETIME_VALUE_OUT_OF_RANGE                         datetime
22012    E   ERRCODE_DIVISION_BY_ZERO                                    division_
22005    E   ERRCODE_ERROR_IN_ASSIGNMENT                                 error_in_
2200B    E   ERRCODE_ESCAPE_CHARACTER_CONFLICT                           escape_c
22022    E   ERRCODE_INDICATOR_OVERFLOW                                  indicato
22015    E   ERRCODE_INTERVAL_FIELD_OVERFLOW                             interval_
2201E    E   ERRCODE_INVALID_ARGUMENT_FOR_LOG                            invalid_a
22014    E   ERRCODE_INVALID_ARGUMENT_FOR_NTILE                          invalid_a
22016    E   ERRCODE_INVALID_ARGUMENT_FOR_NTH_VALUE                     invalid_a
2201F    E   ERRCODE_INVALID_ARGUMENT_FOR_POWER_FUNCTION                 invalid_a
2201G    E   ERRCODE_INVALID_ARGUMENT_FOR_WIDTH_BUCKET_FUNCTION          invalid_a
22018    E   ERRCODE_INVALID_CHARACTER_VALUE_FOR_CAST                    invalid_c
22007    E   ERRCODE_INVALID_DATETIME_FORMAT                            invalid_d
22019    E   ERRCODE_INVALID_ESCAPE_CHARACTER                           invalid_e
2200D    E   ERRCODE_INVALID_ESCAPE_OCTET                               invalid_e
22025    E   ERRCODE_INVALID_ESCAPE_SEQUENCE                            invalid_e
22P06    E   ERRCODE_NONSTANDARD_USE_OF_ESCAPE_CHARACTER                 nonstanda
22010    E   ERRCODE_INVALID_INDICATOR_PARAMETER_VALUE                   invalid_i
22023    E   ERRCODE_INVALID_PARAMETER_VALUE                             invalid_p
2201B    E   ERRCODE_INVALID_REGULAR_EXPRESSION                          invalid_r
2201W    E   ERRCODE_INVALID_ROW_COUNT_IN_LIMIT_CLAUSE                   invalid_r
2201X    E   ERRCODE_INVALID_ROW_COUNT_IN_RESULT_OFFSET_CLAUSE           invalid_r
2202H    E   ERRCODE_INVALID_TABLESAMPLE_ARGUMENT                       invalid_t
2202G    E   ERRCODE_INVALID_TABLESAMPLE_REPEAT                         invalid_t
22009    E   ERRCODE_INVALID_TIME_ZONE_DISPLACEMENT_VALUE               invalid_t
2200C    E   ERRCODE_INVALID_USE_OF_ESCAPE_CHARACTER                   invalid_u
2200G    E   ERRCODE_MOST_SPECIFIC_TYPE_MISMATCH                        most_spec
22004    E   ERRCODE_NULL_VALUE_NOT_ALLOWED                             null_valu
22002    E   ERRCODE_NULL_VALUE_NO_INDICATOR_PARAMETER                  null_valu
22003    E   ERRCODE_NUMERIC_VALUE_OUT_OF_RANGE                         numeric_v
```

```
22026    E   ERRCODE_STRING_DATA_LENGTH_MISMATCH                    string_da
22001    E   ERRCODE_STRING_DATA_RIGHT_TRUNCATION                   string_da
22011    E   ERRCODE_SUBSTRING_ERROR                                substring
22027    E   ERRCODE_TRIM_ERROR                                     trim_erro
22024    E   ERRCODE_UNTERMINATED_C_STRING                          untermina
2200F    E   ERRCODE_ZERO_LENGTH_CHARACTER_STRING                   zero_leng
22P01    E   ERRCODE_FLOATING_POINT_EXCEPTION                       floating_
22P02    E   ERRCODE_INVALID_TEXT_REPRESENTATION                    invalid_t
22P03    E   ERRCODE_INVALID_BINARY_REPRESENTATION                  invalid_b
22P04    E   ERRCODE_BAD_COPY_FILE_FORMAT                          bad_copy_
22P05    E   ERRCODE_UNTRANSLATABLE_CHARACTER                      untransla
2200L    E   ERRCODE_NOT_AN_XML_DOCUMENT                           not_an_xm
2200M    E   ERRCODE_INVALID_XML_DOCUMENT                          invalid_x
2200N    E   ERRCODE_INVALID_XML_CONTENT                           invalid_x
2200S    E   ERRCODE_INVALID_XML_COMMENT                           invalid_x
2200T    E   ERRCODE_INVALID_XML_PROCESSING_INSTRUCTION           invalid_x
22030    E   ERRCODE_RANGE_PARTITION_VALUE_ERROR                   inserted_
```

Class 23 - Integrity Constraint Violation

```
23000    E   ERRCODE_INTEGRITY_CONSTRAINT_VIOLATION               integrity
23001    E   ERRCODE_RESTRICT_VIOLATION                           restrict_
23502    E   ERRCODE_NOT_NULL_VIOLATION                           not_null_
23503    E   ERRCODE_FOREIGN_KEY_VIOLATION                        foreign_k
23505    E   ERRCODE_UNIQUE_VIOLATION                             unique_vi
23514    E   ERRCODE_CHECK_VIOLATION                              check_vio
23P01    E   ERRCODE_EXCLUSION_VIOLATION                         exclusion
```

Class 24 - Invalid Cursor State

```
24000    E   ERRCODE_INVALID_CURSOR_STATE                        invalid_c
```

Class 25 - Invalid Transaction State

```
25000    E   ERRCODE_INVALID_TRANSACTION_STATE                              invalid_t
25001    E   ERRCODE_ACTIVE_SQL_TRANSACTION                                 active_sq
25002    E   ERRCODE_BRANCH_TRANSACTION_ALREADY_ACTIVE                       branch_tr
25008    E   ERRCODE_HELD_CURSOR_REQUIRES_SAME_ISOLATION_LEVEL               held_curs
25003    E   ERRCODE_INAPPROPRIATE_ACCESS_MODE_FOR_BRANCH_TRANSACTION       inappropr
25004    E   ERRCODE_INAPPROPRIATE_ISOLATION_LEVEL_FOR_BRANCH_TRANSACTION  inappropr
25005    E   ERRCODE_NO_ACTIVE_SQL_TRANSACTION_FOR_BRANCH_TRANSACTION       no_active
25006    E   ERRCODE_READ_ONLY_SQL_TRANSACTION                              read_only
25007    E   ERRCODE_SCHEMA_AND_DATA_STATEMENT_MIXING_NOT_SUPPORTED         schema_an
25P01    E   ERRCODE_NO_ACTIVE_SQL_TRANSACTION                              no_active
25P02    E   ERRCODE_IN_FAILED_SQL_TRANSACTION                             in_failed
```

Class 26 - Invalid SQL Statement Name

```
# (prepared statements)
26000    E   ERRCODE_INVALID_SQL_STATEMENT_NAME                           invalid_s
```

Class 27 - Triggered Data Change Violation

```
27000    E   ERRCODE_TRIGGERED_DATA_CHANGE_VIOLATION                       triggered
```

Class 28 - Invalid Authorization Specification

```
28000    E   ERRCODE_INVALID_AUTHORIZATION_SPECIFICATION                   invalid_a
28P01    E   ERRCODE_INVALID_PASSWORD                                      invalid_p
```

Class 2B - Dependent Privilege Descriptors Still Exist

```
2B000    E   ERRCODE_DEPENDENT_PRIVILEGE_DESCRIPTORS_STILL_EXIST           dependent
2BP01    E   ERRCODE_DEPENDENT_OBJECTS_STILL_EXIST                        dependent
```

Class 2D - Invalid Transaction Termination

```
2D000    E   ERRCODE_INVALID_TRANSACTION_TERMINATION                    invalid_t
```

Class 2F - SQL Routine Exception

```
2F000    E   ERRCODE_SQL_ROUTINE_EXCEPTION                              sql_routi
2F005    E   ERRCODE_S_R_E_FUNCTION_EXECUTED_NO_RETURN_STATEMENT        function_
2F002    E   ERRCODE_S_R_E_MODIFYING_SQL_DATA_NOT_PERMITTED             modifying
2F003    E   ERRCODE_S_R_E_PROHIBITED_SQL_STATEMENT_ATTEMPTED           prohibite
2F004    E   ERRCODE_S_R_E_READING_SQL_DATA_NOT_PERMITTED              reading_s
```

Class 34 - Invalid Cursor Name

```
34000    E   ERRCODE_INVALID_CURSOR_NAME                               invalid_c
```

Class 38 - External Routine Exception

```
38000    E    ERRCODE_EXTERNAL_ROUTINE_EXCEPTION                        external
38001    E    ERRCODE_E_R_E_CONTAINING_SQL_NOT_PERMITTED                containi
38002    E    ERRCODE_E_R_E_MODIFYING_SQL_DATA_NOT_PERMITTED            modifyi
38003    E    ERRCODE_E_R_E_PROHIBITED_SQL_STATEMENT_ATTEMPTED          prohibit
38004    E    ERRCODE_E_R_E_READING_SQL_DATA_NOT_PERMITTED              reading_
```

Class 39 - External Routine Invocation Exception

```
39000   E    ERRCODE_EXTERNAL_ROUTINE_INVOCATION_EXCEPTION             external_
39001    E   ERRCODE_E_R_I_E_INVALID_SQLSTATE_RETURNED                 invalid_s
```

```
39004    E   ERRCODE_E_R_I_E_NULL_VALUE_NOT_ALLOWED                    null_valu
39P01    E   ERRCODE_E_R_I_E_TRIGGER_PROTOCOL_VIOLATED                 trigger_p
39P02    E   ERRCODE_E_R_I_E_SRF_PROTOCOL_VIOLATED                    srf_proto
39P03    E   ERRCODE_E_R_I_E_EVENT_TRIGGER_PROTOCOL_VIOLATED          event_tri
```

Class 3B - Savepoint Exception

```
3B000    E   ERRCODE_SAVEPOINT_EXCEPTION                             savepoint
3B001    E   ERRCODE_S_E_INVALID_SPECIFICATION                       invalid_s
```

Class 3D - Invalid Catalog Name

```
3D000    E   ERRCODE_INVALID_CATALOG_NAME                           invalid_c
```

Class 3F - Invalid Schema Name

```
3F000    E   ERRCODE_INVALID_SCHEMA_NAME                            invalid_s
```

Class 40 - Transaction Rollback

```
40000    E   ERRCODE_TRANSACTION_ROLLBACK                           transacti
40002    E   ERRCODE_T_R_INTEGRITY_CONSTRAINT_VIOLATION             transacti
40001    E   ERRCODE_T_R_SERIALIZATION_FAILURE                      serializa
40003    E   ERRCODE_T_R_STATEMENT_COMPLETION_UNKNOWN               statement
40P01    E   ERRCODE_T_R_DEADLOCK_DETECTED                          deadlock_
```

Class 42 - Syntax Error or Access Rule Violation

```
42000    E   ERRCODE_SYNTAX_ERROR_OR_ACCESS_RULE_VIOLATION                syntax_er
42601    E   ERRCODE_SYNTAX_ERROR                                         syntax_er
42501    E   ERRCODE_INSUFFICIENT_PRIVILEGE                              insuffici
42846    E   ERRCODE_CANNOT_COERCE                                        cannot_co
42803    E   ERRCODE_GROUPING_ERROR                                       grouping_
42P20    E   ERRCODE_WINDOWING_ERROR                                      windowing
42P19    E   ERRCODE_INVALID_RECURSION                                    invalid_r
42830    E   ERRCODE_INVALID_FOREIGN_KEY                                  invalid_f
42602    E   ERRCODE_INVALID_NAME                                         invalid_n
42622    E   ERRCODE_NAME_TOO_LONG                                        name_too_
42939    E   ERRCODE_RESERVED_NAME                                        reserved_
42804    E   ERRCODE_DATATYPE_MISMATCH                                    datatype_
42P18    E   ERRCODE_INDETERMINATE_DATATYPE                               indetermi
42P21    E   ERRCODE_COLLATION_MISMATCH                                   collation
42P22    E   ERRCODE_INDETERMINATE_COLLATION                              indetermi
42809    E   ERRCODE_WRONG_OBJECT_TYPE                                    wrong_obj

# 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
34000    E   ERRCODE_UNDEFINED_CURSOR
3D000    E   ERRCODE_UNDEFINED_DATABASE
42883    E   ERRCODE_UNDEFINED_FUNCTION                                  undefined
26000    E   ERRCODE_UNDEFINED_PSTATEMENT
3F000    E   ERRCODE_UNDEFINED_SCHEMA
42P01    E   ERRCODE_UNDEFINED_TABLE                                     undefined
42P02    E   ERRCODE_UNDEFINED_PARAMETER                                 undefined
42P23    E   ERRCODE_RELATION_NOT_FOUND                                  relation_
42704    E   ERRCODE_UNDEFINED_OBJECT                                    undefined
42701    E   ERRCODE_DUPLICATE_COLUMN                                    duplicate
42P03    E   ERRCODE_DUPLICATE_CURSOR                                    duplicate
42P04    E   ERRCODE_DUPLICATE_DATABASE                                  duplicate
42723    E   ERRCODE_DUPLICATE_FUNCTION                                  duplicate
42P05    E   ERRCODE_DUPLICATE_PSTATEMENT                               duplicate
42P06    E   ERRCODE_DUPLICATE_SCHEMA                                    duplicate
```

```
42P07    E   ERRCODE_DUPLICATE_TABLE                                  duplicate
42712    E   ERRCODE_DUPLICATE_ALIAS                                  duplicate
42710    E   ERRCODE_DUPLICATE_OBJECT                                 duplicate
42711    E   ERRCODE_DUPLICATE_TARGET_ROW                            duplicate
42702    E   ERRCODE_AMBIGUOUS_COLUMN                                 ambiguous
42725    E   ERRCODE_AMBIGUOUS_FUNCTION                               ambiguous
42P08    E   ERRCODE_AMBIGUOUS_PARAMETER                              ambiguous
42P09    E   ERRCODE_AMBIGUOUS_ALIAS                                  ambiguous
42P10    E   ERRCODE_INVALID_COLUMN_REFERENCE                         invalid_c
42611    E   ERRCODE_INVALID_COLUMN_DEFINITION                        invalid_c
42P11    E   ERRCODE_INVALID_CURSOR_DEFINITION                       invalid_c
42P12    E   ERRCODE_INVALID_DATABASE_DEFINITION                     invalid_c
42P13    E   ERRCODE_INVALID_FUNCTION_DEFINITION                     invalid_f
42P14    E   ERRCODE_INVALID_PSTATEMENT_DEFINITION                   invalid_p
42P15    E   ERRCODE_INVALID_SCHEMA_DEFINITION                       invalid_s
42P16    E   ERRCODE_INVALID_TABLE_DEFINITION                        invalid_t
42P17    E   ERRCODE_INVALID_OBJECT_DEFINITION                       invalid_o
42P42    E   ERRCODE_BACKEND_ONLY                                    backend_c
```

Class 44 - WITH CHECK OPTION Violation

```
44000    E   ERRCODE_WITH_CHECK_OPTION_VIOLATION                     with_chec
```

Class 53 - Insufficient Resources

```
# (PostgreSQL-specific error class)
53000    E   ERRCODE_INSUFFICIENT_RESOURCES                          insuffici
53100    E   ERRCODE_DISK_FULL                                       disk_full
53200    E   ERRCODE_OUT_OF_MEMORY                                   out_of_me
53300    E   ERRCODE_TOO_MANY_CONNECTIONS                            too_many_
53400    E   ERRCODE_CONFIGURATION_LIMIT_EXCEEDED                    configura
```

Class 54 - Program Limit Exceeded

```
# this is for wired-in limits, not resource exhaustion problems
54000    E   ERRCODE_PROGRAM_LIMIT_EXCEEDED                              program_
54001    E   ERRCODE_STATEMENT_TOO_COMPLEX                              statement
54011    E   ERRCODE_TOO_MANY_COLUMNS                                   too_many_
54023    E   ERRCODE_TOO_MANY_ARGUMENTS                                 too_many_
```

Class 55 - Object Not In Prerequisite State

```
55000    E   ERRCODE_OBJECT_NOT_IN_PREREQUISITE_STATE                   object_no
55006    E   ERRCODE_OBJECT_IN_USE                                      object_in
55P02    E   ERRCODE_CANT_CHANGE_RUNTIME_PARAM                         cant_chan
55P03    E   ERRCODE_LOCK_NOT_AVAILABLE                                lock_not_
```

Class 57 - Operator Intervention

```
57000    E   ERRCODE_OPERATOR_INTERVENTION                             operator_
57014    E   ERRCODE_QUERY_CANCELED                                    query_can
57P01    E   ERRCODE_ADMIN_SHUTDOWN                                    admin_shu
57P02    E   ERRCODE_CRASH_SHUTDOWN                                    crash_shu
57P03    E   ERRCODE_CANNOT_CONNECT_NOW                               cannot_co
57P04    E   ERRCODE_DATABASE_DROPPED                                 database_
```

Class 58 - System Error

```
58000    E   ERRCODE_SYSTEM_ERROR                                      system_er
58030    E   ERRCODE_IO_ERROR                                          io_error
58P01    E   ERRCODE_UNDEFINED_FILE                                    undefined
58P02    E   ERRCODE_DUPLICATE_FILE                                   duplicate
```

Class F0 - Configuration File Error

```
F0000    E   ERRCODE_CONFIG_FILE_ERROR                                  config_fi
F0001    E   ERRCODE_LOCK_FILE_EXISTS                                   lock_file
```

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_colum
HV002    E   ERRCODE_FDW_DYNAMIC_PARAMETER_VALUE_NEEDED                 fdw_dynam
HV010    E   ERRCODE_FDW_FUNCTION_SEQUENCE_ERROR                        fdw_funct
HV021    E   ERRCODE_FDW_INCONSISTENT_DESCRIPTOR_INFORMATION            fdw_incon
HV024    E   ERRCODE_FDW_INVALID_ATTRIBUTE_VALUE                        fdw_inval
HV007    E   ERRCODE_FDW_INVALID_COLUMN_NAME                           fdw_inval
HV008    E   ERRCODE_FDW_INVALID_COLUMN_NUMBER                         fdw_inval
HV004    E   ERRCODE_FDW_INVALID_DATA_TYPE                             fdw_inval
HV006    E   ERRCODE_FDW_INVALID_DATA_TYPE_DESCRIPTORS                 fdw_inval
HV091    E   ERRCODE_FDW_INVALID_DESCRIPTOR_FIELD_IDENTIFIER           fdw_inval
HV00B    E   ERRCODE_FDW_INVALID_HANDLE                                fdw_inval
HV00C    E   ERRCODE_FDW_INVALID_OPTION_INDEX                          fdw_inval
HV00D    E   ERRCODE_FDW_INVALID_OPTION_NAME                           fdw_inval
HV090    E   ERRCODE_FDW_INVALID_STRING_LENGTH_OR_BUFFER_LENGTH        fdw_inval
HV00A    E   ERRCODE_FDW_INVALID_STRING_FORMAT                         fdw_inval
HV009    E   ERRCODE_FDW_INVALID_USE_OF_NULL_POINTER                   fdw_inval
HV014    E   ERRCODE_FDW_TOO_MANY_HANDLES                              fdw_too_m
HV001    E   ERRCODE_FDW_OUT_OF_MEMORY                                 fdw_out_o
HV00P    E   ERRCODE_FDW_NO_SCHEMAS                                     fdw_no_sc
HV00J    E   ERRCODE_FDW_OPTION_NAME_NOT_FOUND                         fdw_optio
HV00K    E   ERRCODE_FDW_REPLY_HANDLE                                  fdw_reply
HV00Q    E   ERRCODE_FDW_SCHEMA_NOT_FOUND                              fdw_schem
HV00R    E   ERRCODE_FDW_TABLE_NOT_FOUND                               fdw_table
HV00L    E   ERRCODE_FDW_UNABLE_TO_CREATE_EXECUTION                    fdw_unabl
HV00M    E   ERRCODE_FDW_UNABLE_TO_CREATE_REPLY                        fdw_unabl
HV00N    E   ERRCODE_FDW_UNABLE_TO_ESTABLISH_CONNECTION                fdw_unabl
```

Class P0: PL/pgSQL Error

```
P0000    E   ERRCODE_PLPGSQL_ERROR                                      plpgsql_e
P0001    E   ERRCODE_RAISE_EXCEPTION                                    raise_exc
```

```
P0002    E    ERRCODE_NO_DATA_FOUND                                    no_data_
P0003    E    ERRCODE_TOO_MANY_ROWS                                    too_many_
P0004    E    ERRCODE_ASSERT_FAILURE                                   assert_fa
```

Class XX: Internal Error

```
# (PostgreSQL-specific error class)
XX000    E    ERRCODE_INTERNAL_ERROR                                   internal_
XX001    E    ERRCODE_DATA_CORRUPTED                                   data_cor
XX002    E    ERRCODE_INDEX_CORRUPTED                                  index_co
```

Parent topic: Yellowbrick Documentation

Glossary

appliance
: The Yellowbrick Data Warehouse. This appliance consists of a self-sufficient hardware platform with Yellowbrick software pre-installed and pre-configured. The appliance is custom-built to support various analytic applications. Everything you need to create and use a Yellowbrick database is included in the appliance.

bulk loader (ybload)
: The client program used for running bulk loads on Yellowbrick tables.

capacity expansion
: The process of adding blades to an appliance.

chassis
: The appliance hardware that contains the shelf of compute blades, InfiniBand switches, and power supplies. The appliance comes in single-, dual-, and triple-chassis configurations.

cluster
: See HA cluster, worker cluster.

Cluster Manager
: Software that manages the compute blades.

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 worker node file systems.

compute blade
: A blade with dedicated storage, CPUs, and memory, deployed as a worker node. Parallel worker processes on the blades execute the query workload for the appliance.

compute node
: See worker node.

control node

See manager node.

cross-database queries

Queries that reference tables and views in different databases.

cumulative backup

A backup that records all the changes to a database since the last full or cumulative backup.

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.

external table

A table that is stored outside the database in an NFS-mounted file system.

fabric blade

Hardware assembly that provides high-speed interconnect access among the compute blades and the manager node.

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.

full backup

A backup of the entire contents of a database.

HA cluster

Cluster of two manager nodes that provide high availability (HA) for the appliance.

horizon

A starting point for a new backup may be thought of as a horizon. Full and cumulative backups reset the horizon, but incremental backups do not; therefore, a new full or cumulative backup picks up all of the changes that were already captured by an unbroken series of incremental backups.

incremental backup

A backup that records all the changes to a database since the last backup of any kind (full, cumulative, or incremental).

InfiniBand switches

Network fabric or interconnect for the cluster.

leader node

See manager node.

Lime compiler

Compiler on the control node that generates C++ code for incoming SQL commands and queries. This code is shipped to the worker nodes for processing.

manager node

A server that communicates with the outside world and assigns work to the processes on the worker nodes. The appliance comes with two manager nodes. The second node is a failover node for the first, should the first node fail or require maintenance. Only one manager node is in use at any given time.

object storage

Remote source files supported by `ybload`, such as Azure and S3.

parity

A Yellowbrick database is resilient to the loss of individual compute blades. All of the data is stored in a way that allows it to be reconstructed from the failed blade(s), using file-level RAID6. Parity-protected files provide the required degree of redundancy.

PG

PostgreSQL front-end database, running on the manager node. Connects via internal JDBC.

primary manager node

The manager node that is currently active, serving user requests.

replicated table

A table that is copied rather than distributed across all of the analytic blades so that all worker processes have immediate local access to all of its data.

replication

The process of maintaining a replica of a physical Yellowbrick database on a secondary Yellowbrick appliance.

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.

shard
A unit of data storage on a worker node.

shard store
See column store.

shelf
The analytic engine for the appliance is a shelf of compute blades that are provisioned as worker nodes.

snapshot
A logical reference to the state of a database at a given point in time. Backup snapshots result from both backups taken with the `ybbackup` tool and database replication cycles.

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 Management Console (SMC)
UI for monitoring activity on the appliance, analyzing query plans and statistics to troubleshoot performance issues, running system management operations, running queries, and so on.

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 about the appliance.

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 table
: A table in the system catalog that is dynamically generated based on current activity in the appliance. These tables capture statistics for queries, CPU and memory utilization, and file system information.

worker cluster
: Interconnected shelf of worker nodes (compute blades) that process Yellowbrick queries and loads in parallel. A standard configuration has a cluster of up to 15 blades.

worker node
: A compute blade that is actively in use for Yellowbrick queries and load operations.

worker processes
: Processes launched on the worker nodes (blades). In general, Yellowbrick queries use massively parallel processing, with all of the nodes sharing the work. One process on a given worker node is elected to run the final phase of a given query.

ybbackup, ybrestore, ybbackupctl
: Yellowbrick backup and restore client tools.

ybcli
: Yellowbrick CLI for system management.

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.

ybunload

Yellowbrick client tool for unloading data.

Yellowbrick appliance

See appliance

Yellowbrick database

A physical database created with the Yellowbrick appliance.

Parent topic: Yellowbrick Documentation

Notices and Licenses

This section contains notices and licenses for third-party components that are used by Yellowbrick Data products.

In This Section

- Third Party Licenses
- OpenSSL License
- Third Party Library Notices

Parent topic: Yellowbrick Documentation

Third Party Licenses

```
The following third-party components are included in binary form in Yellowbrick Data
and redistributed under the following licenses permitted by the original author/orga

    --------------------------------------------------------------------------------
    Apache Avro Mapred API (1.7.5)
    Apache CXF JAX-RS Client (3.2.4)
    Apache CXF JAX-RS Extensions: Providers (3.2.4)
    Apache Hadoop Common (2.7.3)
    Apache Log4j (1.2.17)
    Apache Parquet Hadoop (1.9.0)
    AWS Java SDK for Amazon S3 (1.11.11)
    com.fasterxml.jackson.core:jackson-databind (2.9.7)
    Commons Compress (1.5)
    Commons Lang (2.6)
    Curator Recipes (4.0.1)
    FindBugs-jsr305 (3.0.1)
    IntelliJ IDEA Annotations (15.0)
    Jackson-annotations (2.9.7)
    Jackson-core (2.9.7)
    JCommander (1.48)
    Jettison (1.3.7)
    Kite Data MapReduce Module (1.0.0)
    LZ4 and xxHash (1.4.1)
    Netty/Buffer (4.1.24.Final)
    Netty/Codec (4.1.24.Final)
    Netty/Common (4.1.24.Final)
    Netty/Handler (4.1.24.Final)
    Netty/Transport (4.1.24.Final)
    org.apache.hadoop:hadoop-mapreduce-client-common (2.7.3)
    org.apache.sqoop:sqoop (1.4.6)
    org.apache.zookeeper:zookeeper (3.4.6)
    Spark Project SQL (2.0.1)

    The Apache Software License, Version 2.0
    URL: http://www.apache.org/licenses/LICENSE-2.0.html

    --------------------------------------------------------------------------------
    com.jcabi:jcabi-log (0.17.1)

        Copyright (c) 2012-2017, jcabi.com
        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. 3) Neither the name of the jcabi.com 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.


BSD License
URL: http://www.jcabi.com/LICENSE.txt

--------------------------------------------------------------------------------
javax.ws.rs:javax.ws.rs-api (2.1)

Common Development and Distribution License 1.1,GPL v2 with CLASSPATH Exception
URL: https://javaee.github.io/glassfish/LICENSE

--------------------------------------------------------------------------------
jsoup Java HTML Parser (1.11.3)

    The MIT License

    Copyright (c) 2009-2018 Jonathan Hedley <jonathan@hedley.net>

    Permission is hereby granted, free of charge, to any person obtaining a copy
    of this software and associated documentation files (the "Software"), to dea
```

```
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 a
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 FRO
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN T
SOFTWARE.


The jsoup code-base (including source and compiled packages) are distribute
under the open source MIT license as described below.


Copyright © 2009 - 2017 Jonathan Hedley <https://jhy.io/> (jonathan@hedley.
<mailto:jonathan@hedley.net>)


Permission is hereby granted, free of charge, to any person obtaining a cop
of this software and associated documentation files (the "Software"), to dea
in the Software without restriction, including without limitation the rights
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copi
of the Software, and to permit persons to whom the Software is furnished to
so, subject to the following conditions:


The above copyright notice and this permission notice shall be included in a
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 FRO
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN T
SOFTWARE.
```

```
MIT License
URL: https://jsoup.org/license

--------------------------------------------------------------------------------
org.postgresql:postgresql (9.4.1208)

    Copyright (c) 1997, PostgreSQL Global Development Group
    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.


(none)
URL: https://raw.githubusercontent.com/pgjdbc/pgjdbc/master/LICENSE

--------------------------------------------------------------------------------
PostgreSQL JDBC Driver (9.1-901-1.jdbc4)

    Copyright (c) 1997, PostgreSQL Global Development Group
    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.


BSD License
URL: https://raw.githubusercontent.com/pgjdbc/pgjdbc/master/LICENSE

--------------------------------------------------------------------------------
Protocol Buffers [Core] (3.5.1)

    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.

    Code generated by the Protocol Buffer compiler is owned by the owner
    of the input file used when generating it.  This code is not
    standalone and requires a support library to be linked with it.  This
    support library is itself covered by the above license.


BSD License
URL: https://raw.githubusercontent.com/protocolbuffers/protobuf/master/LICENSE

--------------------------------------------------------------------------------
SLF4J API Module (1.7.25)
SLF4J LOG4J-12 Binding (1.7.25)

    Copyright (c) 2004-2017 QOS.ch
    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 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.
```

```
MIT License
URL: https://raw.githubusercontent.com/qos-ch/slf4j/master/LICENSE.txt

--------------------------------------------------------------------------------
angular-animate:1.5.5
MIT

    The MIT License

    Copyright (c) 2010-2018 Google, Inc. http://angularjs.org

    Permission is hereby granted, free of charge, to any person obtaining a copy
    of this software and associated documentation files (the "Software"), to dea
    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 FRO
    OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
    THE SOFTWARE.



URL: https://raw.githubusercontent.com/angular/angular.js/master/LICENSE

--------------------------------------------------------------------------------
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 dea
```

```
    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 a
    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 TH
    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 dea
    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 dea
    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 dea
    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 FRO
    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
    this software and associated documentation files (the "Software"), to deal i
    the Software without restriction, including without limitation the rights to
    use, copy, modify, merge, publish, distribute, sublicense, and/or sell copie
    the Software, and to permit persons to whom the Software is furnished to do
    subject to the following conditions:

    The above copyright notice and this permission notice shall be included in a
    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, FIT
    FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS
    COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETH
    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 dea
    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 a
    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 TH
    SOFTWARE.



URL: LICENSE

--------------------------------------------------------------------------------
angular-messages:1.5.5
MIT

    The MIT License

    Copyright (c) 2010-2018 Google, Inc. http://angularjs.org

    Permission is hereby granted, free of charge, to any person obtaining a copy
    of this software and associated documentation files (the "Software"), to dea
    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 FRO
    OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
    THE SOFTWARE.



URL: https://raw.githubusercontent.com/angular/angular.js/master/LICENSE

--------------------------------------------------------------------------------
angular-mocks:1.5.0
MIT

    The MIT License

    Copyright (c) 2010-2018 Google, Inc. http://angularjs.org

    Permission is hereby granted, free of charge, to any person obtaining a copy
    of this software and associated documentation files (the "Software"), to dea
    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 FRO
    OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
    THE SOFTWARE.
```

```
URL: https://raw.githubusercontent.com/angular/angular.js/master/LICENSE

--------------------------------------------------------------------------------
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 dea
    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 a
    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 FRO
    OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN TH
    SOFTWARE.



URL: LICENSE

--------------------------------------------------------------------------------
angular-promise-extras:0.1.3
ISC

    Copyright (c) 2016, James Pike <github@chilon.net>

    Permission to use, copy, modify, and/or distribute this software for any pur

    THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES WI


URL: https://raw.githubusercontent.com/ohjames/angular-promise-extras/master/LIC
```

```
--------------------------------------------------------------------------------
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 dea
    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

    The MIT License

    Copyright (c) 2010-2018 Google, Inc. http://angularjs.org

    Permission is hereby granted, free of charge, to any person obtaining a copy
    of this software and associated documentation files (the "Software"), to dea
    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 FRO
    OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
    THE SOFTWARE.



URL: https://raw.githubusercontent.com/angular/angular.js/master/LICENSE

--------------------------------------------------------------------------------
angular-sanitize:1.5.5
MIT

    The MIT License

    Copyright (c) 2010-2018 Google, Inc. http://angularjs.org

    Permission is hereby granted, free of charge, to any person obtaining a copy
    of this software and associated documentation files (the "Software"), to dea
    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 FRO
    OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
    THE SOFTWARE.



URL: https://raw.githubusercontent.com/angular/angular.js/master/LICENSE

--------------------------------------------------------------------------------
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 dea
    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 FRO
    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 dea
    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 a
    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 FRO
    OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN T
    SOFTWARE.


URL: LICENSE

--------------------------------------------------------------------------------
angular-tree-repeat:0.0.1
MIT License

    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 dea
    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 FRO
    OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
    THE SOFTWARE.



URL: LICENCE http://opensource.org/licenses/mit-license.php

--------------------------------------------------------------------------------
angular-ui:1.3.2
MIT
```

```
        The MIT License

        Copyright (c) 2012-2017 the AngularUI Team, https://github.com/organizations

        Permission is hereby granted, free of charge, to any person obtaining a copy
        of this software and associated documentation files (the "Software"), to dea
        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 FRO
        OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
        THE SOFTWARE.



    URL: https://raw.githubusercontent.com/angular-ui/bootstrap/master/LICENSE

    ------------------------------------------------------------------------------
    angular-vs-repeat:1.1.7
    MIT License

        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 dea
        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 a
        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 FRO
    OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN T
    SOFTWARE.

URL: LICENSE http://opensource.org/licenses/mit-license.php

--------------------------------------------------------------------------------
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 dea
    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 FRO
    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 dea
    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 FRO
    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

    The MIT License (MIT)

    Copyright (c) 2011-2018 Twitter, Inc.

    Permission is hereby granted, free of charge, to any person obtaining a cop
    of this software and associated documentation files (the "Software"), to dea
    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 FRO
    OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
    THE SOFTWARE.


URL: https://raw.githubusercontent.com/twbs/bootstrap/master/LICENSE
```

```
--------------------------------------------------------------------------------
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 dea
    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
    this software and associated documentation files (the "Software"), to deal i
    the Software without restriction, including without limitation the rights to
    use, copy, modify, merge, publish, distribute, sublicense, and/or sell copie
    the Software, and to permit persons to whom the Software is furnished to do
    subject to the following conditions:

    The above copyright notice and this permission notice shall be included in a
    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, FIT
    FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS
    COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETH
    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 dea
    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 a
    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 TH
    SOFTWARE.


URL: LICENSE

--------------------------------------------------------------------------------
codeflask:^0.3.0
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 dea
    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 a
    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 FRO
    OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN TH
    SOFTWARE.


URL: https://raw.githubusercontent.com/kazzkiq/CodeFlask/master/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

    MIT License

    Copyright (c) 2019 marmelab

    Permission is hereby granted, free of charge, to any person obtaining a copy
    of this software and associated documentation files (the "Software"), to dea
    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 a
    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 FRO
```

```
    OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN TH
    SOFTWARE.


URL: https://raw.githubusercontent.com/marmelab/configurable.js/master/LICENSE.m

--------------------------------------------------------------------------------
css:0.1.34
MIT License

    The MIT License (MIT)

    Copyright (c) 2014-2015 Guy Bedford

    Permission is hereby granted, free of charge, to any person obtaining a copy
    this software and associated documentation files (the "Software"), to deal i
    the Software without restriction, including without limitation the rights to
    use, copy, modify, merge, publish, distribute, sublicense, and/or sell copie
    the Software, and to permit persons to whom the Software is furnished to do
    subject to the following conditions:

    The above copyright notice and this permission notice shall be included in a
    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, FIT
    FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS
    COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETH
    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 http://opensource.org/licenses/mit-license.php

--------------------------------------------------------------------------------
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
```

```
    Copyright 2010-2017 Mike Bostock
    All rights reserved.

    Redistribution and use in source and binary forms, with or without modificat
    are permitted provided that the following conditions are met:

    * Redistributions of source code must retain the above copyright notice, thi
      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 f
      endorse or promote products derived from this software without specific pr
      written permission.

    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
    ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIE
    WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
    DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE
    ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAG
    (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
    LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
    ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
    (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THI
    SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.


URL: https://raw.githubusercontent.com/d3/d3/master/LICENSE

--------------------------------------------------------------------------------
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 modificat
    are permitted provided that the following conditions are met:

    * Redistributions of source code must retain the above copyright notice, thi
      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 t
      endorse or promote products derived from this software without specific pr
      written permission.

    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
    ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIE
    WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
    DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE
    ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAG
    (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
    LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
    ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
    (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THI
    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

    The above copyright notice and this permission notice shall be included in a

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR I

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 dea
    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 dea
    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 FRO
    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
```

```
    Copyright (c) 2014 marmelab

    Permission is hereby granted, free of charge, to any person obtaining a copy
    of this software and associated documentation files (the "Software"), to dea
    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: https://raw.githubusercontent.com/marmelab/EventDrops/master/LICENSE

--------------------------------------------------------------------------------
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

    The above copyright notice and this permission notice shall be included in a

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR I

      [1]: http://eligrey.com


URL: LICENSE.md

--------------------------------------------------------------------------------
font-awesome:4.5.0
(OFL-1.1 AND MIT)
```

```
    Font Awesome Free License
    -------------------------

    Font Awesome Free is free, open source, and GPL friendly. You can use it fo
    commercial projects, open source projects, or really almost whatever you wa
    Full Font Awesome Free license: https://fontawesome.com/license/free.

    # Icons: CC BY 4.0 License (https://creativecommons.org/licenses/by/4.0/)
    In the Font Awesome Free download, the CC BY 4.0 license applies to all ico
    packaged as SVG and JS file types.

    # Fonts: SIL OFL 1.1 License (https://scripts.sil.org/OFL)
    In the Font Awesome Free download, the SIL OFL license applies to all icons
    packaged as web and desktop font files.

    # Code: MIT License (https://opensource.org/licenses/MIT)
    In the Font Awesome Free download, the MIT license applies to all non-font a
    non-icon files.

    # Attribution
    Attribution is required by MIT, SIL OFL, and CC BY licenses. Downloaded Font
    Awesome Free files already contain embedded comments with sufficient
    attribution, so you shouldn't need to do anything additional when using thes
    files normally.

    We've kept attribution comments terse, so we ask that you do not actively wo
    to remove them from files, especially code. They're a great way for folks to
    learn about Font Awesome.

    # Brand Icons
    All brand icons are trademarks of their respective owners. The use of these
    trademarks does not indicate endorsement of the trademark holder by Font
    Awesome, nor vice versa. **Please do not use brand logos for any purpose exc
    to represent the company, product, or service to which they refer.**


URL: https://raw.githubusercontent.com/FortAwesome/Font-Awesome/master/LICENSE.t

--------------------------------------------------------------------------------
formattor:0.0.2
MIT
```

```
URL:

--------------------------------------------------------------------------------
humanize:0.0.9
MIT License

    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 http://opensource.org/licenses/mit-license.php

--------------------------------------------------------------------------------
jolokia:1.3.2
Apache License

    Apache Software License, Version 2.0

URL: http://www.apache.org/licenses/LICENSE-2.0.html

--------------------------------------------------------------------------------
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
MIT License

    (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 http://opensource.org/licenses/mit-license.php

--------------------------------------------------------------------------------
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 dea
    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 a
    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 TH
    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

    Copyright (c) JS Foundation 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: https://raw.githubusercontent.com/moment/moment/master/LICENSE

--------------------------------------------------------------------------------
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
    this software and associated documentation files (the "Software"), to deal i
    the Software without restriction, including without limitation the rights to
    use, copy, modify, merge, publish, distribute, sublicense, and/or sell copie
    the Software, and to permit persons to whom the Software is furnished to do
    subject to the following conditions:
```

```
    The above copyright notice and this permission notice shall be included in a
    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, FIT
    FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS
    COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETH
    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 modificat

    Redistributions of source code must retain the above copyright notice, this
    Redistributions in binary form must reproduce the above copyright notice, th
    Neither the name of the esvit nor the names of its contributors may be used
    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"

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 dea
    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 a
    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 TH
    SOFTWARE.


URL: LICENSE.txt

------------------------------------------------------------------------------
pako:1.0.0
MIT

    (The MIT License)

    Copyright (C) 2014-2017 by Vitaly Puzrin and Andrei Tuputcyn

    Permission is hereby granted, free of charge, to any person obtaining a copy
    of this software and associated documentation files (the "Software"), to dea
    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: https://raw.githubusercontent.com/nodeca/pako/master/LICENSE

--------------------------------------------------------------------------------
prismjs:1.9.0
MIT

    MIT LICENSE

    Copyright (c) 2012 Lea Verou

    Permission is hereby granted, free of charge, to any person obtaining a copy
    of this software and associated documentation files (the "Software"), to dea
    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: https://raw.githubusercontent.com/PrismJS/prism/master/LICENSE

--------------------------------------------------------------------------------
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
```

```
    The above copyright notice and this permission notice shall be included in a

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR I


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
    this software and associated documentation files (the "Software"), to deal i
    the Software without restriction, including without limitation the rights to
    use, copy, modify, merge, publish, distribute, sublicense, and/or sell copie
    of the Software, and to permit persons to whom the Software is furnished to
    so, subject to the following conditions:

    The above copyright notice and this permission notice shall be included in a
    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 TH
    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 dea
    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 a
    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 TH
    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 dea
    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 a
    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 FRO
    OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN T
    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
    this software and associated documentation files (the "Software"), to deal i
    the Software without restriction, including without limitation the rights to
    use, copy, modify, merge, publish, distribute, sublicense, and/or sell copie
    the Software, and to permit persons to whom the Software is furnished to do
    subject to the following conditions:

    The above copyright notice and this permission notice shall be included in a
    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, FIT
    FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS
    COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETH
    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

    MIT License

    Copyright (c) 2017 Toastr Maintainers

    Permission is hereby granted, free of charge, to any person obtaining a copy
```

```
    of this software and associated documentation files (the "Software"), to dea
    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 a
    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 TH
    SOFTWARE.


URL: https://raw.githubusercontent.com/CodeSeven/toastr/master/LICENSE

----------------------------------------------------------------------------
typicons:2.0.7
Artwork: CCA: https://creativecommons.org/licenses/by-sa/3.0/legalcode,FONT: SIL



URL:

----------------------------------------------------------------------------
urijs:1.17.0
MIT

    The MIT License (MIT)

    Copyright (c) 2011 Rodney Rehm

    Permission is hereby granted, free of charge, to any person obtaining a copy
    of this software and associated documentation files (the "Software"), to dea
    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 FRO
    OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
    THE SOFTWARE.

URL: https://raw.githubusercontent.com/medialize/URI.js/master/LICENSE.txt

--------------------------------------------------------------------------------
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
    this software and associated documentation files (the "Software"), to deal i
    the Software without restriction, including without limitation the rights to
    use, copy, modify, merge, publish, distribute, sublicense, and/or sell copie
    the Software, and to permit persons to whom the Software is furnished to do
    subject to the following conditions:

    The above copyright notice and this permission notice shall be included in a
    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, FIT
    FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS
    COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETH
    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
MIT License

    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 dea
    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 http://opensource.org/licenses/mit-license.php
```

Parent topic: Notices and Licenses

OpenSSL License

The Yellowbrick implementation of the `ENCRYPT` and `DECRYPT` functions contains code under the OpenSSL license. Here is a copy of the current LICENSE file for the OpenSSL toolkit.

```
LICENSE ISSUES
  ==============

  The OpenSSL toolkit stays under a double license, i.e. both the conditions of
  the OpenSSL License and the original SSLeay license apply to the toolkit.
  See below for the actual license texts.

  OpenSSL License
  ---------------

/* ====================================================================
 * Copyright (c) 1998-2018 The OpenSSL Project.  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.
 *
 * 3. All advertising materials mentioning features or use of this
 *    software must display the following acknowledgment:
 *    "This product includes software developed by the OpenSSL Project
 *    for use in the OpenSSL Toolkit. (http://www.openssl.org/)"
 *
 * 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to
 *    endorse or promote products derived from this software without
 *    prior written permission. For written permission, please contact
 *    openssl-core@openssl.org.
 *
 * 5. Products derived from this software may not be called "OpenSSL"
 *    nor may "OpenSSL" appear in their names without prior written
 *    permission of the OpenSSL Project.
```

```
 *
 * 6. Redistributions of any form whatsoever must retain the following
 *    acknowledgment:
 *    "This product includes software developed by the OpenSSL Project
 *    for use in the OpenSSL Toolkit (http://www.openssl.org/)"
 *
 * THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY
 * EXPRESSED 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 OpenSSL PROJECT OR
 * ITS 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.
 * ====================================================================
 *
 * This product includes cryptographic software written by Eric Young
 * (eay@cryptsoft.com).  This product includes software written by Tim
 * Hudson (tjh@cryptsoft.com).
 *
 */

 Original SSLeay License
 -----------------------

/* Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com)
 * All rights reserved.
 *
 * This package is an SSL implementation written
 * by Eric Young (eay@cryptsoft.com).
 * The implementation was written so as to conform with Netscapes SSL.
 *
 * This library is free for commercial and non-commercial use as long as
 * the following conditions are aheared to.  The following conditions
 * apply to all code found in this distribution, be it the RC4, RSA,
 * lhash, DES, etc., code; not just the SSL code.  The SSL documentation
 * included with this distribution is covered by the same copyright terms
 * except that the holder is Tim Hudson (tjh@cryptsoft.com).
 *
 * Copyright remains Eric Young's, and as such any Copyright notices in
 * the code are not to be removed.
```

```
 * If this package is used in a product, Eric Young should be given attribution
 * as the author of the parts of the library used.
 * This can be in the form of a textual message at program startup or
 * in documentation (online or textual) provided with the package.
 *
 * 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 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.
 * 3. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 *    "This product includes cryptographic software written by
 *     Eric Young (eay@cryptsoft.com)"
 *    The word 'cryptographic' can be left out if the rouines from the library
 *    being used are not cryptographic related :-).
 * 4. If you include any Windows specific code (or a derivative thereof) from
 *    the apps directory (application code) you must include an acknowledgement:
 *    "This product includes software written by Tim Hudson (tjh@cryptsoft.com)"
 *
 * THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ``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 AUTHOR 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.
 *
 * The licence and distribution terms for any publically available version or
 * derivative of this code cannot be changed.  i.e. this code cannot simply be
 * copied and put under another distribution licence
 * [including the GNU Public Licence.]
 */
```

Parent topic: Notices and Licenses

Third Party Library Notices

```
THIRD PARTY LIBRARY NOTICES:

    --------------------------------------------------------------------------------
    Apache Avro Mapred API
    Copyright 2009-2013 The Apache Software Foundation

    This product includes software developed at
    The Apache Software Foundation (http://www.apache.org/).

    --------------------------------------------------------------------------------
    Apache CXF
    Copyright 2006-2018 The Apache Software Foundation

    This product includes software developed at
    The Apache Software Foundation (http://www.apache.org/).

    --------------------------------------------------------------------------------
    Curator Recipes
    Copyright 2011-2018 The Apache Software Foundation

    This product includes software developed at
    The Apache Software Foundation (http://www.apache.org/).

    --------------------------------------------------------------------------------
    Spark Project SQL
    Copyright 2016 The Apache Software Foundation

    This product includes software developed at
    The Apache Software Foundation (http://www.apache.org/).

    --------------------------------------------------------------------------------
    # Jackson JSON processor

    Jackson is a high-performance, Free/Open Source JSON processing library.
    It was originally written by Tatu Saloranta (tatu.saloranta@iki.fi), and has
    been in development since 2007.
    It is currently developed by a community of developers, as well as supported
    commercially by FasterXML.com.

    ## Licensing
```

```
Jackson core and extension components may be licensed under different licenses.
To find the details that apply to this artifact see the accompanying LICENSE fi
For more information, including possible other licensing options, contact
FasterXML.com (http://fasterxml.com).

## Credits

A list of contributors may be found from CREDITS file, which is included
in some artifacts (usually source distributions); but is always available
from the source code management (SCM) system project uses.


---------------------------------------------------------------------------
# Jackson JSON processor

Jackson is a high-performance, Free/Open Source JSON processing library.
It was originally written by Tatu Saloranta (tatu.saloranta@iki.fi), and has
been in development since 2007.
It is currently developed by a community of developers, as well as supported
commercially by FasterXML.com.

## Licensing

Jackson core and extension components may licensed under different licenses.
To find the details that apply to this artifact see the accompanying LICENSE fi
For more information, including possible other licensing options, contact
FasterXML.com (http://fasterxml.com).

## Credits

A list of contributors may be found from CREDITS file, which is included
in some artifacts (usually source distributions); but is always available
from the source code management (SCM) system project uses.


---------------------------------------------------------------------------
Apache Commons Compress
Copyright 2002-2013 The Apache Software Foundation

This product includes software developed at
The Apache Software Foundation (http://www.apache.org/).


---------------------------------------------------------------------------
Apache Commons Lang
Copyright 2001-2011 The Apache Software Foundation

This product includes software developed by
```

```
      The Apache Software Foundation (http://www.apache.org/).

      --------------------------------------------------------------------------------
      Apache log4j
      Copyright 2007 The Apache Software Foundation

      This product includes software developed at
      The Apache Software Foundation (http://www.apache.org/).

      --------------------------------------------------------------------------------
      This product includes software developed by The Apache Software
      Foundation (http://www.apache.org/).

      The binary distribution of this product bundles binaries of
      org.iq80.leveldb:leveldb-api (https://github.com/dain/leveldb), which has the
      following notices:
      * Copyright 2011 Dain Sundstrom <dain@iq80.com>
      * Copyright 2011 FuseSource Corp. http://fusesource.com

      The binary distribution of this product bundles binaries of
      org.fusesource.hawtjni:hawtjni-runtime (https://github.com/fusesource/hawtjni),
      which has the following notices:
      * This product includes software developed by FuseSource Corp.
        http://fusesource.com
      * This product includes software developed at
        Progress Software Corporation and/or its  subsidiaries or affiliates.
      * This product includes software developed by IBM Corporation and others.

      The binary distribution of this product bundles binaries of
      Gson 2.2.4,
      which has the following notices:

                                  The Netty Project
                                  =================

      Please visit the Netty web site for more information:

        * http://netty.io/

      Copyright 2014 The Netty Project

      The Netty Project licenses this file to you 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.

  Also, please refer to each LICENSE.<component>.txt file, which is located in
  the 'license' directory of the distribution file, for the license terms of the
  components that this product depends on.

  -------------------------------------------------------------------------------
  This product contains the extensions to Java Collections Framework which has
  been derived from the works by JSR-166 EG, Doug Lea, and Jason T. Greene:

    * LICENSE:
      * license/LICENSE.jsr166y.txt (Public Domain)
    * HOMEPAGE:
      * http://gee.cs.oswego.edu/cgi-bin/viewcvs.cgi/jsr166/
      * http://viewvc.jboss.org/cgi-bin/viewvc.cgi/jbosscache/experimental/jsr166/

  This product contains a modified version of Robert Harder's Public Domain
  Base64 Encoder and Decoder, which can be obtained at:

    * LICENSE:
      * license/LICENSE.base64.txt (Public Domain)
    * HOMEPAGE:
      * http://iharder.sourceforge.net/current/java/base64/

  This product contains a modified portion of 'Webbit', an event based
  WebSocket and HTTP server, which can be obtained at:

    * LICENSE:
      * license/LICENSE.webbit.txt (BSD License)
    * HOMEPAGE:
      * https://github.com/joewalnes/webbit

  This product contains a modified portion of 'SLF4J', a simple logging
  facade for Java, which can be obtained at:

    * LICENSE:
      * license/LICENSE.slf4j.txt (MIT License)
    * HOMEPAGE:
      * http://www.slf4j.org/
```

```
This product contains a modified portion of 'ArrayDeque', written by Josh
Bloch of Google, Inc:

  * LICENSE:
    * license/LICENSE.deque.txt (Public Domain)

This product contains a modified portion of 'Apache Harmony', an open source
Java SE, which can be obtained at:

  * LICENSE:
    * license/LICENSE.harmony.txt (Apache License 2.0)
  * HOMEPAGE:
    * http://archive.apache.org/dist/harmony/

This product contains a modified version of Roland Kuhn's ASL2
AbstractNodeQueue, which is based on Dmitriy Vyukov's non-intrusive MPSC queue.
It can be obtained at:

  * LICENSE:
    * license/LICENSE.abstractnodequeue.txt (Public Domain)
  * HOMEPAGE:
    * https://github.com/akka/akka/blob/wip-2.2.3-for-scala-2.11/akka-actor/src/

This product contains a modified portion of 'jbzip2', a Java bzip2 compression
and decompression library written by Matthew J. Francis. It can be obtained at:

  * LICENSE:
    * license/LICENSE.jbzip2.txt (MIT License)
  * HOMEPAGE:
    * https://code.google.com/p/jbzip2/

This product contains a modified portion of 'libdivsufsort', a C API library to
the suffix array and the Burrows-Wheeler transformed string for any input string
a constant-size alphabet written by Yuta Mori. It can be obtained at:

  * LICENSE:
    * license/LICENSE.libdivsufsort.txt (MIT License)
  * HOMEPAGE:
    * https://code.google.com/p/libdivsufsort/

This product contains a modified portion of Nitsan Wakart's 'JCTools', Java Conc
 which can be obtained at:

  * LICENSE:
```

```
    * license/LICENSE.jctools.txt (ASL2 License)
  * HOMEPAGE:
    * https://github.com/JCTools/JCTools

This product optionally depends on 'JZlib', a re-implementation of zlib in
pure Java, which can be obtained at:

  * LICENSE:
    * license/LICENSE.jzlib.txt (BSD style License)
  * HOMEPAGE:
    * http://www.jcraft.com/jzlib/

This product optionally depends on 'Compress-LZF', a Java library for encoding a
decoding data in LZF format, written by Tatu Saloranta. It can be obtained at:

  * LICENSE:
    * license/LICENSE.compress-lzf.txt (Apache License 2.0)
  * HOMEPAGE:
    * https://github.com/ning/compress

This product optionally depends on 'lz4', a LZ4 Java compression
and decompression library written by Adrien Grand. It can be obtained at:

  * LICENSE:
    * license/LICENSE.lz4.txt (Apache License 2.0)
  * HOMEPAGE:
    * https://github.com/jpountz/lz4-java

This product optionally depends on 'lzma-java', a LZMA Java compression
and decompression library, which can be obtained at:

  * LICENSE:
    * license/LICENSE.lzma-java.txt (Apache License 2.0)
  * HOMEPAGE:
    * https://github.com/jponge/lzma-java

This product contains a modified portion of 'jfastlz', a Java port of FastLZ co
and decompression library written by William Kinney. It can be obtained at:

  * LICENSE:
    * license/LICENSE.jfastlz.txt (MIT License)
  * HOMEPAGE:
    * https://code.google.com/p/jfastlz/

This product contains a modified portion of and optionally depends on 'Protocol
```

```
interchange format, which can be obtained at:

  * LICENSE:
    * license/LICENSE.protobuf.txt (New BSD License)
  * HOMEPAGE:
    * http://code.google.com/p/protobuf/

This product optionally depends on 'Bouncy Castle Crypto APIs' to generate
a temporary self-signed X.509 certificate when the JVM does not provide the
equivalent functionality.  It can be obtained at:

  * LICENSE:
    * license/LICENSE.bouncycastle.txt (MIT License)
  * HOMEPAGE:
    * http://www.bouncycastle.org/

This product optionally depends on 'Snappy', a compression library produced
by Google Inc, which can be obtained at:

  * LICENSE:
    * license/LICENSE.snappy.txt (New BSD License)
  * HOMEPAGE:
    * http://code.google.com/p/snappy/

This product optionally depends on 'JBoss Marshalling', an alternative Java
serialization API, which can be obtained at:

  * LICENSE:
    * license/LICENSE.jboss-marshalling.txt (GNU LGPL 2.1)
  * HOMEPAGE:
    * http://www.jboss.org/jbossmarshalling

This product optionally depends on 'Caliper', Google's micro-
benchmarking framework, which can be obtained at:

  * LICENSE:
    * license/LICENSE.caliper.txt (Apache License 2.0)
  * HOMEPAGE:
    * http://code.google.com/p/caliper/

This product optionally depends on 'Apache Commons Logging', a logging
framework, which can be obtained at:

  * LICENSE:
    * license/LICENSE.commons-logging.txt (Apache License 2.0)
```

```
    * HOMEPAGE:
      * http://commons.apache.org/logging/

  This product optionally depends on 'Apache Log4J', a logging framework, which
  can be obtained at:

    * LICENSE:
      * license/LICENSE.log4j.txt (Apache License 2.0)
    * HOMEPAGE:
      * http://logging.apache.org/log4j/

  This product optionally depends on 'Aalto XML', an ultra-high performance
  non-blocking XML processor, which can be obtained at:

    * LICENSE:
      * license/LICENSE.aalto-xml.txt (Apache License 2.0)
    * HOMEPAGE:
      * http://wiki.fasterxml.com/AaltoHome

  This product contains a modified version of 'HPACK', a Java implementation of
  the HTTP/2 HPACK algorithm written by Twitter. It can be obtained at:

    * LICENSE:
      * license/LICENSE.hpack.txt (Apache License 2.0)
    * HOMEPAGE:
      * https://github.com/twitter/hpack

  This product contains a modified portion of 'Apache Commons Lang', a Java libra
  provides utilities for the java.lang API, which can be obtained at:

    * LICENSE:
      * license/LICENSE.commons-lang.txt (Apache License 2.0)
    * HOMEPAGE:
      * https://commons.apache.org/proper/commons-lang/

  The binary distribution of this product bundles binaries of
  Commons Codec 1.4,
  which has the following notices:
   * src/test/org/apache/commons/codec/language/DoubleMetaphoneTest.javacontains
    ===============================================================================
    The content of package org.apache.commons.codec.language.bm has been translat
    from the original php source code available at http://stevemorse.org/phonetic
    with permission from the original authors.
    Original source copyright:Copyright (c) 2008 Alexander Beider & Stephen P. Mo
```

```
The binary distribution of this product bundles binaries of
Commons Lang 2.6,
which has the following notices:
 * This product includes software from the Spring Framework,under the Apache Li

The binary distribution of this product bundles binaries of
Apache Log4j 1.2.17,
which has the following notices:
 * ResolverUtil.java
    Copyright 2005-2006 Tim Fennell
  Dumbster SMTP test server
    Copyright 2004 Jason Paul Kitchen
  TypeUtil.java
    Copyright 2002-2012 Ramnivas Laddad, Juergen Hoeller, Chris Beams

The binary distribution of this product bundles binaries of
Jetty 6.1.26,
which has the following notices:
 * ============================================================
    Jetty Web Container
    Copyright 1995-2016 Mort Bay Consulting Pty Ltd.
   ============================================================

   The Jetty Web Container is Copyright Mort Bay Consulting Pty Ltd
   unless otherwise noted.

   Jetty is dual licensed under both

     * The Apache 2.0 License
       http://www.apache.org/licenses/LICENSE-2.0.html

         and

     * The Eclipse Public 1.0 License
       http://www.eclipse.org/legal/epl-v10.html

   Jetty may be distributed under either license.

   ------
   Eclipse

   The following artifacts are EPL.
    * org.eclipse.jetty.orbit:org.eclipse.jdt.core

   The following artifacts are EPL and ASL2.
```

```
 * org.eclipse.jetty.orbit:javax.security.auth.message


The following artifacts are EPL and CDDL 1.0.
 * org.eclipse.jetty.orbit:javax.mail.glassfish


------
Oracle

The following artifacts are CDDL + GPLv2 with classpath exception.
https://glassfish.dev.java.net/nonav/public/CDDL+GPL.html

 * javax.servlet:javax.servlet-api
 * javax.annotation:javax.annotation-api
 * javax.transaction:javax.transaction-api
 * javax.websocket:javax.websocket-api

------
Oracle OpenJDK

If ALPN is used to negotiate HTTP/2 connections, then the following
artifacts may be included in the distribution or downloaded when ALPN
module is selected.

 * java.sun.security.ssl

These artifacts replace/modify OpenJDK classes.  The modifications
are hosted at github and both modified and original are under GPL v2 with
classpath exceptions.
http://openjdk.java.net/legal/gplv2+ce.html


------
OW2

The following artifacts are licensed by the OW2 Foundation according to the
terms of http://asm.ow2.org/license.html

org.ow2.asm:asm-commons
org.ow2.asm:asm


------
Apache
```

```
The following artifacts are ASL2 licensed.

org.apache.taglibs:taglibs-standard-spec
org.apache.taglibs:taglibs-standard-impl



------
MortBay

The following artifacts are ASL2 licensed.  Based on selected classes from
following Apache Tomcat jars, all ASL2 licensed.

org.mortbay.jasper:apache-jsp
  org.apache.tomcat:tomcat-jasper
  org.apache.tomcat:tomcat-juli
  org.apache.tomcat:tomcat-jsp-api
  org.apache.tomcat:tomcat-el-api
  org.apache.tomcat:tomcat-jasper-el
  org.apache.tomcat:tomcat-api
  org.apache.tomcat:tomcat-util-scan
  org.apache.tomcat:tomcat-util

org.mortbay.jasper:apache-el
  org.apache.tomcat:tomcat-jasper-el
  org.apache.tomcat:tomcat-el-api



------
Mortbay

The following artifacts are CDDL + GPLv2 with classpath exception.

https://glassfish.dev.java.net/nonav/public/CDDL+GPL.html

org.eclipse.jetty.toolchain:jetty-schemas

------
Assorted

The UnixCrypt.java code implements the one way cryptography used by
Unix systems for simple password protection.  Copyright 1996 Aki Yoshida,
modified April 2001  by Iris Van den Broeke, Daniel Deville.
Permission to use, copy, modify and distribute UnixCrypt
for non-commercial or commercial purposes and without fee is
```

```
      granted provided that the copyright notice appears in all copies./

    The binary distribution of this product bundles binaries of
    Snappy for Java 1.0.4.1,
    which has the following notices:
     * This product includes software developed by Google
        Snappy: http://code.google.com/p/snappy/ (New BSD License)

       This product includes software developed by Apache
        PureJavaCrc32C from apache-hadoop-common http://hadoop.apache.org/
        (Apache 2.0 license)

       This library containd statically linked libstdc++. This inclusion is allowed
       "GCC RUntime Library Exception"
       http://gcc.gnu.org/onlinedocs/libstdc++/manual/license.html

       == Contributors ==
         * Tatu Saloranta
           * Providing benchmark suite
         * Alec Wysoker
           * Performance and memory usage improvement

    The binary distribution of this product bundles binaries of
    Xerces2 Java Parser 2.9.1,
    which has the following notices:
     * ========================================================================
       ==  NOTICE file corresponding to section 4(d) of the Apache License,   ==
       ==  Version 2.0, in this case for the Apache Xerces Java distribution. ==
       ========================================================================

       Apache Xerces Java
       Copyright 1999-2007 The Apache Software Foundation

       This product includes software developed at
       The Apache Software Foundation (http://www.apache.org/).

       Portions of this software were originally based on the following:
         - software copyright (c) 1999, IBM Corporation., http://www.ibm.com.
         - software copyright (c) 1999, Sun Microsystems., http://www.sun.com.
         - voluntary contributions made by Paul Eng on behalf of the
           Apache Software Foundation that were originally developed at iClick, Inc
           software copyright (c) 1999.
```

Parent topic: Notices and Licenses

5.2.32 Yellowbrick Release Notes

Release Version: 5.2.32-43426.02f7540e

Release Date:

In This Section

Upgrade Requirements

Read the following information before beginning an upgrade to Version 5.2.

> **Important**
>
> Customers should always contact Customer Support ahead of the planned upgrade date to review the process and schedule the upgrade.

Upgrades from Version 5.x

If you are upgrading from a version prior to 5.2.10, additional steps may be required to upgrade the cluster. Customer Support will walk you through the process.

For upgrades from 5.2.9 or earlier: Before starting the upgrade, on the version you are upgrading from, run:

```
ALTER SYSTEM SET autovacuum TO 'off';
SELECT pg_reload_conf();
```

After a successful upgrade, when the database is up, run:

```
ALTER SYSTEM SET autovacuum TO 'on';
SELECT pg_reload_conf();
```

For upgrades from 5.2.0, 5.2.1, and 5.2.2: An additional step at the end of the upgrade process is required to update view definitions. Technical Support will walk you through the process.

See also Replication Compatibility and Upgrade Process.

Upgrades from Version 4.x

Be sure to read Changes in Behavior and Compatibility.

For upgrades from 4.1.15 or earlier: Before starting the upgrade, on the version you are upgrading from, run:

```
ALTER SYSTEM SET autovacuum TO 'off';
SELECT pg_reload_conf();
```

After a successful upgrade, when the database is up, run:

```
ALTER SYSTEM SET autovacuum TO 'on';
SELECT pg_reload_conf();
```

Changes in Behavior in This Version

5.2.31 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.

5.2.31 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.

5.2.28 Precision of % on Decimal

The result of the modulo operation had the resulting precision `MAX(p1, p2)`. This was changed to `MAX(p1 - s1, p2 - s2) + MAX(s1, s2)`.

5.2.23 Behavior of the CHAR Data Type

The CHAR data type implementation in Yellowbrick 5.x has resulted in some behavioral inconsistencies.

5.2.23 Default Behavior of WLM Restart Rules

By default, a subset of recoverable error codes triggers an attempted restart, and an attempted restart occurs only once per query.

For details, see "Restarting Queries" in the updated Version 5.2.23 documentation.

You may need to update your existing WLM query restart rules as a result.

5.2.18 Query Plans for GROUP BY in Decorrelated Subqueries

In previous versions, the GROUP BY clause in a decorrelated subquery of the following form:

```
SELECT ...
  FROM t1
  WHERE x IN (
    SELECT y
    FROM t2
    WHERE t2.x = t1.x
    GROUP BY y
  )
```

was always executed as part of the query plan. In Version 5.2.18, a GROUP BY clause in this kind of query is only executed if statistics suggest that its inclusion will improve performance. Therefore, you may see a change in the plan for this type of query.

5.2.11 Increased Memory Estimation for Some Statements

Queries may experience slightly higher memory estimation, up to a maximum of approximately 500MB more than was used in Versions 5.2.8 and earlier. The actual increase depends on how much disk space is available on each worker. This change in estimation corrects an issue in Version 5.2.9 where memory estimation was set too high and out-of-memory (OOM) conditions could occur when a query started spilling.

This may affect the WLM pool that your existing WLM rules assign statements to.

5.2.9 Deprecated Feature: Restartable Replication

The restartable replication feature that was introduced as a tech preview in Version 5.2.0 is deprecated and disabled in Version 5.2.9. This feature is not supported under any circumstances for use on production systems; if you would like to use it purely for testing purposes, contact Customer Support.

5.2.9 Query Planning Costing Changes

Default constants for costing have been changed to optimize memory consumption for queries. This may affect the WLM pools statements are assigned to under your existing WLM rules.

5.2.9 Explicit Casts for Numeric Calculations are not Ignored

In previous versions, an explicit cast for a numeric expression was ignored, and the return type was adjusted to fit the precision and scale of the actual value. In Version 5.2.9, such explicit casts are respected, and the return type of the expression is based on the requested precision and scale.

For example, in Version 5.2.9 the following CTAS statement returns DECIMAL(21,10):

```
yellowbrick_test=# create table t as
select 3.14159::NUMERIC(20,10) + 1::NUMERIC(1,0) from sys.const;
SELECT 1
yellowbrick_test=# \d t
Table "public.t"
  Column   | Type           | Modifiers
----------+----------------+-----------
  ?column? | numeric(21,10) |
```

The same statement in earlier 5.2.x versions returns DECIMAL(11,10):

```
yellowbrick_test=# create table t as
select 3.14159::NUMERIC(20,10) + 1::NUMERIC(1,0) from sys.const;
SELECT 1
yellowbrick_test=# \d t
Table "public.t"
  Column  | Type           | Modifiers
----------+----------------+-----------
?column? | numeric(11,10) |
```

5.2.4 Increased Memory Requirements for table scans

In Version 5.2.4, all queries that involve table scans will see an increase in the amount of memory required for execution. The amount of increase depends on the shape of the query plan (number of tables, columns, column types, and so on). A rough estimate of the range is from 300 MiB to a few GiB for quereis with a large number of columns or very wide string columns..

You may need to tune your WLM rules, particularly if pool assignments are not based on the planner memory estimated sizes. If your WLM rules are based on planner memory estimates, you may see statements now running in larger pools. Whatever changes you make, we advise a dry-run of these changes on a development system.

5.2.4 Cursor Behavior Changes

In Version 5.2.4, a number of fixes result in changes to the behavior of cursors including thier type for WLM rule and sys.log_query classifcation. For details about supported cursor moves and fetches, see the descriptions of the DECLARE, FETCH, and MOVE commands in the Version 5.2.4 documentation.

Note: Holdable cursors execute their underlying queries when they are declared, and these queries are tagged as DECLARE operations, not as SELECT operations. Therefore, these queries cannot be restarted by WLM, and they are not subject to the WLM global row limit.

5.2.4 Updated System Catalog

5.2.4 introduces a new version of the system catalog. If you are using database replication, you need to upgrade the source and target of the replicated system at the same time. See Replication Compatibility and Upgrade Process.

5.2.0 System View Updates

The following system views are changed in Version 5.2:

- sys.query and sys.log_query have significant column changes from prior versions.
- In sys.* views and tables the column "user_name" has been changed to "username".
- sys.replica has a new **is_table_level_restart** column.
- sys.replica_status and sys.log_replica_status both have a new **retries** column.

5.2.0 ybcli Changes

- The SSD section of the **blade status** output has been reformatted and now includes the SSD firmware version. Note that on the Andromeda platform the **blade status** output includes an additional line with BMC and FPGA information.

5.2.0 SQL Changes

- In CREATE TABLE, the ON keyword is now supported for DISTRIBUTE ON RANDOM and DISTRIBUTE ON REPLICATE. Previously, the ON keyword was only supported for hash distribution (DISTRIBUTE ON column).
- The default transaction isolation level can be changed only by internal system users. The only supported transaction isolation level that superusers can set is `read committed`.
- The DEALLOCATE command now has an IF EXISTS option.
- The ADD_MONTHS function always returns the last day of the result month when the input expression is the last day of the month. For example, 2000 was a leap year. Therefore, 2000-02-28 + 1 month returns 2000-03-28, and 2000-02-29 + 1 month returns 2000-03-31.

Changes in Compatibility in This Version

5.2.0 ybtools Compatibility

Yellowbrick recommends that you always upgrade ybtools to match the Yellowbrick server version you are running (for example, Version 5.2). Using Version 5.2 of ybtools with an older server version (such as Version 4.1 or 5.1) may result in error messages for some commands. Using previous versions of ybtools with a Version 5.2 database is likely to work but is deprecated and not supported. See also the following section: Backward Compatibility for Backup and Restore.

Important: **On CentOS and Red Hat client platforms, you must first remove any existing 3.x version of ybtools. Then you can proceed with the installation of the Version 5.2 ybtools. You cannot upgrade directly from an earlier 3.x version of ybtools to Version 5.2.**

5.2.0 Replication Compatibility and Upgrade Process

Important: Both systems used for database replication (source and target) must be running the same *major version* of the database software. For example, you cannot replicate from a 4.x source to a 5.x target or vice versa.

In Version 5.2.19, you cannot replicate to a target system on any earlier version; both systems must be running Version 5.2.19 of the database software. For example, you cannot replicate from a 5.2.19 source to a 5.2.18 target. This behavior is enforced because of a change in the system catalog for Version 5.2.18.

You *can* replicate a Version 5.x database *to* a later 5.x database (such as 5.1.x to 5.2.19). For example, if you are upgrading from Version 5.1.x, and you want to reverse replication in Version 5.2 (using PROMOTE), follow these steps:

1. Pause the replica on the Version 5.1.x source system.
2. Upgrade the target system to Version 5.2.19.
3. Verify that replication works from 5.1.x to 5.2.19.
4. Attempt a promote on the 5.2.19 target. This operation will fail with an expected error because you cannot replicate to a system that is running an earlier version of the software. The replica will remain paused on the 5.1.x source system.
5. Upgrade the source system to 5.2.19.
6. Attempt the promote again on the 5.2.19 target system. This operation should succeed.
7. Attempt a new replication cycle from the reversed source to the new target (both 5.2.18 systems). The replication cycle should succeed on the new target.

5.2.0 Backward Compatibility for Backup and Restore

A backup taken in Version 3.2.x, 3.3.x, 4.x, or 5.x prior to a 5.2 upgrade can be restored in Version 5.2.

After a 5.2 upgrade or installation, the previous version of the backup and restore tools is compatible with Version 5.2. The Version 5.2 ybtools package installs two sets of backup and restore tools: legacy tools and new 5.2 tools.

The legacy tools are deprecated and will be removed in a future release, along with the ability to restore backups taken in previous releases. There are significant differences in the functionality and client tool options between the legacy tools and the new tools. The tool sets are distinguished by their names:

ybtools Version	Backup	Restore	List/Manage Backups
New in Version 4.x and later	ybbackup	ybrestore	ybbackupctl
Legacy in Version 4.x and later	ybbackup-old	ybrestore-old	ybbackup-list-old
Version 3.x	ybbackup	ybrestore	ybbackup-list

To use the legacy tools in the 5.2 package, be sure to use the correct tool name in the command. For example, to return help for the legacy version of ybbackup:

```
$ ybbackup-old --help
Yellowbrick Backup and Restore version 5.2.0-33445
...
```

Important: To use backup and restore in Version 5.2, you must follow these steps:

1. Use the legacy version of ybtools (-old) to restore any 3.x backups into 5.2 databases. Use the new 4.x or 5.x version of ybtools to restore any 4.x or 5.x backups into 5.2 databases.
2. Take new full backups of your 5.2 databases with the new 5.2 ybtools.
3. Stop using the legacy ybtools for backup and restore purposes. Use only the new tools.

What's New in This Version

5.2.31 Replication improvements, bug fixes, security improvements and new alerts

- Added a new alert to detect and report stale backup chains
- Introduced the `Compute Blade Filesystem Usage` alert to monitor the filesystem usage on individual compute blades, providing more granular visibility. See the alerting section for details
- See issues fixed
- List of addressed CVEs can be found here

5.2.30 Bug fixes, security and alerting improvements

New alerts to monitor key system components such as the Yellowbrick Row Store (YRS).

List of issues fixed.

5.2.29 Added the ability to manage appliance password policies using YBCLI

The ability to change minimum password length and rentention of appliance users has been added to YBCLI. For more information see the updated 5.2.29 YBCLI documentation.

5.2.29 A number of CVEs have been addressed in this release

For details about the CVE fixes in this release, see the CVE documentation.

5.2.28 The timezone database has been updated to IANA 2023

This is an important update to bring timezone information up-to-date. Some important changes:

- Mexico will no longer observe DST except near the US border.
- Iran no longer observes DST after 2022.
- Europe/Kyiv is now accepted as a valid timezone.

5.2.27 Improved cluster reconfiguration time

Cluster reconfiguration time on restart, particularly for appliances with a large number of chassis has been improved.

5.2.26 DLE_DST and LE_DST Functions

These two functions have been added. They return the difference between two character strings in terms of the number of changes required to convert one to the other. The DLE_DST function uses the Damerau-Levenshtein edit distance algorithm and the LE_DST function uses the Levenshtein edit distance algorithm to compute the "distance" between the strings.

5.2.25 Kerberos SSO Support in ybools on Windows

The Yellowbrick client tools now support Kerberos SSO on Windows. To enable and configure Kerberos on a Yellowbrick appliance, contact Yellowbrick Customer Support.

5.2.23 JSON_INGEST and JSON_LOOKUP Functions

Two new SQL functions for JSON manipulation, JSON_INGEST and JSON_LOOKUP, are available in Version 5.2.23. See the updated Version 5.2.23 documentation for details and examples.

5.2.22 Kerberos Support in ybtools

The Yellowbrick client tools (ybtools) now have GSSAPI support, which enables support for Kerberos authentication. To enable and configure Kerberos on a Yellowbrick appliance, contact Yellowbrick Customer Support.

5.2.22 Type-Safe Casting Functions

Three new SQL functions are supported for compatibility with applications developed in other databases. These functions implement type-safe casting from VARCHAR strings to DATE, TIMESTAMP, and DOUBLE data types:

- SAFE_TO_DATE
- SAFE_TO_TIMESTAMP
- SAFE_TO_DOUBLE

For more details, see the SQL Functions section in the Version 5.2.x documentation.

5..2.21 Kerberos Authentication

Version 5.2.21 supports Kerberos Version 5 network authentication. To enable and configure Kerberos on a Yellowbrick appliance, contact Yellowbrick Customer Support.

5.2.15 Yellowbrick Client Upgrades for log4j

To ensure that the Yellowbrick client tools are not subject to known log4j vulnerabilities, the log4j jar used by ybtools has been upgraded to version 2.17.1.

5.2.13 Disabling ANALYZE Operations for ybload

Two configuration parameters are available to control the use of ANALYZE operations that occur after data is written to the database:

ybd_analyze_after_loads

> Defaults to ON and triggers an automatic ANALYZE if 10% or more of the rows have changed for the target table when a ybload operation completes. To disable the automatic ANALYZE, set this parameter to OFF. This parameter is new in Version 5.2.13 and applies to workloads that require a large number of continuous trickle-feed ybload operations.
>
> Note that setting this parameter to OFF does not prevent an immediate ANALYZE from being run after intermediate load operations, such as after an INSERT into a target table from a staging table. However, setting ybd_analyze_after_writes to OFF will prevent an immediate ANALYZE in these cases.

ybd_analyze_after_writes

> Defaults to ON and continues to work as described in the Version 5.2.x documentation.

Both parameters and their current settings are visible in the output for SHOW ALL.

5.2.11 Protection from log4j Vulnerabilities

To ensure that the Yellowbrick server and client tools are not subject to known log4j vulnerabilities, unused packages and their dependencies were removed from the log4j distribution used by Yellowbrick. Specifically:

1. The RHEL log4j package and RPMs have been removed from the manager node.
2. The log4j jar used by Yellowbrick on the manager node has been replaced with yb-log-4.1.16.jar, which does not contain the vulnerable classes (JMSAppender, JMSSink, SimpleSocketServer, and SocketServer).
3. The log4j jar used by ybtools has been replaced with yb-log-4.1.16.jar, which does not contain the vulnerable classes (JMSAppender, JMSSink, SimpleSocketServer, and SocketServer).

Note that a logger is still required when integrating with Spark, and it is the responsibility of the Spark runtime to provide it.

5.2.9 TO_NUMBER Function

You can convert a number string to an actual numeric value, using a specified format.

Note that the addition of this function introduces a change to the system catalog version.

5.2.9 Support for 2,000 Columns in a Table

The maximum number of user-defined columns in a table has been increased to 2,000.

5.2.9 Disabling Join Elimination for LEFT Joins

Join elimination is a query optimization that can occur during planning when primary and foreign keys exist on the tables. The query plan may eliminate redundant joins when these key relationships have been declared. However, because Yellowbrick does not *enforce* primary key and foreign key constraints, your tables may have key relationships that are not valid. If this is true, the elimination of joins between those tables may cause queries to return wrong results. |

To prevent this problem from occurring specifically for LEFT joins, you can set the enable_join_elimination configuration parameter to OFF in Version 5.2.9. For more details and examples, see the CREATE TABLE section of

the Version 5.2.9 documentation.

5.2.4 Tech Preview: Setting Configuration Parameters Within Queries

A new SETTING clause may be specified before a query so that it runs with specific values applied for one or more configuration parameters. Contact Technical Support for details.

5.2.4 Management of Lime Cache During Installation

During installation, it is no longer necessary to manually move the lime cache and delete the files later. This operation is now managed automatically by the installer.

5.2.0 Query Planning Enhancements

A number of enhancements have been made in query planning for this release:

- General improvements in join ordering. This includes the flipping of some LEFT OUTER JOINS to RIGHT OUTER JOINS
- Better use of declared primary and foreign keys in query planning
- Improved estimates of join cost and cardinality of matching key values
- Various improvements and adjustments to the EXPLAIN command output

5.2.0 Encryption Keystore Alert

A new user-defined system alert is triggered when the encryption keystore is locked.

5.2.0 ybtools Improvements

- In previous releases, RPMs for ybtools on RHEL and AIX systems had to be installed in the default location (/opt/ybtools). In Version 5.2, you can specify an alternative path.
- The ybdumpschema and ybdumproles help text has been updated for consistency. In previous releases, there were some unnecessary differences in the output for the two tools.

5.2.0 Three-node Configuration for the Andromeda Platform

The Andromeda platform supports a new three-node minimum configuration.

Known Issues in This Version

You may encounter the following issues in Version 5.2.32. Use the workarounds provided in the description and contact Customer Support for additional information.

Issue #	Description
35027	`HTTPS_PROXY` environment variable will not be read by `ybunload` tool when the port is n
33599	In some new versions of the Yellowbrick software, attempting to truncate more than one table i `ERROR:  Cannot start a query more than once` As a temporary workaround, we recommend truncating one table at a time.
27992	Database restore operations sometimes fail and report the following error: `YCATALOGRESTORE: Cannot find mapping for foreign key` If you run into this problem, contact Customer Support for a workaround.
24660	In Versions 5.2.18 and earlier, a rare race condition may occur when using joins that involve sp ultimately causing a worker crash. In Version 5.2.19 and later versions, this type of query no lo fails, but it does return an exception.
20786	Running EXPLAIN ANALYZE on any write statement (INSERT, UPDATE, DELETE) modifies th To work around this problem, run the EXPLAIN ANALYZE statement inside a transaction so yo
20361	The sys.log_query_pre5 view contains query execution summaries for queries run prior to the ! allowed to access this table.

20182	You must create encryption keys in the same database as the tables that will reference them. D to encryption keys in your CREATE TABLE statements. If you do this, you will lose those refere and restored (or replicated).
19980	After upgrading from 4.1.x to 5.2.x, you may encounter the following error during database repl `YCATALOGRESTORE: duplicate key value violates unique constraint "p` If you run into this problem, contact Customer Support for a workaround.
19886	After changing the spill space percentage on your system (Temporary Space Reservation setti restart the database, using ybcli. Otherwise the change will not take effect.
19578	Selecting from the sys.query view returns incorrect results when a WHERE clause condition co example: `SELECT query_id, type, username FROM sys.query WHERE username NOT LIKE 'sys_ybd%';` This problem does not occur for queries against other system views or user tables.
15784, 16097	When the row store becomes full (which triggers a MAJOR alert), DELETE operations may fail queries may be lost.
16101	The SMC does not display row counts in the row store (for example, in the Table Details scree

Issues Fixed in This Version

The following issues are addressed in this version:

Release	Issue #	Description
5.2.32	45910	DCS YRS monitoring can now pause monitoring if no YR
5.2.32	45907	DCS YRS monitoring now caches status for a configurab

5.2.32	45905	Fixed an issue where reloading the configuration could cr were present in `postgres.auto.conf` and audit logg
5.2.32	45825	Improved monitoring of YRS files by adding efficient cour temporary files and ensuring faster status checks in DCS
5.2.31	YBD-39409, YBD-41136	Fixed issues where adding new replicas caused slowdow queries. Improved performance of key replication operations, inclu
5.2.30	40070	Fixed an alerting issue that caused system hardware aler Alerts will now be generated for chassis power, fans, and
5.2.30	38278	Increase visibility of Garbage Collection operations in `sy` This is especially important to understand concurrency an
5.2.30	38140	In rare conditions, a race condition when dropping a data restart. This issue has been resolved.
5.2.30	38129	Improved algorithm for processing duplicated entries in la The plan time for some queries increased quadratically w large `parse_ms` and `plan_ms` in `sys.log_query` The algorithm to parse and plan these types of queries w to 90%. The plan time now grows linearly with the numbe
5.2.30	37056	Fixed a rare race condition in the `DataOutputNode` th `unload` and `yrestore`.
5.2.30	37025	In rare conditions, rows present in the rowstore are disca replicated and there are no rows processed by another w This issue has been resolved.
5.2.30	36925	Disable bloom and pushdown filters for join trees created
5.2.30	36584, 36585	Adjust a fail-safe mechanism to prevent it from causing o

		of `SCHED_YIELD_PROFILE` warnings.
5.2.30	36259, 36799	Fixed a DB frontend crash related to the usage of an inco
5.2.30	23877	Upgrades from 5.2 to 5.4 could lead to `A PRIMARY KE` `yb_alert_definition.profile_name, yb_alert`
5.2.28	29970	Timezone database has been updated to IANA 2023c.
5.2.28	34666	Fixed an issue where the database could take some addi
5.2.28	34599	Improved the performance of auto analyze queries in ord
5.2.28	34537	Fixed wrong precision / scale when simplifying away expl types.
5.2.28	34174	Some backups could error out with: `Abandoned due t` great number of backup chains in a single database that
5.2.28	34145	An issue in the formula for common precision/scale of DE `UNION ALL` nodes use the wrong precision/scale.
5.2.28	34140	Improved support for decorrelating operators mixing corr `select 0 = (select down.b from down whe`
5.2.28	34286	Fix case when a `semi join` being converted into an u `variable not found in subplan target list`
5.2.28	33162	The issue of YB backup failing when executed on a DR s
5.2.28	33236	Improve replication cycle latency when transmitting large
5.2.28	32725	Utility queries (like `CREATE USER` and `GRANT` staten
5.2.28	34803	Fix a crash caused by an incorrect distribution during the

5.2.28	34243	Adhere to IEEE754 standard when comparing floats agai
5.2.28	30809	Utility queries (like `CREATE USER` and `GRANT` staten
5.2.28	33722	Outage caused by trickle inserts on wide table. The large the data to the column store.
5.2.28	33343	Improved force cancellation of queries containing regular
5.2.28	33239	Improved cancellation of queries in the `Plan` stage tha fixed.
5.2.28	28851	Sometimes on a busy system, a query that produces a `Something went wrong try again later`.
5.2.28	20824	In some rare instances, the `sent_rows` field in `sys.` transmitted.
5.2.27	33577	Use of LEFT(f() , N) or RIGHT(f(), N), N < 30 in a CTAS c
5.2.27	33374	Fix an issue where a `semi join` is converted into an columns, resulting in incorrect results.
5.2.27	33234	Fix for CVE-2023-38408
5.2.26	32559	Using the JSON_LOOKUP function with the jpointer optic database instability. In addition, using the JSON_INGEST the ingested JSON column and jpath option returned the `Error: malformed input`
5.2.26	32533	The JSON_LOOKUP function returned a compile error w the first argument.
5.2.26	32530	In some instances, the datepart boundary feature in DAT second, epoch, and millisecond dateparts, causing an ex

5.2.26	32420	An internal error in Postgres prevented "Configure > Use
5.2.26	32092	The GETBIT function returned wrong results when used
5.2.26	31747	The description of the contents of a backup bundle was i
5.2.26	30805	If a temporary table had rows in the rowstore, selecting fr `ERROR: OID out of range`
5.2.26	29699	OS and process interactions would interrupt LDAP, causi `Can't contact LDAP server`
5.2.26	29683	In Version 5.2.26, the database row store alert message is becoming full.
5.2.26	29053	When replicating from 4 chassis to 8 chassis, incrementa
5.2.26	28137	Under certain circumstances, parity rebuild after a drive r
5.2.26	27777	When the ybrelay client requested a commit, the ybrelay database, especially when decompression of the data wa been improved so that all the data is committed
5.2.26	26078	Under certain circumstances after a lime restart, some lir `KE032 Attempt to use spill space above` causing the SMC "Worker CPU Utilization" and "Worker I the sys.log_query view to not be populated.
5.2.24	29270	Joining on strings where both string columns are larger (:
5.2.24	28973	

		In stored procedures performing small inserts to the rows blocks were ignored until a new block was started or the from the results of the full transaction.
5.2.23	29595	Under certain conditions, a query with multiple IN lists wo
5.2.23	29495	Use of the --log-level and --logfilelog-level ybrelay option
5.2.23	28679	Some DDL files were missing from the Windows and Jav configuration.
5.2.23	28655	When output CHAR columns were not referenced consis the query returned the following error: `ERROR: variable not found in subplan ta`
5.2.23	27810	Running a query with multiple COALESCE columns woul
5.2.23	27793	Prepared statements with typemod arguments that were
5.2.23	27628	In some instances, a cursor fetch query in "client wait" st
5.2.23	27470	Default WLM restart rules were applied three times and f subset of error codes triggers a restart, and an attempted
5.2.23	26268	A hung query in idle state was not present in the sys.que pg_termination_backend. In Version 5.2.23, query cance
5.2.23	26155	Under certain conditions, running ybcli commands from th return the following error: `Can not retrieve this platform's hardwa`
5.2.23	20811	For CASE WHEN expressions with CHAR and VARCHA In Version 5.2.23, a VARCHAR type is returned if there is

		expression.
5.2.22	27998	In a query with GROUP BY and LIMIT clauses, a rare rac sometimes result in a cluster restart.
5.2.22	27992	Database restore operations sometimes fail and report th `YCATALOGRESTORE: Cannot find mapping fo` If you run into this problem, contact Customer Support fo
5.2.22	27956	Joining on strings where one string column is small (<=3(wrong results.
5.2.22	27272	Some queries with COUNT(DISTINCT) functions would r
5.2.22	27155	A problem with the rewrite order involving CHAR data typ
5.2.22	27147	A query with multiple subqueries containing the CONCAT
5.2.22	26770	Joins on CHAR columns performed poorly because the p strings, which prevented bloom filters from being applied.
5.2.22	26700	The third-party syslog-ng package was upgraded to fix a files kept on each CMP were rotated, and triggered a DC
5.2.22	26487	The documentation was updated to clarify that SUBSTRI returns a VARCHAR data type, but SUBSTR returns the
5.2.22	26365	Running a stored procedure containing loops and an exp exception handling was involved.
5.2.22	26302	The Yellowbrick installer proceeded as far as the firmwar
5.2.22	26248	A distribution logic issue for SEMI LEFT joins caused red
5.2.22	26082	In some instances, a NOT restriction of correlated subqu
5.2.22	25960	

		After setting a configuration parameter, using SET SESS within the same transaction caused a Postgres crash.
5.2.22	25834	The IN-list rewrite optimization now works with all compa operators were supported for this optimization.
5.2.22	23982	When a regular user executed a long-running SQL stater
5.2.22	23274	Rolling back a transaction with an active backup snapsh
5.2.22	22873	Cluster reconfiguration during a system restart could, in s mechanism could block indefinitely. In Version 5.2.22, the
5.2.21	26228	Some equality conditions on CHAR values with trailing sp
5.2.21	25417	On the Andromeda platform, logins to the SMC were slov service is disabled by default.
5.2.21	25229	For security reasons, the google-gson dependency for ba
5.2.21	25019	A query with multiple rewritten IN lists that constrained th GROUP BY clause returned the following error: `Variable not found in subplan target li`
5.2.21	24674	When a regular user ran a cross-database query that refe query failed with an error: `ERROR: function with OID nnnnnnn does r`
5.2.21	23999	Incorrect implicit casting with string functions resulted in a
5.2.21	20376	Database users granted "Manager" privileges in the SMC
5.2.21	15709	A query plan with a multi-column join and no filtering prec

		a large fact table.
5.2.21	13312	The documentation was updated to clarify the description
5.2.20	24898	CREATE EXTERNAL TABLE failed with the following err `OutOfMemoryError: Direct buffer memory`
5.2.20	24888	Queries against sys.view returned a database lookup err a warning is returned and other views are displayed.
5.2.20	24784	A common table expression (CTE) with an ORDER BY cl
5.2.20	24779	ACLs defined with ALTER DEFAULT PRIVILEGES on the appliance. To change this behavior, add the following opt statement: `user_resolution_mode 'PLACEHOLDER'`
5.2.20	24765, 24660	A rare race condition between spilling and join cleaning c occur for join queries that spill to disk under heavy memo
5.2.19	24157	A worker crash caused by a race condition could occur w
5.2.18	29039	The Yellowbrick client tools (ybtools) now support AIX 7.2
5.2.18	24255	Views created in Version 5.2.15 and earlier that include C
5.2.18	24234	The returned rows counter was not always reset for some reached before the restarted query actually returned the
5.2.18	23977	During an upgrade, some tables appeared to be vacuum return the following error:

		`ERROR: could not access status of trans` `DETAIL: Could not open file "pg_clog/0(`
5.2.18	23954, 23953, 23952, 23951, 23950	Version 5.2.18 addresses some vulnerabilities in the Yell
5.2.18	23912	An IN list rewrite would sometimes return wrong results b
5.2.18	23871	During an upgrade, a table containing duplicate keys that fail.
5.2.18	23869, 23826	A memory allocation issue would cause a crash in the ma
5.2.18	23741	In some instances, a hung monitoring service was not ter and resulting in both manager nodes becoming secondar
5.2.18	23694	Insufficient system pool memory would cause some GC c
5.2.18	23307	A hung query was not present in the sys.query view and operation.
5.2.18	23305	When a blade was being booted, the ybcli blade status c
5.2.18	23094	Using ybcli maintenance mode for DNS changes could ca
5.2.18	22715	Version 5.2.18 improves the ability to set multiple domain
5.2.18	22613	When executed in an aborted transaction, a ROLLBACK sys.query view.
5.2.18	21722	When implicit casting was used for subqueries that includ were applied to the incorrect data type, sometimes result
5.2.18	20015	An upgrade issue would prevent autovacuum from runnin following warning: `WARNING: some databases have not been v`

5.2.17	23255	When estimating range predicates using histograms, the underestimated.
5.2.17	23064	After an upgrade to 5.2.13 or 5.2.14, replication source d READONLY mode. In Version 5.2.17, the system retries source database out of READONLY mode if the target is
5.2.17	22800	Improvements were made to the memory trimming behav
5.2.16	23023	Using DISTINCT inside the MEDIAN function would caus
5.2.16	22886	After an upgrade, some front-end queries would return th `WARNING: out of shared memory.`
5.2.16	22806	The installation preupgrader could terminate early when
5.2.16	22755	Incorrect implicit casting of CHAR values containing only
5.2.16	22750	Parsing a query with a very deeply nested case expressi
5.2.16	22633	Toast tables and indexes related to schema-excluded obj orphaned relations and ultimately causing a database res
5.2.16	22462	A CTAS statement that included a common table express
5.2.16	21746	Out-of-memory (OOM) conditions would sometimes occu were in use.
5.2.16	20674	A CREATE VIEW statement with a CASE expression nov
5.2.16	20377	Entries from yb_deletes tables were included in sys.aalog slower auto-analyze performance.
5.2.16	19908	Selecting all from sys.view took longer to return results th

5.2.15	22308	Character strings inside a NULLIF expression returned w
5.2.15	22168	Misleading histogram statistics were being created for ce slower performance.
5.2.15	22140	The log4j components for ybtools were upgraded to addr
5.2.15	22113	After an upgrade to 5.2, replication failed on the source s "relfilenodes."
5.2.15	15303, 22454	After an upgrade, starting replication sometimes returned `Replication can't proceed due to syste` and "ybcli system status" command incorrectly reported s
5.2.14	22287	Selecting from the sys.query view with a NOT LIKE predi
5.2.14	22117	When a replicated view contained a UDF that was used b database returned a "cache lookup failed" error.
5.2.14	22106	A query with a very large IN list returned a "KE002: Out o
5.2.14	21439	The presence of unicode characters in short character st the strings were compared.
5.2.14	21281	On the Andromeda platform, spilling scalability was impro
5.2.14	20458	Join queries run under low-memory conditions (in a slot c triggered repeated spill cycles and failed with an assertio
5.2.14	17916	The system could be left in a hung state when queries wi 5.2.14, when a regex expression is too complex, the que the problem: `Regex '<pattern>' is too complex. Excee`

		In addition, "yielding" inside the regex engine has been ir complex," system stability is improved for complex patter
5.2.13	21892	The pre-upgrade script returned a Python error when the run a single check on all databases. You can use the --da database. You can run a single check on a single databa are not supported in ybcli.)
5.2.13	21828	The immediate ANALYZE operation that runs after a bulk for ybload.
5.2.13	20896	When a join key was also the target of a constant filter, th
5.2.12	21737	Failure of a GC operation resulted in a "Metadata storage
5.2.12	21734	Performance of scans against the sys.fsinfo virtual table
5.2.12	21717	INSERT statements that required implicit casts on numer
5.2.12	21711	An internal setting for worker spill space could not be retr
5.2.12	21631	In Version 5.2.11, verbose mode was turned off because the installer issue has been fixed and verbose mode is tu
5.2.12	21519, 21522	When a backup contained only a large number of table d the restore service, causing the restore to error out althou
5.2.12	21293	On 3-chassis and 4-chassis systems, the server crashed
5.2.11	21608	In Version 5.2.10, verbose mode was set as the default fo off because it causes a problem during installer rollback.
5.2.11	21389	Certain join queries returned a Generic UnexpectedExce BY clause.
5.2.11	21380	Memory for all joins was over-estimated as a result of a p Estimation.
5.2.11	21069	A rare, intermittent worker crash caused by a race conditi

		involved spilling.
5.2.11	21262, 20958, 20946, 20470	Global parameters set in PostgreSQL configuration files a the results of the system diagnostics command. In Versio package now also includes the contents of sys.vt_cluster
5.2.11	21249, 21233, 21387	To ensure that the Yellowbrick server and client tools are their dependencies were removed from the log4j distribut Vulnerabilities.
5.2.10	21297	To optimize query planning for nested-loop joins, cost cal
5.2.10	21166	Postgres crashed because of a problem with processing command.
5.2.10	21079	When a stored procedure that created a table with a CLU
5.2.10	20640	The ybloadcat and ybdumpcat logs were missing during a upgrades so that pg.log files are redirected to the ybinsta
5.2.10	19375	A restore failure was caused by missing deletes in the ba
5.2.9	21182	For loop joins, the planner sometimes did not choose the
5.2.9	20958	System diagnostics results now include the settings for P ALL and global Postgres configuration files.
5.2.9	20957	The ASCII function is now included in the Version 5.2 doc
5.2.9	20941	A transaction-consistency check in the FSCK code result
5.2.9	20940	Join elimination could cause wrong results for LEFT joins In Version 5.2.9, a workaround is available; see Disablinç
5.2.9	20886	Backups taken over AWS Private Link failed with a timeo
5.2.9	20849	The description of the w.resourcePool property for WLM
5.2.9	20835	The description of the ybbackup --exclude option was imp

5.2.9	20821	A section was added to the Version 5.2 backup and resto encrypted table columns in a restored database.
5.2.9	20786	The EXPLAIN ANALYZE documentation was updated to with EXPLAIN ANALYZE commit permanent changes to rolled back.
5.2.9	20770	If an explicit numeric cast was used in an expression, the Version 5.2.9, the return type is based on the requested
5.2.9	20760	Postgres crashed because of a problem with processing
5.2.9	20750	Restore failures could occur when tables were not restor
5.2.9	20722	At some point in the cycle, a database restore would han network statistics in the log.
5.2.9	20712	The used_bytes column in the sys.storage view is descril used_bytes value is not a precise sum of other values in worker; used_bytes is an approximation, roughly equal to `distributed_bytes + replicated_bytes +`
5.2.9	20710	The SMC showed a value of 0 for temporary space when
5.2.9	20619	Detection of composite primary-key/foreign-key relationsl resulting in the wrong join order being chosen.
5.2.9	20517	The Infiniband route-determination algorithm was change systems.
5.2.9	20466	Because of a memory-estimation problem, certain querie `ERROR: query ran out of memory in pool`

5.2.9	20465	The sys.cluster view logged duplicate rows for the same failed to start.
5.2.9	20407	The NVL/COALESCE function returned wrong results be
5.2.9	20289	The macOS version of ybsql Version 5.2.4 crashed on st
5.2.9	20172	DECIMAL values lost precision when implicit casting was UNION ALL.
5.2.9	17371	A CTAS query that returned an error during parsing or pl of the expected "ctas" type.
5.2.8	20701	Under certain conditions, an INSERT query that was rest For more details, request the Support Bulletin on this top
5.2.8	20639	Timeouts for Postgres shutdown during installation were
5.2.8	20633	The documentation on cross-database queries has been supported.
5.2.7	20653	A security issue made it possible for users to drop databa Security Bulletin on this topic from Customer Support.
5.2.7	20624	The output of ybdumpcat now includes the number of rov
5.2.7	20622	Improvements were made to the preupgrade script to log
5.2.7	20621	Log messages were added to the pg.log when table dele TABLE is executed, check the pg.log for a line of the form `inserted td record oid <oid of dropped`
5.2.7	20538	A rare race condition could cause a restore or replication
5.2.7	20448	A query against the sys.query view with an IN list conditio
5.2.7	20356	

		Under memory pressure, a rare issue occurred with comp
5.2.7	18058	When queries failed with out of memory errors, a race co
5.2.6	20492	Database restore operations could hang when network e were idle for a long time. In Version 5.2.6, the length of ti transfers has been reduced.
5.2.6	20450	The compiler was overly aggressive in its memory consu
5.2.5	20425	Declaring a cursor on a query containing a single count(*
5.2.5	20381, 20394, 20396	In rare instances, the cancellation of a stored procedure
5.2.5	20361	The sys.log_query_pre5 view contains query execution s superusers are allowed to access this table.
5.2.5	20357	Using a WLM throttle rule incorrectly returned error code
5.2.5	20311	A race condition prevented the server from starting up.
5.2.5	20183	The cancellation of a holdable cursor query in "client wait
5.2.5	19658, 20149	An aggregate query ran successfully on the Tinman platf schedule task" error.
5.2.5	18556	When restoring large backup bundles, restore operations
5.3	20182	You must create encryption keys in the same database a references to encryption keys in your CREATE TABLE st database is backed up and restored (or replicated).
5.3	19886	After changing the spill space percentage on your system stop and restart the database, using ybcli. Otherwise the
5.3	19578	Selecting from the sys.query view returns incorrect result operator. For example:

		``` SELECT query_id, type, username FROM sys.query WHERE username NOT LIKE 'sys_ybd%'; ``` This problem does not occur for queries against other sys
5.2.4	20139	Customers requested a way to set configuration paramet Within Single Queries.
5.2.4	20127	A workaround was implemented for a problem with missi details.
5.2.4	20059	Role-specific settings defined with ALTER USER/ROLE v and ybunload.
5.2.4	20057, 19954, 19939, 19728	Multiple issues with cursors have been addressed, includ implementation have been made. See the Version 5.2.4 c
5.2.4	19992	Version 5.2.4 increases memory requirements for all que
5.2.4	19983	Slow query processing of the regexp_replace function co regexp_replace processing has been improved.
5.2.4	19981	A stored procedure had the wrong return type in pg_proc
5.2.4	19980	After upgrading from 4.1.x to 5.2.x, you may encounter th `YCATALOGRESTORE: duplicate key value viol` `"pg_type_typname_nsp_index"` If you run into this
5.2.4	19961	Installation on systems with large lime and query caches
5.2.4	19956	After a successful replication cycle, the statistics on the t
5.2.4	19926	In the SMC, the Monitor Load Processes screen listed jol
5.2.4	19887	An aggregate query that spills to disk ran successfully on an out-of-memory error. In Version 5.2.4, significantly les

5.2.4	19877	When there was data in the row store, dropping a NOT N resulted in an internal error, and the table data became ir
5.2.4	19580	The cluster became unresponsive for a query with a rege
5.2.4	18452	The Execution Timeline in the SMC displayed only the qu
5.2.3	19974	A function was added to repair Version 5.2.x views with ir assistance.
5.2.3	19973	Changes to join ordering introduced in Version 5.2.0 caus LEFT joins became FULL joins. In Version 5.2.3, the origi
5.2.2	19801	A Postgres crash occurred when the result of a correlated function.
5.2.2	19782	Running the pre-upgrade script resulted in a deadlock wh
5.2.2	19755	A Postgres crash occurred with a SIGSEGV error becaus bound views.
5.2.2	19723	During planning for ANTI-joins, statistics were assigned ir
5.2.2	19714	Excessive amounts of data were being captured for remo growth and size of the worker log files, including shorteni 512 characters of SQL text.
5.2.2	19699	An INSERT INTO SELECT or CTAS with an ORDER BY
5.2.1	19697	A restore operation failed to automatically create the rest contained characters that require escaping (such as perio with previous releases can be restored successfully in 5.2
5.2.1	19635	Creating or dropping database in the SMC returned an ex `Connection Error: database "null" does`

5.2.1	19577	A cluster could fail to restart under the following condition • A DELETE was run in the same transaction as a subs • The DELETE and INSERT both targeted the same ta • A WLM rule caused the INSERT to restart. Although the transaction completed, the DELETE was nc FSCK failure occurred.
5.2.1	19280	The "ybcli health blade" command still reported that a bla
5.2.1	16474	When the result of an IN list subquery required a cast, th `Missing constant for test expression` In Version 5.2.1, the cast that is applied to the result of ar
5.2.0	19434	During the activation of a WLM profile with a long timeou query.
5.2.0	19349, 18348	When a user attempted a password change, the SMC loc message.
5.2.0	19072	The Postgres acl_explode() function returned the followir the function: `SQL: ERROR: unrecognized aclright:83886`
5.2.0	18965	RPMs for ybtools on RHEL and AIX systems had to be in can specify an alternative path.
5.2.0	18922	The MONTHS_BETWEEN function did not work fully as 31 days, and the result is an integral value if both month month.
5.2.0	18671	False PSU alerts were being issued from the CMPs.

5.2.0	18529	A problem with the decorrelation of expressions in a sele `SQL Error [XX000]: ERROR: variable not`
5.2.0	18225	User-defined functions were not being preserved across
5.2.0	18046	After an upgrade from Version 3.x, the word BATCH coul
5.2.0	17502	The ybdumpschema tool returned unexpected "depender emitted.
5.2.0	17206, 17020	In Version 5.2, a new user-defined system alert is trigger
5.2.0	16589	Expressions contained in join conditions are now pushed cardinality-increasing joins.
5.2.0	15365	The ADD_MONTHS function returned the wrong result w
5.2.0	14560	When a non-existent fully qualified column was reference references are not allowed." The correct error, returned ir

CVE Fixes in This Version

The following CVEs are addressed in this version:

Release	CVE Fix	Description
5.2.32	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.
5.2.31	CVE-2018-20676	In Bootstrap before 3.4.0, XSS is possible in the tooltip data-viewport attribute.
5.2.31	CVE-2018-14042	In Bootstrap before 4.1.2, XSS is possible in the data-container property of tooltip.
5.2.31	CVE-2018-20677	In Bootstrap before 3.4.0, XSS is possible in the affix configuration target property.
5.2.31	CVE-2018-14041	In Bootstrap before 4.1.2, XSS is possible in the data-target property of scrollspy.
5.2.31	CVE-2016-10735	In Bootstrap 3.x before 3.4.0 and 4.x-beta before 4.0.0-beta.2, XSS is possible in the data-target attribute, a different vulnerability than CVE-2018-14041.
5.2.31	CVE-2019-8331	In Bootstrap before 3.4.1 and 4.3.x before 4.3.1, XSS is possible in the tooltip or popover data-template attribute.
5.2.30	CVE-2022-31197	The PGJDBC implementation of the `java.sql.ResultRow.refreshRow()` method is not performing escaping of column names so a malicious column name that contains a statement terminator, e.g. `;`, could lead to SQL injection. This could lead to executing additional SQL commands as the application's JDBC user. User applications that do not invoke the `ResultSet.refreshRow()` method are not impacted.
5.2.29	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.
5.2.29	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.
5.2.29	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.
5.2.29	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.
5.2.29	CVE-2017-12615	When running Apache Tomcat 5.2.29 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.
5.2.29	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 5.2.29 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.
5.2.29	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.
5.2.29	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.
5.2.29	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.
5.2.29	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.
5.2.29	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.
5.2.29	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.
5.2.27	CVE-2023-38408	undefined
5.2.26	CVE-2021-42392	undefined

Parent topic: Yellowbrick Documentation

© 2018 - 2025 Yellowbrick Data, Inc. All Rights Reserved. | Terms and Privacy Policy