
# Percona Server Documentation

***Release 8.0.18-9***



**Dec 11, 2019**

# CONTENTS

*Percona Server for MySQL* is an enhanced drop-in replacement for *MySQL*. With *Percona Server for MySQL*,

- Your queries will run faster and more consistently.
- You will consolidate servers on powerful hardware.
- You will delay sharding, or avoid it entirely.
- You will save money on hosting fees and power.
- You will spend less time tuning and administering.
- You will achieve higher uptime.
- You will troubleshoot without guesswork.

Does this sound too good to be true? It's not. *Percona Server for MySQL* offers breakthrough performance, scalability, features, and instrumentation. Its self-tuning algorithms and support for extremely high-performance hardware make it the clear choice for companies who demand the utmost performance and reliability from their database server.

# Part I

# Introduction

CHAPTER

ONE

# THE PERCONA XTRADB STORAGE ENGINE

Percona XtraDB is an enhanced version of the *InnoDB* storage engine, designed to better scale on modern hardware. It also includes a variety of other features useful in high-performance environments. It is fully backwards compatible, and so can be used as a drop-in replacement for standard *InnoDB*.

Percona XtraDB includes all of *InnoDB* 's robust, reliable `ACID`-compliant design and advanced `MVCC` architecture, and builds on that solid foundation with more features, more tunability, more metrics, and more scalability. In particular, it is designed to scale better on many cores, to use memory more efficiently, and to be more convenient and useful. The new features are especially designed to alleviate some of *InnoDB*'s limitations. We choose features and fixes based on customer requests and on our best judgment of real-world needs as a high-performance consulting company.

Percona XtraDB engine will not have further binary releases, it is distributed as part of *Percona Server for MySQL*.

CHAPTER

TWO

# LIST OF FEATURES AVAILABLE IN *PERCONA SERVER FOR MYSQL* RELEASES

| *Percona Server for MySQL* 5.6 | *Percona Server for MySQL* 5.7 | *Percona Server for MySQL* 8.0 |
|---|---|---|
| Improved Buffer Pool Scalability | Improved Buffer Pool Scalability | Improved Buffer Pool Scalability |
| Improved InnoDB I/O Scalability | Improved InnoDB I/O Scalability | Improved InnoDB I/O Scalability |
| Multiple Adaptive Hash Search Partitions | Multiple Adaptive Hash Search Partitions | Multiple Adaptive Hash Search Partitions |
| Atomic write support for Fusion-io devices | Atomic write support for Fusion-io devices | Atomic write support for Fusion-io devices |
| Query Cache Enhancements | Query Cache Enhancements | Feature not implemented |
| Improved NUMA support | Improved NUMA support | Feature not implemented |
| Thread Pool | Thread Pool | Thread Pool |
| Suppress Warning Messages | Suppress Warning Messages | Suppress Warning Messages |
| Ability to change database for mysqlbinlog | Ability to change database for mysqlbinlog | Ability to change database for mysqlbinlog |
| Fixed Size for the Read Ahead Area | Fixed Size for the Read Ahead Area | Fixed Size for the Read Ahead Area |
| Improved MEMORY Storage Engine | Improved MEMORY Storage Engine | Improved MEMORY Storage Engine |
| Restricting the number of binlog files | Restricting the number of binlog files | Restricting the number of binlog files |
| Ignoring missing tables in mysqldump | Ignoring missing tables in mysqldump | Ignoring missing tables in mysqldump |
| Too Many Connections Warning | Too Many Connections Warning | Too Many Connections Warning |
| Handle Corrupted Tables | Handle Corrupted Tables | Handle Corrupted Tables |
| Lock-Free SHOW SLAVE STATUS | Lock-Free SHOW SLAVE STATUS | Lock-Free SHOW SLAVE STATUS |
| Expanded Fast Index Creation | Expanded Fast Index Creation | Expanded Fast Index Creation |
| Percona Toolkit UDFs | Percona Toolkit UDFs | Percona Toolkit UDFs |
| Support for Fake Changes | Support for Fake Changes | Support for Fake Changes |
| Kill Idle Transactions | Kill Idle Transactions | Kill Idle Transactions |
| XtraDB changed page tracking | XtraDB changed page tracking | XtraDB changed page tracking |
| Enforcing Storage Engine | Enforcing Storage Engine | Replaced with upstream implementation |
| Utility user | Utility user | Feature not implemented |
| Extending the secure-file-priv server option | Extending the secure-file-priv server option | Extending the secure-file-priv server option |
| Expanded Program Option Modifiers | Expanded Program Option Modifiers | Feature not implemented |
| PAM Authentication Plugin | PAM Authentication Plugin | PAM Authentication Plugin |

Continued on next page

Table 2.1 – continued from previous page

| *Percona Server for MySQL* 5.6 | *Percona Server for MySQL* 5.7 | *Percona Server for MySQL* 8.0 |
|---|---|---|
| Log Archiving for XtraDB | Log Archiving for XtraDB | Log Archiving for XtraDB |
| User Statistics | User Statistics | User Statistics |
| Slow Query Log | Slow Query Log | Slow Query Log |
| Count InnoDB Deadlocks | Count InnoDB Deadlocks | Count InnoDB Deadlocks |
| Log All Client Commands (syslog) | Log All Client Commands (syslog) | Log All Client Commands (syslog) |
| Response Time Distribution | Response Time Distribution | Feature not implemented |
| Show Storage Engines | Show Storage Engines | Show Storage Engines |
| Show Lock Names | Show Lock Names | Show Lock Names |
| Process List | Process List | Process List |
| Misc. INFORMATION_SCHEMA Tables | Misc. INFORMATION_SCHEMA Tables | Misc. INFORMATION_SCHEMA Tables |
| Extended Show Engine InnoDB Status | Extended Show Engine InnoDB Status | Extended Show Engine InnoDB Status |
| Thread Based Profiling | Thread Based Profiling | Thread Based Profiling |
| XtraDB Performance Improvements for I/O-Bound Highly-Concurrent Workloads | XtraDB Performance Improvements for I/O-Bound Highly-Concurrent Workloads | XtraDB Performance Improvements for I/O-Bound Highly-Concurrent Workloads |
| Page cleaner thread tuning | Page cleaner thread tuning | Page cleaner thread tuning |
| Statement Timeout | Statement Timeout | Statement Timeout |
| Extended SELECT INTO OUTFILE/DUMPFILE | Extended SELECT INTO OUTFILE/DUMPFILE | Extended SELECT INTO OUTFILE/DUMPFILE |
| Per-query variable statement | Per-query variable statement | Per-query variable statement |
| Extended mysqlbinlog | Extended mysqlbinlog | Extended mysqlbinlog |
| Slow Query Log Rotation and Expiration | Slow Query Log Rotation and Expiration | Slow Query Log Rotation and Expiration |
| Metrics for scalability measurement | Metrics for scalability measurement | Feature not implemented |
| Audit Log | Audit Log | Audit Log |
| Backup Locks | Backup Locks | Backup Locks |
| CSV engine mode for standard-compliant quote and comma parsing | CSV engine mode for standard-compliant quote and comma parsing | CSV engine mode for standard-compliant quote and comma parsing |
| Super read-only | Super read-only | Super read-only |

## Other Reading

- Changed in Percona Server 5.6
- *Percona Server for MySQL In-Place Upgrading Guide: From 5.7 to 8.0*
- What Is New in MySQL 5.5
- What Is New in MySQL 5.6

CHAPTER

THREE

# *PERCONA SERVER FOR MYSQL* FEATURE COMPARISON

*Percona Server for MySQL* is an enhanced drop-in replacement for *MySQL*. With *Percona Server for MySQL*,

- Your queries will run faster and more consistently.
- You will consolidate servers on powerful hardware.
- You will delay sharding, or avoid it entirely.
- You will save money on hosting fees and power.
- You will spend less time tuning and administering.
- You will achieve higher uptime.
- You will troubleshoot without guesswork.

We provide these benefits by significantly enhancing *Percona Server for MySQL* as compared to the standard *MySQL* database server:

| Features | *Percona Server for MySQL* 8.0.13 | MySQL 8.0.13 |
|---|---|---|
| Open source | Yes | Yes |
| ACID Compliance | Yes | Yes |
| Multi-Version Concurrency Control | Yes | Yes |
| Row-Level Locking | Yes | Yes |
| Automatic Crash Recovery | Yes | Yes |
| Table Partitioning | Yes | Yes |
| Views | Yes | Yes |
| Subqueries | Yes | Yes |
| Triggers | Yes | Yes |
| Stored Procedures | Yes | Yes |
| Foreign Keys | Yes | Yes |
| Window Functions | Yes | Yes |
| Common Table Expressions | Yes | Yes |
| Geospatial Features (GIS, SPRS) | Yes | Yes |
| GTID Replication | Yes | Yes |
| Group Replication | Yes | Yes |
| MyRocks Storage Engine | Yes | No |
| TokuDB Storage Engine | Yes | No |

## Improvements for Developers

| Feature | *Percona Server for MySQL* 8.0.13 | MySQL 8.0.13 |
|---|---|---|
| NoSQL Socket-Level Interface | Yes | Yes |
| X API Support | Yes | Yes |
| JSON Functions | Yes | Yes |
| InnoDB Full-Text Search Improvements | Yes | No |
| Extra Hash/Digest Functions | Yes | No |

## Extra Diagnostic Features

| Feature | *Percona Server for MySQL* 8.0.13 | MySQL 8.0.13 |
|---|---|---|
| INFORMATION_SCHEMA Tables | 95 | 65 |
| Global Performance and Status Counters | 853 | 434 |
| Optimizer Histograms | Yes | Yes |
| Per-Table Performance Counters | Yes | No |
| Per-Index Performance Counters | Yes | No |
| Per-User Performance Counters | Yes | No |
| Per-Client Performance Counters | Yes | No |
| Per-Thread Performance Counters | Yes | No |
| Enhanced SHOW ENGINE INNODB STATUS | Yes | No |
| Temporary tables Information | Yes | No |
| Extended Slow Query Logging | Yes | No |
| User Statistics | Yes | No |

## Performance & Scalability Enhancements

| Feature | *Percona Server for MySQL* 8.0.13 | MySQL 8.0.13 |
|---|---|---|
| InnoDB Resource Groups | Yes | Yes |
| Configurable Page Sizes | Yes | Yes |
| Contention-Aware Transaction Scheduling | Yes | Yes |
| Improved Scalability by Splitting Mutexes | Yes | Yes |
| Improved MEMORY Storage Engine | Yes | No |
| Improved Flushing | Yes | No |
| Parallel Doublewrite Buffer | Yes | No |
| Configurable Fast Index Creation | Yes | No |
| Per-Column Compression for VARCHAR/BLOB and JSON | Yes | No |
| Compressed Columns with Dictionaries | Yes | No |

## Security Features

| Feature | *Percona Server for MySQL* 8.0.13 | MySQL 8.0.13 |
|---|---|---|
| SQL Roles | Yes | Yes |
| SHA-2 Based Password Hashing | Yes | Yes |
| Password Rotation Policy | Yes | Yes |
| PAM Authentication | Yes | Enterprise Only |
| Audit Logging Plugin | Yes | Enterprise Only |

## Encryption Features

| Feature | *Percona Server for MySQL* 8.0.13 | MySQL 8.0.13 |
|---|---|---|
| Storing Keyring in a File | Yes | Yes |
| Storing Keyring in Hashicorp Vault | Yes | No |
| Encrypt InnoDB Data | Yes | Yes |
| Encrypt InnoDB Logs | Yes | Yes |
| Encrypt Built-in InnoDB Tablespaces (General, System, Undo, Temp) | Yes | No |
| Encrypt Binary Logs | Yes | No |
| Encrypt Temporary Files | Yes | No |
| Key Rotation with Scrubbing | Yes | No |
| Enforce Encryption | Yes | No |

## Operational Improvements

| Feature | *Percona Server for MySQL* 8.0.13 | MySQL 8.0.13 |
|---|---|---|
| Atomic DDL | Yes | Yes |
| Transactional Data Dictionary | Yes | Yes |
| Instant DDL | Yes | Yes |
| SET PERSIST | Yes | Yes |
| Invisible Indexes | Yes | Yes |
| Changed Page Tracking | Yes | No |
| Threadpool | Yes | Enterprise Only |
| Backup Locks | Yes | Yes |
| Extended SHOW GRANTS | Yes | No |
| Improved Handling of Corrupted Tables | Yes | No |
| Ability to Kill Idle Transactions | Yes | No |
| Improvements to START TRANSACTION WITH CONSISTENT SNAPSHOT | Yes | No |

CHAPTER

FOUR

# CHANGED IN PERCONA SERVER 8.0

*Percona Server for MySQL* 8.0 is based on *MySQL* 8.0 and incorporates many of the improvements found in *Percona Server for MySQL* 8.0.

## Features ported to *Percona Server for MySQL* 8.0 from *Percona Server for MySQL* 5.7

The following features have been ported to *Percona Server for MySQL* 8.0 from *Percona Server for MySQL* 5.7:

### SHOW ENGINE INNODB STATUS Extensions

- The Redo Log state
- Specifying the InnoDB buffer pool sizes in bytes
- `innodb_print_lock_wait_timeout_info` system variable

### Performance

- *Prefix Index Queries Optimization*
- *Multiple page asynchronous I/O requests*
- *Thread Pool*
- *Priority refill for the buffer pool free list*
- *Multi-threaded LRU flusher*
- *Parallel doublewrite buffer*

### Flexibility

- *InnoDB Full-Text Search Improvements*
- *Improved MEMORY Storage Engine*
- *Extended mysqldump*
- *Extended SELECT INTO OUTFILE/DUMPFILE*
- *Support for PROXY protocol*

- *Compressed columns with dictionaries*

### Management

- *Percona Toolkit UDFs*
- *Kill Idle Transactions*
- *XtraDB changed page tracking*
- *PAM Authentication Plugin*
- *Expanded Fast Index Creation*
- *Backup Locks*
- *Audit Log Plugin*
- *Start transaction with consistent snapshot*
- *Extended SHOW GRANTS*
- *Transparent Data Encryption*

### Reliability

- *Handle Corrupted Tables*
- *Too Many Connections Warning*

### Diagnostics

- *User Statistics*
- *Slow Query Log*
- *Show Storage Engines*
- *Process List*
- *INFORMATION_SCHEMA.[GLOBAL_]TEMP_TABLES*
- *Thread Based Profiling*
- *InnoDB Page Fragmentation Counters*

## Features removed from *Percona Server for MySQL* 8.0

Some features, that were present in *Percona Server for MySQL* 5.7, are removed from *Percona Server for MySQL* 8.0:

### Removed Features

- Slow Query Log Rotation and Expiration
- CSV engine mode for standard-compliant quote and comma parsing
- Utility user

- Expanded program option modifiers
- The `ALL_O_DIRECT` InnoDB flush method: it is not compatible with the new redo logging implementation
- XTRADB_RSEG table from INFORMATION_SCHEMA
- InnoDB memory size information from SHOW ENGINE INNODB STATUS; the same information is available from Performance Schema memory summary tables
- Query cache enhancements

**See also:**

***MySQL* Documentation: Performance Schema Table Description** https://dev.mysql.com/doc/refman/8.0/en/performance-schema-table-descriptions.html

## Removed Syntax

- The `SET STATEMENT ... FOR ...` statement that enabled setting a variable for a single query. For more information see *Replacing SET STATEMENT FOR with the Upstream Equivalent*.
- The `LOCK BINLOG FOR BACKUP` statement due to the introduction of the `log_status` table in Performance Schema of *MySQL* 8.0.

## Removed Plugins

- `SCALABILITY_METRICS`
- `QUERY_RESPONSE_TIME` plugins

The `QUERY_RESPONSE_TIME` plugins have been removed from *Percona Server for MySQL* 8.0 as the Performance Schema of *MySQL* 8.0 provides histogram data for statement execution time.

**See also:**

***MySQL* Documentation: Statement Histogram Summary Tables** https://dev.mysql.com/doc/refman/8.0/en/statement-histogram-summary-tables.html

## Removed System variables

- The innodb_use_global_flush_log_at_trx_commit system variable which enabled setting the global *MySQL* variable innodb_flush_log_at_trx_commit
- pseudo_server_id
- max_slowlog_files
- max_slowlog_size
- innodb_show_verbose_locks: showed the records locked in `SHOW ENGINE INNODB STATUS`
- NUMA support in mysqld_safe
- innodb_kill_idle_trx which was an alias to the `kill_idle_trx` system variable
- The max_binlog_files system variable

# Part II

# Installation

CHAPTER

FIVE

# INSTALLING *PERCONA SERVER FOR MYSQL* 8.0.18-9

This page provides the information on how to you can install *Percona Server for MySQL*. Following options are available:

- *Installing Percona Server for MySQL from Repositories* (recommended)
- Installing *Percona Server for MySQL* from Downloaded *rpm* or *apt* Packages
- *Installing Percona Server for MySQL from a Binary Tarball*
- *Installing Percona Server for MySQL from a Source Tarball*
- *Installing Percona Server for MySQL from the Git Source Tree*
- *Compiling Percona Server for MySQL from Source*
- *Running Percona Server for MySQL in a Docker Container*

Before installing, you might want to read the *Percona Server for MySQL 8.0 Release notes*.

## Installing *Percona Server for MySQL* from Repositories

*Percona* provides repositories for **yum** (`RPM` packages for *Red Hat*, *CentOS* and *Amazon Linux AMI*) and **apt** (`.deb` packages for *Ubuntu* and *Debian*) for software such as *Percona Server for MySQL*, *Percona XtraBackup*, and *Percona Toolkit*. This makes it easy to install and update your software and its dependencies through your operating system's package manager. This is the recommended way of installing where possible.

Following guides describe the installation process for using the official Percona repositories for `.deb` and `.rpm` packages.

### Installing *Percona Server for MySQL* on *Debian* and *Ubuntu*

Ready-to-use packages are available from the *Percona Server for MySQL* software repositories and the Percona downloads page.

Supported Releases:

- Debian 9.0 (stretch)
- Ubuntu 16.04 (xenial)
- Ubuntu 18.04 LTS (bionic)

Supported Platforms:

- x86_64 (also known as `amd64`)

### What's in each DEB package?

| Package | Contains |
|---|---|
| percona-server-server | The database server itself, the `mysqld` binary and associated files. |
| percona-server-common | The files common to the server and client. |
| percona-server-client | The command line client. |
| percona-server-dbg | Debug symbols for the server. |
| percona-server-test | The database test suite. |
| percona-server-source | The server source. |
| libperconaserverclient21-dev | Header files needed to compile software to use the client library. |
| libperconaserver-client21 | The client shared library. The version is incremented when there is an ABI change that requires software using the client library to be recompiled or its source code modified. |

### Installing *Percona Server for MySQL* from Percona `apt` repository

Run the following commands as root or by using the **sudo** command

1. Fetch the repository packages from Percona web:

```
$ wget https://repo.percona.com/apt/percona-release_latest.$(lsb_release -sc)_all.
↪deb
```

2. Install the downloaded package with **dpkg**. To do that, run the following commands as root or with **sudo**:

```
$ sudo dpkg -i percona-release_latest.$(lsb_release -sc)_all.deb
```

3. Once you install this package the Percona repositories should be added. You can check the repository setup in the `/etc/apt/sources.list.d/percona-release.list` file.

4. Enable the repository:

```
$ sudo percona-release setup ps80
```

5. After that you can install the server package:

```
$ sudo apt-get install percona-server-server
```

**Note:** Percona Server for MySQL 8.0 comes with the *TokuDB storage engine* and *MyRocks* storage engines. These storage engines are installed as plugins. You can find more information on how to install and enable the *TokuDB* storage in the *TokuDB Installation* guide. More information about how to install *MyRocks* can be found in the section *Percona MyRocks Installation Guide*.

### Percona `apt` Testing repository

Percona offers pre-release builds from the testing repository. To enable it, run **`percona-release`** with the `testing` argument. Run this command as root or by using the **`sudo`** command.

```
$ sudo percona-release enable ps80 testing
```

### Apt-Pinning the packages

In some cases you might need to "pin" the selected packages to avoid the upgrades from the distribution repositories. You'll need to make a new file `/etc/apt/preferences.d/00percona.pref` and add the following lines in it:

```
Package: *
Pin: release o=Percona Development Team
Pin-Priority: 1001
```

For more information about the pinning you can check the official debian wiki.

### Installing *Percona Server for MySQL* using downloaded deb packages

Download the packages of the desired series for your architecture from the Percona downloads page. The easiest way is to download bundle which contains all the packages. The following example will download *Percona Server for MySQL* *8.0.13-3* release packages for Debian 9.0 (stretch):

```
$ wget https://www.percona.com/downloads/Percona-Server-8.0/Percona-Server-8.0.13-3/
↪binary/debian/stretch/x86_64/percona-server-8.0.13-3-r63dafaf-stretch-x86_64-bundle.
↪tar
```

You should then unpack the bundle to get the packages:

```
$ tar xvf percona-server-8.0.13-3-r63dafaf-stretch-x86_64-bundle.tar
```

After you unpack the bundle you should see the following packages:

```
$ ls *.deb
```

**Output**

```
libperconaserverclient21-dev_8.0.13-3-1.stretch_amd64.deb
libperconaserverclient21_8.0.13-3-1.stretch_amd64.deb
percona-server-dbg_8.0.13-3-1.stretch_amd64.deb
percona-server-client_8.0.13-3-1.stretch_amd64.deb
percona-server-common_8.0.13-3-1.stretch_amd64.deb
percona-server-server_8.0.13-3-1.stretch_amd64.deb
percona-server-source_8.0.13-3-1.stretch_amd64.deb
percona-server-test_8.0.13-3-1.stretch_amd64.deb
percona-server-tokudb_8.0.13-3-1.stretch_amd64.deb
```

Now, you can install *Percona Server for MySQL* using **`dpkg`**. Run this command as root or by using the **`sudo`** command

```
$ sudo dpkg -i *.deb
```

This will install all the packages from the bundle. Another option is to download/specify only the packages you need for running *Percona Server for MySQL* installation (`libperconaserverclient21_8.0.13-3-1.stretch_amd64.deb`, `percona-server-client_8.0.13-3-1.stretch_amd64.deb`, `percona-server-common_8.0.13-3-1.stretch_amd64.deb`, and `percona-server-server_8.0.13-3-1.stretch_amd64.deb`. Optionally, you can install `percona-server-tokudb_8.0.13-3-1.stretch_amd64.deb` if you want the *TokuDB* storage engine).

**Note:** *Percona Server for MySQL* 8.0 comes with the *TokuDB storage engine*. You can find more information on how to install and enable the *TokuDB* storage in the *TokuDB Installation* guide.

**Warning:** When installing packages manually like this, you'll need to make sure to resolve all the dependencies and install missing packages yourself. Following packages will need to be installed before you can manually install Percona Server: `mysql-common`, `libjemalloc1`, `libaio1` and `libmecab2`

### Running *Percona Server for MySQL*

*Percona Server for MySQL* stores the data files in `/var/lib/mysql/` by default. You can find the configuration file that is used to manage *Percona Server for MySQL* in `/etc/mysql/my.cnf`.

**Note:**

***Debian* and *Ubuntu* installation doesn't automatically create a special** `debian-sys-maint` user which can be used by the control scripts to control the *Percona Server for MySQL* `mysqld` and `mysqld_safe` services like it was the case with previous *Percona Server for MySQL* versions. If you still require this user you'll need to create it manually.

Run the following commands as root or by using the **sudo** command

1. Starting the service

   *Percona Server for MySQL* is started automatically after it gets installed unless it encounters errors during the installation process. You can also manually start it by running: `service mysql start`

2. Confirming that service is running. You can check the service status by running: `service mysql status`

3. Stopping the service

   You can stop the service by running: `service mysql stop`

4. Restarting the service. `service mysql restart`

**Note:** Debian 9.0 (stretch) and Ubuntu 18.04 LTS (bionic) come with systemd as the default system and service manager. You can invoke all the above commands with `systemctl` instead of `service`. Currently both are supported.

### Uninstalling *Percona Server for MySQL*

To uninstall *Percona Server for MySQL* you'll need to remove all the installed packages. Removing packages with **apt-get remove** will leave the configuration and data files. Removing the p ackages with **apt-get purge** will remove all the packages with configuration files and data files (all the databases). Depending on your needs you can choose which command better suits you.

1. Stop the *Percona Server for MySQL* service: `service mysql stop`
2. Remove the packages
   (a) Remove the packages. This will leave the data files (databases, tables, logs, configuration, etc.) behind. In case you don't need them you'll need to remove them manually: `apt-get remove percona-server*`
   (b) Purge the packages. **NOTE**: This will remove all the packages and delete all the data files (databases, tables, logs, etc.): `apt-get purge percona-server*`

## Installing *Percona Server for MySQL* on Red Hat Enterprise Linux and CentOS

Ready-to-use packages are available from the *Percona Server for MySQL* software repositories and the download page. The *Percona* **yum** repository supports popular *RPM*-based operating systems, including the *Amazon Linux AMI*.

The easiest way to install the *Percona Yum* repository is to install an *RPM* that configures **yum** and installs the Percona GPG key.

Supported Releases:

- *CentOS* 6 and *RHEL* 6 (Current Stable)
- *CentOS* 7 and *RHEL* 7
- *RHEL* 8
- *Amazon Linux AMI* (works the same as *CentOS* 6)
- *Amazon Linux* 2

---

**Important:** "Current Stable": We support only the current stable RHEL6/CentOS6 release, because there is no official (i.e. RedHat provided) method to support or download the latest OpenSSL on RHEL/CentOS versions prior to 6.5. Similarly, and also as a result thereof, there is no official Percona way to support the latest builds of *Percona Server for MySQL* on RHEL/CentOS versions prior to 6.5. Additionally, many users will need to upgrade to OpenSSL 1.0.1g or later (due to the Heartbleed vulnerability), and this OpenSSL version is not available for download from any official RHEL/Centos repository for versions 6.4 and prior. For any officially unsupported system, src.rpm packages may be used to rebuild *Percona Server for MySQL* for any environment. Please contact our support service if you require further information on this.

---

The *CentOS* repositories should work well with *Red Hat Enterprise Linux* too, provided that **yum** is installed on the server.

---

**Important:**

*CentOS* 6 offers an outdated version of the `curl` library required by the keyring Vault plugin of *Percona Server for MySQL*. The version of the `curl` library in *CentOS* 6, which depends on the `nss` library, is known to create memory corruption issues. This bug is registered in Red Hat Bugzilla. Its current status is *CLOSED WONTFIX*.

If you intend to use the keyring Vault plugin of *Percona Server for MySQL* make sure that you use the latest version of the `curl` library. We recommend that you build it from source configuring with `ssl` but without `nss`:

```
$ ./configuration --with-ssl --without-nss --prefix=<INSTALATION DIRECTORY>
```

As soon as you install `curl`, make sure that *Percona Server for MySQL* will use this version.

**See also:**

**How to install curl and libcurl** https://curl.haxx.se/docs/install.html

Supported Platforms:

- x86_64 (also known as `amd64`)

### What's in each RPM package?

Each of the *Percona Server for MySQL* RPM packages have a particular purpose.

| Package | Contains |
|---|---|
| percona-server-server | The server itself (the `mysqld` binary) |
| percona-server-debuginfo | Debug symbols for the server |
| percona-server-client | The command line client |
| percona-server-devel | the header files needed to compile software using the client library. |
| percona-server-shared | The client shared library. |
| percona-server-shared-compat | Shared libraries for software compiled against old versions of the client library. The following libraries are included in this package: `libmysqlclient.so.12`, `libmysqlclient.so.14`, `libmysqlclient.so.15`, `libmysqlclient.so.16`, and `libmysqlclient.so.18`. |
| percona-server-test | package includes the test suite for *Percona Server for MySQL*. |

### Installing *Percona Server for MySQL* from Percona `yum` repository

Please add sudo to percona-release setup and yum install commands

Run the following commands as root or by using the **sudo** command

1. Install the Percona repository

   You can install Percona yum repository by running the following command as a `root` user or with sudo:

   ```
   $ sudo yum install https://repo.percona.com/yum/percona-release-latest.noarch.rpm
   ```

   You should see some output such as the following:

   ```
   Retrieving http://www.percona.com/downloads/percona-release/redhat/0.1-6/percona-
   ↪release-latest.noarch.rpm
   Preparing...                ########################################### [100%]
   1:percona-release          ########################################### [100%]
   ```

2. Enable the repository:

```
$ sudo percona-release setup ps80
```

3. Install the packages

   You can now install *Percona Server for MySQL* by running:

```
$ sudo yum install percona-server-server
```

**Note:** *Percona Server for MySQL* 8.0 comes with the *TokuDB storage engine* and *MyRocks* storage engines. These storage engines are installed as plugins. You can find more information on how to install and enable the *TokuDB* storage in the *TokuDB Installation* guide. More information about how to install *MyRocks* can be found in the section *Percona MyRocks Installation Guide*.

### Percona *yum* Testing repository

Percona offers pre-release builds from our testing repository. To subscribe to the testing repository, you'll need to enable the testing repository in `/etc/yum.repos.d/percona-release.repo`. To do so, set both `percona-testing-$basearch` and `percona-testing-noarch` to `enabled = 1` (Note that there are 3 sections in this file: release, testing and experimental - in this case it is the second section that requires updating). **NOTE:** You'll need to install the Percona repository first (ref above) if this hasn't been done already.

### Installing *Percona Server for MySQL* using downloaded rpm packages

1. Download the packages of the desired series for your architecture from the download page. The easiest way is to download bundle which contains all the packages. Following example will download *Percona Server for MySQL* 8.0.13-3 release packages for *CentOS* 7:

```
$ wget https://www.percona.com/downloads/Percona-Server-8.0/Percona-Server-8.0.13-
↪3/binary/redhat/7/x86_64/Percona-Server-8.0.13-3-r63dafaf-el7-x86_64-bundle.tar
```

2. You should then unpack the bundle to get the packages: `tar xvf Percona-Server-8.0.13-3-r63dafaf-el7-x86_64-bundle.tar`

   After you unpack the bundle you should see the following packages when running `ls *.rpm`:

   **Output**

```
percona-server-80-debuginfo-8.0.13-3.el7.x86_64.rpm
percona-server-client-80-8.0.13-3.el7.x86_64.rpm
percona-server-devel-80-8.0.13-3.el7.x86_64.rpm
percona-server-server-80-8.0.13-3.el7.x86_64.rpm
percona-server-shared-80-8.0.13-3.el7.x86_64.rpm
percona-server-shared-compat-80-8.0.13-3.el7.x86_64.rpm
percona-server-test-80-8.0.13-3.el7.x86_64.rpm
percona-server-tokudb-80-8.0.13-3.el7.x86_64.rpm
```

   **Note:** For an RHEL 8 package installation, Percona Server requires the mysql module to be disabled.

```
$ sudo yum module disable mysql
```

3. Now you can install *Percona Server for MySQL* 8.0 by running:

```
$ sudo rpm -ivh percona-server-server-80-8.0.13-3.el7.x86_64.rpm \
percona-server-client-80-8.0.13-3.el7.x86_64.rpm \
percona-server-shared-80-8.0.13-3.el7.x86_64.rpm
```

This will install only packages required to run the *Percona Server for MySQL* 8.0. Optionally you can install *TokuDB* storage engine by adding the `percona-server-tokudb-80-8.0.13-3.el7.x86_64.rpm` to the command above. You can find more information on how to install and enable the *TokuDB* storage in the *TokuDB Installation* guide.

To install all the packages (for debugging, testing, etc.) you should run:

```
$ sudo rpm -ivh *.rpm
```

**Note:** When installing packages manually like this, you'll need to make sure to resolve all the dependencies and install missing packages yourself.

### Running *Percona Server for MySQL*

*Percona Server for MySQL* stores the data files in `/var/lib/mysql/` by default. You can find the configuration file that is used to manage *Percona Server for MySQL* in `/etc/my.cnf`.

1. Starting the service

   *Percona Server for MySQL* is not started automatically on *RHEL* and *CentOS* after it gets installed. You should start it by running:

```
$ sudo service mysql start
```

2. Confirming that service is running

   You can check the service status by running:

```
$ sudo service mysql status
```

3. Stopping the service

   You can stop the service by running:

```
$ sudo service mysql stop
```

4. Restarting the service

   You can restart the service by running:

```
$ sudo service mysql restart
```

**Note:** *RHEL* 7 and *CentOS* 7 come with systemd as the default system and service manager so you can invoke all the above commands with `sytemctl` instead of `service`. Currently both are supported.

### Uninstalling *Percona Server for MySQL*

To completely uninstall *Percona Server for MySQL* you'll need to remove all the installed packages and data files.

1. Stop the *Percona Server for MySQL* service: `service mysql stop`
2. Remove the packages:

```
$ sudo yum remove percona-server*
```

3. Remove the data and configuration files

```
rm -rf /var/lib/mysql
rm -f /etc/my.cnf
```

**Warning:** This will remove all the packages and delete all the data files (databases, tables, logs, etc.), you might want to take a backup before doing this in case you need the data.

## Installing *Percona Server for MySQL* from a Binary Tarball

*Percona Server for MySQL* offers multiple tarballs depending on the *OpenSSL* library available in the distribution:

- ssl100 - for *Debian* prior to 9 and *Ubuntu* prior to 14.04 versions (`libssl.so.1.0.0 => /usr/lib/x86_64-linux-gnu/libssl.so.1.0.0 (0x00007f2e389a5000)`);
- ssl101 - for *CentOS* 6 and *CentOS* 7 (`libssl.so.10 => /usr/lib64/libssl.so.10 (0x00007facbe8c4000)`);
- ssl102 - for *Debian* 9 and *Ubuntu* versions starting from 14.04 (`libssl.so.1.1 => /usr/lib/libssl.so.1.1 (0x00007f5e57397000)`;

You can download the binary tarballs from the `Linux - Generic` section on the download page.

Fetch and extract the correct binary tarball. For example for *Debian Wheezy*:

```
$ wget https://www.percona.com/downloads/Percona-Server-LATEST/Percona-Server-8.0.13-
↪3/binary/tarball/Percona-Server-8.0.13-3-Linux.x86_64.ssl102.tar.gz
```

## Installing *Percona Server for MySQL* from a Source Tarball

Fetch and extract the source tarball. For example:

```
$ wget https://www.percona.com/downloads/Percona-Server-LATEST/Percona-Server-8.0.13-
↪3/source/tarball/Percona-Server-8.0.13-3-Linux.x86_64.ssl102.tar.gz
$ tar xfz percona-server-8.0.13-3.tar.gz
```

Next, follow the instructions in *Compiling Percona Server for MySQL from Source* below.

## Installing *Percona Server for MySQL* from the Git Source Tree

Percona uses the Github revision control system for development. To build the latest *Percona Server for MySQL* from the source tree you will need `git` installed on your system.

You can now fetch the latest *Percona Server for MySQL* 8.0 sources.

```
$ git clone https://github.com/percona/percona-server.git
$ cd percona-server
$ git checkout 8.0
$ git submodule init
$ git submodule update
```

If you are going to be making changes to *Percona Server for MySQL* 8.0 and wanting to distribute the resulting work, you can generate a new source tarball (exactly the same way as we do for release):

```
$ cmake .
$ make dist
```

Next, follow the instructions in *Compiling Percona Server for MySQL from Source* below.

## Compiling *Percona Server for MySQL* from Source

After either fetching the source repository or extracting a source tarball (from Percona or one you generated yourself), you will now need to configure and build *Percona Server for MySQL*.

**Important:** Make sure that `gcc` installed on your system is at least of a version in the 4.9 release series.

First, run cmake to configure the build. Here you can specify all the normal build options as you do for a normal *MySQL* build. Depending on what options you wish to compile *Percona Server for MySQL* with, you may need other libraries installed on your system. Here is an example using a configure line similar to the options that Percona uses to produce binaries:

```
$ cmake . -DCMAKE_BUILD_TYPE=RelWithDebInfo -DBUILD_CONFIG=mysql_release -DFEATURE_
↪SET=community
```

Now, compile using make

```
$ make
```

Install:

```
$ make install
```

*Percona Server for MySQL* 8.0 will now be installed on your system.

## Building *Percona Server for MySQL* Debian/Ubuntu packages

If you wish to build your own Debian/Ubuntu (dpkg) packages of *Percona Server for MySQL*, you first need to start with a source tarball, either from the Percona website or by generating your own by following the instructions above( *Installing Percona Server for MySQL from the Git Source Tree*).

Extract the source tarball:

```
$ tar xfz Percona-Server-8.0.13-3-Linux.x86_64.ssl102.tar.gz
$ cd Percona-Server-8.0.13-3
```

Put the debian packaging in the directory that Debian expects it to be in:

```
$ cp -ap build-ps/debian debian
```

Update the changelog for your distribution (here we update for the unstable distribution - sid), setting the version number appropriately. The trailing one in the version number is the revision of the Debian packaging.

```
$ dch -D unstable --force-distribution -v "8.0.13-3-1" "Update to 8.0.13-3"
```

Build the Debian source package:

```
$ dpkg-buildpackage -S
```

Use sbuild to build the binary package in a chroot:

```
$ sbuild -d sid percona-server-8.0_8.0.13-3-1.dsc
```

You can give different distribution options to `dch` and `sbuild` to build binary packages for all Debian and Ubuntu releases.

**Note:** *PAM Authentication Plugin* is not built with the server by default. In order to build the *Percona Server for MySQL* with PAM plugin, additional option `-DWITH_PAM=ON` should be used.

## Running *Percona Server for MySQL* in a Docker Container

Docker images of *Percona Server for MySQL* are hosted publicly on Docker Hub at https://hub.docker.com/r/percona/percona-server/.

For more information about using Docker, see the Docker Docs.

**Note:** Make sure that you are using the latest version of Docker. The ones provided via `apt` and `yum` may be outdated and cause errors.

By default, Docker will pull the image from Docker Hub if it is not available locally.

### Using the *Percona Server for MySQL* Images

The following procedure describes how to run and access Percona Server 8.0 using Docker.

### Starting an Instance of *Percona Server for MySQL* in a Container

To start a container named `ps` running the latest version of *Percona Server for MySQL* 8.0, with the root password set to `root`:

```
[root@docker-host] $ docker run -d \
  --name ps \
  -e MYSQL_ROOT_PASSWORD=root \
  percona/percona-server:8.0
```

**Important:** `root` is not a secure password.

### Accessing the *Percona Server for MySQL* Container

To access the shell in the container:

```
[root@docker-host] $ docker exec -it ps /bin/bash
```

From the shell, you can view the error log:

```
[mysql@ps] $ more /var/log/mysql/error.log
2017-08-29T04:20:22.190474Z 0 [Warning] 'NO_ZERO_DATE', 'NO_ZERO_IN_DATE' and 'ERROR_
↪FOR_DIVISION_BY_ZERO' sql modes should be used with strict mode. They will be␣
↪merged with strict mode in a future release.
2017-08-29T04:20:22.190520Z 0 [Warning] 'NO_AUTO_CREATE_USER' sql mode was not set.
...
```

You can also run the MySQL command-line client to access the database directly:

```
[mysql@ps] $ mysql -uroot -proot
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4
Server version: 8.0.13-3 Percona Server (GPL), Release '17', Revision 'e19a6b7b73f'

Copyright (c) 2009-2017 Percona LLC and/or its affiliates
Copyright (c) 2000, 2017, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other␣
↪names may be trademarks of their respective owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

### Accessing *Percona Server for MySQL* from Application in Another Container

The image exposes the standard MySQL port 3306, so container linking makes Percona Server instance available from other containers. To link a container running your application (in this case, from image named `app/image`) with the Percona Server container, run it with the following command:

```
[root@docker-host] $ docker run -d \
  --name app \
  --link ps \
  app/image:latest
```

This application container will be able to access the Percona Server container via port 3306.

### Environment Variables

When running a Docker container with Percona Server, you can adjust the configuration of the instance by passing one or more environment variables with the `docker run` command.

---

**Note:** These variables will not have any effect if you start the container with a data directory that already contains a database: any pre-existing database will always remain untouched on container startup.

---

The variables are optional, except that you must specify at least one of the following:

- *MYSQL_ALLOW_EMPTY_PASSWORD*: least secure, use only for testing.
- *MYSQL_ROOT_PASSWORD*: more secure, but setting the password on the command line is not recommended for sensitive production setups.
- *MYSQL_RANDOM_ROOT_PASSWORD*: most secure, recommended for production.

---

**Note:** To further secure your instance, use the *MYSQL_ONETIME_PASSWORD* variable if you are running version 5.6 or later.

---

**variable MYSQL_ALLOW_EMPTY_PASSWORD**
Specifies whether to allow the container to be started with a blank password for the MySQL root user. Disabled by default. To enable, set `MYSQL_ALLOW_EMPTY_PASSWORD=yes`.

---

**Note:** Allowing empty root password is not recommended for production, because anyone will have full superuser access to the database.

---

**variable MYSQL_DATABASE**
Specifies the name of the database to be created when running the container. To create a user with full access to this database (`GRANT ALL`), set the *MYSQL_USER* and *MYSQL_PASSWORD* variables.

**variable MYSQL_ONETIME_PASSWORD**
Specifies whether the password for the MySQL root user should be set as expired. Disabled by default. If enabled using `MYSQL_ONETIME_PASSWORD=yes`, the MySQL root password must be changed before using it to log in.

**variable MYSQL_PASSWORD**
Specifies the password for the user with full access to the database specified by the *MYSQL_DATABASE* variable. Setting the *MYSQL_USER* variable is also required.

**variable MYSQL_RANDOM_ROOT_PASSWORD**
Specifies whether a random password for the MySQL root user should be generated. Disabled by default. To enable, set `MYSQL_RANDOM_ROOT_PASSWORD=yes`.

The password will be printed to `stdout` in the container, and it can be viewed using the `docker logs` command.

**variable MYSQL_ROOT_PASSWORD**
Specifies the password for the MySQL root user.

---

**Note:** Setting the MySQL root password on the command line is insecure. It is recommended to set a random password using the *MYSQL_RANDOM_ROOT_PASSWORD* variable.

---

**variable MYSQL_ROOT_PASSWORD_FILE**
Specifies a file that will be read for the root user account. This can be a mounted file when you run your container. This can also be used in the scope of the Docker Secrets (Swarm mode) functionality.

**variable MYSQL_USER**
Specifies the name for the user with full access to the database specified by the *MYSQL_DATABASE* variable. Setting the *MYSQL_PASSWORD* variable is also required.

**variable INIT_TOKUDB**
Specifies whether to allow the container to be started with enabled TokuDB engine. Disabled by default. To enable, set `INIT_TOKUDB=yes`.

**variable `INIT_ROCKSDB`**
: Specifies whether to allow the container to be started with enabled RocksDB engine. Disabled by default. To enable, set `INIT_ROCKSDB=yes`.

### Storing Data

There are two ways to store data used by applications that run in Docker containers:

- Let Docker manage the storage of your data by writing the database files to disk on the host system using its own internal volume management.
- Create a data directory on the host system (outside the container on high performance storage) and mount it to a directory visible from inside the container. This places the database files in a known location on the host system, and makes it easy for tools and applications on the host system to access the files. The user should make sure that the directory exists, and that permissions and other security mechanisms on the host system are set up correctly.

For example, if you create a data directory on a suitable volume on your host system named `/local/datadir`, you run the container with the following command:

```
[root@docker-host] $ docker run -d \
  --name ps \
  -e MYSQL_ROOT_PASSWORD=root \
  -v /local/datadir:/var/lib/mysql \
  percona/percona-server:8.0
```

The `-v /local/datadir:/var/lib/mysql` option mounts the `/local/datadir` directory on the host to `/var/lib/mysql` in the container, which is the default data directory used by *Percona Server for MySQL*.

---

**Note:** If you the Percona Server container instance with a data directory that already contains data (the `mysql` subdirectory where all our system tables are stored), the *`MYSQL_ROOT_PASSWORD`* variable should be omitted from the `docker run` command.

---

---

**Note:** If you have SELinux enabled, assign the relevant policy type to the new data directory, so that the container will be allowed to access it:

```
[root@docker-host] $ chcon -Rt svirt_sandbox_file_t /local/datadir
```

---

### Port Forwarding

Docker allows mapping ports on the container to ports on the host system using the `-p` option. If you run the container with this option, you can connect to the database by connecting your client to a port on the host machine. This can greatly simplify consolidating many instances to a single host.

To map the standard MySQL port 3306 to port 6603 on the host:

```
[root@docker-host] $ docker run -d \
 --name ps \
 -e MYSQL_ROOT_PASSWORD=root \
 -p 6603:3306 \
 percona/percona-server:8.0
```

### Passing Options to *Percona Server for MySQL*

You can pass options to *Percona Server for MySQL* when running the container by appending them to the `docker run` command. For example, to start run *Percona Server for MySQL* with UTF-8 as the default setting for character set and collation for all databases:

```
[root@docker-host] $ docker run -d \
 --name ps \
 -e MYSQL_ROOT_PASSWORD=root \
 percona/percona-server:8.0 \
 --character-set-server=utf8 \
 --collation-server=utf8_general_ci
```

CHAPTER

SIX

# *PERCONA SERVER FOR MYSQL* IN-PLACE UPGRADING GUIDE: FROM 5.7 TO 8.0

In-place upgrades are those which are done using the existing data in the server. Generally speaking, this is stopping the server, installing the new server and starting it with the same data files. While they may not be suitable for high-complexity environments, they may be adequate for many scenarios.

The following is a summary of the more relevant changes in the 8.0 series. It is strongly recommended that you read the following guides as they contain the list of incompatible changes that could cause automatic upgrade to fail:

- changed_in_8.0
- Upgrading MySQL
- Upgrading from MySQL 5.7 to 8.0
- Upgrade Paths
- Preparing your Installation for Upgrade

Starting from release `8.0.15-5`, *Percona Server for MySQL* uses the upstream implementation of binary log encryption. The variable `encrypt_binlog` is removed and the related command line option `--encrypt_binlog` is not supported. It is important that you remove the `encrypt_binlog` variable from your configuration file before you attempt to upgrade either from another release in the *Percona Server for MySQL* 8.0 series or from *Percona Server for MySQL* 5.7. Otherwise, a server boot error will be produced reporting an unknown variable. The implemented binary log encryption is compatible with the old format: the binary log encrypted in a previous version of MySQL 8.0 series or Percona Server for MySQL are supported.

**See also:**

***MySQL* Documentation**

- Encrypting Binary Log Files and Relay Log Files
- binlog_encryption variable

**Warning:** Do not upgrade from 5.7 to 8.0 on a crashed instance. If the server instance has crashed, crash recovery should be run before proceeding with the upgrade.

Note that in *Percona Server for MySQL* 8.0, the `ROW FORMAT` clause is not supported in `CREATE TABLE` and `ALTER TABLE` statements. Instead, use the `tokudb_row_format` variable to set the default compression algorithm.

With partitioned tables that use the TokuDB or MyRocks storage engine, the upgrade only works with native partitioning.

# Upgrading using the Percona repositories

The easiest and recommended way of installing - where possible - is by using the *Percona* repositories.

Instructions for enabling the repositories in a system can be found in:

- *Percona APT Repository*
- *Percona YUM Repository*

## DEB-based distributions

Run the following commands as root or by using the **sudo** command

Having done the full backup (or dump if possible), stop the server running and proceed to do the modifications needed in your configuration file, as explained at the beginning of this guide.

---

**Note:** If you are running *Debian*/*Ubuntu* system with systemd as the default system and service manager you can invoke the above command with **systemctl** instead of **service**. Currently both are supported.

---

Then install the new server with:

Enable the repository:

```
$ percona-release enable ps-80 release
$ apt-get update
```

```
$ apt-get install percona-server-server
```

If you used or *TokuDB* or *MyRocks* storage engines

The *TokuDB* and *MyRocks* storage engines are installed separately. The `percona-server-tokudb` package installs both of them.

```
$ apt-get install percona-server-tokudb
```

If you only used the *MyRocks* storage engine in *Percona Server for MySQL* 5.7, install the `percona-server-rocksdb` package.

```
$ apt-get install percona-server-rocksdb
```

The installation script will *NOT* run automatically **mysql_upgrade** as it was the case in previous versions. You'll need to run the command manually and restart the service after it's finished.

```
$ mysql_upgrade

Checking if update is needed.
```

```
Checking server version.
Running queries to upgrade MySQL server.
Checking system database.
mysql.columns_priv                                 OK
mysql.db                                           OK
mysql.engine_cost                                  OK
...
Upgrade process completed successfully.
Checking if update is needed.

$ service mysql restart
```

## RPM-based distributions

Run the following commands as root or by using the **sudo** command

Having done the full backup (and dump if possible), stop the server: `service mysql stop` and check your installed packages with `rpm -qa | grep Percona-Server`

**Note:** If you're running *RHEL/CentOS* system with systemd as the default system and service manager you can invoke the above command with **systemctl** instead of **service**. Currently both are supported.

**Output of `rpm -qa | grep Percona-Server`**

```
Percona-Server-57-debuginfo-5.7.10-3.1.el7.x86_64
Percona-Server-client-57-5.7.10-3.1.el7.x86_64
Percona-Server-devel-57-5.7.10-3.1.el7.x86_64
Percona-Server-server-57-5.7.10-3.1.el7.x86_64
Percona-Server-shared-57-5.7.10-3.1.el7.x86_64
Percona-Server-shared-compat-57-5.7.10-3.1.el7.x86_64
Percona-Server-test-57-5.7.10-3.1.el7.x86_64
Percona-Server-tokudb-57-5.7.10-3.1.el7.x86_64
```

After checking, proceed to remove them without dependencies:

```
$ rpm -qa | grep Percona-Server | xargs rpm -e --nodeps
```

It is important that you remove them without dependencies as many packages may depend on these (as they replace `mysql`) and will be removed if omitted.

**Important:** `/etc/my.cnf` Backed Up in CentOS 7

In CentOS 7, the `/etc/my.cnf` configuration file is backed up when you uninstall the *Percona Server for MySQL* packages with the `rpm -e --nodeps` command.

The backup file is stored in the same directory with the *_backup* suffix followed by a timestamp: `etc/my.cnf_backup-20181201-1802`.

Substitute `grep '^mysql-'` for `grep 'Percona-Server'` in the previous command and remove the listed packages.

You will have to install the `percona-server-server` package:

```
$ yum install percona-server-server
```

The *TokuDB* and *MyRocks* storage engines are installed separately.

If you used *TokuDB* in *Percona Server for MySQL* 5.7, install the `percona-server-tokudb` package when doing the upgrade. This command installs both

```
$ yum install percona-server-tokudb
```

If you used the *MyRocks* storage engine in *Percona Server for MySQL* 5.7, install the `percona-server-rocksdb` package:

```
$ yum install percona-server-rocksdb
```

Once installed, proceed to modify your configuration file - `my.cnf` - and reinstall the plugins if necessary.

---

**Note:** If you are using *TokuDB* storage engine you'll need to comment out all the *TokuDB* specific variables in your configuration file(s) before starting the server, otherwise the server won't be able to start. *RHEL/CentOS* 7 automatically backs up the previous configuration file to `/etc/my.cnf.rpmsave` and installs the default `my.cnf`. After upgrade/install process completes you can move the old configuration file back (after you remove all the unsupported system variables).

---

You can now start the `mysql` service using `service mysql start` and using `mysql_upgrade` to migrate to the new grant tables, it will rebuild the indexes needed and do the modifications needed:

---

**Note:** If you're using *TokuDB* storage engine you'll need re-enable the storage engine plugin by running the: **`ps-admin --enable-tokudb`** before running `mysql_upgrade` otherwise you'll get errors.

---

```
$ mysql_upgrade
```

---

**Output**

```
Checking if update is needed.
Checking server version.
Running queries to upgrade MySQL server.
Checking system database.
mysql.columns_priv                                OK
mysql.db                                          OK
...
pgrade process completed successfully.
Checking if update is needed.
```

---

Once this is done, just restart the server as usual: `service mysql restart`

After the service has been successfully restarted you can use the new *Percona Server for MySQL* 8.0.

# Upgrading using Standalone Packages

## DEB-based distributions

Having done the full backup (and dump if possible), stop the server. Run this command as root or by using the **sudo** command: `/etc/init.d/mysql stop` and remove the installed packages with their dependencies: `apt-get autoremove percona-server percona-client`

Once removed, proceed to do the modifications needed in your configuration file, as explained at the beginning of this guide.

Then, download the following packages for your architecture:

- `percona-server-server`
- `percona-server-client`
- `percona-server-common`
- `libperconaserverclient21`

The following example will download *Percona Server for MySQL* `8.0.13-3` release packages for *Debian* 9.0:

```
$ wget https://www.percona.com/downloads/Percona-Server-8.9/Percona-Server-8.0.13-3/
↪binary/debian/stretch/x86_64/percona-server-8.0.13-3-r63dafaf-stretch-x86_64-bundle.
↪tar
```

You should then unpack the bundle to get the packages: `tar xvf Percona-Server-8.0.13-3-r63dafaf-stretch-x86_64-bundle.tar`

After you unpack the bundle you should see the following packages:

```
$ ls *.deb

libperconaserverclient21-dev_8.0.13-3-1.stretch_amd64.deb
libperconaserverclient21_8.0.13-3-1.stretch_amd64.deb
percona-server-dbg_8.0.13-3-1.stretch_amd64.deb
percona-server-client_8.0.13-3-1.stretch_amd64.deb
percona-server-common_8.0.13-3-1.stretch_amd64.deb
percona-server-server_8.0.13-3-1.stretch_amd64.deb
percona-server-source_8.0.13-3-1.stretch_amd64.deb
percona-server-test_8.0.13-3-1.stretch_amd64.deb
percona-server-tokudb_8.0.13-3-1.stretch_amd64.deb
```

Now you can install *Percona Server for MySQL* by running:

```
$ sudo dpkg -i *.deb
```

This will install all the packages from the bundle. Another option is to download/specify only the packages you need for running *Percona Server for MySQL* installation (`libperconaserverclient21_8.0.13-3.stretch_amd64.deb`, `percona-server-client-8.0.13-3.stretch_amd64.deb`, `percona-server-common-8.0.13-3.stretch_amd64.deb`, and `percona-server-server-8.0.13-3.stretch_amd64.deb`. Optionally you can install `percona-server-tokudb-8.0.13-3.stretch_amd64.deb` if you want *TokuDB* storage engine).

**Note:** *Percona Server for MySQL* 8.0 comes with the *TokuDB storage engine*. You can find more information on how to install and enable the *TokuDB* storage in the *TokuDB Installation* guide.

**Warning:** When installing packages manually like this, you'll need to make sure to resolve all the dependencies and install missing packages yourself. At least the following packages should be installed before installing *Percona Server for MySQL* 8.0: `libmecab2`, `libjemalloc1`, `zlib1g-dev`, and `libaio1`.

The installation script will not run automatically **`mysql_upgrade`**, so you'll need to run it yourself and restart the service afterwards.

## RPM-based distributions

Having done the full backup (and dump if possible), stop the server (command: `service mysql stop`) and check your installed packages:

```
$ rpm -qa | grep Percona-Server

Percona-Server-57-debuginfo-5.7.10-3.1.el7.x86_64
Percona-Server-client-57-5.7.10-3.1.el7.x86_64
Percona-Server-devel-57-5.7.10-3.1.el7.x86_64
Percona-Server-server-57-5.7.10-3.1.el7.x86_64
Percona-Server-shared-57-5.7.10-3.1.el7.x86_64
Percona-Server-shared-compat-57-5.7.10-3.1.el7.x86_64
Percona-Server-test-57-5.7.10-3.1.el7.x86_64
Percona-Server-tokudb-57-5.7.10-3.1.el7.x86_64
```

You may have the `shared-compat` package, which is for compatibility purposes.

After checked that, proceed to remove them without dependencies: `rpm -qa | grep percona-server | xargs rpm -e --nodeps`

It is important that you remove it without dependencies as many packages may depend on these (as they replace `mysql`) and will be removed if ommited.

Note that this procedure is the same for upgrading from *MySQL* 5.7 to *Percona Server for MySQL* 8.0, just grep `'^mysql-'` instead of `Percona-Server` and remove them.

Download the packages of the desired series for your architecture from the download page. The easiest way is to download bundle which contains all the packages. The following example will download *Percona Server for MySQL* 8.0.13-3 release packages for *CentOS* 7:

```
$ wget https://www.percona.com/downloads/Percona-Server-8.0/Percona-Server-8.0.13-3/
↪binary/redhat/7/x86_64/Percona-Server-8.0.13-3-r63dafaf-el7-x86_64-bundle.tar
```

You should then unpack the bundle to get the packages: `tar xvf Percona-Server-8.0.13-3-r63dafaf-el7-x86_64-bundle.tar`

After you unpack the bundle you should see the following packages: `ls *.rpm`

**Output**

```
percona-server-debuginfo-8.0.13-3.1.el7.x86_64.rpm
percona-server-client-8.0.13-3.1.el7.x86_64.rpm
percona-server-devel-8.0.13-3.1.el7.x86_64.rpm
percona-server-server-8.0.13-3.1.el7.x86_64.rpm
percona-server-shared-8.0.13-3.1.el7.x86_64.rpm
percona-server-shared-compat-8.0.13-3.1.el7.x86_64.rpm
percona-server-test-8.0.13-3.1.el7.x86_64.rpm
percona-server-tokudb-8.0.13-3.1.el7.x86_64.rpm
```

Now, you can install *Percona Server for MySQL* 8.0 by running:

```
rpm -ivh percona-server-server_8.0.13-3.el7.x86_64.rpm \
percona-server-client_8.0.13-3.el7.x86_64.rpm \
percona-server-shared_8.0.13-3.el7.x86_64.rpm
```

This will install only packages required to run the *Percona Server for MySQL* 8.0. Optionally you can install *TokuDB* storage engine by adding the `percona-server-tokudb-8.0.13-3.el7.x86_64.rpm` to the command above. You can find more information on how to install and enable the *TokuDB* storage in the *TokuDB Installation* guide.

To install all the packages (for debugging, testing, etc.) you should run: `rpm -ivh *.rpm`

**Note:** When installing packages manually like this, you'll need to make sure to resolve all the dependencies and install missing packages yourself.

Once installed, proceed to modify your configuration file - `my.cnf` - and install the plugins if necessary. If you're using *TokuDB* storage engine you'll need to comment out all the *TokuDB* specific variables in your configuration file(s) before starting the server, otherwise server won't be able to start. *RHEL/CentOS* 7 automatically backs up the previous configuration file to `/etc/my.cnf.rpmsave` and installs the default `my.cnf`. After upgrade/install process completes you can move the old configuration file back (after you remove all the unsupported system variables).

As the schema of the grant table has changed, the server must be started without reading them: `service mysql start`

Then, use `mysql_upgrade` to migrate to the new grant tables. It will rebuild the indexes needed and do the modifications needed: `mysql_upgrade`

**Note:** If you're using *TokuDB* storage engine you'll need re-enable the storage engine plugin by running the: **`ps-admin --enable-tokudb`** before running `mysql_upgrade` otherwise you'll get errors.

After this is done, just restart the server as usual: `service mysql restart`

## Upgrading from Systems that Use the *TokuDB* or *MyRocks* Storage Engine and Partitioned Tables

Due to the limitation imposed by *MySQL*, it is the storage engine that must provide support for partitioning. *MySQL* 8.0 only provides support for partitioned table for the *InnoDB* storage engine.

If you use partitioned tables with the *TokuDB* or *MyRocks* storage engine, the upgrade may fail if the native partitioning (i.e. provided by the storage engine itself) is not enabled.

Before you attempt the upgrade, check whether or not you have any tables that do not use the native partitioning.

```
$ mysqlcheck -u root --all-databases --check-upgrade
```

If such tables are found **`mysqlcheck`** issues a warning:

**Output of `mysqlcheck` detecting a table that do not use the native partitioning**

```
| comp_test.t1_RocksDB_lz4     OK
| warning  : The partition engine, used by table 'comp_test.t1_RocksDB_lz4',
| is deprecated and will be removed in a future release. Please use native␣
↪partitioning instead.
```

In this case `comp_test.t1_RocksDB_lz4` is not using native partitions. To switch, enable either `rocksdb_enable_native_partition` or `tokudb_enable_native_partition` variable depending on the storage engine that you are using. Then restart the server. Your next step is to alter the tables that are not using the native partitioning with the `UPGRADE PARTITIONING` clause:

```
ALTER TABLE comp_test.t1_RocksDB_lz4 UPGRADE PARTITIONING
```

In this example, the table `comp_test.t1_RocksDB_lz4` to native partitioning. Unless you complete these steps for each table that **`mysqlcheck`** complained about, the upgrade to *MySQL* 8.0 will fail and your error log will contain messages like:

```
2018-12-17T18:34:14.152660Z 2 [ERROR] [MY-013140] [Server] The 'partitioning' feature␣
↪is not available; you need to remove '--skip-partition' or use MySQL built with '-
↪DWITH_PARTITION_STORAGE_ENGINE=1'
2018-12-17T18:34:14.152679Z 2 [ERROR] [MY-013140] [Server] Can't find file: './comp_
↪test/t1_RocksDB_lz4.frm' (errno: 0 - Success)
2018-12-17T18:34:14.152691Z 2 [ERROR] [MY-013137] [Server] Can't find file: './comp_
↪test/t1_RocksDB_lz4.frm' (OS errno: 0 - Success)
```

**See also:**

***MySQL* Documentation: Partitioning Limitations Relating to Storage Engines** https://dev.mysql.com/doc/refman/8.0/en/partitioning-limitations-storage-engines.html

## Performing a Distribution upgrade in-place on a System with installed Percona packages

The recommended process for performing a distribution upgrade on a system with the Percona packages installed is the following:

1. Record the installed Percona packages
2. Backup the data and configurations
3. Uninstall the Percona packages without removing the configurations or data
4. Perform the upgrade by following the distribution upgrade instructions
5. Reboot the system
6. Install the Percona packages intended for the upgraded version of the distribution

# Part III

# Scalability Improvements

CHAPTER

SEVEN

# IMPROVED *INNODB* I/O SCALABILITY

Because *InnoDB* is a complex storage engine it must be configured properly in order to perform at its best. Some points are not configurable in standard *InnoDB*. The goal of this feature is to provide a more exhaustive set of options for *XtraDB*.

## Version Specific Information

- `8.0.12-1` - the feature was ported from *Percona Server for MySQL* 5.7.

## System Variables

**variable `innodb_flush_method`**

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dyn** No

**Variable Type** Enumeration

**Default Value** `fdatasync`

**Allowed Values** `fdatasync`, `O_DSYNC`, `O_DIRECT`, `O_DIRECT_NO_FSYNC`

See *`innodb_flush_method`* in the *MySQL* 8.0 Reference Manual).

This variable affects the parallel doublewrite buffer as follows

| Value | Usage |
|---|---|
| fdatasync | Use `fsync()` to flush parallel doublewrite files. |
| O_SYNC | Use `O_SYNC` to open and flush parallel doublewrite files; Do not use the `fsync()` system call to flush the parallel doublewrite file. |
| O_DIRECT | Use O_DIRECT to open the data files and the `fsync()` system call to flush parallel doublewrite files. |
| O_DIRECT_NO_FSYNC | Use `O_DIRECT` to open the data files but don't use `fsync()` system call to flush doublewrite files. |

## Status Variables

The following information has been added to `SHOW ENGINE INNODB STATUS` to confirm the checkpointing activity:

```
The max checkpoint age
The current checkpoint age target
The current age of the oldest page modification which has not been flushed to disk␣
↪yet.
The current age of the last checkpoint
...
---
LOG
---
Log sequence number 0 1059494372
Log flushed up to   0 1059494372
Last checkpoint at  0 1055251010
Max checkpoint age  162361775
Checkpoint age target 104630090
Modified age        4092465
Checkpoint age      4243362
0 pending log writes, 0 pending chkp writes
...
```

# Part IV

# Performance Improvements

CHAPTER

EIGHT

# MULTIPLE PAGE ASYNCHRONOUS I/O REQUESTS

I/O unit size in *InnoDB* is only one page, even if doing read ahead. 16KB I/O unit size is too small for sequential reads, and much less efficient than larger I/O unit size.

*InnoDB* uses Linux asynchronous I/O (`aio`) by default. By submitting multiple consecutive 16KB read requests at once, Linux internally can merge requests and reads can be done more efficiently.

On a HDD RAID 1+0 environment, more than 1000MB/s disk reads can be achieved by submitting 64 consecutive pages requests at once, while only 160MB/s disk reads is shown by submitting single page request.

With this feature *InnoDB* submits multiple page I/O requests.

## Version Specific Information

- `8.0.12-1` - The feature was ported from *Percona Server for MySQL* 5.7.

## Status Variables

**variable `Innodb_buffered_aio_submitted`**

**Variable Type** Numeric

**Scope** Global

This variable shows the number of submitted buffered asynchronous I/O requests.

## Other Reading

- Making full table scan 10x faster in InnoDB
- Bug #68659 InnoDB Linux native aio should submit more i/o requests at once

CHAPTER

NINE

# THREAD POOL

*MySQL* executes statements using one thread per client connection. Once the number of connections increases past a certain point performance will degrade.

This feature enables the server to keep the top performance even with a large number of client connections by introducing a dynamic thread pool. By using the thread pool server would decrease the number of threads, which will then reduce the context switching and hot locks contentions. Using the thread pool will have the most effect with `OLTP` workloads (relatively short CPU-bound queries).

In order to enable the thread pool variable `thread_handling` should be set up to `pool-of-threads` value. This can be done by adding:

```
thread_handling=pool-of-threads
```

to the *MySQL* configuration file `my.cnf`.

Although the default values for the thread pool should provide good performance, additional tuning can be performed with the dynamic system variables described below.

---

**Note:** Current implementation of the thread pool is built in the server, unlike the upstream version which is implemented as a plugin. Another significant implementation difference is that this implementation doesn't try to minimize the number of concurrent transactions like the `MySQL Enterprise Threadpool`. Because of these things this implementation isn't compatible with the upstream one.

---

## Priority connection scheduling

Even though thread pool puts a limit on the number of concurrently running queries, the number of open transactions may remain high, because connections with already started transactions are put to the end of the queue. Higher number of open transactions has a number of implications on the currently running queries. To improve the performance new *`thread_pool_high_prio_tickets`* variable has been introduced.

This variable controls the high priority queue policy. Each new connection is assigned this many tickets to enter the high priority queue. Whenever a query has to be queued to be executed later because no threads are available, the thread pool puts the connection into the high priority queue if the following conditions apply:

1. The connection has an open transaction in the server.
2. The number of high priority tickets of this connection is non-zero.

If both the above conditions hold, the connection is put into the high priority queue and its tickets value is decremented. Otherwise the connection is put into the common queue with the initial tickets value specified with this option.

Each time the thread pool looks for a new connection to process, first it checks the high priority queue, and picks connections from the common queue only when the high priority one is empty.

The goal is to minimize the number of open transactions in the server. In many cases it is beneficial to give short-running transactions a chance to commit faster and thus deallocate server resources and locks without waiting in the same queue with other connections that are about to start a new transaction, or those that have run out of their high priority tickets.

The default thread pool behavior is to always put events from already started transactions into the high priority queue, as we believe that results in better performance in vast majority of cases.

With the value of `0`, all connections are always put into the common queue, i.e. no priority scheduling is used as in the original implementation in *MariaDB*. The higher is the value, the more chances each transaction gets to enter the high priority queue and commit before it is put in the common queue.

In some cases it is required to prioritize all statements for a specific connection regardless of whether they are executed as a part of a multi-statement transaction or in the autocommit mode. Or vice versa, some connections may require using the low priority queue for all statements unconditionally. To implement this new *`thread_pool_high_prio_mode`* variable has been introduced in *Percona Server for MySQL*.

### Low priority queue throttling

One case that can limit thread pool performance and even lead to deadlocks under high concurrency is a situation when thread groups are oversubscribed due to active threads reaching the oversubscribe limit, but all/most worker threads are actually waiting on locks currently held by a transaction from another connection that is not currently in the thread pool.

What happens in this case is that those threads in the pool that have marked themselves inactive are not accounted to the oversubscribe limit. As a result, the number of threads (both active and waiting) in the pool grows until it hits *`thread_pool_max_threads`* value. If the connection executing the transaction which is holding the lock has managed to enter the thread pool by then, we get a large (depending on the *`thread_pool_max_threads`* value) number of concurrently running threads, and thus, suboptimal performance as a result. Otherwise, we get a deadlock as no more threads can be created to process those transaction(s) and release the lock(s).

Such situations are prevented by throttling the low priority queue when the total number of worker threads (both active and waiting ones) reaches the oversubscribe limit. That is, if there are too many worker threads, do not start new transactions and create new threads until queued events from the already started transactions are processed.

## Handling of Long Network Waits

Certain types of workloads (large result sets, BLOBs, slow clients) can have longer waits on network I/O (socket reads and writes). Whenever server waits, this should be communicated to the Thread Pool, so it can start new query by either waking a waiting thread or sometimes creating a new one. This implementation has been ported from *MariaDB* patch MDEV-156.

## Version Specific Information

- **`8.0.12-1`** `Thread Pool` feature ported from *Percona Server for MySQL* 5.7.

## System Variables

**variable `thread_pool_idle_timeout`**

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dynamic** Yes

**Variable Type** Numeric

**Default Value** 60 (seconds)

This variable can be used to limit the time an idle thread should wait before exiting.

**variable `thread_pool_high_prio_mode`**

**Command Line** Yes

**Config File** Yes

**Scope** Global, Session

**Dynamic** Yes

**Variable Type** String

**Default Value** `transactions`

**Allowed Values** `transactions`, `statements`, `none`

This variable is used to provide more fine-grained control over high priority scheduling either globally or per connection.

The following values are allowed:

- `transactions` (the default). In this mode only statements from already started transactions may go into the high priority queue depending on the number of high priority tickets currently available in a connection (see *`thread_pool_high_prio_tickets`*).
- `statements`. In this mode all individual statements go into the high priority queue, regardless of connection's transactional state and the number of available high priority tickets. This value can be used to prioritize `AUTOCOMMIT` transactions or other kinds of statements such as administrative ones for specific connections. Note that setting this value globally essentially disables high priority scheduling, since in this case all statements from all connections will use a single queue (the high priority one)
- `none`. This mode disables high priority queue for a connection. Some connections (e.g. monitoring) may be insensitive to execution latency and/or never allocate any server resources that would otherwise impact performance in other connections and thus, do not really require high priority scheduling. Note that setting *`thread_pool_high_prio_mode`* to `none` globally has essentially the same effect as setting it to `statements` globally: all connections will always use a single queue (the low priority one in this case).

**variable `thread_pool_high_prio_tickets`**

**Command Line** Yes

**Config File** Yes

**Scope** Global, Session

**Dynamic** Yes

**Variable Type** Numeric

**Default Value** 4294967295

This variable controls the high priority queue policy. Each new connection is assigned this many tickets to enter the high priority queue. Setting this variable to `0` will disable the high priority queue.

**variable `thread_pool_max_threads`**

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dynamic** Yes

**Variable Type** Numeric

**Default Value** 100000

This variable can be used to limit the maximum number of threads in the pool. Once this number is reached no new threads will be created.

**variable `thread_pool_oversubscribe`**

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dynamic** Yes

**Variable Type** Numeric

**Default Value** 3

The higher the value of this parameter the more threads can be run at the same time, if the values is lower than `3` it could lead to more sleeps and wake-ups.

**variable `thread_pool_size`**

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dynamic** Yes

**Variable Type** Numeric

**Default Value** Number of processors

This variable can be used to define the number of threads that can use the CPU at the same time.

**variable `thread_pool_stall_limit`**

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dynamic** No

**Variable Type** Numeric

**Default Value** 500 (ms)

The number of milliseconds before a running thread is considered stalled. When this limit is reached thread pool will wake up or create another thread. This is being used to prevent a long-running query from monopolizing the pool.

### Upgrading from a version before 8.0.14 to 8.0.14 or higher

Starting with the release of version *8.0.141*, *Percona Server for MySQL* uses the upstram implementation of the admin_port. The variables *extra_port* and *extra_max_connections* are removed and not supported. It is essential to remove the `extra_port` and `extra_max_connections` variables from your configuration file before you attempt to upgrade from a release before *8.0.14* to *Percona Server for MySQL* version *8.0.14* or higher. Otherwise, a server produces a boot error and refuses to start.

**See also:**

***MySQL* Documentation:**

- admin_port

**variable `extra_port`**

**Version_info** removed in *8.0.14*

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dynamic** No

**Variable Type** Numeric

**Default Value** 0

This variable can be used to specify an additional port for *Percona Server for MySQL* to listen on. This port can be used in case no new connections can be established due to all worker threads being busy or being locked when `pool-of-threads` feature is enabled.

To connect to the extra port following command can be used:

```
mysql --port='extra-port-number' --protocol=tcp
```

**variable `extra_max_connections`**

**Version_info** removed in *8.0.14*

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dynamic** Yes

**Variable Type** Numeric

**Default Value** 1

This variable can be used to specify the maximum allowed number of connections plus one extra `SUPER` users connection on the *extra_port*. This can be used with the *extra_port* variable to access the server in case no new connections can be established due to all worker threads being busy or being locked when `pool-of-threads` feature is enabled.

## Status Variables

**variable `Threadpool_idle_threads`**

**Variable Type** Numeric

**Scope** Global

This status variable shows the number of idle threads in the pool.

**variable `Threadpool_threads`**

**Variable Type** Numeric

**Scope** Global

This status variable shows the number of threads in the pool.

## Other Reading

- Thread pool in MariaDB 5.5
- Thread pool implementation in Oracle MySQL

CHAPTER

TEN

# XTRADB PERFORMANCE IMPROVEMENTS FOR I/O-BOUND HIGHLY-CONCURRENT WORKLOADS

## Priority refill for the buffer pool free list

In highly-concurrent I/O-bound workloads the following situation may happen:

1. Buffer pool free lists are used faster than they are refilled by the LRU cleaner thread.
2. Buffer pool free lists become empty and more and more query and utility (i.e. purge) threads stall, checking whether a buffer pool free list has became non-empty, sleeping, performing single-page LRU flushes.
3. The number of buffer pool free list mutex waiters increases.
4. When the LRU manager thread (or a single page LRU flush by a query thread) finally produces a free page, it is starved from putting it on the buffer pool free list as it must acquire the buffer pool free list mutex too. However, being one thread in up to hundreds, the chances of a prompt acquisition are low.

This is addressed by delegating all the LRU flushes to the to the LRU manager thread, never attempting to evict a page or perform a LRU single page flush by a query thread, and introducing a backoff algorithm to reduce buffer pool free list mutex pressure on empty buffer pool free lists. This is controlled through a new system variable *innodb_empty_free_list_algorithm*.

**variable `innodb_empty_free_list_algorithm`**

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dynamic** Yes

**Values** legacy, backoff

**Default Value** legacy

When `legacy` option is set, server will use the upstream algorithm and when the `backoff` is selected, *Percona* implementation will be used.

## Multi-threaded LRU flusher

*Percona Server for MySQL* features a true multi-threaded LRU flushing. In this scheme, each buffer pool instance has its own dedicated LRU manager thread that is tasked with performing LRU flushes and evictions to refill the free list of that buffer pool instance. Existing multi-threaded flusher no longer does any LRU flushing and is tasked with flush list flushing only.

- All threads still synchronize on each coordinator thread iteration. If a particular flushing job is stuck on one of the worker threads, the rest will idle until the stuck one completes.
- The coordinator thread heuristics focus on flush list adaptive flushing without considering the state of free lists, which might be in need of urgent refill for a subset of buffer pool instances on a loaded server.
- LRU flushing is serialized with flush list flushing for each buffer pool instance, introducing the risk that the right flushing mode will not happen for a particular instance because it is being flushed in the other mode.

The following *InnoDB* metrics are no longer accounted, as their semantics do not make sense under the current LRU flushing design: `buffer_LRU_batch_flush_avg_time_slot`, `buffer_LRU_batch_flush_avg_pass`, `buffer_LRU_batch_flush_avg_time_thread`, `buffer_LRU_batch_flush_avg_time_est`.

The need for *InnoDB* recovery thread writer threads is also removed, consequently all associated code is deleted.

## Parallel doublewrite buffer

The legacy doublewrite buffer is shared between all the buffer pool instances and all the flusher threads. It collects all the page write requests into a single buffer, and, when the buffer fills, writes it out to disk twice, blocking any new write requests until the writes complete. This becomes a bottleneck with increased flusher parallelism, limiting the effect of extra cleaner threads. In addition, single page flushes, if they are performed, are subject to above and also contend on the doublewrite mutex.

To address these issues *Percona Server for MySQL* uses private doublewrite buffers for each buffer pool instance, for each batch flushing mode (LRU or flush list). For example, with four buffer pool instances, there will be eight doublewrite shards. Only one flusher thread can access any shard at a time, and each shard is added to and flushed completely independently from the rest. This does away with the mutex and the event wait does not block other threads from proceeding anymore, it only waits for the asynchronous I/O to complete. The only inter-thread synchronization is between the flusher thread and I/O completion threads.

The new doublewrite buffer is contained in a new file, where all the shards are contained, at different offsets. This file is created on startup, and removed on a clean shutdown. If it's found on a crashed instance startup, its contents are read and any torn pages are restored. If it's found on a clean instance startup, the server startup is aborted with an error message.

The location of the doublewrite file is governed by a new *innodb_parallel_doublewrite_path* global, read-only system variable. It defaults to `xb_doublewrite` in the data directory. The variable accepts both absolute and relative paths. In the latter case they are treated as relative to the data directory. The doublewrite file is not a tablespace from *InnoDB* internals point of view.

The legacy *InnoDB* doublewrite buffer in the system tablespace continues to address doublewrite needs of single page flushes, and they are free to use the whole of that buffer (128 pages by default) instead of the last eight pages as currently used. Note that single page flushes will not happen in *Percona Server for MySQL* unless *innodb_empty_free_list_algorithm* is set to `legacy` value.

The existing system tablespace is not touched in any way for this feature implementation, ensuring that cleanly-shutdown instances may be freely moved between different server flavors.

### Interaction with `innodb_flush_method`

Regardless of *innodb_flush_method* setting, the parallel doublewrite file is opened with `O_DIRECT` flag to remove OS caching, then its access is further governed by the exact value set: if it's set to `O_DSYNC`, the parallel doublewrite is opened with `O_SYNC` flag too. Further, if it's one of `O_DSYNC`, `O_DIRECT_NO_FSYNC`, then the doublewrite file is not flushed after a batch of writes to it is completed. With other *innodb_flush_method* values the doublewrite buffer is flushed only if setting `O_DIRECT` has failed.

**variable `innodb_parallel_doublewrite_path`**

**Command Line** Yes

**Scope** Global

**Dynamic** No

**Variable Type** String

**Default Value** `xb_doublewrite`

This variable is used to specify the location of the parallel doublewrite file. It accepts both absolute and relative paths. In the latter case they are treated as relative to the data directory.

*Percona Server for MySQL* has introduced several options, only available in builds compiled with `UNIV_PERF_DEBUG` C preprocessor define.

**variable `innodb_sched_priority_master`**

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dynamic** Yes

**Variable Type** Boolean

## Version Specific Information

- `8.0.12-1` Feature ported from *Percona Server for MySQL* 5.7

## Other Reading

- Page cleaner thread tuning
- Bug #74637 - make dirty page flushing more adaptive
- Bug #67808 - in innodb engine, double write and multi-buffer pool instance reduce concurrency
- Bug #69232 - buf_dblwr->mutex can be splited into two

CHAPTER

ELEVEN

# PREFIX INDEX QUERIES OPTIMIZATION

*Percona Server for MySQL* has ported Prefix Index Queries Optimization feature from Facebook patch for *MySQL*.

Prior to this *InnoDB* would always fetch the clustered index for all prefix columns in an index, even when the value of a particular record was smaller than the prefix length. This implementation optimizes that case to use the record from the secondary index and avoid the extra lookup.

## Status Variables

**variable `Innodb_secondary_index_triggered_cluster_reads`**

**Variable Type** Numeric

**Scope** Global

This variable shows the number of times secondary index lookup triggered cluster lookup.

**variable `Innodb_secondary_index_triggered_cluster_reads_avoided`**

**Variable Type** Numeric

**Scope** Global

This variable shows the number of times prefix optimization avoided triggering cluster lookup.

## Version Specific Information

- `8.0.12-1`: The feature was ported from *Percona Server for MySQL* 5.7

CHAPTER

TWELVE

# LIMITING THE ESTIMATION OF RECORDS IN A QUERY

**Availability** This feature is **Experimental** quality.

This page describes an alternative when running queries against a large number of table partitions. When a query runs, InnoDB estimates the records in each partition. This process can result in more pages read and more disk I/O, if the buffer pool must fetch the pages from disk. This process increases the query time if there are a large number of partitions.

The addition of two variables make it possible to override records_in_range which effectively bypasses the process.

**Warning:** The use of these variables may result in improper index selection by the optimizer.

**variable `innodb_records_in_range`**

**Command Line** `--innodb-records-in-range`

**Dynamic** Yes

**Scope** Global

**Variable Type** Numeric

**Default Value** 0

The variable provides a method to limit the number of records estimated for a query.

```
mysql> SET @@GLOBAL.innodb_records_in_range=100;
100
```

**variable `innodb_force_index_records_in_range`**

**Command Line** `--innodb-force-index-records-in-range`

**Dynamic** Yes

**Scope** Global

**Variable Type** Numeric

**Default Value** 0

This variable provides a method to override the *records_in_range* result when a FORCE INDEX is used in a query.

```
mysql> SET @@GLOBAL.innodb_force_index_records_in_range=100;
100
```

# Part V

# Flexibility Improvements

CHAPTER

# THIRTEEN

# SUPPRESS WARNING MESSAGES

This feature is intended to provide a general mechanism (using `log_warnings_silence`) to disable certain warning messages to the log file. Currently, it is only implemented for disabling message #1592 warnings. This feature does not influence warnings delivered to a client. Please note that warning code needs to be a string:

```
mysql> SET GLOBAL log_warnings_suppress = '1592';
Query OK, 0 rows affected (0.00 sec)
```

## Version Specific Information

- `8.0.12-1`: The feature was ported from *Percona Server for MySQL* 5.7

## System Variables

**variable `log_warnings_suppress`**

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dynamic** Yes

**Variable Type** SET

**Default Value** `(empty string)`

**Range** `(empty string),1592`

It is intended to provide a more general mechanism for disabling warnings than existed previously with variable `suppress_log_warning_1592`. When set to the empty string, no warnings are disabled. When set to `1592`, warning #1592 messages (unsafe statement for binary logging) are suppressed. In the future, the ability to optionally disable additional warnings may also be added.

## Related Reading

- MySQL bug 42851
- MySQL InnoDB replication
- InnoDB Startup Options and System Variables

- InnoDB Error Handling

CHAPTER

# FOURTEEN

# IMPROVED MEMORY STORAGE ENGINE

As of MySQL 5.5.15, a *Fixed Row Format* (FRF) is still being used in the MEMORY storage engine. The fixed row format imposes restrictions on the type of columns as it assigns on advance a limited amount of memory per row. This renders a VARCHAR field in a CHAR field in practice and makes impossible to have a TEXT or BLOB field with that engine implementation.

To overcome this limitation, the *Improved MEMORY Storage Engine* is introduced in this release for supporting **true** VARCHAR, VARBINARY, TEXT and BLOB fields in MEMORY tables.

This implementation is based on the *Dynamic Row Format* (DFR) introduced by the mysql-heap-dynamic-rows patch.

DFR is used to store column values in a variable-length form, thus helping to decrease memory footprint of those columns and making possible BLOB and TEXT fields and real VARCHAR and VARBINARY.

Unlike the fixed implementation, each column value in DRF only uses as much space as required. This is, for variable-length values, up to 4 bytes is used to store the actual value length, and then only the necessary number of blocks is used to store the value.

Rows in DFR are represented internally by multiple memory blocks, which means that a single row can consist of multiple blocks organized into one set. Each row occupies at least one block, there can not be multiple rows within a single block. Block size can be configured when creating a table (see below).

This DFR implementation has two caveats regarding to ordering and indexes.

## Caveats

### Ordering of Rows

In the absence of ORDER BY, records may be returned in a different order than the previous MEMORY implementation.

This is not a bug. Any application relying on a specific order without an ORDER BY clause may deliver unexpected results. A specific order without ORDER BY is a side effect of a storage engine and query optimizer implementation which may and will change between minor *MySQL* releases.

### Indexing

It is currently impossible to use indexes on BLOB columns due to some limitations of the *Dynamic Row Format*. Trying to create such an index will fail with the following error:

```
BLOB column '<name>' can't be used in key specification with the used table type.
```

# Restrictions

For performance reasons, a mixed solution is implemented: the fixed format is used at the beginning of the row, while the dynamic one is used for the rest of it.

The size of the fixed-format portion of the record is chosen automatically on `CREATE TABLE` and cannot be changed later. This, in particular, means that no indexes can be created later with `CREATE INDEX` or `ALTER TABLE` when the dynamic row format is used.

All values for columns used in indexes are stored in fixed format at the first block of the row, then the following columns are handled with `DRF`.

This sets two restrictions to tables:

- the order of the fields and therefore,
- the minimum size of the block used in the table.

## Ordering of Columns

The columns used in fixed format must be defined before the dynamic ones in the `CREATE TABLE` statement. If this requirement is not met, the engine will not be able to add blocks to the set for these fields and they will be treated as fixed.

## Minimum Block Size

The block size has to be big enough to store all fixed-length information in the first block. If not, the `CREATE TABLE` or `ALTER TABLE` statements will fail (see below).

# Limitations

*MyISAM* tables are still used for query optimizer internal temporary tables where the `MEMORY` tables could be used now instead: for temporary tables containing large `VARCHAR``s, ``BLOB`, and `TEXT` columns.

# Setting Row Format

Taking the restrictions into account, the *Improved MEMORY Storage Engine* will choose `DRF` over `FRF` at the moment of creating the table according to following criteria:

- There is an implicit request of the user in the column types **OR**
- There is an explicit request of the user **AND** the overhead incurred by `DFR` is beneficial.

## Implicit Request

The implicit request by the user is taken when there is at least one `BLOB` or `TEXT` column in the table definition. If there are none of these columns and no relevant option is given, the engine will choose `FRF`.

For example, this will yield the use of the dynamic format:

```
mysql> CREATE TABLE t1 (f1 VARCHAR(32), f2 TEXT, PRIMARY KEY (f1)) ENGINE=HEAP;
```

While this will not:

```
mysql> CREATE TABLE t1 (f1 VARCHAR(16), f2 VARCHAR(16), PRIMARY KEY (f1)) ENGINE=HEAP;
```

### Explicit Request

The explicit request is set with one of the following options in the `CREATE TABLE` statement:

- `KEY_BLOCK_SIZE = <value>`
  - Requests the DFR with the specified block size (in bytes)

Despite its name, the `KEY_BLOCK_SIZE` option refers to a block size used to store data rather then indexes. The reason for this is that an existing `CREATE TABLE` option is reused to avoid introducing new ones.

*The Improved MEMORY Engine* checks whether the specified block size is large enough to keep all key column values. If it is too small, table creation will abort with an error.

After `DRF` is requested explicitly and there are no `BLOB` or `TEXT` columns in the table definition, the *Improved MEMORY Engine* will check if using the dynamic format provides any space saving benefits as compared to the fixed one:

- if the fixed row length is less than the dynamic block size (plus the dynamic row overhead - platform dependent) **OR**
- there isn't any variable-length columns in the table or `VARCHAR` fields are declared with length 31 or less,

the engine will revert to the fixed format as it is more space efficient in such case. The row format being used by the engine can be checked using `SHOW TABLE STATUS`.

## Examples

On a 32-bit platform:

```
mysql> CREATE TABLE t1 (f1 VARCHAR(32), f2 VARCHAR(32), f3 VARCHAR(32), f4␣
↪VARCHAR(32),
                        PRIMARY KEY (f1)) KEY_BLOCK_SIZE=124 ENGINE=HEAP;

mysql> SHOW TABLE STATUS LIKE 't1';
Name  Engine  Version    Rows Avg_row_length  Data_length     Max_data_length Index_
↪length    Data_free       Auto_increment  Create_time     Update_time     Check_
↪time      Collation       Checksum        Create_options  Comment
t1    MEMORY  10         X    0       X       0       0       NULL    NULL    NULL  ␣
↪NULL    latin1_swedish_ci       NULL    row_format=DYNAMIC KEY_BLOCK_SIZE=124
```

On a 64-bit platform:

```
mysql> CREATE TABLE t1 (f1 VARCHAR(32), f2 VARCHAR(32), f3 VARCHAR(32), f4␣
↪VARCHAR(32),
                        PRIMARY KEY (f1)) KEY_BLOCK_SIZE=124 ENGINE=HEAP;

mysql> SHOW TABLE STATUS LIKE 't1';
Name  Engine  Version    Rows Avg_row_length  Data_length     Max_data_length Index_
↪length    Data_free       Auto_increment  Create_time     Update_time     Check_
↪time      Collation       Checksum        Create_options  Comment
t1    MEMORY  10         X    0       X       0       0       NULL    NULL    NULL  ␣
↪NULL    latin1_swedish_ci       NULL    KEY_BLOCK_SIZE=124
```

## Implementation Details

*MySQL MEMORY* tables keep data in arrays of fixed-size chunks. These chunks are organized into two groups of `HP_BLOCK` structures:

- `group1` contains indexes, with one `HP_BLOCK` per key (part of `HP_KEYDEF`),
- `group2` contains record data, with a single `HP_BLOCK` for all records.

While columns used in indexes are usually small, other columns in the table may need to accommodate larger data. Typically, larger data is placed into `VARCHAR` or `BLOB` columns.

*The Improved MEMORY Engine* implements the concept of dataspace, `HP_DATASPACE`, which incorporates the `HP_BLOCK` structures for the record data, adding more information for managing variable-sized records.

Variable-size records are stored in multiple "chunks", which means that a single record of data (a database "row") can consist of multiple chunks organized into one "set", contained in `HP_BLOCK` structures.

In variable-size format, one record is represented as one or many chunks depending on the actual data, while in fixed-size mode, one record is always represented as one chunk. The index structures would always point to the first chunk in the chunkset.

Variable-size records are necessary only in the presence of variable-size columns. The *Improved Memory Engine* will be looking for `BLOB` or `VARCHAR` columns with a declared length of 32 or more. If no such columns are found, the table will be switched to the fixed-size format. You should always put such columns at the end of the table definition in order to use the variable-size format.

Whenever data is being inserted or updated in the table, the *Improved Memory Engine* will calculate how many chunks are necessary.

For `INSERT` operations, the engine only allocates new chunksets in the recordspace. For `UPDATE` operations it will modify the length of the existing chunkset if necessary, unlinking unnecessary chunks at the end, or allocating and adding more if a larger length is needed.

When writing data to chunks or copying data back to a record, fixed-size columns are copied in their full format, while `VARCHAR` and `BLOB` columns are copied based on their actual length, skipping any `NULL` values.

When allocating a new chunkset of N chunks, the engine will try to allocate chunks one-by-one, linking them as they become allocated. For allocating a single chunk, it will attempt to reuse a deleted (freed) chunk. If no free chunks are available, it will try to allocate a new area inside a `HP_BLOCK`.

When freeing chunks, the engine will place them at the front of a free list in the dataspace, each one containing a reference to the previously freed chunk.

The allocation and contents of the actual chunks varies between fixed and variable-size modes:

- Format of a fixed-size chunk:
  - `uchar[]`
    - With `sizeof=chunk_dataspace_length`, but at least `sizeof(uchar*)` bytes. It keeps actual data or pointer to the next deleted chunk, where `chunk_dataspace_length` equals to full record length
  - `uchar`
    - Status field (1 means "in use", 0 means "deleted")
- Format of a variable-size chunk:
  - `uchar[]`

    - With `sizeof=chunk_dataspace_length`, but at least `sizeof(uchar*)` bytes. It keeps actual data or pointer to the next deleted chunk, where `chunk_dataspace_length` is set according to table's `key_block_size`

  - `uchar*`
    - Pointer to the next chunk in this chunkset, or NULL for the last chunk
  - `uchar`
    - Status field (1 means "first", 0 means "deleted", 2 means "linked")

Total chunk length is always aligned to the next `sizeof(uchar*)`.

## See Also

- Dynamic row format for MEMORY tables

CHAPTER

FIFTEEN

# EXTENDED `MYSQLDUMP`

## Backup Locks support

When used together with the `--single-transaction` option, the *`lock-for-backup`* option makes `mysqldump` issue `LOCK TABLES FOR BACKUP` before starting the dump operation to prevent unsafe statements that would normally result in an inconsistent backup.

More information can be found on the *Backup Locks* feature documentation.

## Compressed Columns support

**`mysqldump`** supports the *Compressed columns with dictionaries* feature. More information about the relevant options can be found on the *Compressed columns with dictionaries* feature page.

## Taking backup by descending primary key order

`--order-by-primary-desc` tells `mysqldump` to take the backup by descending primary key order (`PRIMARY KEY DESC`) which can be useful if the storage engine is using the reverse order column for a primary key.

## RocksDB support

**`mysqldump`** detects when MyRocks is installed and available. If there is a session variable named *`rocksdb_skip_fill_cache`* **`mysqldump`** sets it to **1**.

**`mysqldump`** will now automatically enable session the variable *`rocksdb_bulk_load`* if it is supported by the target server.

## Version Specific Information

- `8.0.12-1`: The feature was ported from *Percona Server for MySQL* 5.7

CHAPTER

SIXTEEN

# EXTENDED `SELECT INTO OUTFILE/DUMPFILE`

*Percona Server for MySQL* has extended the `SELECT INTO ... OUTFILE` and `SELECT INTO DUMPFILE` commands to add the support for UNIX sockets and named pipes. Before this was implemented the database would return an error for such files.

This feature allows using `LOAD DATA LOCAL INFILE` in combination with `SELECT INTO OUTFILE` to quickly load multiple partitions across the network or in other setups, without having to use an intermediate file which wastes space and I/O.

## Version Specific Information

- `8.0.12-1` - Feature ported from *Percona Server for MySQL* 5.7.

## Other Reading

- *MySQL* bug: #44835

CHAPTER

SEVENTEEN

# EXTENDED `MYSQLBINLOG`

*Percona Server for MySQL* has implemented compression support for **`mysqlbinlog`**. This is similar to support that both `mysql` and `mysqldump` programs include (the `-C`, `--compress` options "Use compression in server/client protocol"). Using the compressed protocol helps reduce the bandwidth use and speed up transfers.

*Percona Server for MySQL* has also implemented support for `SSL`. **`mysqlbinlog`** now accepts the `SSL` connection options as all the other client programs. This feature can be useful with `--read-from-remote-server` option. Following `SSL` options are now available:

- `--ssl` - Enable SSL for connection (automatically enabled with other flags).
- `--ssl-ca=name` - CA file in PEM format (check OpenSSL docs, implies –ssl).
- `--ssl-capath=name` - CA directory (check OpenSSL docs, implies –ssl).
- `--ssl-cert=name` - X509 cert in PEM format (implies –ssl).
- `--ssl-cipher=name` - SSL cipher to use (implies –ssl).
- `--ssl-key=name` - X509 key in PEM format (implies –ssl).
- `--ssl-verify-server-cert` - Verify server's "Common Name" in its cert against hostname used when connecting. This option is disabled by default.

## Version Specific Information

- `8.0.12-1`: The feature was ported from *Percona Server for MySQL* 5.7

CHAPTER

EIGHTEEN

# SUPPORT FOR PROXY PROTOCOL

The proxy protocol allows an intermediate proxying server speaking proxy protocol (ie. HAProxy) between the server and the ultimate client (i.e. mysql client etc) to provide the source client address to the server, which normally would only see the proxying server address instead.

As the proxy protocol amounts to spoofing the client address, it is disabled by default, and can be enabled on per-host or per-network basis for the trusted source addresses where trusted proxy servers are known to run. Unproxied connections are not allowed from these source addresses.

**Note:** You need to ensure proper firewall ACL's in place when this feature is enabled.

Proxying is supported for TCP over IPv4 and IPv6 connections only. UNIX socket connections can not be proxied and do not fall under the effect of proxy-protocol-networks='*'.

As a special exception, it is forbidden for the proxied IP address to be `127.0.0.1` or `::1`.

## Version Specific Information

- `8.0.12-1`: Feature ported from *Percona Server for MySQL* 5.7.

## System Variables

**variable `proxy_protocol_networks`**

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dynamic** No

**Default Value** `(empty string)`

This variable is a global-only, read-only variable, which is either a `*` (to enable proxying globally, a non-recommended setting), or a list of comma-separated IPv4 and IPv6 network and host addresses, for which proxying is enabled. Network addresses are specified in CIDR notation, i.e. `192.168.0.0/24`. To prevent source host spoofing, the setting of this variable must be as restrictive as possible to include only trusted proxy hosts.

## Related Reading

- PROXY protocol specification

CHAPTER

NINETEEN

# COMPRESSED COLUMNS WITH DICTIONARIES

The `per-column compression` feature is a data type modifier, independent from user-level SQL and *InnoDB* data compression, that causes the data stored in the column to be compressed on writing to storage and decompressed on reading. For all other purposes, the data type is identical to the one without the modifier, i.e. no new data types are created. Compression is done by using the `zlib` library.

Additionally, it is possible to pre-define a set of strings for each compressed column to achieve a better compression ratio on relatively small individual data items.

This feature provides:

- a better compression ratio for text data which consist of a large number of predefined words (e.g. JSON or XML) using compression methods with static dictionaries
- a way to select columns in the table to compress (in contrast to the *InnoDB* row compression method)

This feature is based on a patch provided by Weixiang Zhai.

## Specifications

The feature is limited to InnoDB/XtraDB storage engine and to columns of the following data types:

- `BLOB` (including `TINYBLOB`, `MEDIUMBLOB`, `LONGBLOG`)
- `TEXT` (including `TINYTEXT`, `MEDUUMTEXT`, `LONGTEXT`)
- `VARCHAR` (including `NATIONAL VARCHAR`)
- `VARBINARY`
- `JSON`

A compressed column is declared by using the syntax that extends the existing `COLUMN_FORMAT` modifier: `COLUMN_FORMAT COMPRESSED`. If this modifier is applied to an unsupported column type or storage engine, an error is returned.

The compression can be specified:

- when creating a table: `CREATE TABLE ... (..., foo BLOB COLUMN_FORMAT COMPRESSED, ...);`
- when altering a table and modifying a column to the compressed format: `ALTER TABLE ... MODIFY [COLUMN] ... COLUMN_FORMAT COMPRESSED`, or `ALTER TABLE ... CHANGE [COLUMN] ... COLUMN_FORMAT COMPRESSED`.

Unlike Oracle MySQL, compression is applicable to generated stored columns. Use this syntax extension as follows:

```
mysql> CREATE TABLE t1(
       id INT,
       a BLOB,
       b JSON COLUMN_FORMAT COMPRESSED,
       g BLOB GENERATED ALWAYS AS (a) STORED COLUMN_FORMAT COMPRESSED WITH␣
↪COMPRESSION_DICTIONARY numbers
    ) ENGINE=InnoDB;
```

To decompress a column, specify a value other than `COMPRESSED` to `COLUMN_FORMAT`: `FIXED`, `DYNAMIC`, or `DEFAULT`. If there is a column compression/decompression request in an `ALTER TABLE`, it is forced to the `COPY` algorithm.

Two new variables: *innodb_compressed_columns_zip_level* and *innodb_compressed_columns_threshold* have been implemented.

## Compression dictionary support

To achieve a better compression ratio on relatively small individual data items, it is possible to predefine a compression dictionary, which is a set of strings for each compressed column.

Compression dictionaries can be represented as a list of words in the form of a string (comma or any other character can be used as a delimiter although not required). In other words, `a,bb,ccc`, `a bb ccc` and `abbccc` will have the same effect. However, the latter is more compact. Quote symbol quoting is handled by regular SQL quoting. The maximum supported dictionary length is 32506 bytes (`zlib` limitation).

The compression dictionary is stored in a new system *InnoDB* table. As this table is of the data dictionary kind, concurrent reads are allowed, but writes are serialized, and reads are blocked by writes. Table read through old read views are not supported, similar to *InnoDB* internal DDL transactions.

### Interaction with `innodb_force_recovery` variable

Compression dictionary operations are treated like DDL operations with the exception when `innodb_force_value` is set to `3`: with values less than `3`, compression dictionary operations are allowed, and with values >= `3`, they are forbidden.

---

**Note:** Prior to *Percona Server for MySQL 8.0.15-6* using Compression dictionary operations with `innodb_force_recovery` variable set to value > 0 would result in an error.

---

### Example

In order to use the compression dictionary you need to create it. This can be done by running:

```
mysql> SET @dictionary_data = 'one' 'two' 'three' 'four';
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE COMPRESSION_DICTIONARY numbers (@dictionary_data);
Query OK, 0 rows affected (0.00 sec)
```

To create a table that has both compression and compressed dictionary support you should run:

```
mysql> CREATE TABLE t1(
        id INT,
        a BLOB COLUMN_FORMAT COMPRESSED,
        b BLOB COLUMN_FORMAT COMPRESSED WITH COMPRESSION_DICTIONARY numbers
      ) ENGINE=InnoDB;
```

The following example shows how to insert a sample of JSON data into the table:

```
SET @json_value =
'[\n'
' {\n'
' "one" = 0,\n'
' "two" = 0,\n'
' "three" = 0,\n'
' "four" = 0\n'
' },\n'
' {\n'
' "one" = 0,\n'
' "two" = 0,\n'
' "three" = 0,\n'
' "four" = 0\n'
' },\n'
' {\n'
' "one" = 0,\n'
' "two" = 0,\n'
' "three" = 0,\n'
' "four" = 0\n'
' },\n'
' {\n'
' "one" = 0,\n'
' "two" = 0,\n'
' "three" = 0,\n'
' "four" = 0\n'
' }\n'
']\n'
;
```

```
mysql> INSERT INTO t1 VALUES(0, @json_value, @json_value);
Query OK, 1 row affected (0.01 sec)
```

## INFORMATION_SCHEMA Tables

This feature implements two new INFORMATION_SCHEMA tables.

**table** INFORMATION_SCHEMA.**COMPRESSION_DICTIONARY**

**Columns**

- **dict_version** (*BIGINT(21) UNSIGNED*) – dictionary version
- **dict_name** (*VARCHAR(64)*) – dictionary name
- **dict_data** (*BLOB*) – compression dictionary string

This table provides a view over the internal compression dictionary. The SUPER privilege is required to query it.

**table** INFORMATION_SCHEMA.**COMPRESSION_DICTIONARY_TABLES**

**Columns**

- **table_schema** (*BIGINT(21)_UNSIGNED*) – table schema
- **table_name** (*BIGINT(21)_UNSIGNED*) – table ID from INFORMATION_SCHEMA.INNODB_SYS_TABLES
- **column_name** (*BIGINT(21)_UNSIGNED*) – column position (starts from 0 as in INFORMATION_SCHEMA.INNODB_SYS_COLUMNS)
- **dict_name** (*BIGINT(21)_UNSIGNED*) – dictionary ID

This table provides a view over the internal table that stores the mapping between the compression dictionaries and the columns using them. The SUPER privilege is require to query it.

## Limitations

Compressed columns cannot be used in indices (neither on their own nor as parts of composite keys).

---

**Note:** CREATE TABLE t2 AS SELECT * FROM t1 will create a new table with a compressed column, whereas CREATE TABLE t2 AS SELECT CONCAT(a,'') AS a FROM t1 will not create compressed columns.

At the same time, after executing CREATE TABLE t2 LIKE t1 statement, t2.a will have COMPRESSED attribute.

---

ALTER TABLE ... DISCARD/IMPORT TABLESPACE is not supported for tables with compressed columns. To export and import tablespaces with compressed columns, you need to uncompress them first with: ALTER TABLE ... MODIFY ... COLUMN_FORMAT DEFAULT.

## mysqldump command line parameters

By default, with no additional options, mysqldump will generate a *MySQL* compatible SQL output.

All /*!50633 COLUMN_FORMAT COMPRESSED */ and /*!50633 COLUMN_FORMAT COMPRESSED WITH COMPRESSION_DICTIONARY <dictionary> */ won't be in the dump.

When a new option enable-compressed-columns is specified, all /*!50633 COLUMN_FORMAT COMPRESSED */ will be left intact and all /*!50633 COLUMN_FORMAT COMPRESSED WITH COMPRESSION_DICTIONARY <dictionary> */ will be transformed into /*!50633 COLUMN_FORMAT COMPRESSED */. In this mode the dump will contain the necessary SQL statements to create compressed columns, but without dictionaries.

When a new enable-compressed-columns-with-dictionaries option is specified, dump will contain all compressed column attributes and compression dictionary.

Moreover, the following dictionary creation fragments will be added before CREATE TABLE statements which are going to use these dictionaries for the first time.

```
/*!50633 DROP COMPRESSION_DICTIONARY IF EXISTS <dictionary>; */
/*!50633 CREATE COMPRESSION_DICTIONARY <dictionary>(...); */
```

Two new options add-drop-compression-dictionary and skip-add-drop-compression-dictionary will control if /*!50633 DROP COMPRESSION_DICTIONARY IF EXISTS <dictionary> */ part from

previous paragraph will be skipped or not. By default, `add-drop-compression-dictionary` mode will be used.

When both `enable-compressed-columns-with-dictionaries` and `--tab=<dir>` (separate file for each table) options are specified, necessary compression dictionaries will be created in each output file using the following fragment (regardless of the values of `add-drop-compression-dictionary` and `skip-add-drop-compression-dictionary` options).

```
/*!50633 CREATE COMPRESSION_DICTIONARY IF NOT EXISTS <dictionary>(...); */
```

## Version Specific Information

- *8.0.13-3* Feature ported from *Percona Server for MySQL* 5.7.

## System Variables

**variable `innodb_compressed_columns_zip_level`**

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dynamic** Yes

**Variable Type** Numeric

**Default Value** 6

**Range** 0-9

This variable is used to specify the compression level used for compressed columns. Specifying `0` will use no compression, `1` the fastest and `9` the best compression. Default value is `6`.

**variable `innodb_compressed_columns_threshold`**

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dynamic** Yes

**Variable Type** Numeric

**Default Value** 96

**Range** 1 - 2^64-1 (or 2^32-1 for 32-bit release)

By default a value being inserted will be compressed if its length exceeds *innodb_compressed_columns_threshold* bytes. Otherwise, it will be stored in raw (uncompressed) form.

Please also notice that because of the nature of some data, its compressed representation can be longer than the original value. In this case it does not make sense to store such values in compressed form as *Percona Server for MySQL* would have to waste both memory space and CPU resources for unnecessary decompression. Therefore, even if the length of such non-compressible values exceeds *innodb_compressed_columns_threshold*, they will be stored in an uncompressed form (however, an attempt to compress them will still be made).

This parameter can be tuned in order to skip unnecessary attempts of data compression for values that are known in advance by the user to have bad compression ratio of their first N bytes.

**See also:**

**How to find a good/optimal dictionary for zlib 'setDictionary' when processing a given set of data?** http://stackoverflow.com/questions/2011653/how-to-find-a-good-optimal-dictionary-for-zlib-setdictionary-when-processing-a

CHAPTER

TWENTY

# INNODB FULL-TEXT SEARCH IMPROVEMENTS

CHAPTER

TWENTYONE

# IGNORING STOPWORD LIST

By default all Full-Text Search indexes check the stopwords list, to see if any indexed elements *contain* one of the words on that list.

Using this list for n-gram indexes isn't always suitable, as an example, any item that contains `a` or `i` will be ignored. Another word that can't be searched is `east`, this one will find no matches because `a` is on the FTS stopword list.

To resolve this issue, *Percona Server for MySQL* has the *innodb_ft_ignore_stopwords* variable to control whether *InnoDB* Full-Text Search should ignore the stopword list.

Although this variable is introduced to resolve n-gram issues, it affects all Full-Text Search indexes as well.

Being a stopword doesn't just mean to be a one of the predefined words from the list. Tokens shorter than innodb_ft_min_token_size or longer than innodb_ft_max_token_size are also considered stopwords. Therefore, when *innodb_ft_ignore_stopwords* is set to `ON` even for non-ngram FTS, `innodb_ft_min_token_size` / `innodb_ft_max_token_size` will be ignored meaning that in this case very short and very long words will also be indexed.

## System Variables

**variable innodb_ft_ignore_stopwords**

**Command Line** Yes

**Config File** Yes

**Scope** Session, Global

**Dynamic** Yes

**Variable Type** Boolean

**Default Value** `OFF`

When enabled, this variable will instruct *InnoDB* Full Text Search parser to ignore the stopword list when building/updating an FTS index.

CHAPTER

TWENTYTWO

# BINLOGGING AND REPLICATION IMPROVEMENTS

Due to continuous development, *Percona Server for MySQL* incorporated a number of improvements related to replication and binary logs handling. This resulted in replication specifics, which distinguishes it from *MySQL*.

## Safety of statements with a `LIMIT` clause

### Summary of the Fix

*MySQL* considers all `UPDATE/DELETE/INSERT ... SELECT` statements with `LIMIT` clause to be unsafe, no matter wether they are really producing non-deterministic result or not, and switches from statement-based logging to row-based one. *Percona Server for MySQL* is more accurate, it acknowledges such instructions as safe when they include `ORDER BY PK` or `WHERE` condition. This fix has been ported from the upstream bug report #42415 (#44).

## Performance improvement on relay log position update

### Summary of the Fix

*MySQL* always updated relay log position in multi-source replications setups regardless of whether the committed transaction has already been executed or not. Percona Server omitts relay log position updates for the already logged GTIDs.

### Details

Particularly, such unconditional relay log position updates caused additional fsync operations in case of `relay-log-info-repository=TABLE`, and with the higher number of channels transmitting such duplicate (already executed) transactions the situation became proportionally worse. Bug fixed #1786 (upstream #85141).

## Performance improvement on master and connection status updates

### Summary of the Fix

Slave nodes configured to update master status and connection information only on log file rotation did not experience the expected reduction in load. *MySQL* was additionaly updating this information in case of multi-source replication when slave had to skip the already executed GTID event.

### Details

The configuration with `master_info_repository=TABLE` and `sync_master_info=0` makes slave to update master status and connection information in this table on log file rotation and not after each sync_master_info event, but it didn't work on multi-source replication setups. Heartbeats sent to the slave to skip GTID events which it had already executed previously, were evaluated as relay log rotation events and reacted with `mysql.slave_master_info` table sync. This inaccuracy could produce huge (up to 5 times on some setups) increase in write load on the slave, before this problem was fixed in *Percona Server for MySQL*. Bug fixed #1812 (upstream #85158).

## Writing `FLUSH` Commands to the Binary Log

`FLUSH` commands, such as `FLUSH SLOW LOGS`, are not written to the binary log if the system variable *`binlog_skip_flush_commands`* is set to **ON**.

In addition, the following changes were implemented in the behavior of `read_only` and `super_read_only` modes:

- When `read_only` is set to **ON**, any `FLUSH ...` command executed by a normal user (without the `SUPER` privilege) are not written to the binary log regardless of the value of the `binlog_skip_flush_command` variable.
- When `super_read_only` is set to **ON**, any `FLUSH ...` command executed by any user (even by those with the `SUPER` privilege) are not written to the binary log regardless of the value of the `binlog_skip_flush_command` variable.

An attempt to run a `FLUSH` command without either `SUPER` or `RELOAD` privileges results in the `ER_SPECIFIC_ACCESS_DENIED_ERROR` exception regardless of the value of the `binlog_skip_flush_command` variable.

**variable `binlog_skip_flush_commands`**

**Version Info**

- ***`8.0.15-5`*** – Introduced

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dynamic** Yes

**Default Value** OFF

When `binlog_skip_flush_command` is set to **ON**, `FLUSH ...` commands are not written to the binary log. See *Writing FLUSH Commands to the Binary Log* for more information about what else affects the writing of `FLUSH` commands to the binary log.

---

**Note:** `FLUSH LOGS`, `FLUSH BINARY LOGS`, `FLUSH TABLES WITH READ LOCK`, and `FLUSH TABLES ... FOR EXPORT` are not written to the binary log no matter what value the `binlog_skip_flush_command` variable contains. The `FLUSH` command is not recorded to the binary log and the value of `binlog_skip_flush_command` is ignored if the `FLUSH` command is run with the `NO_WRITE_TO_BINLOG` keyword (or its alias `LOCAL`).

**See also:**

***MySQL* Documentation: FLUSH Syntax** https://dev.mysql.com/doc/refman/8.0/en/flush.html

CHAPTER

TWENTYTHREE

# EXTENDED SET VAR OPTIMIZER HINT

*Percona Server for MySQL* 8.0 extends the `SET_VAR` introduced in *MySQL* 8.0 effectively replacing the `SET STATEMENT ... FOR` statement. `SET_VAR` is an optimizer hint that can be applied to session variables.

*Percona Server for MySQL* 8.0 extends the `SET_VAR` hint to support the following:

- The `OPTIMIZE TABLE` statement
- MyISAM session variables
- Plugin or Storage Engine variables
- InnoDB Session variables
- The `ALTER TABLE` statement
- `CALL stored_proc()` statement
- The `ANALYZE TABLE` statement
- The `CHECK TABLE` statement
- The `LOAD INDEX` statement (used for MyISAM)
- The `CREATE TABLE` statement

*Percona Server for MySQL* 8.0 also supports setting the following variables by using `SET_VAR`:

- innodb_lock_wait_timeout
- innodb_tmpdir
- innodb_ft_user_stopword_table
- block_encryption_mode
- histogram_generation_max_mem_size
- myisam_sort_buffer_size
- myisam_repair_threads
- myisam_stats_method
- preload_buffer_size (used by MyISAM only)

**See also:**

***MySQL* Documentation: Variable-setting hint syntax** https://dev.mysql.com/doc/refman/8.0/en/optimizer-hints.html#optimizer-hints-set-var

# Part VI

# Reliability Improvements

CHAPTER

# TWENTYFOUR

# TOO MANY CONNECTIONS WARNING

This feature issues the warning `Too many connections` to the log, if `log_error_verbosity` is set to `2` or higher.

## Version-Specific Information

- `8.0.12-1`: Feature ported from *Percona Server for MySQL* 5.7.

CHAPTER

TWENTYFIVE

# HANDLE CORRUPTED TABLES

When a server subsystem tries to access a corrupted table, the server may crash. If this outcome is not desirable when a corrupted table is encountered, set the new system *innodb_corrupt_table_action* variable to a value which allows the ongoing operation to continue without crashing the server.

The server error log registers attempts to access corrupted table pages.

### Interacting with the innodb_force_recovery variable

The *innodb_corrupt_table_action* variable may work in conjunction with the innodb_force_recovery variable which considerably reduces the effect of *InnoDB* subsystems running in the background.

If the innodb_force_recovery variable is set to a low value and you expect the server to crash, it may still be running due to a non-default value of the *innodb_corrupt_table_action* variable.

For more information about the innodb_force_recovery variable, see Forcing InnoDB Recovery from the MySQL Reference Manual.

This feature adds a new system variable.

## Version Specific Information

- 8.0.12-1: Feature ported from *Percona Server for MySQL* 5.7.

## System Variables

**variable innodb_corrupt_table_action**

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dynamic** Yes

**Variable Type** ULONG

**Range** assert, warn, salvage

- With the default value *XtraDB* will intentionally crash the server with an assertion failure as it would normally do when detecting corrupted data in a single-table tablespace.

- If the `warn` value is used it will pass corruption of the table as `corrupt table` instead of crashing itself. For this to work `innodb_file_per_table` should be enabled. All file I/O for the datafile after detected as corrupt is disabled, except for the deletion.
- When the option value is `salvage`, *XtraDB* allows read access to a corrupted tablespace, but ignores corrupted pages".

# Part VII

# Management Improvements

CHAPTER

# TWENTYSIX

# *PERCONA TOOLKIT* UDFS

Three *Percona Toolkit* UDFs that provide faster checksums are provided:

- `libfnv1a_udf`
- `libfnv_udf`
- `libmurmur_udf`

## Version Specific Information

- `8.0.12-1`: Feature ported from *Percona Server for MySQL* 5.7.

## Other Information

- Author / Origin: Baron Schwartz

## Installation

These UDFs are part of the *Percona Server for MySQL* packages. To install one of the UDFs into the server, execute one of the following commands, depending on which UDF you want to install:

```
mysql -e "CREATE FUNCTION fnv1a_64 RETURNS INTEGER SONAME 'libfnv1a_udf.so'"
mysql -e "CREATE FUNCTION fnv_64 RETURNS INTEGER SONAME 'libfnv_udf.so'"
mysql -e "CREATE FUNCTION murmur_hash RETURNS INTEGER SONAME 'libmurmur_udf.so'"
```

Executing each of these commands will install its respective UDF into the server.

## Troubleshooting

If you get the error:

```
ERROR 1126 (HY000): Can't open shared library 'fnv_udf.so' (errno: 22 fnv_udf.so:␣
↪cannot open shared object file: No such file or directory)
```

Then you may need to copy the .so file to another location in your system. Try both `/lib` and `/usr/lib`. Look at your environment's `$LD_LIBRARY_PATH` variable for clues. If none is set, and neither `/lib` nor `/usr/lib` works, you may need to set `LD_LIBRARY_PATH` to `/lib` or `/usr/lib`.

## Other Reading

- *Percona Toolkit* documentation

CHAPTER

# TWENTYSEVEN

# KILL IDLE TRANSACTIONS

This feature limits the age of idle transactions, for all transactional storage engines. If a transaction is idle for more seconds than the threshold specified, it will be killed. This prevents users from blocking *InnoDB* purge by mistake.

## Version Specific Information

- **`8.0.12-1:`** Feature ported from *Percona Server for MySQL* 5.7

## System Variables

**variable `kill_idle_transaction`**

**Scope** `GLOBAL`

**Config** `YES`

**Dynamic** `YES`

**Variable Type** `INTEGER`

**Default Value** 0 (disabled)

**Units** Seconds

If non-zero, any idle transaction will be killed after being idle for this many seconds.

CHAPTER

TWENTYEIGHT

# XTRADB CHANGED PAGE TRACKING

*XtraDB* now tracks the pages that have changes written to them according to the redo log. This information is written out in special changed page bitmap files. This information can be used to speed up incremental backups using Percona XtraBackup by removing the need to scan whole data files to find the changed pages. Changed page tracking is done by a new *XtraDB* worker thread that reads and parses log records between checkpoints. The tracking is controlled by a new read-only server variable *innodb_track_changed_pages*.

Bitmap filename format used for changed page tracking is `ib_modified_log_<seq>_<startlsn>.xdb`. The first number is the sequence number of the bitmap log file and the *startlsn* number is the starting LSN number of data tracked in that file. Example of the bitmap log files should look like this:

```
ib_modified_log_1_0.xdb
ib_modified_log_2_1603391.xdb
```

Sequence number can be used to easily check if all the required bitmap files are present. Start LSN number will be used in *XtraBackup* and `INFORMATION_SCHEMA` queries to determine which files have to be opened and read for the required LSN interval data. The bitmap file is rotated on each server restart and whenever the current file size reaches the predefined maximum. This maximum is controlled by a new *innodb_max_bitmap_file_size* variable.

Old bitmap files may be safely removed after a corresponding incremental backup is taken. For that there are server *User statements for handling the XtraDB changed page bitmaps*. Removing the bitmap files from the filesystem directly is safe too, as long as care is taken not to delete data for not-yet-backuped LSN range.

This feature will be used for implementing faster incremental backups that use this information to avoid full data scans in *Percona XtraBackup*.

## User statements for handling the XtraDB changed page bitmaps

New statements have been introduced for handling the changed page bitmap tracking. All of these statements require `SUPER` privilege.

- `FLUSH CHANGED_PAGE_BITMAPS` - this statement can be used for synchronous bitmap write for immediate catch-up with the log checkpoint. This is used by innobackupex to make sure that XtraBackup indeed has all the required data it needs.
- `RESET CHANGED_PAGE_BITMAPS` - this statement will delete all the bitmap log files and restart the bitmap log file sequence.
- `PURGE CHANGED_PAGE_BITMAPS BEFORE <lsn>` - this statement will delete all the change page bitmap files up to the specified log sequence number.

## Additional information in SHOW ENGINE INNODB STATUS

When log tracking is enabled, the following additional fields are displayed in the LOG section of the `SHOW ENGINE INNODB STATUS` output:

- "Log tracked up to:" displays the LSN up to which all the changes have been parsed and stored as a bitmap on disk by the log tracking thread
- "Max tracked LSN age:" displays the maximum limit on how far behind the log tracking thread may be.

## INFORMATION_SCHEMA Tables

This table contains a list of modified pages from the bitmap file data. As these files are generated by the log tracking thread parsing the log whenever the checkpoint is made, it is not real-time data.

**table** `INFORMATION_SCHEMA.`**`INNODB_CHANGED_PAGES`**

**Columns**

- **`space_id`** (*`INT(11)`*) – space id of modified page
- **`page_id`** (*`INT(11)`*) – id of modified page
- **`start_lsn`** (*`BIGINT(21)`*) – start of the interval
- **`end_lsn`** (*`BIGINT(21)`*) – end of the interval

The `start_lsn` and the `end_lsn` columns denote between which two checkpoints this page was changed at least once. They are also equal to checkpoint LSNs.

Number of records in this table can be limited by using the variable *`innodb_max_changed_pages`*.

## Version Specific Information

- **`8.0.12-1`** Feature ported from *Percona Server for MySQL* 5.7.

## System Variables

**variable `innodb_max_changed_pages`**

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dynamic** Yes

**Variable Type** Numeric

**Default Value** 1000000

**Range** 1 - 0 (unlimited)

This variable is used to limit the result row count for the queries from *`INNODB_CHANGED_PAGES`* table.

**variable `innodb_track_changed_pages`**

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dynamic** No

**Variable Type** Boolean

**Default Value** 0 - False

**Range** 0-1

This variable is used to enable/disable *XtraDB changed page tracking* feature.

**variable `innodb_max_bitmap_file_size`**

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dynamic** Yes

**Variable Type** Numeric

**Default Value** 104857600 (100 MB)

**Range** 4096 (4KB) - 18446744073709551615 (16EB)

This variable is used to control maximum bitmap size after which the file will be rotated.

CHAPTER

TWENTYNINE

# PAM AUTHENTICATION PLUGIN

This page has been moved or been replaced. The new page is located here:

*PAM Authentication Plugin*

Please update any bookmarks that point to the old page.

CHAPTER

THIRTY

# EXPANDED FAST INDEX CREATION

**Availability** This feature is **Experimental** qualtiy.

Percona has implemented several changes related to *MySQL*'s fast index creation feature. Fast index creation was implemented in *MySQL* as a way to speed up the process of adding or dropping indexes on tables with many rows.

This feature implements a session variable that enables extended fast index creation. Besides optimizing DDL directly, *expand_fast_index_creation* may also optimize index access for subsequent DML statements because using it results in much less fragmented indexes.

## The `mysqldump` Command

A new option, `--innodb-optimize-keys`, was implemented in **`mysqldump`**. It changes the way *InnoDB* tables are dumped, so that secondary and foreign keys are created after loading the data, thus taking advantage of fast index creation. More specifically:

- `KEY`, `UNIQUE KEY`, and `CONSTRAINT` clauses are omitted from `CREATE TABLE` statements corresponding to *InnoDB* tables.
- An additional `ALTER TABLE` is issued after dumping the data, in order to create the previously omitted keys.

### `ALTER TABLE`

When `ALTER TABLE` requires a table copy, secondary keys are now dropped and recreated later, after copying the data. The following restrictions apply:

- Only non-unique keys can be involved in this optimization.
- If the table contains foreign keys, or a foreign key is being added as a part of the current `ALTER TABLE` statement, the optimization is disabled for all keys.

### `OPTIMIZE TABLE`

Internally, `OPTIMIZE TABLE` is mapped to `ALTER TABLE ... ENGINE=innodb` for *InnoDB* tables. As a consequence, it now also benefits from fast index creation, with the same restrictions as for `ALTER TABLE`.

## Caveats

*InnoDB* fast index creation uses temporary files in tmpdir for all indexes being created. So make sure you have enough tmpdir space when using *expand_fast_index_creation*. It is a session variable, so you can temporarily switch it off if you are short on tmpdir space and/or don't want this optimization to be used for a specific table.

There's also a number of cases when this optimization is not applicable:

- `UNIQUE` indexes in `ALTER TABLE` are ignored to enforce uniqueness where necessary when copying the data to a temporary table;
- `ALTER TABLE` and `OPTIMIZE TABLE` always process tables containing foreign keys as if *expand_fast_index_creation* is OFF to avoid dropping keys that are part of a FOREIGN KEY constraint;
- **mysqldump --innodb-optimize-keys** ignores foreign keys because *InnoDB* requires a full table rebuild on foreign key changes. So adding them back with a separate `ALTER TABLE` after restoring the data from a dump would actually make the restore slower;
- **mysqldump --innodb-optimize-keys** ignores indexes on `AUTO_INCREMENT` columns, because they must be indexed, so it is impossible to temporarily drop the corresponding index;
- **mysqldump --innodb-optimize-keys** ignores the first UNIQUE index on non-nullable columns when the table has no `PRIMARY KEY` defined, because in this case *InnoDB* picks such an index as the clustered one.

## System Variables

**variable expand_fast_index_creation**

**Command Line** Yes

**Config File** No

**Scope** Local/Global

**Dynamic** Yes

**Variable Type** Boolean

**Default Value** OFF

**Range** ON/OFF

**See also:**

**Improved InnoDB fast index creation** http://www.mysqlperformanceblog.com/2011/11/06/improved-innodb-fast-index-creation/

**Thinking about running OPTIMIZE on your InnoDB Table? Stop!** http://www.mysqlperformanceblog.com/2010/12/09/thinking-about-running-optimize-on-your-innodb-table-stop/

CHAPTER

# THIRTYONE

# BACKUP LOCKS

*Percona Server for MySQL* offers the `LOCK TABLES FOR BACKUP` statement as a lightweight alternative to `FLUSH TABLES WITH READ LOCK` for both physical and logical backups.

## LOCK TABLES FOR BACKUP

`LOCK TABLES FOR BACKUP` uses a new MDL lock type to block updates to non-transactional tables and DDL statements for all tables. If there is an active `LOCK TABLES FOR BACKUP` lock then all DDL statements and all updates to MyISAM, CSV, MEMORY, ARCHIVE, *TokuDB*, and *MyRocks* tables will be blocked in the `Waiting for backup lock` status, visible in `PERFORMANCE_SCHEMA` or `PROCESSLIST`.

`LOCK TABLES FOR BACKUP` has no effect on `SELECT` queries for all mentioned storage engines. Against *InnoDB*, *MyRocks*, Blackhole and Federated tables, the `LOCK TABLES FOR BACKUP` is not applicable to the `INSERT`, `REPLACE`, `UPDATE`, `DELETE` statements: Blackhole tables obviously have no relevance to backups, and Federated tables are ignored by both logical and physical backup tools.

Unlike `FLUSH TABLES WITH READ LOCK`, `LOCK TABLES FOR BACKUP` does not flush tables, i.e. storage engines are not forced to close tables and tables are not expelled from the table cache. As a result, `LOCK TABLES FOR BACKUP` only waits for conflicting statements to complete (i.e. DDL and updates to non-transactional tables). It never waits for SELECTs, or UPDATEs to *InnoDB* or *MyRocks* tables to complete, for example.

If an "unsafe" statement is executed in the same connection that is holding a `LOCK TABLES FOR BACKUP` lock, it fails with the following error:

```
ERROR 1880 (HY000): Can't execute the query because you have a conflicting backup lock

UNLOCK TABLES releases the lock acquired by LOCK TABLES FOR BACKUP.
```

The intended use case for *Percona XtraBackup* is:

```
LOCK TABLES FOR BACKUP
... copy .frm, MyISAM, CSV, etc. ...
UNLOCK TABLES
... get binlog coordinates ...
... wait for redo log copying to finish ...
```

## Privileges

The `LOCK TABLES FOR BACKUP` requires the `BACKUP_ADMIN` privilege.

## Interaction with other global locks

The `LOCK TABLES FOR BACKUP` has no effect if the current connection already owns a `FLUSH TABLES WITH READ LOCK` lock, as it is a more restrictive lock. If `FLUSH TABLES WITH READ LOCK` is executed in a connection that has acquired `LOCK TABLES FOR BACKUP`, `FLUSH TABLES WITH READ LOCK` fails with an error.

If the server is operating in the read-only mode (i.e. `read_only` set to `1`), statements that are unsafe for backups will be either blocked or fail with an error, depending on whether they are executed in the same connection that owns `LOCK TABLES FOR BACKUP` lock, or other connections.

## MyISAM index and data buffering

*MyISAM* key buffering is normally write-through, i.e. by the time each update to a *MyISAM* table is completed, all index updates are written to disk. The only exception is delayed key writing feature which is disabled by default.

When the global system variable `delay_key_write` is set to `ALL`, key buffers for all *MyISAM* tables are not flushed between updates, so a physical backup of those tables may result in broken *MyISAM* indexes. To prevent this, `LOCK TABLES FOR BACKUP` will fail with an error if `delay_key_write` is set to `ALL`. An attempt to set `delay_key_write` to `ALL` when there's an active backup lock will also fail with an error.

Another option to involve delayed key writing is to create *MyISAM* tables with the DELAY_KEY_WRITE option and set the `delay_key_write` variable to `ON` (which is the default). In this case, `LOCK TABLES FOR BACKUP` will not be able to prevent stale index files from appearing in the backup. Users are encouraged to set `delay_key_writes` to `OFF` in the configuration file, `my.cnf`, or repair *MyISAM* indexes after restoring from a physical backup created with backup locks.

*MyISAM* may also cache data for bulk inserts, e.g. when executing multi-row INSERTs or `LOAD DATA` statements. Those caches, however, are flushed between statements, so have no effect on physical backups as long as all statements updating *MyISAM* tables are blocked.

## The mysqldump Command

`mysqldump` has also been extended with a new option, *`lock-for-backup`* (disabled by default). When used together with the `--single-transaction` option, the option makes `mysqldump` issue `LOCK TABLES FOR BACKUP` before starting the dump operation to prevent unsafe statements that would normally result in an inconsistent backup.

When used without the `single-transaction` option, *`lock-for-backup`* is automatically converted to `lock-all-tables`.

The option *`lock-for-backup`* is mutually exclusive with `lock-all-tables`, i.e. specifying both on the command line will lead to an error.

If the backup locks feature is not supported by the target server, but *`lock-for-backup`* is specified on the command line, `mysqldump` aborts with an error.

## Version Specific Information

- **`8.0.12-1`** Feature ported from *Percona Server for MySQL* 5.7.

## System Variables

**variable `have_backup_locks`**

**Command Line** Yes

**Config File** No

**Scope** Global

**Dynamic** No

**Variable Type** Boolean

**Default Value** YES

This is a server variable implemented to help other utilities decide what locking strategy can be implemented for a server. When available, the backup locks feature is supported by the server and the variable value is always `YES`.

## Status Variables

**variable `Com_lock_tables_for_backup`**

**Variable Type** Numeric

**Scope** Global/Session

This status variable indicates the number of times the corresponding statements have been executed.

## Client Command Line Parameter

**option `lock-for-backup`**

**Command Line** Yes

**Scope** Global

**Dynamic** No

**Variable Type** String

**Default Value** Off

When used together with the `--single-transaction` option, the option makes `mysqldump` issue `LOCK TABLES FOR BACKUP` before starting the dump operation to prevent unsafe statements that would normally result in an inconsistent backup.

CHAPTER

# THIRTYTWO

# AUDIT LOG PLUGIN

Percona Audit Log Plugin provides monitoring and logging of connection and query activity that were performed on specific server. Information about the activity will be stored in the XML log file where each event will have its `NAME` field, its own unique `RECORD_ID` field and a `TIMESTAMP` field. This implementation is alternative to the MySQL Enterprise Audit Log Plugin

Audit Log plugin produces the log of following events:

- **Audit** - Audit event indicates that audit logging started or finished. `NAME` field will be `Audit` when logging started and `NoAudit` when logging finished. Audit record also includes server version and command-line arguments.

  Example of the Audit event:

```
<AUDIT_RECORD
"NAME"="Audit"
"RECORD"="1_2014-04-29T09:29:40"
"TIMESTAMP"="2014-04-29T09:29:40 UTC"
"MYSQL_VERSION"="5.6.17-65.0-655.trusty"
"STARTUP_OPTIONS"="--basedir=/usr --datadir=/var/lib/mysql --plugin-dir=/usr/lib/
↪mysql/plugin --user=mysql --log-error=/var/log/mysql/error.log --pid-file=/var/
↪run/mysqld/mysqld.pid --socket=/var/run/mysqld/mysqld.sock --port=3306"
"OS_VERSION"="x86_64-debian-linux-gnu",
/>
```

- **Connect/Disconnect** - Connect record event will have `NAME` field `Connect` when user logged in or login failed, or `Quit` when connection is closed. Additional fields for this event are `CONNECTION_ID`, `STATUS`, `USER`, `PRIV_USER`, `OS_LOGIN`, `PROXY_USER`, `HOST`, and `IP`. `STATUS` will be `0` for successful logins and non-zero for failed logins.

  Example of the Disconnect event:

```
<AUDIT_RECORD
"NAME"="Quit"
"RECORD"="24_2014-04-29T09:29:40"
"TIMESTAMP"="2014-04-29T10:20:13 UTC"
"CONNECTION_ID"="49"
"STATUS"="0"
"USER"=""
"PRIV_USER"=""
"OS_LOGIN"=""
"PROXY_USER"=""
"HOST"=""
"IP"=""
"DB"=""
/>
```

- **Query** - Additional fields for this event are: `COMMAND_CLASS` (values come from the `com_status_vars` array in the `sql/mysqld.cc`` file in a MySQL source distribution. Examples are `select`, `alter_table`, `create_table`, etc.), `CONNECTION_ID`, `STATUS` (indicates error when non-zero), `SQLTEXT` (text of SQL-statement), `USER`, `HOST`, `OS_USER`, `IP`. Possible values for the `NAME` name field for this event are `Query`, `Prepare`, `Execute`, `Change user`, etc.

  Example of the Query event:

```
<AUDIT_RECORD
"NAME"="Query"
"RECORD"="23_2014-04-29T09:29:40"
"TIMESTAMP"="2014-04-29T10:20:10 UTC"
"COMMAND_CLASS"="select"
"CONNECTION_ID"="49"
"STATUS"="0"
"SQLTEXT"="SELECT * from mysql.user"
"USER"="root[root] @ localhost []"
"HOST"="localhost"
"OS_USER"=""
"IP"=""
/>
```

## Installation

Audit Log plugin is shipped with *Percona Server for MySQL*, but it is not installed by default. To enable the plugin you must run the following command:

```
INSTALL PLUGIN audit_log SONAME 'audit_log.so';
```

You can check if the plugin is loaded correctly by running:

```
SHOW PLUGINS;
```

Audit log should be listed in the output:

```
+-------------------------------+----------+--------------------+--------------+-----
↪----+
| Name                          | Status   | Type               | Library      |␣
↪License |
+-------------------------------+----------+--------------------+--------------+-----
↪----+
...
| audit_log                     | ACTIVE   | AUDIT              | audit_log.so | GPL␣
↪    |
+-------------------------------+----------+--------------------+--------------+-----
↪----+
```

## Log Format

The audit log plugin supports four log formats: `OLD`, `NEW`, `JSON`, and `CSV`. `OLD` and `NEW` formats are based on XML, where the former outputs log record properties as XML attributes and the latter as XML tags. Information logged is the same in all four formats. The log format choice is controlled by *`audit_log_format`* variable.

Example of the `OLD` format:

```
<AUDIT_RECORD
 "NAME"="Query"
 "RECORD"="2_2014-04-28T09:29:40"
 "TIMESTAMP"="2014-04-28T09:29:40 UTC"
 "COMMAND_CLASS"="install_plugin"
 "CONNECTION_ID"="47"
 "STATUS"="0"
 "SQLTEXT"="INSTALL PLUGIN audit_log SONAME 'audit_log.so'"
 "USER"="root[root] @ localhost []"
 "HOST"="localhost"
 "OS_USER"=""
 "IP"=""
/>
```

Example of the `NEW` format:

```
<AUDIT_RECORD>
 <NAME>Quit</NAME>
 <RECORD>10902_2014-04-28T11:02:54</RECORD>
 <TIMESTAMP>2014-04-28T11:02:59 UTC</TIMESTAMP>
 <CONNECTION_ID>36</CONNECTION_ID>
 <STATUS>0</STATUS>
 <USER></USER>
 <PRIV_USER></PRIV_USER>
 <OS_LOGIN></OS_LOGIN>
 <PROXY_USER></PROXY_USER>
 <HOST></HOST>
 <IP></IP>
 <DB></DB>
</AUDIT_RECORD>
```

Example of the `JSON` format:

```
{"audit_record":{"name":"Query","record":"4707_2014-08-27T10:43:52","timestamp":"2014-
↪08-27T10:44:19 UTC","command_class":"show_databases","connection_id":"37","status
↪":0,"sqltext":"show databases","user":"root[root] @ localhost []","host":"localhost
↪","os_user":"","ip":""}}
```

Example of the `CSV` format:

```
"Query","49284_2014-08-27T10:47:11","2014-08-27T10:47:23 UTC","show_databases","37",0,
↪"show databases","root[root] @ localhost []","localhost","",""
```

## Streaming the audit log to syslog

To stream the audit log to syslog you'll need to set *audit_log_handler* variable to `SYSLOG`. To control the syslog file handler, the following variables can be used: *audit_log_syslog_ident*, *audit_log_syslog_facility*, and *audit_log_syslog_priority* These variables have the same meaning as appropriate parameters described in the syslog(3) manual.

---

**Note:** Variables: *audit_log_strategy*, *audit_log_buffer_size*, *audit_log_rotate_on_size*, *audit_log_rotations* have effect only with `FILE` handler.

---

# Filtering by user

The filtering by user feature adds two new global variables: *audit_log_include_accounts* and *audit_log_exclude_accounts* to specify which user accounts should be included or excluded from audit logging.

**Warning:** Only one of these variables can contain a list of users to be either included or excluded, while the other needs to be NULL. If one of the variables is set to be not NULL (contains a list of users), the attempt to set another one will fail. Empty string means an empty list.

**Note:** Changes of *audit_log_include_accounts* and *audit_log_exclude_accounts* do not apply to existing server connections.

## Example

Following example shows adding users who will be monitored:

```
mysql> SET GLOBAL audit_log_include_accounts = 'user1@localhost,root@localhost';
Query OK, 0 rows affected (0.00 sec)
```

If you you try to add users to both include and exclude lists server will show you the following error:

```
mysql> SET GLOBAL audit_log_exclude_accounts = 'user1@localhost,root@localhost';
ERROR 1231 (42000): Variable 'audit_log_exclude_accounts' can't be set to the value␣
↪of 'user1@localhost,root@localhost'
```

To switch from filtering by included user list to the excluded one or back, first set the currently active filtering variable to NULL:

```
mysql> SET GLOBAL audit_log_include_accounts = NULL;
Query OK, 0 rows affected (0.00 sec)

mysql> SET GLOBAL audit_log_exclude_accounts = 'user1@localhost,root@localhost';
Query OK, 0 rows affected (0.00 sec)

mysql> SET GLOBAL audit_log_exclude_accounts = "'user'@'host'";
Query OK, 0 rows affected (0.00 sec)

mysql> SET GLOBAL audit_log_exclude_accounts = '''user''@''host''';
Query OK, 0 rows affected (0.00 sec)

mysql> SET GLOBAL audit_log_exclude_accounts = '\'user\'@\'host\'';
Query OK, 0 rows affected (0.00 sec)
```

To see what users are currently in the on the list you can run:

```
mysql> SELECT @@audit_log_exclude_accounts;
+------------------------------+
| @@audit_log_exclude_accounts |
+------------------------------+
| 'user'@'host'                |
```

```
+-----------------------------+
1 row in set (0.00 sec)
```

Account names from `mysql.user` table are the one that are logged in the audit log. For example when you create a user:

```
mysql> CREATE USER 'user1'@'%' IDENTIFIED BY '111';
Query OK, 0 rows affected (0.00 sec)
```

This is what you'll see when `user1` connected from `localhost`:

```
 <AUDIT_RECORD
  NAME="Connect"
  RECORD="4971917_2016-08-22T09:09:10"
  TIMESTAMP="2016-08-22T09:12:21 UTC"
  CONNECTION_ID="6"
  STATUS="0"
  USER="user1" ;; this is a 'user' part of account in 8.0
  PRIV_USER="user1"
  OS_LOGIN=""
  PROXY_USER=""
  HOST="localhost" ;; this is a 'host' part of account in 8.0
  IP=""
  DB=""
/>
```

To exclude `user1` from logging in *Percona Server for MySQL* 8.0 you must set:

```
SET GLOBAL audit_log_exclude_accounts = 'user1@%';
```

The value can be `NULL` or comma separated list of accounts in form `user@host` or `'user'@'host'` (if user or host contains comma).

## Filtering by SQL command type

The filtering by SQL command type adds two new global variables: *audit_log_include_commands* and *audit_log_exclude_commands* to specify which command types should be included or excluded from audit logging.

**Warning:** Only one of these variables can contain a list of command types to be either included or excluded, while the other needs to be `NULL`. If one of the variables is set to be not `NULL` (contains a list of command types), the attempt to set another one will fail. Empty string means an empty list.

**Note:** If both *audit_log_exclude_commands* and *audit_log_include_commands* are `NULL` all commands will be logged.

### Example

The available command types can be listed by running:

```
mysql> SELECT name FROM performance_schema.setup_instruments WHERE name LIKE
↪"statement/sql/%" ORDER BY name;
+---------------------------------------+
| name                                  |
+---------------------------------------+
| statement/sql/alter_db                |
| statement/sql/alter_db_upgrade        |
| statement/sql/alter_event             |
| statement/sql/alter_function          |
| statement/sql/alter_procedure         |
| statement/sql/alter_server            |
| statement/sql/alter_table             |
| statement/sql/alter_tablespace        |
| statement/sql/alter_user              |
| statement/sql/analyze                 |
| statement/sql/assign_to_keycache      |
| statement/sql/begin                   |
| statement/sql/binlog                  |
| statement/sql/call_procedure          |
| statement/sql/change_db               |
| statement/sql/change_master           |
...
| statement/sql/xa_rollback             |
| statement/sql/xa_start                |
+---------------------------------------+
145 rows in set (0.00 sec)
```

You can add commands to the include filter by running:

```
mysql> SET GLOBAL audit_log_include_commands= 'set_option,create_db';
```

If you now create a database:

```
mysql> CREATE DATABASE world;
```

You'll see it the audit log:

```
<AUDIT_RECORD
  NAME="Query"
  RECORD="10724_2016-08-18T12:34:22"
  TIMESTAMP="2016-08-18T15:10:47 UTC"
  COMMAND_CLASS="create_db"
  CONNECTION_ID="61"
  STATUS="0"
  SQLTEXT="create database world"
  USER="root[root] @ localhost []"
  HOST="localhost"
  OS_USER=""
  IP=""
  DB=""
/>
```

To switch command type filtering type from included type list to excluded one or back, first reset the currently-active list to `NULL`:

```
mysql> SET GLOBAL audit_log_include_commands = NULL;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> SET GLOBAL audit_log_exclude_commands= 'set_option,create_db';
Query OK, 0 rows affected (0.00 sec)
```

**Note:** Invocation of stored procedures have command type `call_procedure`, and all the statements executed within the procedure have the same type `call_procedure` as well.

## Filtering by database

The filtering by an SQL database is implemented via two global variables: *audit_log_include_databases* and *audit_log_exclude_databases* to specify which databases should be included or excluded from audit logging.

**Warning:** Only one of these variables can contain a list of databases to be either included or excluded, while the other needs to be `NULL`. If one of the variables is set to be not `NULL` (contains a list of databases), the attempt to set another one will fail. Empty string means an empty list.

If query is accessing any of databases listed in *audit_log_include_databases*, the query will be logged. If query is accessing only databases listed in *audit_log_exclude_databases*, the query will not be logged. `CREATE TABLE` statements are logged unconditionally.

**Note:** Changes of *audit_log_include_databases* and *audit_log_exclude_databases* do not apply to existing server connections.

### Example

To add databases to be monitored you should run:

```
mysql> SET GLOBAL audit_log_include_databases = 'test,mysql,db1';
Query OK, 0 rows affected (0.00 sec)

mysql> SET GLOBAL audit_log_include_databases= 'db1,```db3"`';
Query OK, 0 rows affected (0.00 sec)
```

If you you try to add databases to both include and exclude lists server will show you the following error:

```
mysql> SET GLOBAL audit_log_exclude_databases = 'test,mysql,db1';
ERROR 1231 (42000): Variable 'audit_log_exclude_databases can't be set to the value␣
↪of 'test,mysql,db1'
```

To switch from filtering by included database list to the excluded one or back, first set the currently active filtering variable to `NULL`:

```
mysql> SET GLOBAL audit_log_include_databases = NULL;
Query OK, 0 rows affected (0.00 sec)

mysql> SET GLOBAL audit_log_exclude_databases = 'test,mysql,db1';
Query OK, 0 rows affected (0.00 sec)
```

## System Variables

**variable `audit_log_strategy`**

**Command Line** Yes

**Scope** Global

**Dynamic** No

**Variable Type** String

**Default Value** ASYNCHRONOUS

**Allowed values** `ASYNCHRONOUS`, `PERFORMANCE`, `SEMISYNCHRONOUS`, `SYNCHRONOUS`

This variable is used to specify the audit log strategy, possible values are:

- `ASYNCHRONOUS` - (default) log using memory buffer, do not drop messages if buffer is full
- `PERFORMANCE` - log using memory buffer, drop messages if buffer is full
- `SEMISYNCHRONOUS` - log directly to file, do not flush and sync every event
- `SYNCHRONOUS` - log directly to file, flush and sync every event

This variable has effect only when *`audit_log_handler`* is set to `FILE`.

**variable `audit_log_file`**

**Command Line** Yes

**Scope** Global

**Dynamic** No

**Variable Type** String

**Default Value** audit.log

This variable is used to specify the filename that's going to store the audit log. It can contain the path relative to the datadir or absolute path.

**variable `audit_log_flush`**

**Command Line** Yes

**Scope** Global

**Dynamic** Yes

**Variable Type** String

**Default Value** OFF

When this variable is set to `ON` log file will be closed and reopened. This can be used for manual log rotation.

**variable `audit_log_buffer_size`**

**Command Line** Yes

**Scope** Global

**Dynamic** No

**Variable Type** Numeric

**Default Value** 1 Mb

This variable can be used to specify the size of memory buffer used for logging, used when *audit_log_strategy* variable is set to ASYNCHRONOUS or PERFORMANCE values. This variable has effect only when *audit_log_handler* is set to FILE.

**variable audit_log_exclude_accounts**

**Command Line** Yes

**Scope** Global

**Dynamic** Yes

**Variable Type** String

This variable is used to specify the list of users for which *Filtering by user* is applied. The value can be NULL or comma separated list of accounts in form user@host or 'user'@'host' (if user or host contains comma). If this variable is set, then *audit_log_include_accounts* must be unset, and vice versa.

**variable audit_log_exclude_commands**

**Command Line** Yes

**Scope** Global

**Dynamic** Yes

**Variable Type** String

This variable is used to specify the list of commands for which *Filtering by SQL command type* is applied. The value can be NULL or comma separated list of commands. If this variable is set, then *audit_log_include_commands* must be unset, and vice versa.

**variable audit_log_exclude_databases**

**Command Line** Yes

**Scope** Global

**Dynamic** Yes

**Variable Type** String

This variable is used to specify the list of commands for which *Filtering by database* is applied. The value can be NULL or comma separated list of commands. If this variable is set, then *audit_log_include_databases* must be unset, and vice versa.

**variable audit_log_format**

**Command Line** Yes

**Scope** Global

**Dynamic** No

**Variable Type** String

**Default Value** OLD

**Allowed values** OLD, NEW, CSV, JSON

This variable is used to specify the audit log format. The audit log plugin supports four log formats: OLD, NEW, JSON, and CSV. OLD and NEW formats are based on XML, where the former outputs log record properties as XML attributes and the latter as XML tags. Information logged is the same in all four formats.

**variable audit_log_include_accounts**

**Command Line** Yes

**Scope** Global

**Dynamic** Yes

**Variable Type** String

This variable is used to specify the list of users for which *Filtering by user* is applied. The value can be `NULL` or comma separated list of accounts in form `user@host` or `'user'@'host'` (if user or host contains comma). If this variable is set, then *`audit_log_exclude_accounts`* must be unset, and vice versa.

**variable `audit_log_include_commands`**

**Command Line** Yes

**Scope** Global

**Dynamic** Yes

**Variable Type** String

This variable is used to specify the list of commands for which *Filtering by SQL command type* is applied. The value can be `NULL` or comma separated list of commands. If this variable is set, then *`audit_log_exclude_commands`* must be unset, and vice versa.

**variable `audit_log_include_databases`**

**Command Line** Yes

**Scope** Global

**Dynamic** Yes

**Variable Type** String

This variable is used to specify the list of commands for which *Filtering by database* is applied. The value can be `NULL` or comma separated list of commands. If this variable is set, then *`audit_log_exclude_databases`* must be unset, and vice versa.

**variable `audit_log_policy`**

**Command Line** Yes

**Scope** Global

**Dynamic** Yes

**Variable Type** String

**Default Value** ALL

**Allowed values** `ALL`, `LOGINS`, `QUERIES`, `NONE`

This variable is used to specify which events should be logged. Possible values are:

- `ALL` - all events will be logged
- `LOGINS` - only logins will be logged
- `QUERIES` - only queries will be logged
- `NONE` - no events will be logged

**variable `audit_log_rotate_on_size`**

**Command Line** Yes

**Scope** Global

**Dynamic** No

**Variable Type** Numeric

**Default Value** 0 (don't rotate the log file)

This variable is used to specify the maximum audit log file size. Upon reaching this size the log will be rotated. The rotated log files will be present in the same same directory as the current log file. A sequence number will be appended to the log file name upon rotation. This variable has effect only when *audit_log_handler* is set to FILE.

**variable audit_log_rotations**

**Command Line** Yes

**Scope** Global

**Dynamic** No

**Variable Type** Numeric

**Default Value** 0

This variable is used to specify how many log files should be kept when *audit_log_rotate_on_size* variable is set to non-zero value. This variable has effect only when *audit_log_handler* is set to FILE.

**variable audit_log_handler**

**Command Line** Yes

**Scope** Global

**Dynamic** No

**Variable Type** String

**Default Value** FILE

**Allowed values** FILE, SYSLOG

This variable is used to configure where the audit log will be written. If it is set to FILE, the log will be written into a file specified by *audit_log_file* variable. If it is set to SYSLOG, the audit log will be written to syslog.

**variable audit_log_syslog_ident**

**Command Line** Yes

**Scope** Global

**Dynamic** No

**Variable Type** String

**Default Value** percona-audit

This variable is used to specify the ident value for syslog. This variable has the same meaning as the appropriate parameter described in the syslog(3) manual.

**variable audit_log_syslog_facility**

**Command Line** Yes

**Scope** Global

**Dynamic** No

**Variable Type** String

**Default Value** LOG_USER

This variable is used to specify the facility value for syslog. This variable has the same meaning as the appropriate parameter described in the syslog(3) manual.

**variable `audit_log_syslog_priority`**

**Command Line** Yes

**Scope** Global

**Dynamic** No

**Variable Type** String

**Default Value** LOG_INFO

This variable is used to specify the `priority` value for syslog. This variable has the same meaning as the appropriate parameter described in the syslog(3) manual.

## Status Variables

**variable `Audit_log_buffer_size_overflow`**

**Variable Type** Numeric

**Scope** Global

The number of times an audit log entry was either dropped or written directly to the file due to its size being bigger than *`audit_log_buffer_size`* variable.

## Version Specific Information

- `8.0.12-1` Feature ported from *Percona Server for MySQL* 5.7
- *`8.0.15-6`* *`Audit_log_buffer_size_overflow`* variable implemented

CHAPTER

# THIRTYTHREE

# START TRANSACTION WITH CONSISTENT SNAPSHOT

*Percona Server for MySQL* has ported *MariaDB* enhancement for `START TRANSACTION WITH CONSISTENT SNAPSHOTS` feature to *MySQL* 5.6 group commit implementation. This enhancement makes binary log positions consistent with *InnoDB* transaction snapshots.

This feature is quite useful to obtain logical backups with correct positions without running a `FLUSH TABLES WITH READ LOCK`. Binary log position can be obtained by two newly implemented status variables: *`Binlog_snapshot_file`* and *`Binlog_snapshot_position`*. After starting a transaction using the `START TRANSACTION WITH CONSISTENT SNAPSHOT`, these two variables will provide you with the binlog position corresponding to the state of the database of the consistent snapshot so taken, irrespectively of which other transactions have been committed since the snapshot was taken.

## Snapshot Cloning

The *Percona Server for MySQL* implementation extends the `START TRANSACTION WITH CONSISTENT SNAPSHOT` syntax with the optional `FROM SESSION` clause:

```
START TRANSACTION WITH CONSISTENT SNAPSHOT FROM SESSION <session_id>;
```

When specified, all participating storage engines and binary log instead of creating a new snapshot of data (or binary log coordinates), create a copy of the snapshot which has been created by an active transaction in the specified session. `session_id` is the session identifier reported in the `Id` column of `SHOW PROCESSLIST`.

Currently snapshot cloning is only supported by *XtraDB* and the binary log. As with the regular `START TRANSACTION WITH CONSISTENT SNAPSHOT`, snapshot clones can only be created with the `REPEATABLE READ` isolation level.

For *XtraDB*, a transaction with a cloned snapshot will only see data visible or changed by the donor transaction. That is, the cloned transaction will see no changes committed by transactions that started after the donor transaction, not even changes made by itself. Note that in case of chained cloning the donor transaction is the first one in the chain. For example, if transaction A is cloned into transaction B, which is in turn cloned into transaction C, the latter will have read view from transaction A (i.e. the donor transaction). Therefore, it will see changes made by transaction A, but not by transaction B.

## mysqldump

`mysqldump` has been updated to use new status variables automatically when they are supported by the server and both `--single-transaction` and `--master-data` are specified on the command line. Along with the `mysqldump` improvements introduced in *Backup Locks* there is now a way to generate `mysqldump` backups that are guaranteed to be consistent without using `FLUSH TABLES WITH READ LOCK` even if `--master-data` is requested.

## System Variables

**variable `have_snapshot_cloning`**

**Command Line** Yes

**Config File** No

**Scope** Global

**Dynamic** No

**Variable Type** Boolean

This server variable is implemented to help other utilities detect if the server supports the `FROM SESSION` extension. When available, the snapshot cloning feature and the syntax extension to `START TRANSACTION WITH CONSISTENT SNAPSHOT` are supported by the server, and the variable value is always `YES`.

## Status Variables

**variable `Binlog_snapshot_file`**

**Variable Type** String

**Scope** Global

**variable `Binlog_snapshot_position`**

**Variable Type** Numeric

**Scope** Global

These status variables are only available when the binary log is enabled globally.

## Other Reading

- MariaDB Enhancements for START TRANSACTION WITH CONSISTENT SNAPSHOT

CHAPTER

# THIRTYFOUR

# EXTENDED SHOW GRANTS

In Oracle *MySQL* `SHOW GRANTS` displays only the privileges granted explicitly to the named account. Other privileges might be available to the account, but they are not displayed. For example, if an anonymous account exists, the named account might be able to use its privileges, but `SHOW GRANTS` will not display them. *Percona Server for MySQL* offers the `SHOW EFFECTIVE GRANTS` command to display all the effectively available privileges to the account, including those granted to a different account.

## Example

If we create the following users:

```
mysql> CREATE USER grantee@localhost IDENTIFIED BY 'grantee1';
Query OK, 0 rows affected (0.50 sec)

mysql> CREATE USER grantee IDENTIFIED BY 'grantee2';
Query OK, 0 rows affected (0.09 sec)

mysql> CREATE DATABASE db2;
Query OK, 1 row affected (0.20 sec)

mysql> GRANT ALL PRIVILEGES ON db2.* TO grantee WITH GRANT OPTION;
Query OK, 0 rows affected (0.12 sec)
```

- `SHOW EFFECTIVE GRANTS` output before the change:

```
mysql> SHOW EFFECTIVE GRANTS;
+-------------------------------------------------------------------------------------------
↪---------------------------+
| Grants for grantee@localhost
↪                           |
+-------------------------------------------------------------------------------------------
↪---------------------------+
| GRANT USAGE ON *.* TO 'grantee'@'localhost' IDENTIFIED BY PASSWORD
↪'*9823FF338D44DAF02422CF24DD1F879FB4F6B232' |
+-------------------------------------------------------------------------------------------
↪---------------------------+
1 row in set (0.04 sec)
```

Although the grant for the `db2` database isn't shown, `grantee` user has enough privileges to create the table in that database:

```
user@trusty:~$ mysql -ugrantee -pgrantee1 -h localhost
```

```
mysql> CREATE TABLE db2.t1(a int);
Query OK, 0 rows affected (1.21 sec)
```

- The output of `SHOW EFFECTIVE GRANTS` after the change shows all the privileges for the `grantee` user:

```
mysql> SHOW EFFECTIVE GRANTS;
+-------------------------------------------------------------------------------------------------
↪----------------------------+
| Grants for grantee@localhost
↪                            |
+-------------------------------------------------------------------------------------------------
↪----------------------------+
| GRANT USAGE ON *.* TO 'grantee'@'localhost' IDENTIFIED BY PASSWORD
↪'*9823FF338D44DAF02422CF24DD1F879FB4F6B232' |
| GRANT ALL PRIVILEGES ON `db2`.* TO 'grantee'@'%' WITH GRANT OPTION
↪                            |
+-------------------------------------------------------------------------------------------------
↪----------------------------+
2 rows in set (0.00 sec)
```

## Version-Specific Information

- `8.0.12-1`: Feature ported from *Percona Server for MySQL* 5.7.

## Other reading

- #53645 - `SHOW GRANTS` not displaying all the applicable grants

CHAPTER

THIRTYFIVE

# DATA AT REST ENCRYPTION

This page has been moved or been replaced. The new page is located here:

*Transparent Data Encryption*

Please update any bookmarks that point to the old page.

CHAPTER

THIRTYSIX

# SSL IMPROVEMENTS

This page has been moved or been replaced. The new page is located here:

*SSL Improvements*

Please update any bookmarks that point to the old page.

CHAPTER

# THIRTYSEVEN

# UTILITY USER

**Availability** This feature is **Experimental** quality.

*Percona Server for MySQL* has implemented ability to have a *MySQL* user who has system access to do administrative tasks but limited access to user schema. This feature is especially useful to those operating *MySQL* As A Service.

This user has a mixed and special scope of abilities and protection:

- Utility user will not appear in the mysql.user table and can not be modified by any other user, including root.
- Utility user will not appear in *USER_STATISTICS*, *CLIENT_STATISTICS* or *THREAD_STATISTICS* tables or in any performance_schema tables.
- Utility user's queries may appear in the general and slow logs.
- Utility user doesn't have the ability create, modify, delete or see any schemas or data not specified (except for information_schema).
- Utility user may modify all visible, non read-only system variables (see Expanded Program Option Modifiers functionality).
- Utility user may see, create, modify and delete other system users only if given access to the mysql schema.
- Regular users may be granted proxy rights to the utility user but any attempt to impersonate the utility user will fail. The utility user may not be granted proxy rights on any regular user. For example running: GRANT PROXY ON utility_user TO regular_user; will not fail, but any actual attempt to impersonate as the utility user will fail. Running: *GRANT PROXY ON regular_user TO utility_user;* will fail when utility_user is an exact match or is more specific than than the utility user specified.

When the server starts, it will note in the log output that the utility user exists and the schemas that it has access to.

In order to have the ability for a special type of MySQL user, which will have a very limited and special amount of control over the system and can not be see or modified by any other user including the root user, three new options have been added.

Option *utility_user* specifies the user which the system will create and recognize as the utility user. The host in the utility user specification follows conventions described in the MySQL manual, i.e. it allows wildcards and IP masks. Anonymous user names are not permitted to be used for the utility user name.

This user must not be an exact match to any other user that exists in the mysql.user table. If the server detects that the user specified with this option exactly matches any user within the mysql.user table on start up, the server will report an error and shut down gracefully. If host name wildcards are used and a more specific user specification is identified on start up, the server will report a warning and continue.

Example: `--utility_user` =frank@% and frank@localhost exists within the mysql.user table.

If a client attempts to create a MySQL user that matches this user specification exactly or if host name wildcards are used for the utility user and the user being created has the same name and a more specific host, the creation attempt will fail with an error.

Example: `--utility_user` =frank@% and CREATE USER 'frank@localhost';

As a result of these requirements, it is strongly recommended that a very unique user name and reasonably specific host be used and that any script or tools test that they are running within the correct user by executing 'SELECT CURRENT_USER()' and comparing the result against the known utility user.

Option *`utility_user_password`* specifies the password for the utility user and MUST be specified or the server will shut down gracefully with an error.

Example: `--utility_user_password` ='Passw0rD'

Option *`utility_user_schema_access`* specifies the name(s) of the schema(s) that the utility user will have access to read write and modify. If a particular schema named here does not exist on start up it will be ignored. If a schema by the name of any of those listed in this option is created after the server is started, the utility user will have full access to it.

Example: `--utility_user_schema_access` =schema1,schema2,schema3

Option *`utility_user_privileges`* allows a comma-separated list of extra access privileges to grant to the utility user.

Example: `--utility-user-privileges` ="CREATE,DROP,LOCK TABLES"

## Version Specific Information

- **`8.0.17-8`** Feature ported from *Percona Server for MySQL* 5.7

## System Variables

**variable `utility_user`**

**Command Line** Yes

**Config File** utility_user=<user@host>

**Scope** Global

**Dynamic** No

**Variable Type** String

**Default Value** NULL

Specifies a MySQL user that will be added to the internal list of users and recognized as the utility user.

**variable `utility_user_password`**

**Command Line** Yes

**Config File** utility_user_password=<password>

**Scope** Global

**Dynamic** No

**Variable Type** String

**Default Value** NULL

Specifies the password required for the utility user.

**variable `utility_user_schema_access`**

**Command Line** Yes

**Config File** utility_user_schema_access=<schema>,<schema>,<schema>

**Scope** Global

**Dynamic** No

**Variable Type** String

**Default Value** NULL

Specifies the schemas that the utility user has access to in a comma delimited list.

**variable `utility_user_privileges`**

**Command Line** Yes

**Config File** utility_user_privileges=<privilege1>,<privilege2>,<privilege3>

**Scope** Global

**Dynamic** No

**Variable Type** String

**Default Value** NULL

This variable can be used to specify a comma-separated list of extra access privileges to grant to the utility user. Supported values for the privileges list are: `SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, RELOAD, SHUTDOWN, PROCESS, FILE, GRANT, REFERENCES, INDEX, ALTER, SHOW DATABASES, SUPER, CREATE TEMPORARY TABLES, LOCK TABLES, EXECUTE, REPLICATION SLAVE, REPLICATION CLIENT, CREATE VIEW, SHOW VIEW, CREATE ROUTINE, ALTER ROUTINE, CREATE USER, EVENT, TRIGGER, CREATE TABLESPACE`

# Part VIII

# Security Improvements

CHAPTER

# THIRTYEIGHT

# PAM AUTHENTICATION PLUGIN

Percona PAM Authentication Plugin is a free and Open Source implementation of the *MySQL*'s authentication plugin. This plugin acts as a mediator between the *MySQL* server, the *MySQL* client, and the PAM stack. The server plugin requests authentication from the PAM stack, forwards any requests and messages from the PAM stack over the wire to the client (in cleartext) and reads back any replies for the PAM stack.

> PAM plugin uses dialog as its client side plugin. Dialog plugin can be loaded to any client application that uses `libperconaserverclient/libmysqlclient` library.

Here are some of the benefits that Percona dialog plugin offers over the default one:

- It correctly recognizes whether PAM wants input to be echoed or not, while the default one always echoes the input on the user's console.
- It can use the password which is passed to *MySQL* client via "-p" parameter.
- Dialog client installation bug has been fixed.
- This plugin works on *MySQL* and *Percona Server for MySQL*.

Percona offers two versions of this plugin:

- Full PAM plugin called *auth_pam*. This plugin uses *dialog.so*. It fully supports the PAM protocol with arbitrary communication between client and server.
- Oracle-compatible PAM called *auth_pam_compat*. This plugin uses *mysql_clear_password* which is a part of Oracle MySQL client. It also has some limitations, such as, it supports only one password input. You must use `-p` option in order to pass the password to *auth_pam_compat*.

These two versions of plugins are physically different. To choose which one you want used, you must use *IDENTIFIED WITH 'auth_pam'* for auth_pam, and *IDENTIFIED WITH 'auth_pam_compat'* for auth_pam_compat.

## Installation

This plugin requires manual installation because it isn't installed by default.

```
mysql> INSTALL PLUGIN auth_pam SONAME 'auth_pam.so';
```

After the plugin has been installed it should be present in the plugins list. To check if the plugin has been correctly installed and active

```
mysql> SHOW PLUGINS;
...
...
| auth_pam                      | ACTIVE   | AUTHENTICATION     | auth_pam.so | GPL ␣
↪   |
```

## Configuration

In order to use the plugin, authentication method should be configured. Simple setup can be to use the standard UNIX authentication method (`pam_unix`).

**Note:** To use `pam_unix`, mysql will need to be added to the shadow group in order to have enough privileges to read the /etc/shadow.

A sample */etc/pam.d/mysqld* file:

```
auth       required     pam_unix.so
account    required     pam_unix.so
```

For added information in the system log, you can expand it to be:

```
auth       required     pam_warn.so
auth       required     pam_unix.so audit
account    required     pam_unix.so audit
```

## Creating a user

After the PAM plugin has been configured, users can be created with the PAM plugin as authentication method

```
mysql> CREATE USER 'newuser'@'localhost' IDENTIFIED WITH auth_pam;
```

This will create a user `newuser` that can connect from `localhost` who will be authenticated using the PAM plugin. If the `pam_unix` method is being used user will need to exist on the system.

## Supplementary groups support

*Percona Server for MySQL* has implemented PAM plugin support for supplementary groups. Supplementary or secondary groups are extra groups a specific user is member of. For example user `joe` might be a member of groups: `joe` (his primary group) and secondary groups `developers` and `dba`. A complete list of groups and users belonging to them can be checked with `cat /etc/group` command.

This feature enables using secondary groups in the mapping part of the authentication string, like "`mysql, developers=joe, dba=mark`". Previously only primary groups could have been specified there. If user is a member of both `developers` and `dba`, PAM plugin will map it to the `joe` because `developers` matches first.

## Known issues

Default mysql stack size is not enough to handle `pam_ecryptfs` module. Workaround is to increase the *MySQL* stack size by setting the thread-stack variable to at least `512KB` or by increasing the old value by `256KB`.

PAM authentication can fail with `mysqld: pam_unix(mysqld:account): Fork failed: Cannot allocate memory` error in the `/var/log/secure` even when there is enough memory available. Current workaround is to set vm.overcommit_memory to `1`:

```
echo 1 > /proc/sys/vm/overcommit_memory
```

and by adding the `vm.overcommit_memory = 1` to `/etc/sysctl.conf` to make the change permanent after reboot. Authentication of internal (i.e. non PAM) accounts continues to work fine when `mysqld` reaches this memory utilization level. *NOTE:* Setting the `vm.overcommit_memory` to `1` will cause kernel to perform no memory overcommit handling which could increase the potential for memory overload and invoking of OOM killer.

## Version Specific Information

- `8.0.12-1` Feature ported from *Percona Server for MySQL* 5.7.

CHAPTER

THIRTYNINE

# TRANSPARENT DATA ENCRYPTION

Data security is a concern for institutions and organizations. `Transparent Data Encryption (TDE)` or `Data at Rest Encryption` encrypts data files. Data at rest is any data which is not accessed or changed frequently, stored on different types of storage devices. Encryption ensures that if an unauthorized user accesses the data files from the file system, the user cannot read contents.

If the user uses master key encryption, the MySQL keyring plugin stores the InnoDB master key, used for the master key encryption implemented by *MySQL*. The master key is also used to encrypt redo logs, and undo logs, along with the tablespaces.

The InnoDB tablespace encryption has the following components:

- The database instance has a master key for tablespaces and a master key for binary log encryption.
- Each tablespace has a tablespace key. The key is used to encrypt the Tablespace data pages. Encrypted tablespace keys are written on tablespace header. In the master key implementation, the tablespace key cannot be changed unless you rebuild the table.

Two separate keys allow the master key to be rotated in a minimal operation. When the master key is rotated, each tablespace key is decrypted and re-encrypted with the new master key. Only the first page of every tablespace (.ibd) file is read and written during the key rotation.

An InnoDB tablespace file is comprised of multiple logical and physical pages. Page 0 is the tablespace header page and keeps the metadata for the tablespace. The encryption information is stored on page 0 and the tablespace key is encrypted.

A buffer pool page is not encrypted. An encrypted page is decrypted at the I/O layer and added to the buffer pool and used to access the data. The page is encrypted by the I/O layer before the page is flushed to disk.

---

**Note:** *Percona XtraBackup* version 8 supports the backup of encrypted general tablespaces. Features which are not Generally Available (GA) in *Percona Server for MySQL* are not supported in version 8.

---

**See also:**

*Information about HashiCorp Vault*

*Using the Keyring Plugin*

*Encrypting File-Per-Tablespace Tables*

*Encrypting a Schema or a General Tablespace*

*Encrypting the System Tablespace*

*Encrypting Temporary Files*

*Verifying the Encryption for Tables, Tablespaces, and Schemas*

CHAPTER

FORTY

# INFORMATION ABOUT HASHICORP VAULT

The `keyring_vault` plugin can store the encryption keys inside the HashiCorp Vault.

**Important:** The `keyring_vault` plugin works with kv secrets engine version 1.

**See also:**

HashiCorp Documentation:

Installing Vault https://www.vaultproject.io/docs/install/index.html

KV Secrets Engine - Version 1 https://www.vaultproject.io/docs/secrets/kv/kv-v1.html

Production Hardening https://learn.hashicorp.com/vault/operations/production-hardening

**See also:**

*Using the Keyring Plugin*

*Rotating the Master Key*

CHAPTER

# FORTYONE

# USING THE KEYRING PLUGIN

*Percona Server for MySQL* may use the following plugins:

- *keyring_file* stores the keyring data locally
- *keyring_vault* provides an interface for the database with a HashiCorp Vault server to store key and secure encryption keys.

**Note:** The `keyring_file` plugin should not be used for regulatory compliance.

To install the plugin, follow the installing and uninstalling plugins instructions.

## Loading the Keyring Plugin

You should load the plugin at server startup with the `-early-plugin-load` option to enable keyrings.

We recommend the plugin should be loaded in the configuration file to facilitate recovery for encrypted tables. Also, the redo log and undo log encryption cannot be used without `--early-plugin-load`. The normal plugin load happens too late in startup.

To use the keyring_vault, you can add this option to your configuration file:

```
[mysqld]
early-plugin-load="keyring_vault=keyring_vault.so"
loose-keyring_vault_config="/home/mysql/keyring_vault.conf"
```

**Note:** The keyring_vault extension, ".so" and the file location for the vault configuration should be changed to match your operating system's extension and operating system location.

You could also run the following command which loads the keyring_file plugin:

```
$ mysqld --early-plugin-load="keyring_file=keyring_file.so"
```

**Warning:** Only one keyring plugin should be enabled at a time. Enabling multiple keyring plugins is not supported and may result in data loss.

**Note:** If server is started with different plugins loaded early, the `--early-plugin-load` option should contain the plugin names in a double-quoted list with each plugin name separated by a semicolon. The use of double quotes ensures the semicolons do not create issues when the list is executed in a script.

**See also:**

***MySQL* Documentation:**

- Installing a Keyring Plugin
- The ' –early-plugin-load Option

Apart from installing the plugin you also must set the *keyring_vault_config* variable to point to the keyring_vault configuration file.

The *keyring_vault_config* file has the following information:

- `vault_url` - the Vault server address
- **`secret_mount_point` - the mount point name where the *keyring_vault*** stores the keys
- `token` - a token generated by the Vault server
- `vault_ca [optional]` - if the machine does not trust the Vault's CA certificate, this variable points to the CA certificate used to sign the Vault's certificates

This is an example of a configuration file:

```
vault_url = https://vault.public.com:8202
secret_mount_point = secret
token = 58a20c08-8001-fd5f-5192-7498a48eaf20
vault_ca = /data/keyring_vault_confs/vault_ca.crt
```

**Warning:** Each `secret_mount_point` must be used by only one server. If multiple server use the same secret_mount_point, the behavior is unpredictable.

The first time a key is fetched from a *keyring*, the *keyring_vault* communicates with the Vault server to retrieve the key type and data.

A user-created key deletion is only possible with the use of the keyring_udf plugin and deletes the key from the in-memory hash map and the Vault server. You cannot delete system keys, such as the master key.

This plugin supports the SQL interface for keyring key management described in General-Purpose Keyring Key-Management Functions manual.

The plugin library contains keyring user-defined functions (UDFs) which allow access to the internal keyring service functions. To enable the functions you must enable the `keyring_udf` plugin:

```
mysql> INSTALL PLUGIN keyring_udf SONAME 'keyring_udf.so';
```

**Note:** The `keyring_udf` plugin must be installed. Attempts to use the UDFs without the `keyring_udf` plugin generates an error.

You must also create keyring encryption UDFs.

## System Variables

**variable `keyring_vault_config`**

**Command Line** `--keyring-vault-config`

**Dynamic** Yes

**Scope** Global

**Variable Type** Text

**Default Value**

This variable is used to define the location of the Keyring Vault plugin configuration file.

**variable `keyring_vault_timeout`**

**Command Line** `--keyring-vault-timeout`

**Dynamic** Yes

**Scope** Global

**Variable Type** Numeric

**Default Value** `15`

Set the duration in seconds for the Vault server connection timeout. The default value is `15`. The allowed range is from `0` to `86400`. The timeout can be also disabled to wait an infinite amount of time by setting this variable to `0`.

**See also:**

*Information about HashiCorp Vault*

*Rotating the Master Key*

CHAPTER

FORTYTWO

# ROTATING THE MASTER KEY

The Master key should be periodically rotated. You should rotate the key if you believe the key has been compromised. The Master key rotation changes the Master key and tablespace keys are re-encrypted and updated in the tablespace headers. The operation does not affect tablespace data.

If the master key rotation is interrupted, the rotation operation is rolled forward when the server restarts. InnoDB reads the encryption data from the tablespace header, if certain tablespace keys have been encrypted with the prior master key, InnoDB retrieves the master key from the keyring to decrypt the tablespace key. InnoDB re-encrypts the tablespace key with the new Master key.

To allow for Master Key rotation, you can encrypt an already encrypted InnoDB system tablespace with a new master key by running the following `ALTER INSTANCE` statement:

```
mysql> ALTER INSTANCE ROTATE INNODB MASTER KEY;
```

The rotation operation must complete before any tablespace encryption operation can begin.

---

**Note:** The rotation re-encrypts each tablespace key. The tablespace key is not changed. If you want to change a tablespace key, you should disable and then re-enable encryption.

---

CHAPTER

FORTYTHREE

# ENCRYPTING FILE-PER-TABLESPACE TABLES

InnoDB can use a tablespace file for each InnoDB table and creates and stores the table data and the indexes in a single data file. In this tablespace configuration, each table is stored in an .ibd file.

If you require a specific table to be encrypted, configure the InnoDB table stored in `innodb_file_per_table` tablespace. The default value is enabled for the *innodb_file_per_table* option, unless you have explicitly specified the `innodb_file_per_table` to be `OFF` in your my.cnf file.

The architecture for data at rest encryption has two tiers:

- Master key
- Tablespace keys.

For encryption, you must have the keyring plugin installed and enabled. The file_per_table tablespace inherits the schema default encryption setting,unless you explicitly define encryption in the `CREATE TABLE` statement.

An example of the `CREATE TABLE` statement:

```
mysql> CREATE TABLE myexample (id INT, mytext varchar(255)) ENCRYPTION='Y';
```

An example of an `ALTER TABLE` statement.

```
mysql> ALTER TABLE myexample ENCRYPTION='Y';
```

Without the `ENCRYPTION` option in the *ALTER TABLE* statement, the table's encryption state does not change. An encrypted table remains encrypted. An unencrypted table remains unencrypted.

**See also:**

*MySQL* Documentation: - File-Per-Table Encryption

**See also:**

*Encrypting a Schema or a General Tablespace*

*Encrypting Temporary Files*

CHAPTER

FORTYFOUR

# ENCRYPTING A SCHEMA OR A GENERAL TABLESPACE

*Percona Server for MySQL* uses the same encryption architecture as *MySQL*, a two-tier system consisting of a master key and tablespace keys. The master key can be changed, or rotated in the keyring, as needed. Each of the tablespace keys, when decrypted, remain the same.

The feature requires the keyring plugin.

## Setting the Default for Schemas and General Tablespace Encryption

The tables in a general tablespace are either all encrypted or all unencrypted. A tablespace cannot contain a mixture of encrypted tables and unencrypted tables.

In versions before *Percona Server for MySQL* 8.0.16-7, use the variable *innodb_encrypt_tables*.

**variable innodb_encrypt_tables**

**Command Line** `--innodb-encrypt-tables`

**Removed** version 8.0.16-7

**Dynamic** Yes

**Scope** Global

**Variable Type** Text

**Default Value** `OFF`

The variable is considered **deprecated** and was removed in version 8.0.16-7. The default setting is "OFF".

The encryption of a schema or a general tablespace is determined by the *default_table_encryption* variable unless you specify the ENCRYPTION clause in the CREATE SCHEMA or CREATE TABLESPACE statement. This variable is implemented in *Percona Server for MySQL* version 8.0.16-7.

You can set the *default_table_encryption* variable in an individual connection.

```
mysql> SET default_table_encryption=ON;
```

### System Variable

**variable default_table_encryption**

**Command Line** `default-table-encryption`

**Dynamic** Yes

**Scope** Session

**Variable Type** Text

**Default Value** OFF

Defines the default encryption setting for schemas and general tablespaces. The variable allows you to create or alter schemas or tablespaces without specifying the ENCRYPTION clause. The default encryption setting applies only to schemas and general tablespaces and is not applied to the MySQL system tablespace.

The variable has the following possible values:

### ON

New tables are encrypted. To create unencrypted tables add `ENCRYPTION="N"` to the `CREATE TABLE` or `ALTER TABLE` statement.

### OFF

By default, new tables are unencrypted. To create encrypted tables add `ENCRYPTION="Y"` to the `CREATE TABLE` or `ALTER TABLE` statement.

### FORCE

New tables are created with the Master key. Using the *ENCRYPTION=NO* to *CREATE TABLE* or *ALTER TABLE* generates an error and the table is not created or altered.

To encrypt an unencrypted table with an *ALTER TABLE* statement the *ENCRYPTION=YES* must be explicitly used.

### KEYRING_ON

**Availablilty** This value is **Experimental** quality.

New tables are created with the keyring as the default encryption. You may specify a numeric key identifier and use a specific *percona-innodb-* key from the keyring instead of the default key:

```
mysql> CREATE TABLE ... ENCRYPTION=`KEYRING` ENCRYPTION_KEY=ID=NEW_ID
```

*NEW_ID* is an unsigned 32-bit integer that refers to the numerical part of the *percona-innodb-* key. When you assign a numerical identifier in the *ENCRYPTION_KEY_ID* clause, the server uses the latest version of the corresponding key. For example, *ENCRYPTION_KEY_ID=2* refers to the latest version of the *percona_innodb-2* key from the keyring.

### FORCE_KEYRING

**Availablilty** This value is **Experimental** quality.

New tables are created encrypted and the keyring encryption is enforced.

### ONLINE_TO_KEYRING

**Availablilty** This value is **Experimental** quality.

It is only possible to apply the keyring encryption when creating or altering tables.

**Note:** The *ALTER TABLE* statement changes the current encryption mode only if you use the *ENCRYPTION* clause.

**See also:**

MySQL Documentation: default_table_encryption https://dev.mysql.com/doc/refman/8.0/en/server-system-variables.html

### Merge-sort-encryption

**variable `innodb_encrypt_online_alter_logs`**

**Command Line** `--innodb_encrypt-online-alter-logs`

**Dynamic** Yes

**Scope** Global

**Variable Type** Boolean

**Default Value** OFF

This variable simultaneously turns on the encryption of files used by InnoDB for full text search using parallel sorting, building indexes using merge sort, and online DDL logs created by InnoDB for online DDL. Encryption is available for file merges used in queries and backend processes.

## Setting Tablespace *ENCRYPTION* without the Default Setting

If you do not set the default encryption setting, you can create general tablespaces with the `ENCRYPTION` setting.

```
mysql> CREATE TABLESPACE tablespace_name ENCRYPTION='Y';
```

All tables contained in the tablespace are either encrypted or not encrypted. You cannot encrypted only some of the tables in a general tablespace. This feature extends the CREATE TABLESPACE statement to accept the `ENCRYPTION='Y/N'` option.

**Note:** Prior to *Percona Server for MySQL* 8.0.13, the `ENCRYPTION` option was specific to the `CREATE TABLE` or `SHOW CREATE TABLE` statement. As of *Percona Server for MySQL* 8.0.13, this option is a tablespace attribute and no longer allowed with the `CREATE TABLE` or `SHOW CREATE TABLE` statement except for file-per-table tablespaces.

In an encrypted general tablespace, an attempt to create an unencrypted table generates the following error:

```
mysql> CREATE TABLE t3 (a INT, b TEXT) TABLESPACE foo ENCRYPTION='N';
ERROR 1478 (HY0000): InnoDB: Tablespace 'foo' can contain only ENCRYPTED tables.
```

An attempt to create or to move any tables, including partitioned ones, to a general tablespace with an incompatible encryption setting are diagnosed and the process is aborted.

If you must move tables between incompatible tablespaces, create tables with the same structure in another tablespace and run `INSERT INTO SELECT` from each of the source tables into the destination tables.

## Exporting an Encrypted General Tablespace

You can only export encrypted file-per-table tablespaces

**See also:**

*Encrypting File-Per-Tablespace Tables*

*Encrypting the System Tablespace*

*Encrypting Temporary Files*

*Verifying the Encryption for Tables, Tablespaces, and Schemas*

CHAPTER

FORTYFIVE

# ENCRYPTING THE SYSTEM TABLESPACE

*Percona Server for MySQL* supports system tablespace encryption. The InnoDB system tablespace may be encrypted with the Master key encryption or may use keyring encryption with encryption threads.

**See also:**

*Working with Background Encryption Threads*.

The limitation is the following:

- You cannot convert the system tablespace from the encrypted state to the unencrypted state, or the unencrypted state to the encrypted state. If a conversion is needed, you should create a new instance with the system tablespace in the required state and transfer the user tables to that instance.

**Important:** A server instance initialized with the encrypted InnoDB system tablespace cannot be downgraded. It is not possible to parse encrypted InnoDB system tablespace pages in a version of *Percona Server for MySQL* lower than the version where the InnoDB system tablespace has been encrypted.

To enable system tablespace encryption, edit the my.cnf file with the following:

- Add the `innodb_sys_tablespace_encrypt`
- Edit the *innodb_sys_tablespace_encrypt* value to "ON"

System tablespace encryption can only be enabled with the `--initialize` option

You can create an encrypted table as follows:

```
mysql> CREATE TABLE table_name TABLESPACE=innodb_system ENCRYPTION='Y';
```

## System Variables

**variable `innodb_sys_tablespace_encrypt`**

**Command Line** `--innodb-sys-tablespace-encrypt`

**Dynamic** No

**Scope** Global

**Variable Type** Boolean

**Default Value** `OFF`

Enables the encryption of the InnoDB system tablespace.

**See also:**

*MySQL* Documentation: mysql system Tablespace Encryption https://dev.mysql.com/doc/refman/8.0/en/innodb-data-encryption.html#innodb-mysql-tablespace-encryption-enabling-disabling

***MySQL* Documentation: `--initialize` option** https://dev.mysql.com/doc/refman/8.0/en/server-options.html#option_mysqld_initialize

## Re-Encrypt the System Tablespace

You can re-encrypt the system tablespace key with master key rotation. When the master key is rotated, the tablespace key is decrypted and re-encrypte with the new master key. Only the first page of the tablespace (.ibd) file is read and written during the key rotation. The tables in the tablespace are not re-encrypted.

The command is as follows:

```
mysql> ALTER INSTANCE ROTATE INNODB MASTER KEY;
```

**See also:**

*Rotating the Master Key*

*Using the Keyring Plugin*

CHAPTER

# FORTYSIX

# ENCRYPTING TEMPORARY FILES

**Availability** This feature is of **Experimental** quality.

For InnoDB user-created temporary tables, created in a temporary tablespace file, use the *innodb_temp_tablespace_encrypt* variable.

**variable innodb_temp_tablespace_encrypt**

**Command Line** `innodb-temp-tablespace-encrypt`

**Dynamic** Yes

**Scope** Global

**Variable Type** Boolean

**Default Value** `OFF`

When this variable is set to `ON`, the server encrypts the global temporary tablespace (:file: *ibtmp** files) and the session temporary tablespaces (:file: *#innodb_temp/temp_*.ibt* files). The variable does not enforce the encryption of currently open temporary files and does not rebuild the system temporary tablespace to encrypt data which has already been written.

The `CREATE TEMPORARY TABLE` does not support the `ENCRYPTION` clause. The `TABLESPACE` clause cannot be set to innodb_temporary.

The global temporary tablespace datafile `ibtmp1` contains temporary table undo logs while intrinsic temporary tables and user-created temporary tables are located in the encrypted session temporary tablespace.

To create new temporary tablespaces unencrypted, the following variables must be set to `OFF` at runtime:

- *innodb_temp_tablespace_encrypt*
- *default_table_encryption*

Any existing encrypted user-created temporary files and intrinsic temporary tables remain in an encrypted session.

Temporary tables are only destroyed when the session is disconnected.

The *default_table_encryption* setting in my.cnf determines if a temporary table is encrypted.

If the *innodb_temp_tablespace_encrypt* = “OFF” and the *default_table_encryption* =”ON”, the user-created temporary tables are encrypted. The temporary tablespace datafile `ibtmp1`, which contains undo logs, is not encrypted.

If the `innodb_temp_tablespace_encrypt` is “ON” for the system tablespace, InnoDB generates an encryption key and encrypts the system temporary tablespace. If you reset the encryption to “OFF”, all subsequent pages are written to an unencrypted tablespace. Any generated keys are not erased to allow encrypted tables and undo data to be decrypted.

**Important:** To use this option, the keyring plugin must be loaded. If the keyring is not loaded the server generates an error and refuses to create new temporary tables.

Temporary files are currently used in *Percona Server for MySQL* for the following purposes:

- Filesort - for example, when you run a *SELECT* statement with *SQL_BIG_RESULT* hints
- Binary log transactional caches
- Group Replication caches

For each temporary file, an encryption key is generated locally and only maintained in memory for the lifetime of the temporary file and the key is discarded afterwards.

## System Variables

**variable `encrypt_tmp_files`**

**Command Line** `--encrypt_tmp_files`

**Dynamic** No

**Scope** Global

**Variable Type** Boolean

**Default Value** `OFF`

This variable turns "ON" the encryption of temporary files created by *Percona Server for MySQL*.

**See also:**

*MySQL* Documentation https://dev.mysql.com/doc/refman/8.0/en/create-temporary-table.html

**See also:**

*Using the Keyring Plugin*

*Encrypting the System Tablespace*

CHAPTER

# FORTYSEVEN

# ENCRYPTING BINARY LOGS

Binary log encryption at rest ensures the server-generated binary logs are encrypted in persistent storage.

After you have enabled the `binlog_encryption` option and the keyring is available, you can encrypt new binary logs and relay logs on disk. Only the data content is encrypted.

In replication, the master sends the stream of decrypted binary log events to a slave, in transport the data is encrypted by SSL connections. Master and slaves use separate keyring storages and are able to use different keyring plugins.

When an encrypted binary log is dumped, and the operation involves decryption, and done using `mysqlbinlog` with `--read-from-remote-server` option.

---

**Note:** The *–read-from-remote-server* option only applies to the binary logs. Encrypted relay logs can not be dumped or decrypted with this option.

---

Attempting a binary log encryption without the keyring generates a MySQL error.

The Binary log encryption uses two tiers:

- File password
- Binary log encryption key

The file password encrypts the content of a single binary file or relay log file. The binary log encryption key is used to encrypt the file password and is stored in the keyring.

## Enabling Binary Log Encryption

To enable the `binlog_encryption` option you must set the option in a startup configuration file, such as the my.cnf file.

```
binlog_encryption=ON
```

## Verifying the Encryption Setting

To verify if the binary log encryption option is enabled, run the following statement:

```
mysql>SHOW BINARY LOGS;

+------------------+----------------+---------------+
| Log_name         | File_size      | Encrypted     |
```

```
+------------------+----------------+---------------+
| binlog.00011     | 72367          | No            |
| binlog:00012     | 71503          | No            |
| binlog:00013     | 73762          | Yes           |
+------------------+----------------+---------------+
```

**See also:**

*MySQL* Documentation: Encrypting Binary Log Files and Relay Log Files

## Upgrading from *Percona Server for MySQL* 8.0.15-5 to any Higher Version

Starting from release *8.0.15-5*, *Percona Server for MySQL* uses the upstream implementation of binary log encryption. The variable *encrypt-binlog* is removed and the related command line option *–encrypt-binlog* is not supported. It is important to remove the *encrypt-binlog* variable from your configuration file before you attempt to upgrade either from another release in the *Percona Server for MySQL* 8.0 series or from *Percona Server for MySQL* 5.7. Otherwise, a server boot error will be generated reporting an unknown variable. The implemented binary log encryption is compatible with the older format. The encrypted binary log used in a previous version of MySQL 8.0 series or Percona Server for MySQL series is supported.

**variable `encrypt_binlog`**

**Version-info** removed in *8.0.15-5*

**Command Line** `--encrypt-binlog`

**Dynamic** No

**Scope** Global

**Variable Type** Boolean

**Default Value** `OFF`

The variable turns on binary log and relay log encryption.

**See also:**

*Encrypting File-Per-Tablespace Tables*

*Encrypting a Schema or a General Tablespace*

*Encrypting the System Tablespace*

*Encrypting Temporary Files*

*Encrypting Doublewrite Buffers*

*Encrypting the Redo Log*

*Encrypting the Undo Tablespace*

CHAPTER

# FORTYEIGHT

# ENCRYPTING THE REDO LOG

The Redo log can be encrypted with the :variable: *innodb_redo_log_encrypt* variable. The default value for the variable is `OFF`. The Redo log uses the tablespace encryption key.

**variable `innodb_redo_log_encrypt`**

> **Command Line** `--innodb-redo-log-encrypt`
>
> **Dynamic** Yes
>
> **Scope** Global
>
> **Variable Type** Text
>
> **Default Value** OFF

Determines the encryption for redo log data for tables. The encryption of redo log data, by default, is 'OFF'.

When you enable *innodb_redo_log_encrypt* any existing redo log pages stay unencrypted, and new pages are encrypted when they are written to disk. If you disable *innodb_redo_log_encrypt*, any encrypted pages remain encrypted, but new pages are unencrypted.

As implemented in `8.0.16-7`, the supported values for :variable: *innodb_redo_log_encrypt* are the following:

- ON
- OFF
- master_key
- keyring_key

The keyring_key is an **Experimental** value.

**See also:**

For more information on the keyring_key - *Working with Background Encryption Threads*

---

**Note:** For *innodb_redo_log_encrypt*, the "ON" value is a compatibility alias for master_key.

---

After starting the server, an attempt to encrypt the redo log fails in the following conditions:

- Server started with no keyring specified
- Server started with a keyring, but you have specified a different redo log encryption method that what the same server previously used.

**See also:**

*Encrypting File-Per-Tablespace Tables*

*Encrypting a Schema or a General Tablespace*

CHAPTER

FORTYNINE

# ENCRYPTING THE UNDO TABLESPACE

The undo data may contain sensitive information about the database operations.

You can encrypt the data in an undo log using the *innodb_undo_log_encrypt* option. You can change the setting for this variable in the configuration file, as a startup parameter, or during runtime as a global variable. The undo data encryption must be enabled; the feature is disabled by default.

**variable innodb_undo_log_encrypt**

**Command Line** --innodb_undo-log_encrypt

**Dynamic** Yes

**Scope** Global

**Variable Type** Boolean

**Default Value** OFF

Defines if an undo log data is encrypted. The default for the undo log is "OFF", which disables the encryption.

You can create up to 127 undo tablespaces and you can, with the server running, add or reduce the number of undo tablespaces.

---

**Note:** If you disable encryption, any encrypted undo data remains encrypted. To remove this data, truncate the undo tablespace.

---

**See also:**

*MySQL* Documentation

innodb_undo_log_encrypt

## How to Enable Encryption on an Undo Log

You enable encryption for an undo log by adding the following to the my.cnf file:

```
[mysqld]
innodb_undo_log_encrypt=ON
```

**See also:**

*Encrypting the Redo Log*

CHAPTER

FIFTY

# WORKING WITH BACKGROUND ENCRYPTION THREADS

**Availabiliity** This feature is **Experimental**.

Encryption threads in the background allow you to perform some encryption and decryption tasks in real-time.

You would use encryption threads for the following purposes:

- Encryption threads can encrypt existing tablespaces. Encryption threads allow encryption to be applied to all or some of the existing tablespaces, you can exclude tablespaces from rotation, in a background process. You can encrypt existing tablespaces with the Master key, but you must do this operation by tablespace.
- Encryption threads encrypt tables with a key from a keyring. The Master key encrypts tables by a key and is stored in the encryption header of the tablespace.
- Encryption threads allow key rotation. In an encryption thread rotation, the operation re-encrypts each tablespace page by page. The Master key rotation does not re-encrypt each page, only the tablespace encryption header.

If you have tablespaces encrypted with the Master key and you enable encryption threads, the tablespaces are re-encrypted with the keyring key in a background process.

---

**Note:** While encryption threads are enabled, you cannot convert the tablespaces to Master key encryption. To convert the tablespaces, you must disable the encryption threads.

---

**Availability** This feature is **Experimental** quality.

**variable `innodb_encryption_threads`**

**Command Line** `--innodb-encryption-threads`

**Dynamic** Yes

**Scope** Global

**Variable Type** Numeric

**Default Value** 0

This variable works in combination with the *`default_table_encryption`* variable set to `ONLINE_TO_KEYRING`. This variable configures the number of threads for background encryption. For the online encryption, the value must be greater than **zero**.

**variable `innodb_online_encryption_rotate_key_age`**

**Command Line** `--innodb-online-encryption-rotate-key-age`

**Dynamic** Yes

**Scope** Global

**Variable Type** Numeric

**Default Value** 1

Defines the rotation for the re-encryption of a table encrypted using KEYRING. The value of this variable determines the how frequently the encrypted tables are re-encrypted.

For example, the following values would trigger a re-encryption in the following intervals:

- The value is **1**, the table is re-encrypted on each key rotation.
- The value is **2**, the table is re-encrypted on every other key rotation.
- The value is **10**, the table is re-encrypted on every tenth key rotation.

You should select the value which best fits your operational requirements.

## Using Keyring Encryption

**Availability** This feature is **Experimental** quality.

Keyring management is enabled for each table, per file table, separately when you set encryption in the `ENCRYPTION` clause to `KEYRING` in the supported SQL statement.

- CREATE TABLE ... ENCRYPTION='KEYRING'
- ALTER TABLE ... ENCRYPTION='KEYRING'

---

**Note:** Running an `ALTER TABLE ... ENCRYPTION='N'` on a table created with `ENCRYPTION='KEYRING'` converts the table to the existing MySQL schema, tablespace, or table encryption state.

---

**See also:**

*Using the Keyring Plugin*

CHAPTER

FIFTYONE

# ENCRYPTING DOUBLEWRITE BUFFERS

The two types of doublewrite buffers used in *Percona Server for MySQL* are encrypted differently.

When the InnoDB system tablespace is encrypted, the `doublewrite buffer` pages are encrypted as well. The key which was used to encrypt the InnoDB system tablespace is also used to encrypt the doublewrite buffer.

*Percona Server for MySQL* encrypts the `parallel doublewrite buffer` with the respective tablespace keys. Only encrypted tablespace pages are written as encrypted in the parallel doublewrite buffer. Unencrypted tablespace pages will be written as unencrypted.

**variable `innodb_parallel_dblwr_encrypt`**

**Command Line** `--innodb-parallel-dblwr-encrypt`

**Dynamic** Yes

**Scope** Global

**Variable Type** Boolean

**Default Value** `OFF`

Enables the encryption of the parallel doublewrite buffer. For encryption, uses the key of the tablespace where the parallel doublewrite buffer is used.

**See also:**

*Encrypting the System Tablespace*

*Encrypting a Schema or a General Tablespace*

*Encrypting File-Per-Tablespace Tables*

CHAPTER

FIFTYTWO

# VERIFYING THE ENCRYPTION FOR TABLES, TABLESPACES, AND SCHEMAS

If a general tablespace contains tables, check the table information to see if the table is encrypted. When the general tablespace contains no tables, you may verify if the tablespace is encrypted or not.

For single tablespaces, verify the ENCRYPTION option using *INFORMATION_SCHEMA.TABLES* and the *CREATE OPTIONS* settings.

```
mysql> SELECT TABLE_SCHEMA, TABLE_NAME, CREATE_OPTIONS FROM
       INFORMATION_SCHEMA.TABLES WHERE CREATE_OPTIONS LIKE '%ENCRYPTION%';

+---------------------+-------------------+------------------------------+
| TABLE_SCHEMA        | TABLE_NAME        | CREATE_OPTIONS               |
+---------------------+-------------------+------------------------------+
|sample               | t1                | ENCRYPTION="Y"               |
+---------------------+-------------------+------------------------------+
```

A `flag` field in the `INFORMATION_SCHEMA.INNODB_TABLESPACES` has bit number 13 set if the tablespace is encrypted. This bit can be checked with the `flag & 8192` expression in the following way:

```
SELECT space, name, flag, (flag & 8192) != 0 AS encrypted FROM
INFORMATION_SCHEMA.INNODB_TABLESPACES WHERE name in ('foo', 'test/t2', 'bar',
'noencrypt');
```

**Output**

```
+-------+-----------+-------+-----------+
| space | name      | flag  | encrypted |
+-------+-----------+-------+-----------+
|    29 | foo       | 10240 |      8192 |
|    30 | test/t2   |  8225 |      8192 |
|    31 | bar       | 10240 |      8192 |
|    32 | noencrypt |  2048 |         0 |
+-------+-----------+-------+-----------+
4 rows in set (0.01 sec)
```

**Availabiliity** This feature is **Experimental**.

The encrypted table metadata is contained in the INFORMATION_SCHEMA.INNODB_TABLESPACES_ENCRYPTION table. You must have the `Process` privilege to view the table information.

**Note:** This table is **Experimental** and may change in future releases.

```
>desc INNODB_TABLESPACES_ENCRYPTION:

+-----------------------------+---------------------+-----+----+---------+-------+
| Field                       | Type                | Null| Key| Default| Extra|
+-----------------------------+---------------------+-----+----+---------+-------+
| SPACE                       | int(11) unsigned    | NO  |    |         |       |
| NAME                        | varchar(655)        | YES |    |         |       |
| ENCRYPTION_SCHEME           | int(11) unsigned    | NO  |    |         |       |
| KEYSERVER_REQUESTS          | int(11) unsigned    | NO  |    |         |       |
| MIN_KEY_VERSION             | int(11) unsigned    | NO  |    |         |       |
| CURRENT_KEY_VERSION         | int(11) unsigned    | NO  |    |         |       |
| KEY_ROTATION_PAGE_NUMBER    | bigint(21) unsigned| YES |    |         |       |
| KEY_ROTATION_MAX_PAGE_NUMBER| bigint(21) unsigned| YES |    |         |       |
| CURRENT_KEY_ID              | int(11) unsigned    | NO  |    |         |       |
| ROTATING_OR_FLUSHING        | int(1) unsigned     | NO  |    |         |       |
+-----------------------------+---------------------+-----+----+---------+-------+
```

To identify encryption-enabled schemas, query the INFORMATION_SCHEMA.SCHEMATA table:

```
mysql> SELECT SCHEMA_NAME, DEFAULT_ENCRYPTION FROM
INFORMATION_SCHEMA.SCHEMATA WHERE DEFAULT_ENCRYPTION='YES';

+-----------------------------+---------------------------------+
| SCHEMA_NAME                 | DEFAULT_ENCRYPTION              |
+-----------------------------+---------------------------------+
| samples                     | YES                             |
+-----------------------------+---------------------------------+
```

**Note:** The `SHOW CREATE SCHEMA` statement returns the `DEFAULT ENCRYPTION` clause.

**See also:**

***MariaDB* Documentation** https://mariadb.com/kb/en/library/information-schema-innodb_tablespaces_encryption-table/

CHAPTER

# FIFTYTHREE

# DATA SCRUBBING

**Availability** This feature is **Experimental** quality

While data encryption ensures that the existing data are not stored in plain form, the data scrubbing literally removes the data once the user decides they should be deleted. Compare this behavior with how the `DELETE` statement works which only marks the affected data as *deleted* - the space claimed by this data is overwritten with new data later.

Once enabled, data scrubbing works automatically on each tablespace separately. To enable data scrubbing, you need to set the following variables:

- `innodb-background-scrub-data-uncompressed`
- `innodb-background-scrub-data-compressed`

Uncompressed tables can also be scrubbed immediately, independently of key rotation or background threads. This can be enabled by setting the variable `innodb-immediate-scrub-data-uncompressed`. This option is not supported for compressed tables.

Note that data scrubbing is made effective by setting the `innodb_online_encryption_threads` variable to a value greater than **zero**.

## System Variables

**variable `innodb_background_scrub_data_compressed`**

**Command Line** `--innodb-background-scrub-data-compressed`

**Dynamic** Yes

**Scope** Global

**Variable Type** Boolean

**Default Value** `OFF`

**variable `innodb_background_scrub_data_uncompressed`**

**Command Line** `--innodb-background-scrub-data-uncompressed`

**Dynamic** Yes

**Scope** Global

**Variable Type** Boolean

**Default Value** `OFF`

**See also:**

**Vault Documentation** https://www.vaultproject.io/docs/index.html

**General-Purpose Keyring Key-Management Functions** https://dev.mysql.com/doc/refman/8.0/en/keyring-udfs-general-purpose.html

CHAPTER

# FIFTYFOUR

# SSL IMPROVEMENTS

By default, *Percona Server for MySQL* passes elliptic-curve crypto-based ciphers to OpenSSL, such as ECDHE-RSA-AES128-GCM-SHA256.

**Note:** Although documented as supported, elliptic-curve crypto-based ciphers do not work with *MySQL*.

**See also:**

***MySQL* Bug System (solved for *Percona Server for MySQL*):** #82935 Cipher ECDHE-RSA-AES128-GCM-SHA256 listed in man/Ssl_cipher_list, not supported

CHAPTER

# FIFTYFIVE

# DATA MASKING

This feature is **Experimental** quality.

This feature was implemented in *Percona Server for MySQL* version `8.0.17-8`.

The Percona Data Masking plugin is a free and Open Source implementation of the *MySQL*'s data masking plugin. Data Masking provides a set of functions to hide sensitive data with modified content.

The data masking functions are the following:

| Type | Description | Sample |
|---|---|---|
| mask_inner() | Masks the inner part of a string. The string ends are not masked. | `mysql> SELECT mask_inner('123456789', 1, 1);`<br>`+-----------+-----------+-----------+-----+`<br>`\| mask_inner("123456789", 1, 1) \|`<br>`+-----------+-----------+-----------+-----+`<br>`\|1XXXXXXX9\|`<br>`+-----------+-----------+-----------+-----+` |
| mask_outer() | Masks the outer part of the string. The inner section is not masked. | `mysql> SELECT mask_outer('123456789', 2, 2);`<br>`+-----------+-----------+-----------+-----+`<br>`\| mask_outer("123456789", 2, 2). \|`<br>`+-----------+-----------+-----------+-----+`<br>`\|XX34567XX\|`<br>`+-----------+-----------+-----------+-----+` |

**See also:**

*MySQL* Documentation https://dev.mysql.com/doc/refman/8.0/en/data-masking-reference.html

# Part IX

# Diagnostics Improvements

CHAPTER

# FIFTYSIX

# USER STATISTICS

This feature adds several `INFORMATION_SCHEMA` tables, several commands, and the userstat variable. The tables and commands can be used to understand the server activity better and identify the source of the load.

The functionality is disabled by default, and must be enabled by setting `userstat` to `ON`. It works by keeping several hash tables in memory. To avoid contention over global mutexes, each connection has its own local statistics, which are occasionally merged into the global statistics, and the local statistics are then reset to 0.

## Version Specific Information

- **`8.0.12-1:`** Feature ported from *Percona Server for MySQL* 5.7.

## Other Information

- **Author/Origin:** *Google*; *Percona* added the `INFORMATION_SCHEMA` tables and the *`userstat`* variable.

## System Variables

**variable `userstat`**

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dynamic** Yes

**Variable Type** BOOLEAN

**Default Value** OFF

**Range** ON/OFF

Enables or disables collection of statistics. The default is `OFF`, meaning no statistics are gathered. This is to ensure that the statistics collection doesn't cause any extra load on the server unless desired.

**variable `thread_statistics`**

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dynamic** Yes

**Variable Type** BOOLEAN

**Default Value** OFF

**Range** ON/OFF

Enables or disables collection of thread statistics. The default is `OFF`, meaning no thread statistics are gathered. This is to ensure that the statistics collection doesn't cause any extra load on the server unless desired. Variable *`userstat`* needs to be enabled as well in order for thread statistics to be collected.

## INFORMATION_SCHEMA Tables

**table** `INFORMATION_SCHEMA.`**`CLIENT_STATISTICS`**

**Columns**

- **`CLIENT`** – The IP address or hostname from which the connection originated.
- **`TOTAL_CONNECTIONS`** – The number of connections created for this client.
- **`CONCURRENT_CONNECTIONS`** – The number of concurrent connections for this client.
- **`CONNECTED_TIME`** – The cumulative number of seconds elapsed while there were connections from this client.
- **`BUSY_TIME`** – The cumulative number of seconds there was activity on connections from this client.
- **`CPU_TIME`** – The cumulative CPU time elapsed, in seconds, while servicing this client''s connections.
- **`BYTES_RECEIVED`** – The number of bytes received from this client's connections.
- **`BYTES_SENT`** – The number of bytes sent to this client's connections.
- **`BINLOG_BYTES_WRITTEN`** – The number of bytes written to the binary log from this client's connections.
- **`ROWS_FETCHED`** – The number of rows fetched by this client's connections.
- **`ROWS_UPDATED`** – The number of rows updated by this client's connections.
- **`TABLE_ROWS_READ`** – The number of rows read from tables by this client's connections. (It may be different from `ROWS_FETCHED`.)
- **`SELECT_COMMANDS`** – The number of `SELECT` commands executed from this client's connections.
- **`UPDATE_COMMANDS`** – The number of `UPDATE` commands executed from this client's connections.
- **`OTHER_COMMANDS`** – The number of other commands executed from this client's connections.
- **`COMMIT_TRANSACTIONS`** – The number of `COMMIT` commands issued by this client's connections.
- **`ROLLBACK_TRANSACTIONS`** – The number of `ROLLBACK` commands issued by this client's connections.

- **DENIED_CONNECTIONS** – The number of connections denied to this client.
- **LOST_CONNECTIONS** – The number of this client's connections that were terminated uncleanly.
- **ACCESS_DENIED** – The number of times this client's connections issued commands that were denied.
- **EMPTY_QUERIES** – The number of times this client's connections sent empty queries to the server.

This table holds statistics about client connections. The Percona version of the feature restricts this table's visibility to users who have the `SUPER` or `PROCESS` privilege.

Example:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.CLIENT_STATISTICS\G
*************************** 1. row ***************************
                CLIENT: 10.1.12.30
     TOTAL_CONNECTIONS: 20
CONCURRENT_CONNECTIONS: 0
        CONNECTED_TIME: 0
             BUSY_TIME: 93
              CPU_TIME: 48
        BYTES_RECEIVED: 5031
            BYTES_SENT: 276926
  BINLOG_BYTES_WRITTEN: 217
          ROWS_FETCHED: 81
          ROWS_UPDATED: 0
       TABLE_ROWS_READ: 52836023
       SELECT_COMMANDS: 26
       UPDATE_COMMANDS: 1
        OTHER_COMMANDS: 145
   COMMIT_TRANSACTIONS: 1
 ROLLBACK_TRANSACTIONS: 0
    DENIED_CONNECTIONS: 0
      LOST_CONNECTIONS: 0
         ACCESS_DENIED: 0
         EMPTY_QUERIES: 0
```

**table** `INFORMATION_SCHEMA.`**INDEX_STATISTICS**

**Columns**

- **TABLE_SCHEMA** – The schema (database) name.
- **TABLE_NAME** – The table name.
- **INDEX_NAME** – The index name (as visible in `SHOW CREATE TABLE`).
- **ROWS_READ** – The number of rows read from this index.

This table shows statistics on index usage. An older version of the feature contained a single column that had the `TABLE_SCHEMA`, `TABLE_NAME` and `INDEX_NAME` columns concatenated together. The *Percona* version of the feature separates these into three columns. Users can see entries only for tables to which they have `SELECT` access.

This table makes it possible to do many things that were difficult or impossible previously. For example, you can use it to find unused indexes and generate DROP commands to remove them.

Example:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INDEX_STATISTICS
   WHERE TABLE_NAME='tables_priv';
+--------------+-----------------------+--------------------+-----------+
| TABLE_SCHEMA | TABLE_NAME            | INDEX_NAME         | ROWS_READ |
+--------------+-----------------------+--------------------+-----------+
| mysql        | tables_priv           | PRIMARY            |         2 |
+--------------+-----------------------+--------------------+-----------+
```

**Note:** Current implementation of index statistics doesn't support partitioned tables.

**table** INFORMATION_SCHEMA.**TABLE_STATISTICS**

**Columns**

- **TABLE_SCHEMA** – The schema (database) name.
- **TABLE_NAME** – The table name.
- **ROWS_READ** – The number of rows read from the table.
- **ROWS_CHANGED** – The number of rows changed in the table.
- **ROWS_CHANGED_X_INDEXES** – The number of rows changed in the table, multiplied by the number of indexes changed.

This table is similar in function to the INDEX_STATISTICS table.

Example:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.TABLE_STATISTICS
   WHERE TABLE_NAME=``tables_priv``;
+--------------+-------------------------------+-----------+--------------+-----------
↪-------------+
| TABLE_SCHEMA | TABLE_NAME                    | ROWS_READ | ROWS_CHANGED | ROWS_
↪CHANGED_X_INDEXES |
+--------------+-------------------------------+-----------+--------------+-----------
↪-------------+
| mysql        | tables_priv                   |         2 |            0 |          ␣
↪           0 |
+--------------+-------------------------------+-----------+--------------+-----------
↪-------------+
```

**Note:** Current implementation of table statistics doesn't support partitioned tables.

**table** INFORMATION_SCHEMA.**THREAD_STATISTICS**

**Columns**

- **THREAD_ID** – Thread ID
- **TOTAL_CONNECTIONS** – The number of connections created from this thread.
- **CONNECTED_TIME** – The cumulative number of seconds elapsed while there were connections from this thread.
- **BUSY_TIME** – The cumulative number of seconds there was activity from this thread.
- **CPU_TIME** – The cumulative CPU time elapsed while servicing this thread.
- **BYTES_RECEIVED** – The number of bytes received from this thread.

- **BYTES_SENT** – The number of bytes sent to this thread.
- **BINLOG_BYTES_WRITTEN** – The number of bytes written to the binary log from this thread.
- **ROWS_FETCHED** – The number of rows fetched by this thread.
- **ROWS_UPDATED** – The number of rows updated by this thread.
- **TABLE_ROWS_READ** – The number of rows read from tables by this tread.
- **SELECT_COMMANDS** – The number of `SELECT` commands executed from this thread.
- **UPDATE_COMMANDS** – The number of `UPDATE` commands executed from this thread.
- **OTHER_COMMANDS** – The number of other commands executed from this thread.
- **COMMIT_TRANSACTIONS** – The number of `COMMIT` commands issued by this thread.
- **ROLLBACK_TRANSACTIONS** – The number of `ROLLBACK` commands issued by this thread.
- **DENIED_CONNECTIONS** – The number of connections denied to this thread.
- **LOST_CONNECTIONS** – The number of thread connections that were terminated uncleanly.
- **ACCESS_DENIED** – The number of times this thread issued commands that were denied.
- **EMPTY_QUERIES** – The number of times this thread sent empty queries to the server.
- **TOTAL_SSL_CONNECTIONS** – The number of thread connections that used SSL.

In order for this table to be populated with statistics, additional variable *`thread_statistics`* should be set to `ON`.

**table** `INFORMATION_SCHEMA.`**USER_STATISTICS**

**Columns**

- **USER** – The username. The value `#mysql_system_user#` appears when there is no username (such as for the slave SQL thread).
- **TOTAL_CONNECTIONS** – The number of connections created for this user.
- **CONCURRENT_CONNECTIONS** – The number of concurrent connections for this user.
- **CONNECTED_TIME** – The cumulative number of seconds elapsed while there were connections from this user.
- **BUSY_TIME** – The cumulative number of seconds there was activity on connections from this user.
- **CPU_TIME** – The cumulative CPU time elapsed, in seconds, while servicing this user's connections.
- **BYTES_RECEIVED** – The number of bytes received from this user's connections.
- **BYTES_SENT** – The number of bytes sent to this user's connections.
- **BINLOG_BYTES_WRITTEN** – The number of bytes written to the binary log from this user's connections.
- **ROWS_FETCHED** – The number of rows fetched by this user's connections.
- **ROWS_UPDATED** – The number of rows updated by this user's connections.

- **TABLE_ROWS_READ** – The number of rows read from tables by this user's connections. (It may be different from ROWS_FETCHED.)
- **SELECT_COMMANDS** – The number of SELECT commands executed from this user's connections.
- **UPDATE_COMMANDS** – The number of UPDATE commands executed from this user's connections.
- **OTHER_COMMANDS** – The number of other commands executed from this user's connections.
- **COMMIT_TRANSACTIONS** – The number of COMMIT commands issued by this user's connections.
- **ROLLBACK_TRANSACTIONS** – The number of ROLLBACK commands issued by this user's connections.
- **DENIED_CONNECTIONS** – The number of connections denied to this user.
- **LOST_CONNECTIONS** – The number of this user's connections that were terminated uncleanly.
- **ACCESS_DENIED** – The number of times this user's connections issued commands that were denied.
- **EMPTY_QUERIES** – The number of times this user's connections sent empty queries to the server.

This table contains information about user activity. The *Percona* version of the patch restricts this table's visibility to users who have the SUPER or PROCESS privilege.

The table gives answers to questions such as which users cause the most load, and whether any users are being abusive. It also lets you measure how close to capacity the server may be. For example, you can use it to find out whether replication is likely to start falling behind.

Example:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.USER_STATISTICS\G
*************************** 1. row ***************************
                  USER: root
     TOTAL_CONNECTIONS: 5592
CONCURRENT_CONNECTIONS: 0
        CONNECTED_TIME: 6844
             BUSY_TIME: 179
              CPU_TIME: 72
        BYTES_RECEIVED: 603344
            BYTES_SENT: 15663832
  BINLOG_BYTES_WRITTEN: 217
          ROWS_FETCHED: 9793
          ROWS_UPDATED: 0
       TABLE_ROWS_READ: 52836023
       SELECT_COMMANDS: 9701
       UPDATE_COMMANDS: 1
        OTHER_COMMANDS: 2614
   COMMIT_TRANSACTIONS: 1
 ROLLBACK_TRANSACTIONS: 0
    DENIED_CONNECTIONS: 0
      LOST_CONNECTIONS: 0
         ACCESS_DENIED: 0
         EMPTY_QUERIES: 0
```

## Commands Provided

- FLUSH CLIENT_STATISTICS
- FLUSH INDEX_STATISTICS
- FLUSH TABLE_STATISTICS
- FLUSH THREAD_STATISTICS
- FLUSH USER_STATISTICS

These commands discard the specified type of stored statistical information.

- SHOW CLIENT_STATISTICS
- SHOW INDEX_STATISTICS
- SHOW TABLE_STATISTICS
- SHOW THREAD_STATISTICS
- SHOW USER_STATISTICS

These commands are another way to display the information you can get from the INFORMATION_SCHEMA tables. The commands accept WHERE clauses. They also accept but ignore LIKE clauses.

## Status Variables

**variable Com_show_client_statistics**

**Variable Type** numeric

**Scope** Global/Session

The *Com_show_client_statistics* statement counter variable indicates the number of times the statement SHOW CLIENT_STATISTICS has been executed.

**variable Com_show_index_statistics**

**Variable Type** numeric

**Scope** Global/Session

The *Com_show_index_statistics* statement counter variable indicates the number of times the statement SHOW INDEX_STATISTICS has been executed.

**variable Com_show_table_statistics**

**Variable Type** numeric

**Scope** Global/Session

The *Com_show_table_statistics* statement counter variable indicates the number of times the statement SHOW TABLE_STATISTICS has been executed.

**variable Com_show_thread_statistics**

**Variable Type** numeric

**Scope** Global/Session

The *Com_show_thread_statistics* statement counter variable indicates the number of times the statement SHOW THREAD_STATISTICS has been executed.

**variable `Com_show_user_statistics`**

**Variable Type** numeric

**Scope** Global/Session

The *`Com_show_user_statistics`* statement counter variable indicates the number of times the statement `SHOW USER_STATISTICS` has been executed.

CHAPTER

FIFTYSEVEN

# SLOW QUERY LOG

This feature adds microsecond time resolution and additional statistics to the slow query log output. It lets you enable or disable the slow query log at runtime, adds logging for the slave SQL thread, and adds fine-grained control over what and how much to log into the slow query log.

You can use *Percona-Toolkit*'s pt-query-digest tool to aggregate similar queries together and report on those that consume the most execution time.

## Version Specific Information

- **`8.0.12-1`:**
  - Feature ported from *Percona Server for MySQL* 5.7.

## System Variables

**variable `log_slow_filter`**

**Command Line** Yes

**Config File** Yes

**Scope** Global, Session

**Dynamic** Yes

Filters the slow log by the query's execution plan. The value is a comma-delimited string, and can contain any combination of the following values:

- `full_scan`: The query performed a full table scan.
- `full_join`: The query performed a full join (a join without indexes).
- `tmp_table`: The query created an implicit internal temporary table.
- `tmp_table_on_disk`: The query's temporary table was stored on disk.
- `filesort`: The query used a filesort.
- `filesort_on_disk`: The filesort was performed on disk.

Values are OR'ed together. If the string is empty, then the filter is disabled. If it is not empty, then queries will only be logged to the slow log if their execution plan matches one of the types of plans present in the filter.

For example, to log only queries that perform a full table scan, set the value to `full_scan`. To log only queries that use on-disk temporary storage for intermediate results, set the value to `tmp_table_on_disk, filesort_on_disk`.

**variable `log_slow_rate_type`**

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dynamic** Yes

**Variable Type** Enumerated

**Default Value** `session`

**Range** `session`, `query`

Specifies semantic of *`log_slow_rate_limit`* - `session` or `query`.

**variable `log_slow_rate_limit`**

**Command Line** Yes

**Config File** Yes

**Scope** Global, session

**Dynamic** Yes

**Default Value** 1

**Range** 1-1000

Behavior of this variable depends from *`log_slow_rate_type`*.

Specifies that only a fraction of `session/query` should be logged. Logging is enabled for every nth `session/query`. By default, n is 1, so logging is enabled for every `session/query`. Please note: when *`log_slow_rate_type`* is `session` rate limiting is disabled for the replication thread.

**Logging all queries might consume I/O bandwidth and cause the log file to grow large.**

- When *`log_slow_rate_type`* is `session`, this option lets you log full sessions, so you have complete records of sessions for later analysis; but you can rate-limit the number of sessions that are logged. Note that this feature will not work well if your application uses any type of connection pooling or persistent connections. Note that you change *`log_slow_rate_limit`* in `session` mode, you should reconnect for get effect.
- When *`log_slow_rate_type`* is `query`, this option lets you log just some queries for later analysis. For example, if you set the value to 100, then one percent of queries will be logged.

Note that every query has global unique `query_id` and every connection can has it own (session) *`log_slow_rate_limit`*. Decision "log or no" calculated in following manner:

- if `log_slow_rate_limit` is 1 - log every query
- If `log_slow_rate_limit` > 1 - randomly log every 1/`log_slow_rate_limit` query.

This allows flexible setup logging behavior.

For example, if you set the value to 100, then one percent of `sessions/queries` will be logged. In *Percona Server for MySQL* information about the *`log_slow_rate_limit`* has been added to the slow query log. This means that if the *`log_slow_rate_limit`* is effective it will be reflected in the slow query log for each written query. Example of the output looks like this:

```
Log_slow_rate_type: query  Log_slow_rate_limit: 10
```

**variable `log_slow_sp_statements`**

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dynamic** Yes

**Variable Type** Boolean

**Default Value** TRUE

**Range** TRUE/FALSE

If `TRUE`, statements executed by stored procedures are logged to the slow if it is open.

***Percona Server for MySQL* implemented improvements for logging of stored procedures to the slow query log:**

- Each query from a stored procedure is now logged to the slow query log individually
- `CALL` itself isn't logged to the slow query log anymore as this would be counting twice for the same query which would lead to incorrect results
- Queries that were called inside of stored procedures are annotated in the slow query log with the stored procedure name in which they run.

Example of the improved stored procedure slow query log entry:

```
mysql> DELIMITER //
mysql> CREATE PROCEDURE improved_sp_log()
       BEGIN
        SELECT * FROM City;
        SELECT * FROM Country;
       END//
mysql> DELIMITER ;
mysql> CALL improved_sp_log();
```

When we check the slow query log after running the stored procedure ,with variable:*log_slow_sp_statements* set to `TRUE`, it should look like this:

```
# Time: 150109 11:38:55
# User@Host: root[root] @ localhost []
# Thread_id: 40  Schema: world  Last_errno: 0  Killed: 0
# Query_time: 0.012989  Lock_time: 0.000033  Rows_sent: 4079  Rows_examined: 4079
↪Rows_affected: 0  Rows_read: 4079
# Bytes_sent: 161085
# Stored routine: world.improved_sp_log
SET timestamp=1420803535;
SELECT * FROM City;
# User@Host: root[root] @ localhost []
# Thread_id: 40  Schema: world  Last_errno: 0  Killed: 0
# Query_time: 0.001413  Lock_time: 0.000017  Rows_sent: 4318  Rows_examined: 4318
↪Rows_affected: 0  Rows_read: 4318
# Bytes_sent: 194601
# Stored routine: world.improved_sp_log
SET timestamp=1420803535;
```

If variable *log_slow_sp_statements* is set to FALSE:

- Entry is added to a slow-log for a CALL statement only and not for any of the individual statements run in that stored procedure
- Execution time is reported for the CALL statement as the total execution time of the CALL including all its statements

If we run the same stored procedure with the variable *log_slow_sp_statements* is set to FALSE slow query log should look like this:

```
# Time: 150109 11:51:42
# User@Host: root[root] @ localhost []
# Thread_id: 40  Schema: world  Last_errno: 0  Killed: 0
# Query_time: 0.013947  Lock_time: 0.000000  Rows_sent: 4318  Rows_examined: 4318
↪Rows_affected: 0  Rows_read: 4318
# Bytes_sent: 194612
SET timestamp=1420804302;
CALL improved_sp_log();
```

**Note:** Support for logging stored procedures doesn't involve triggers, so they won't be logged even if this feature is enabled.

**variable `log_slow_verbosity`**

**Command Line** Yes

**Config File** Yes

**Scope** Global, session

**Dynamic** Yes

Specifies how much information to include in your slow log. The value is a comma-delimited string, and can contain any combination of the following values:

- `microtime`: Log queries with microsecond precision.
- `query_plan`: Log information about the query's execution plan.
- `innodb`: Log *InnoDB* statistics.
- `minimal`: Equivalent to enabling just `microtime`.
- `standard`: Equivalent to enabling `microtime,innodb`.
- `full`: Equivalent to all other values OR'ed together without the `profiling` and `profiling_use_getrusage` options.
- `profiling`: Enables profiling of all queries in all connections.
- `profiling_use_getrusage`: Enables usage of the getrusage function.

Values are OR'ed together.

For example, to enable microsecond query timing and *InnoDB* statistics, set this option to `microtime,innodb` or `standard`. To turn all options on, set the option to `full`.

**variable `slow_query_log_use_global_control`**

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dynamic** Yes

**Default Value** None

Specifies which variables have global scope instead of local. For such variables, the global variable value is used in the current session, but without copying this value to the session value. Value is a "flag" variable - you can specify multiple values separated by commas

- `none`: All variables use local scope
- `log_slow_filter`: Global variable *`log_slow_filter`* has effect (instead of local)
- `log_slow_rate_limit`: Global variable *`log_slow_rate_limit`* has effect (instead of local)
- `log_slow_verbosity`: Global variable *`log_slow_verbosity`* has effect (instead of local)
- `long_query_time`: Global variable `long_query_time` has effect (instead of local)
- `min_examined_row_limit`: Global variable `min_examined_row_limit` has effect (instead of local)
- `all` Global variables has effect (instead of local)

**variable `slow_query_log_always_write_time`**

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dynamic** Yes

**Default Value** 10

This variable can be used to specify the query execution time after which the query will be written to the slow query log. It can be used to specify an additional execution time threshold for the slow query log, that, when exceeded, will cause a query to be logged unconditionally, that is, *`log_slow_rate_limit`* will not apply to it.

# Other Information

## Changes to the Log Format

The feature adds more information to the slow log output. Here is a sample log entry:

```
# Time: 130601  8:01:06.058915
# User@Host: root[root] @ localhost []  Id:    42
# Schema: imdb  Last_errno: 0  Killed: 0
# Query_time: 7.725616  Lock_time: 0.000328  Rows_sent: 4  Rows_examined: 1543720 ␣
↪Rows_affected: 0
# Bytes_sent: 272  Tmp_tables: 0  Tmp_disk_tables: 0  Tmp_table_sizes: 0
# Full_scan: Yes  Full_join: No  Tmp_table: No  Tmp_table_on_disk: No
# Filesort: No  Filesort_on_disk: No  Merge_passes: 0
SET timestamp=1370073666;
SELECT id,title,production_year FROM title WHERE title = 'Bambi';
```

Another example (*`log_slow_verbosity`* =`profiling`):

```
# Time: 130601  8:03:20.700441
# User@Host: root[root] @ localhost []  Id:    43
# Schema: imdb  Last_errno: 0  Killed: 0
# Query_time: 7.815071  Lock_time: 0.000261  Rows_sent: 4  Rows_examined: 1543720
↪Rows_affected: 0
# Bytes_sent: 272
# Profile_starting: 0.000125 Profile_starting_cpu: 0.000120
Profile_checking_permissions: 0.000021 Profile_checking_permissions_cpu: 0.000021
Profile_Opening_tables: 0.000049 Profile_Opening_tables_cpu: 0.000048 Profile_init: 0.
↪000048
Profile_init_cpu: 0.000049 Profile_System_lock: 0.000049 Profile_System_lock_cpu: 0.
↪000048
Profile_optimizing: 0.000024 Profile_optimizing_cpu: 0.000024 Profile_statistics: 0.
↪000036
Profile_statistics_cpu: 0.000037 Profile_preparing: 0.000029 Profile_preparing_cpu: 0.
↪000029
Profile_executing: 0.000012 Profile_executing_cpu: 0.000012 Profile_Sending_data: 7.
↪814583
Profile_Sending_data_cpu: 7.811634 Profile_end: 0.000013 Profile_end_cpu: 0.000012
Profile_query_end: 0.000014 Profile_query_end_cpu: 0.000014 Profile_closing_tables: 0.
↪000023
Profile_closing_tables_cpu: 0.000023 Profile_freeing_items: 0.000051
Profile_freeing_items_cpu: 0.000050 Profile_logging_slow_query: 0.000006
Profile_logging_slow_query_cpu: 0.000006
# Profile_total: 7.815085 Profile_total_cpu: 7.812127
SET timestamp=1370073800;
SELECT id,title,production_year FROM title WHERE title = 'Bambi';
```

Notice that the `Killed: `` keyword is followed by zero when the query successfully completes. If the query was killed, the ``Killed:` keyword is followed by a number other than zero:

| Killed Numeric Code | Exception |
|---|---|
| 0 | NOT_KILLED |
| 1 | KILL_BAD_DATA |
| 1053 | ER_SERVER_SHUTDOWN (see *MySQL* Documentation) |
| 1317 | ER_QUERY_INTERRUPTED (see *MySQL* Documentation) |
| 3024 | ER_QUERY_TIMEOUT (see *MySQL* Documentation) |
| Any other number | KILLED_NO_VALUE (Catches all other cases) |

**See also:**

***MySQL* Documentation: *MySQL* Server Error Codes** https://dev.mysql.com/doc/refman/8.0/en/server-error-reference.html

## Connection and Schema Identifier

Each slow log entry now contains a connection identifier, so you can trace all the queries coming from a single connection. This is the same value that is shown in the Id column in `SHOW FULL PROCESSLIST` or returned from the `CONNECTION_ID()` function.

Each entry also contains a schema name, so you can trace all the queries whose default database was set to a particular schema.

```
# Id: 43  Schema: imdb
```

### Microsecond Time Resolution and Extra Row Information

This is the original functionality offered by the `microslow` feature. `Query_time` and `Lock_time` are logged with microsecond resolution.

The feature also adds information about how many rows were examined for `SELECT` queries, and how many were analyzed and affected for `UPDATE`, `DELETE`, and `INSERT` queries,

```
# Query_time: 0.962742  Lock_time: 0.000202  Rows_sent: 4  Rows_examined: 1543719
↪Rows_affected: 0
```

Values and context:

- `Rows_examined`: Number of rows scanned - `SELECT`
- `Rows_affected`: Number of rows changed - `UPDATE`, `DELETE`, `INSERT`

### Memory Footprint

The feature provides information about the amount of bytes sent for the result of the query and the number of temporary tables created for its execution - differentiated by whether they were created on memory or on disk - with the total number of bytes used by them.

```
# Bytes_sent: 8053  Tmp_tables: 1  Tmp_disk_tables: 0  Tmp_table_sizes: 950528
```

Values and context:

- `Bytes_sent`: The amount of bytes sent for the result of the query
- `Tmp_tables`: Number of temporary tables created on memory for the query
- `Tmp_disk_tables`: Number of temporary tables created on disk for the query
- `Tmp_table_sizes`: Total Size in bytes for all temporary tables used in the query

### Query Plan Information

Each query can be executed in various ways. For example, it may use indexes or do a full table scan, or a temporary table may be needed. These are the things that you can usually see by running `EXPLAIN` on the query. The feature will now allow you to see the most important facts about the execution in the log file.

```
# Full_scan: Yes  Full_join: No  Tmp_table: No  Tmp_table_on_disk: No
# Filesort: No  Filesort_on_disk: No  Merge_passes: 0
```

The values and their meanings are documented with the *`log_slow_filter`* option.

### *InnoDB* Usage Information

The final part of the output is the *InnoDB* usage statistics. *MySQL* currently shows many per-session statistics for operations with `SHOW SESSION STATUS`, but that does not include those of *InnoDB*, which are always global and shared by all threads. This feature lets you see those values for a given query.

```
#   InnoDB_IO_r_ops: 6415  InnoDB_IO_r_bytes: 105103360  InnoDB_IO_r_wait: 0.001279
#   InnoDB_rec_lock_wait: 0.000000  InnoDB_queue_wait: 0.000000
#   InnoDB_pages_distinct: 6430
```

Values:

- `innodb_IO_r_ops`: Counts the number of page read operations scheduled. The actual number of read operations may be different, but since this can be done asynchronously, there is no good way to measure it.
- `innodb_IO_r_bytes`: Similar to innodb_IO_r_ops, but the unit is bytes.
- `innodb_IO_r_wait`: Shows how long (in seconds) it took *InnoDB* to actually read the data from storage.
- `innodb_rec_lock_wait`: Shows how long (in seconds) the query waited for row locks.
- `innodb_queue_wait`: Shows how long (in seconds) the query spent either waiting to enter the *InnoDB* queue or inside that queue waiting for execution.
- `innodb_pages_distinct`: Counts approximately the number of unique pages the query accessed. The approximation is based on a small hash array representing the entire buffer pool, because it could take a lot of memory to map all the pages. The inaccuracy grows with the number of pages accessed by a query, because there is a higher probability of hash collisions.

If the query did not use *InnoDB* tables, that information is written into the log instead of the above statistics.

## Related Reading

- Impact of logging on MySQL's performance
- log_slow_filter Usage
- Blueprint in Launchpad

CHAPTER

FIFTYEIGHT

# EXTENDED SHOW ENGINE *INNODB* STATUS

This feature reorganizes the output of `SHOW ENGINE INNODB STATUS` to improve readability and to provide additional information. The variable *`innodb_show_locks_held`* controls the umber of locks held to print for each *InnoDB* transaction.

This feature modified the `SHOW ENGINE INNODB STATUS` command as follows:

- Added extended information about *InnoDB* internal hash table sizes (in bytes) in the `BUFFER POOL AND MEMORY` section; also added buffer pool size in bytes.
- Added additional LOG section information.

## Other Information

- Author / Origin: Baron Schwartz, http://lists.mysql.com/internals/35174

## System Variables

**variable `innodb_show_locks_held`**

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dynamic** Yes

**Variable Type** ULONG

**Default Value** 10

**Range** 0 - 1000

Specifies the number of locks held to print for each *InnoDB* transaction in `SHOW ENGINE INNODB STATUS`.

**variable `innodb_print_lock_wait_timeout_info`**

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dynamic** Yes

**Variable Type** Boolean

**Default Value** OFF

Makes *InnoDB* to write information about all lock wait timeout errors into the log file.

This allows to find out details about the failed transaction, and, most importantly, the blocking transaction. Query string can be obtained from `performance_schema.events_statements_current` table, based on the `PROCESSLIST_ID` field, which corresponds to `thread_id` from the log output.

Taking into account that blocking transaction is often a multiple statement one, folowing query can be used to obtain blocking thread statements history:

```
SELECT s.SQL_TEXT FROM performance_schema.events_statements_history s
INNER JOIN performance_schema.threads t ON t.THREAD_ID = s.THREAD_ID
WHERE t.PROCESSLIST_ID = %d
UNION
SELECT s.SQL_TEXT FROM performance_schema.events_statements_current s
INNER JOIN performance_schema.threads t ON t.THREAD_ID = s.THREAD_ID
WHERE t.PROCESSLIST_ID = %d;
```

(PROCESSLIST_ID in this example is exactly the thread id from error log output).

## Status Variables

The status variables here contain information available in the output of `SHOW ENGINE INNODB STATUS`, organized by the sections `SHOW ENGINE INNODB STATUS` displays. If you are familiar with the output of `SHOW ENGINE INNODB STATUS`, you will probably already recognize the information these variables contain.

### BACKGROUND THREAD

The following variables contain information in the `BACKGROUND THREAD` section of the output from `SHOW ENGINE INNODB STATUS`. An example of that output is:

```
-----------------
BACKGROUND THREAD
-----------------
srv_master_thread loops: 1 srv_active, 0 srv_shutdown, 11844 srv_idle
srv_master_thread log flush and writes: 11844
```

*InnoDB* has a master thread which performs background tasks depending on the server state, once per second. If the server is under workload, the master thread runs the following: performs background table drops; performs change buffer merge, adaptively; flushes the redo log to disk; evicts tables from the dictionary cache if needed to satisfy its size limit; makes a checkpoint. If the server is idle: performs background table drops, flushes and/or checkpoints the redo log if needed due to the checkpoint age; performs change buffer merge at full I/O capacity; evicts tables from the dictionary cache if needed; and makes a checkpoint.

**variable `Innodb_master_thread_active_loops`**

**Variable Type** Numeric

**Scope** Global

This variable shows the number of times the above one-second loop was executed for active server states.

**variable `Innodb_master_thread_idle_loops`**

**Variable Type** Numeric

**Scope** Global

This variable shows the number of times the above one-second loop was executed for idle server states.

**variable `Innodb_background_log_sync`**

**Variable Type** Numeric

**Scope** Global

This variable shows the number of times the *InnoDB* master thread has written and flushed the redo log.

## SEMAPHORES

The following variables contain information in the `SEMAPHORES` section of the output from `SHOW ENGINE INNODB STATUS`. An example of that output is:

```
----------
SEMAPHORES
----------
OS WAIT ARRAY INFO: reservation count 9664, signal count 11182
Mutex spin waits 20599, rounds 223821, OS waits 4479
RW-shared spins 5155, OS waits 1678; RW-excl spins 5632, OS waits 2592
Spin rounds per wait: 10.87 mutex, 15.01 RW-shared, 27.19 RW-excl
```

## INSERT BUFFER AND ADAPTIVE HASH INDEX

The following variables contain information in the `INSERT BUFFER AND ADAPTIVE HASH INDEX` section of the output from `SHOW ENGINE INNODB STATUS`. An example of that output is:

```
-------------------------------------
INSERT BUFFER AND ADAPTIVE HASH INDEX
-------------------------------------
Ibuf: size 1, free list len 6089, seg size 6091,
44497 inserts, 44497 merged recs, 8734 merges
0.00 hash searches/s, 0.00 non-hash searches/s
```

**variable `Innodb_ibuf_free_list`**

**Variable Type** Numeric

**Scope** Global

**variable `Innodb_ibuf_segment_size`**

**Variable Type** Numeric

**Scope** Global

## LOG

The following variables contain information in the `LOG` section of the output from `SHOW ENGINE INNODB STATUS`. An example of that output is:

```
LOG
---
Log sequence number 10145937666
Log flushed up to   10145937666
Pages flushed up to 10145937666
```

```
Last checkpoint at  10145937666
Max checkpoint age    80826164
Checkpoint age target 78300347
Modified age          0
Checkpoint age        0
0 pending log writes, 0 pending chkp writes
9 log i/o's done, 0.00 log i/o's/second
Log tracking enabled
Log tracked up to   10145937666
Max tracked LSN age 80826164
```

**variable `Innodb_lsn_current`**

**Variable Type** Numeric

**Scope** Global

This variable shows the current log sequence number.

**variable `Innodb_lsn_flushed`**

**Variable Type** Numeric

**Scope** Global

This variable shows the current maximum LSN that has been written and flushed to disk.

**variable `Innodb_lsn_last_checkpoint`**

**Variable Type** Numeric

**Scope** Global

This variable shows the LSN of the latest completed checkpoint.

**variable `Innodb_checkpoint_age`**

**Variable Type** Numeric

**Scope** Global

This variable shows the current *InnoDB* checkpoint age, i.e., the difference between the current LSN and the LSN of the last completed checkpoint.

**variable `Innodb_checkpoint_max_age`**

**Variable Type** Numeric

**Scope** Global

This variable shows the maximum allowed checkppoint age above which the redo log is close to full and a checkpoint must happen before any further redo log writes.

## BUFFER POOL AND MEMORY

The following variables contain information in the `BUFFER POOL AND MEMORY` section of the output from `SHOW ENGINE INNODB STATUS`. An example of that output is:

```
----------------------
BUFFER POOL AND MEMORY
----------------------
Total memory allocated 137363456; in additional pool allocated 0
Total memory allocated by read views 88
```

```
Internal hash tables (constant factor + variable factor)
    Adaptive hash index 2266736         (2213368 + 53368)
    Page hash           139112 (buffer pool 0 only)
    Dictionary cache    729463  (554768 + 174695)
    File system         824800  (812272 + 12528)
    Lock system         333248  (332872 + 376)
    Recovery system     0       (0 + 0)
Dictionary memory allocated 174695
Buffer pool size        8191
Buffer pool size, bytes 134201344
Free buffers            7481
Database pages          707
Old database pages      280
Modified db pages       0
Pending reads 0
Pending writes: LRU 0, flush list 0 single page 0
Pages made young 0, not young 0
0.00 youngs/s, 0.00 non-youngs/s
Pages read 707, created 0, written 1
0.00 reads/s, 0.00 creates/s, 0.00 writes/s
No buffer pool page gets since the last printout
Pages read ahead 0.00/s, evicted without access 0.00/s, Random read ahead 0.00/s
LRU len: 707, unzip_LRU len: 0
```

**variable `Innodb_mem_adaptive_hash`**

**Variable Type** Numeric

**Scope** Global

This variable shows the current size, in bytes, of the adaptive hash index.

**variable `Innodb_mem_dictionary`**

**Variable Type** Numeric

**Scope** Global

This variable shows the current size, in bytes, of the *InnoDB* in-memory data dictionary info.

**variable `Innodb_mem_total`**

**Variable Type** Numeric

**Scope** Global

This variable shows the total amount of memory, in bytes, *InnoDB* has allocated in the process heap memory.

**variable `Innodb_buffer_pool_pages_LRU_flushed`**

**Variable Type** Numeric

**Scope** Global

This variable shows the total number of buffer pool pages which have been flushed from the LRU list, i.e., too old pages which had to be flushed in order to make buffer pool room to read in new data pages.

**variable `Innodb_buffer_pool_pages_made_not_young`**

**Variable Type** Numeric

**Scope** Global

This variable shows the number of times a buffer pool page was not marked as accessed recently in the LRU list because of `innodb_old_blocks_time` variable setting.

**variable `Innodb_buffer_pool_pages_made_young`**

**Variable Type** Numeric

**Scope** Global

This variable shows the number of times a buffer pool page was moved to the young end of the LRU list due to its access, to prevent its eviction from the buffer pool.

**variable `Innodb_buffer_pool_pages_old`**

**Variable Type** Numeric

**Scope** Global

This variable shows the total number of buffer pool pages which are considered to be old according to the Making the Buffer Pool Scan Resistant manual page.

## TRANSACTIONS

The following variables contain information in the `TRANSACTIONS` section of the output from `SHOW INNODB STATUS`. An example of that output is:

```
------------
TRANSACTIONS
------------
Trx id counter F561FD
Purge done for trx's n:o < F561EB undo n:o < 0
History list length 19
LIST OF TRANSACTIONS FOR EACH SESSION:
---TRANSACTION 0, not started, process no 993, OS thread id 140213152634640
mysql thread id 15933, query id 32109 localhost root
show innodb status
---TRANSACTION F561FC, ACTIVE 29 sec, process no 993, OS thread id 140213152769808␣
↪updating or deleting
mysql tables in use 1, locked 1
```

**variable `Innodb_max_trx_id`**

**Variable Type** Numeric

**Scope** Global

This variable shows the next free transaction id number.

**variable `Innodb_oldest_view_low_limit_trx_id`**

**Variable Type** Numeric

**Scope** Global

This variable shows the highest transaction id, above which the current oldest open read view does not see any transaction changes. Zero if there is no open view.

**variable `Innodb_purge_trx_id`**

**Variable Type** Numeric

**Scope** Global

This variable shows the oldest transaction id whose records have not been purged yet.

**variable `Innodb_purge_undo_no`**

**Variable Type** Numeric

**Scope** Global

## INFORMATION_SCHEMA Tables

The following table contains information about the oldest active transaction in the system.

**table** INFORMATION_SCHEMA.**XTRADB_READ_VIEW**

**Columns**

- **READ_VIEW_LOW_LIMIT_TRX_NUMBER** – This is the highest transactions number at the time the view was created.
- **READ_VIEW_UPPER_LIMIT_TRX_ID** – This is the highest transactions ID at the time the view was created. This means that it should not see newer transactions with IDs bigger than or equal to that value.
- **READ_VIEW_LOW_LIMIT_TRX_ID** – This is the latest committed transaction ID at the time the oldest view was created. This means that it should see all transactions with IDs smaller than or equal to that value.

The following table contains information about the memory usage for InnoDB/XtraDB hash tables.

**table** INFORMATION_SCHEMA.**XTRADB_INTERNAL_HASH_TABLES**

**Columns**

- **INTERNAL_HASH_TABLE_NAME** – Hash table name
- **TOTAL_MEMORY** – Total amount of memory
- **CONSTANT_MEMORY** – Constant memory
- **VARIABLE_MEMORY** – Variable memory

## Other reading

- SHOW INNODB STATUS walk through
- Table locks in SHOW INNODB STATUS

CHAPTER

# FIFTYNINE

# SHOW STORAGE ENGINES

This feature changes the comment field displayed when the `SHOW STORAGE ENGINES` command is executed and *XtraDB* is the storage engine.

Before the Change:

```
mysql> show storage engines;
+------------+---------+----------------------------------------------------------------
↪--+--------------+------+------------+
| Engine     | Support | Comment                                                        ␣
↪  | Transactions | XA   | Savepoints |
+------------+---------+----------------------------------------------------------------
↪--+--------------+------+------------+
| InnoDB     | YES     | Supports transactions, row-level locking, and foreign keys  ␣
↪  | YES          | YES  | YES        |
...
+------------+---------+----------------------------------------------------------------
↪--+--------------+------+------------+
```

After the Change:

```
mysql> show storage engines;
+------------+---------+----------------------------------------------------------------
↪--------------+--------------+------+------------+
| Engine     | Support | Comment                                                        ␣
↪              | Transactions |   XA | Savepoints |
+------------+---------+----------------------------------------------------------------
↪--------------+--------------+------+------------+
| InnoDB     | YES     | Percona-XtraDB, Supports transactions, row-level locking,␣
↪and foreign keys |          YES | YES  | YES        |
...
+------------+---------+----------------------------------------------------------------
↪--------------+--------------+------+------------+
```

## Version-Specific Information

- `8.0.12-1`: Feature ported from *Percona Server for MySQL* 5.7

CHAPTER

SIXTY

# PROCESS LIST

This page describes Percona changes to both the standard *MySQL* `SHOW PROCESSLIST` command and the standard *MySQL* `INFORMATION_SCHEMA` table `PROCESSLIST`.

## Version Specific Information

- `8.0.12-1`:
  - Feature ported from *Percona Server for MySQL* 5.7

## INFORMATION_SCHEMA Tables

**table** `INFORMATION_SCHEMA.`**`PROCESSLIST`**

This table implements modifications to the standard *MySQL* `INFORMATION_SCHEMA` table `PROCESSLIST`.

**Columns**

- **`ID`** – The connection identifier.
- **`USER`** – The *MySQL* user who issued the statement.
- **`HOST`** – The host name of the client issuing the statement.
- **`DB`** – The default database, if one is selected, otherwise NULL.
- **`COMMAND`** – The type of command the thread is executing.
- **`TIME`** – The time in seconds that the thread has been in its current state.
- **`STATE`** – An action, event, or state that indicates what the thread is doing.
- **`INFO`** – The statement that the thread is executing, or NULL if it is not executing any statement.
- **`TIME_MS`** – The time in milliseconds that the thread has been in its current state.
- **`ROWS_EXAMINED`** – The number of rows examined by the statement being executed (*NOTE:* This column is not updated for each examined row so it does not necessarily show an up-to-date value while the statement is executing. It only shows a correct value after the statement has completed.).
- **`ROWS_SENT`** – The number of rows sent by the statement being executed.
- **`TID`** – The Linux Thread ID. For Linux, this corresponds to light-weight process ID (LWP ID) and can be seen in the `ps -L` output. In case when *Thread Pool* is enabled, "TID" is not null for only currently executing statements and statements received via "extra" connection.

## Example Output

Table *PROCESSLIST*:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.PROCESSLIST;

+----+------+-----------+--------------------+---------+------+-----------+-----------
↪---------------+---------+-----------+---------------+
| ID | USER | HOST      | DB                 | COMMAND | TIME | STATE     | INFO
↪                | TIME_MS | ROWS_SENT | ROWS_EXAMINED |
+----+------+-----------+--------------------+---------+------+-----------+-----------
↪---------------+---------+-----------+---------------+
| 12 | root | localhost | information_schema | Query   |    0 | executing | select *
↪from processlist |       0 |         0 |             0 |
+----+------+-----------+--------------------+---------+------+-----------+-----------
↪---------------+---------+-----------+---------------+
```

CHAPTER

SIXTYONE

# MISC. INFORMATION_SCHEMA TABLES

This page lists the `INFORMATION_SCHEMA` tables added to standard *MySQL* by *Percona Server for MySQL* that don't exist elsewhere in the documentation.

## Temporary tables

**Note:** This feature implementation is considered ALPHA quality.

Only the temporary tables that were explicitly created with *CREATE TEMPORARY TABLE* or *ALTER TABLE* are shown, and not the ones created to process complex queries.

**table** `INFORMATION_SCHEMA.`**`GLOBAL_TEMPORARY_TABLES`**

**Version Info**

- **`8.0.12-1`** – Feature ported from *Percona Server for MySQL* 5.7

**Columns**

- **`SESSION_ID`** – *MySQL* connection id
- **`TABLE_SCHEMA`** – Schema in which the temporary table is created
- **`TABLE_NAME`** – Name of the temporary table
- **`ENGINE`** – Engine of the temporary table
- **`NAME`** – Internal name of the temporary table
- **`TABLE_ROWS`** – Number of rows of the temporary table
- **`AVG_ROW_LENGTH`** – Average row length of the temporary table
- **`DATA_LENGTH`** – Size of the data (Bytes)
- **`INDEX_LENGTH`** – Size of the indexes (Bytes)
- **`CREATE_TIME`** – Date and time of creation of the temporary table
- **`UPDATE_TIME`** – Date and time of the latest update of the temporary table

This table holds information on the temporary tables that exist for all connections. You don't need the `SUPER` privilege to query this table.

**table** `INFORMATION_SCHEMA.`**`TEMPORARY_TABLES`**

**Version Info**

- **8.0.12-1** – Feature ported from *Percona Server for MySQL* 5.7

**Columns**

- **SESSION_ID** – *MySQL* connection id
- **TABLE_SCHEMA** – Schema in which the temporary table is created
- **TABLE_NAME** – Name of the temporary table
- **ENGINE** – Engine of the temporary table
- **NAME** – Internal name of the temporary table
- **TABLE_ROWS** – Number of rows of the temporary table
- **AVG_ROW_LENGTH** – Average row length of the temporary table
- **DATA_LENGTH** – Size of the data (Bytes)
- **INDEX_LENGTH** – Size of the indexes (Bytes)
- **CREATE_TIME** – Date and time of creation of the temporary table
- **UPDATE_TIME** – Date and time of the latest update of the temporary table

This table holds information on the temporary tables existing for the running connection.

CHAPTER

SIXTYTWO

# THREAD BASED PROFILING

*Percona Server for MySQL* now uses thread based profiling by default, instead of process based profiling. This was implemented because with process based profiling, threads on the server, other than the one being profiled, can affect the profiling information.

Thread based profiling is using the information provided by the kernel getrusage function. Since the 2.6.26 kernel version, thread based resource usage is available with the **RUSAGE_THREAD**. This means that the thread based profiling will be used if you're running the 2.6.26 kernel or newer, or if the **RUSAGE_THREAD** has been ported back.

This feature is enabled by default if your system supports it, in other cases it uses process based profiling.

## Version Specific Information

- `8.0.12-1`: Feature ported from *Percona Server for MySQL* 5.7

CHAPTER

SIXTYTHREE

# *INNODB* PAGE FRAGMENTATION COUNTERS

*InnoDB* page fragmentation is caused by random insertion or deletion from a secondary index. This means that the physical ordering of the index pages on the disk is not same as the index ordering of the records on the pages. As a consequence this means that some pages take a lot more space and that queries which require a full table scan can take a long time to finish.

To provide more information about the *InnoDB* page fragmentation *Percona Server for MySQL* now provides the following counters as status variables: *Innodb_scan_pages_contiguous*, *Innodb_scan_pages_disjointed*, *Innodb_scan_data_size*, *Innodb_scan_deleted_recs_size*, and *Innodb_scan_pages_total_seek_distance*.

## Version Specific Information

- `8.0.12-1`: The feature was ported from *Percona Server for MySQL* 5.7

## Status Variables

**variable `Innodb_scan_pages_contiguous`**

**Variable Type** Numeric

**Scope** Session

This variable shows the number of contiguous page reads inside a query.

**variable `Innodb_scan_pages_disjointed`**

**Variable Type** Numeric

**Scope** Session

This variable shows the number of disjointed page reads inside a query.

**variable `Innodb_scan_data_size`**

**Variable Type** Numeric

**Scope** Session

This variable shows the size of data in all *InnoDB* pages read inside a query (in bytes) - calculated as the sum of `page_get_data_size(page)` for every page scanned.

**variable `Innodb_scan_deleted_recs_size`**

**Variable Type** Numeric

**Scope** Session

This variable shows the size of deleted records (marked as `deleted` in `page_delete_rec_list_end()`) in all *InnoDB* pages read inside a query (in bytes) - calculated as the sum of `page_header_get_field(page, PAGE_GARBAGE)` for every page scanned.

**variable `Innodb_scan_pages_total_seek_distance`**

**Variable Type** Numeric

**Scope** Session

This variable shows the total seek distance when moving between pages.

## Related Reading

- InnoDB: look after fragmentation
- Defragmenting a Table

# Part X

# TokuDB

CHAPTER

SIXTYFOUR

# TOKUDB INTRODUCTION

**Availability** *TokuDB* is deprecated in the 8.0 series and will be supported through the 8.0 series until further notice. This storage engine will not be included in the next major release of *Percona Server for MySQL*. We recommend MyRocks as a long-term migration path.

*TokuDB* is a highly scalable, zero-maintenance downtime MySQL storage engine that delivers indexing-based query acceleration, improved replication performance, unparalleled compression, and live schema modification. The *TokuDB* storage engine is a scalable, ACID and MVCC compliant storage engine that provides indexing-based query improvements, offers online schema modifications, and reduces slave lag for both hard disk drives and flash memory. This storage engine is specifically designed for high performance on write-intensive workloads which is achieved with Fractal Tree indexing.

*Percona Server for MySQL* is compatible with the separately available *TokuDB* storage engine package. The *TokuDB* engine must be separately downloaded and then enabled as a plug-in component. This package can be installed alongside with standard *Percona Server for MySQL* releases and does not require any specially adapted version of *Percona Server for MySQL*.

**Warning:** Only the Percona supplied *TokuDB* engine should be used with *Percona Server for MySQL*. A *TokuDB* engine downloaded from other sources is not compatible. *TokuDB* file formats are not the same across *MySQL* variants. Migrating from one variant to any other variant requires a logical data dump and reload.

Additional features unique to *TokuDB* include:

- Up to 25x Data Compression
- Fast Inserts
- Eliminates Slave Lag with *Read Free Replication*
- Hot Schema Changes
- Hot Index Creation - *TokuDB* tables support insertions, deletions and queries with no down time while indexes are being added to that table
- Hot column addition, deletion, expansion, and rename - *TokuDB* tables support insertions, deletions and queries without down-time when an alter table adds, deletes, expands, or renames columns
- On-line Backup

For more information on installing and using *TokuDB* click on the following links:

# TokuDB Installation

*Percona Server for MySQL* is compatible with the separately available *TokuDB* storage engine package. The *TokuDB* engine must be separately downloaded and then enabled as a plug-in component. This package can be installed alongside with standard *Percona Server for MySQL* 8.0 releases and does not require any specially adapted version of *Percona Server for MySQL*.

The *TokuDB* storage engine is a scalable, ACID and MVCC compliant storage engine that provides indexing-based query improvements, offers online schema modifications, and reduces slave lag for both hard disk drives and flash memory. This storage engine is specifically designed for high performance on write-intensive workloads which is achieved with Fractal Tree indexing. To learn more about Fractal Tree indexing, you can visit the following Wikipedia page.

**Warning:** Only the Percona supplied *TokuDB* engine should be used with *Percona Server for MySQL* 8.0. A *TokuDB* engine downloaded from other sources is not compatible. *TokuDB* file formats are not the same across *MySQL* variants. Migrating from one variant to any other variant requires a logical data dump and reload.

## Prerequisites

### `libjemalloc` library

*TokuDB* storage engine requires `libjemalloc` library 3.3.0 or greater. If the version in the distribution repository is lower than that you can use one from *Percona Software Repositories* or download it from somewhere else.

If the `libjemalloc` wasn't installed and enabled before it will be automatically installed when installing the *TokuDB* storage engine package by using the **apt`** or **yum** package manager, but *Percona Server for MySQL* instance should be restarted for `libjemalloc` to be loaded. This way `libjemalloc` will be loaded with `LD_PRELOAD`. You can also enable `libjemalloc` by specifying `malloc-lib` variable in the `[mysqld_safe]` section of the `my.cnf` file:

```
[mysqld_safe]
malloc-lib= /path/to/jemalloc
```

### Transparent huge pages

*TokuDB* won't be able to start if the transparent huge pages are enabled. Transparent huge pages is feature available in the newer kernel versions. You can check if the Transparent huge pages are enabled with: `cat /sys/kernel/mm/transparent_hugepage/enabled`

**Output**

```
[always] madvise never
```

If transparent huge pages are enabled and you try to start the TokuDB engine you'll get the following message in you `error.log`:

```
Transparent huge pages are enabled, according to /sys/kernel/mm/redhat_transparent_
↪hugepage/enabled
Transparent huge pages are enabled, according to /sys/kernel/mm/transparent_hugepage/
↪enabled
```

You can disable transparent huge pages permanently by passing `transparent_hugepage=never` to the kernel in your bootloader (**NOTE**: For this change to take an effect you'll need to reboot your server).

You can disable the transparent huge pages by running the following command as root (**NOTE**: Setting this will last only until the server is rebooted):

```
echo never > /sys/kernel/mm/transparent_hugepage/enabled
echo never > /sys/kernel/mm/transparent_hugepage/defrag
```

## Installation

The *TokuDB* storage engine for *Percona Server for MySQL* is currently available in our *apt* and *yum* repositories.

You can install the *Percona Server for MySQL* with the *TokuDB* engine by using the respective package manager:

**yum** `yum install percona-server-tokudb.x86_64`

**apt** `apt install percona-server-tokudb`

## Enabling the TokuDB Storage Engine

Once the *TokuDB* server package is installed, the following output is shown:

---

**Output**

- This release of Percona Server is distributed with TokuDB storage engine. * Run the following script to enable the TokuDB storage engine in Percona Server:

  ```
  ps-admin --enable-tokudb -u <mysql_admin_user>
  -p[mysql_admin_pass] [-S <socket>] [-h <host> -P <port>]
  ```

  - See http://www.percona.com/doc/percona-server/8.0/tokudb/tokudb_installation.html for more installation details
  - See http://www.percona.com/doc/percona-server/8.0/tokudb/tokudb_intro.html for an introduction to TokuDB

---

*Percona Server for MySQL* has implemented **ps-admin** to make the enabling the *TokuDB* storage engine easier. This script will automatically disable Transparent huge pages, if they're enabled, and install and enable the *TokuDB* storage engine with all the required plugins. You need to run this script as root or with **sudo**. After you run the script with required parameters:

```
$ ps-admin --enable-tokudb -uroot -pPassw0rd
```

Following output will be displayed:

```
Checking if Percona server is running with jemalloc enabled...
>> Percona server is running with jemalloc enabled.

Checking transparent huge pages status on the system...
>> Transparent huge pages are currently disabled on the system.

Checking if thp-setting=never option is already set in config file...
>> Option thp-setting=never is not set in the config file.
>> (needed only if THP is not disabled permanently on the system)
```

```
Checking TokuDB plugin status...
>> TokuDB plugin is not installed.

Adding thp-setting=never option into /etc/mysql/my.cnf
>> Successfuly added thp-setting=never option into /etc/mysql/my.cnf

Installing TokuDB engine...
>> Successfuly installed TokuDB plugin.
```

If the script returns no errors, *TokuDB* storage engine should be successfully enabled on your server. You can check it out by running `SHOW ENGINES;`

**Output**

```
...
| TokuDB | YES | Tokutek TokuDB Storage Engine with Fractal Tree(tm) Technology | YES␣
↪| YES | YES |
...
```

## Enabling the TokuDB Storage Engine Manually

If you don't want to use **`ps-admin`** you'll need to manually install the storage engine ad required plugins.

```
INSTALL PLUGIN tokudb SONAME 'ha_tokudb.so';
INSTALL PLUGIN tokudb_file_map SONAME 'ha_tokudb.so';
INSTALL PLUGIN tokudb_fractal_tree_info SONAME 'ha_tokudb.so';
INSTALL PLUGIN tokudb_fractal_tree_block_map SONAME 'ha_tokudb.so';
INSTALL PLUGIN tokudb_trx SONAME 'ha_tokudb.so';
INSTALL PLUGIN tokudb_locks SONAME 'ha_tokudb.so';
INSTALL PLUGIN tokudb_lock_waits SONAME 'ha_tokudb.so';
INSTALL PLUGIN tokudb_background_job_status SONAME 'ha_tokudb.so';
```

After the engine has been installed it should be present in the engines list. To check if the engine has been correctly installed and active: `SHOW ENGINES;`

**Output**

```
...
| TokuDB | YES | Tokutek TokuDB Storage Engine with Fractal Tree(tm) Technology | YES␣
↪| YES | YES |
...
```

To check if all the *TokuDB* plugins have been installed correctly you should run: `SHOW PLUGINS;`

**Output**

```
...
| TokuDB                        | ACTIVE   | STORAGE ENGINE     | ha_tokudb.so | GPL ␣
↪    |
| TokuDB_file_map               | ACTIVE   | INFORMATION SCHEMA | ha_tokudb.so | GPL ␣
↪    |
| TokuDB_fractal_tree_info      | ACTIVE   | INFORMATION SCHEMA | ha_tokudb.so | GPL ␣
↪    |
```

```
| TokuDB_fractal_tree_block_map | ACTIVE   | INFORMATION SCHEMA | ha_tokudb.so | GPL ␣
↪   |
| TokuDB_trx                    | ACTIVE   | INFORMATION SCHEMA | ha_tokudb.so | GPL ␣
↪   |
| TokuDB_locks                  | ACTIVE   | INFORMATION SCHEMA | ha_tokudb.so | GPL ␣
↪   |
| TokuDB_lock_waits             | ACTIVE   | INFORMATION SCHEMA | ha_tokudb.so | GPL ␣
↪   |
| TokuDB_background_job_status  | ACTIVE   | INFORMATION SCHEMA | ha_tokudb.so | GPL ␣
↪   |
...
```

### TokuDB Version

*TokuDB* storage engine version can be checked with: `SELECT @@tokudb_version;`

**Output**

```
+------------------+
| @@tokudb_version |
+------------------+
| 8.0.13-3         |
+------------------+
1 row in set (0.00 sec)
```

### Upgrade

Before upgrading to *Percona Server for MySQL* 8.0, make sure that your system is ready by running **`mysqlcheck`**: `mysqlcheck -u root -p --all-databases --check-upgrade`

> **Warning:** With partitioned tables that use the *TokuDB* or *MyRocks* storage engine, the upgrade only works with native partitioning.

**See also:**

***MySQL* Documentation: Preparing Your Installation for Upgrade** https://dev.mysql.com/doc/refman/8.0/en/upgrade-prerequisites.html

## Using TokuDB

> **Warning:** Do not move or modify any *TokuDB* files. You will break the database, and need to recover the database from a backup.

## Fast Insertions and Richer Indexes

TokuDB's fast indexing enables fast queries through the use of rich indexes, such as covering and clustering indexes. It's worth investing some time to optimize index definitions to get the best performance from *MySQL* and *TokuDB*. Here are some resources to get you started:

- "Understanding Indexing" by Zardosht Kasheff (video)
- Rule of Thumb for Choosing Column Order in Indexes
- Covering Indexes: Orders-of-Magnitude Improvements
- Introducing Multiple Clustering Indexes
- Clustering Indexes vs. Covering Indexes
- How Clustering Indexes Sometimes Helps UPDATE and DELETE Performance
- *High Performance MySQL, 3rd Edition* by Baron Schwartz, Peter Zaitsev, Vadim Tkachenko, Copyright 2012, O'Reilly Media. See Chapter 5, *Indexing for High Performance*.

## Clustering Secondary Indexes

One of the keys to exploiting TokuDB's strength in indexing is to make use of clustering secondary indexes.

*TokuDB* allows a secondary key to be defined as a clustering key. This means that all of the columns in the table are clustered with the secondary key. *Percona Server for MySQL* parser and query optimizer support Multiple Clustering Keys when *TokuDB* engine is used. This means that the query optimizer will avoid primary clustered index reads and replace them by secondary clustered index reads in certain scenarios.

The parser has been extended to support following syntax:

```
CREATE TABLE ... ( ..., CLUSTERING KEY identifier (column list), ...
CREATE TABLE ... ( ..., UNIQUE CLUSTERING KEY identifier (column list), ...
CREATE TABLE ... ( ..., CLUSTERING UNIQUE KEY identifier (column list), ...
CREATE TABLE ... ( ..., CONSTRAINT identifier UNIQUE CLUSTERING KEY identifier␣
↪(column list), ...
CREATE TABLE ... ( ..., CONSTRAINT identifier CLUSTERING UNIQUE KEY identifier␣
↪(column list), ...

CREATE TABLE ... (... column type CLUSTERING [UNIQUE] [KEY], ...)
CREATE TABLE ... (... column type [UNIQUE] CLUSTERING [KEY], ...)

ALTER TABLE ..., ADD CLUSTERING INDEX identifier (column list), ...
ALTER TABLE ..., ADD UNIQUE CLUSTERING INDEX identifier (column list), ...
ALTER TABLE ..., ADD CLUSTERING UNIQUE INDEX identifier (column list), ...
ALTER TABLE ..., ADD CONSTRAINT identifier UNIQUE CLUSTERING INDEX identifier (column␣
↪list), ...
ALTER TABLE ..., ADD CONSTRAINT identifier CLUSTERING UNIQUE INDEX identifier (column␣
↪list), ...

CREATE CLUSTERING INDEX identifier ON ...
```

To define a secondary index as clustering, simply add the word `CLUSTERING` before the key definition. For example:

```
CREATE TABLE foo (
  column_a INT,
  column_b INT,
  column_c INT,
```

```
  PRIMARY KEY index_a (column_a),
  CLUSTERING KEY index_b (column_b)) ENGINE = TokuDB;
```

In the previous example, the primary table is indexed on *column_a*. Additionally, there is a secondary clustering index (named *index_b*) sorted on *column_b*. Unlike non-clustered indexes, clustering indexes include all the columns of a table and can be used as covering indexes. For example, the following query will run very fast using the clustering *index_b*:

```
SELECT column_c
  FROM foo
  WHERE column_b BETWEEN 10 AND 100;
```

This index is sorted on *column_b*, making the `WHERE` clause fast, and includes *column_c*, which avoids lookups in the primary table to satisfy the query.

*TokuDB* makes clustering indexes feasible because of its excellent compression and very high indexing rates. For more information about using clustering indexes, see Introducing Multiple Clustering Indexes.

## Hot Index Creation

TokuDB enables you to add indexes to an existing table and still perform inserts and queries on that table while the index is being created.

The `ONLINE` keyword is not used. Instead, the value of the *tokudb_create_index_online* client session variable is examined.

Hot index creation is invoked using the `CREATE INDEX` command after setting *tokudb_create_index_online* to `on` as follows:

```
mysql> SET tokudb_create_index_online=on;
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE INDEX index ON foo (field_name);
```

Alternatively, using the `ALTER TABLE` command for creating an index will create the index offline (with the table unavailable for inserts or queries), regardless of the value of *tokudb_create_index_online*. The only way to hot create an index is to use the `CREATE INDEX` command.

Hot creating an index will be slower than creating the index offline, and progress depends how busy the mysqld server is with other tasks. Progress of the index creation can be seen by using the `SHOW PROCESSLIST` command (in another client). Once the index creation completes, the new index will be used in future query plans.

If more than one hot `CREATE INDEX` is issued for a particular table, the indexes will be created serially. An index creation that is waiting for another to complete will be shown as *Locked* in `SHOW PROCESSLIST`. We recommend that each `CREATE INDEX` be allowed to complete before the next one is started.

## Hot Column Add, Delete, Expand, and Rename (HCADER)

*TokuDB* enables you to add or delete columns in an existing table, expand `char`, `varchar`, `varbinary`, and `integer` type columns in an existing table, or rename an existing column in a table with little blocking of other updates and queries. HCADER typically blocks other queries with a table lock for no more than a few seconds. After that initial short-term table locking, the system modifies each row (when adding, deleting, or expanding columns) later, when the row is next brought into main memory from disk. For column rename, all the work is done during the seconds of downtime. On-disk rows need not be modified.

To get good performance from HCADER, observe the following guidelines:

- The work of altering the table for column addition, deletion, or expansion is performed as subsequent operations touch parts of the Fractal Tree, both in the primary index and secondary indexes.

  You can force the column addition, deletion, or expansion work to be performed all at once using the standard syntax of `OPTIMIZE TABLE X`, when a column has been added to, deleted from, or expanded in table X. It is important to note that as of *TokuDB* version 7.1.0, `OPTIMIZE TABLE` is also hot, so that a table supports updates and queries without blocking while an `OPTIMIZE TABLE` is being performed. Also, a hot `OPTIMIZE TABLE` does not rebuild the indexes, since *TokuDB* indexes do not age. Rather, they flush all background work, such as that induced by a hot column addition, deletion, or expansion.

- Each hot column addition, deletion, or expansion operation must be performed individually (with its own SQL statement). If you want to add, delete, or expand multiple columns use multiple statements.

- Avoid adding, deleting, or expanding a column at the same time as adding or dropping an index.

- The time that the table lock is held can vary. The table-locking time for HCADER is dominated by the time it takes to flush dirty pages, because MySQL closes the table after altering it. If a checkpoint has happened recently, this operation is fast (on the order of seconds). However, if the table has many dirty pages, then the flushing stage can take on the order of minutes.

- Avoid dropping a column that is part of an index. If a column to be dropped is part of an index, then dropping that column is slow. To drop a column that is part of an index, first drop the indexes that reference the column in one alter table statement, and then drop the column in another statement.

- Hot column expansion operations are only supported to `char`, `varchar`, `varbinary`, and `integer` data types. Hot column expansion is not supported if the given column is part of the primary key or any secondary keys.

- Rename only one column per statement. Renaming more than one column will revert to the standard MySQL blocking behavior. The proper syntax is as follows:

```
ALTER TABLE table
  CHANGE column_old column_new
  DATA_TYPE REQUIRED_NESS DEFAULT
```

  Here's an example of how that might look:

```
ALTER TABLE table
  CHANGE column_old column_new
  INT(10) NOT NULL;
```

Notice that all of the column attributes must be specified. `ALTER TABLE table CHANGE column_old column_new;` induces a slow, blocking column rename.

- Hot column rename does not support the following data types: `TIME`, `ENUM`, `BLOB`, `TINYBLOB`, `MEDIUMBLOB`, `LONGBLOB`. Renaming columns of these types will revert to the standard MySQL blocking behavior.

- Temporary tables cannot take advantage of HCADER. Temporary tables are typically small anyway, so altering them using the standard method is usually fast.

## Compression Details

*TokuDB* offers different levels of compression, which trade off between the amount of CPU used and the compression achieved. Standard compression uses less CPU but generally compresses at a lower level, high compression uses more CPU and generally compresses at a higher level. We have seen compression up to 25x on customer data.

Compression in *TokuDB* occurs on background threads, which means that high compression need not slow down your database. Indeed, in some settings, we've seen higher overall database performance with high compression.

**Note:** We recommend that users use standard compression on machines with six or fewer cores, and high compression on machines with more than six cores.

The ultimate choice depends on the particulars of how a database is used, and we recommend that users use the default settings unless they have profiled their system with high compression in place.

The table is compressed using whichever row format is specified in the session variable *tokudb_row_format*. If no row format is set nor is *tokudb_row_format*, the QUICKLZ compression algorithm is used.

The row_format and *tokudb_row_format* variables accept the following values:

| Value | Description |
|---|---|
| TOKUDB_DEFAULT | Sets the compression to the default behavior. As of TokuDB 7.1.0, the default behavior is to compress using the zlib library. In the future this behavior may change. |
| TOKUDB_FAST | Sets the compression to use the quicklz library. |
| TOKUDB_SMALL | Sets the compression to use the lzma library. |
| TOKUDB_ZLIB | Compress using the zlib library, which provides mid-range compression and CPU utilization. |
| TOKUDB_QUICKLZ | Compress using the quicklz library, which provides light compression and low CPU utilization. |
| TOKUDB_LZMA | Compress using the lzma library, which provides the highest compression and high CPU utilization. |
| TOKUDB_SNAPPY | This compression is using snappy library and aims for very high speeds and reasonable compression. |
| TOKUDB_UNCOMPRESSED | This setting turns off compression and is useful for tables with data that cannot be compressed. |

## Read Free Replication

*TokuDB* slaves can be configured to perform significantly less read IO in order to apply changes from the master. By utilizing the power of Fractal Tree indexes:

- insert/update/delete operations can be configured to eliminate read-modify-write behavior and simply inject messages into the appropriate Fractal Tree indexes
- update/delete operations can be configured to eliminate the IO required for uniqueness checking

To enable Read Free Replication, the servers must be configured as follows:

- On the replication master:
  - Enable row based replication: set BINLOG_FORMAT=ROW
- On the replication slave(s):
  - The slave must be in read-only mode: set read_only=1
  - Disable unique checks: set tokudb_rpl_unique_checks=0
  - Disable lookups (read-modify-write): set tokudb_rpl_lookup_rows=0

**Note:** You can modify one or both behaviors on the slave(s).

**Note:** As long as the master is using row based replication, this optimization is available on a *TokuDB* slave. This means that it's available even if the master is using *InnoDB* or *MyISAM* tables, or running non-TokuDB binaries.

**Warning:** *TokuDB* Read Free Replication will not propagate `UPDATE` and `DELETE` events reliably if *TokuDB* table is missing the primary key which will eventually lead to data inconsistency on the slave.

## Transactions and ACID-compliant Recovery

By default, *TokuDB* checkpoints all open tables regularly and logs all changes between checkpoints, so that after a power failure or system crash, *TokuDB* will restore all tables into their fully ACID-compliant state. That is, all committed transactions will be reflected in the tables, and any transaction not committed at the time of failure will be rolled back.

The default checkpoint period is every 60 seconds, and this specifies the time from the beginning of one checkpoint to the beginning of the next. If a checkpoint requires more than the defined checkpoint period to complete, the next checkpoint begins immediately. It is also related to the frequency with which log files are trimmed, as described below. The user can induce a checkpoint at any time by issuing the `FLUSH LOGS` command. When a database is shut down normally it is also checkpointed and all open transactions are aborted. The logs are trimmed at startup.

## Managing Log Size

*TokuDB* keeps log files back to the most recent checkpoint. Whenever a log file reaches 100 MB, a new log file is started. Whenever there is a checkpoint, all log files older than the checkpoint are discarded. If the checkpoint period is set to be a very large number, logs will get trimmed less frequently. This value is set to 60 seconds by default.

*TokuDB* also keeps rollback logs for each open transaction. The size of each log is proportional to the amount of work done by its transaction and is stored compressed on disk. Rollback logs are trimmed when the associated transaction completes.

## Recovery

Recovery is fully automatic with *TokuDB*. *TokuDB* uses both the log files and rollback logs to recover from a crash. The time to recover from a crash is proportional to the combined size of the log files and uncompressed size of rollback logs. Thus, if there were no long-standing transactions open at the time of the most recent checkpoint, recovery will take less than a minute.

## Disabling the Write Cache

When using any transaction-safe database, it is essential that you understand the write-caching characteristics of your hardware. *TokuDB* provides transaction safe (ACID compliant) data storage for *MySQL*. However, if the underlying operating system or hardware does not actually write data to disk when it says it did, the system can corrupt your database when the machine crashes. For example, *TokuDB* can not guarantee proper recovery if it is mounted on an NFS volume. It is always safe to disable the write cache, but you may be giving up some performance.

For most configurations you must disable the write cache on your disk drives. On ATA/SATA drives, the following command should disable the write cache:

```
$ hdparm -W0 /dev/hda
```

There are some cases when you can keep the write cache, for example:

- Write caching can remain enabled when using XFS, but only if XFS reports that disk write barriers work. If you see one of the following messages in /var/log/messages, then you must disable the write cache:
  - `Disabling barriers, not supported with external log device`

  - `Disabling barriers, not supported by the underlying device`
  - `Disabling barriers, trial barrier write failed`

  XFS write barriers appear to succeed for single disks (with no LVM), or for very recent kernels (such as that provided by Fedora 12). For more information, see the XFS FAQ.

In the following cases, you must disable the write cache:

- If you use the ext3 filesystem
- If you use LVM (although recent Linux kernels, such as Fedora 12, have fixed this problem)
- If you use Linux's software RAID
- If you use a RAID controller with battery-backed-up memory. This may seem counter-intuitive. For more information, see the XFS FAQ

In summary, you should disable the write cache, unless you have a very specific reason not to do so.

### Progress Tracking

*TokuDB* has a system for tracking progress of long running statements, thereby removing the need to define triggers to track statement execution, as follows:

- Bulk Load: When loading large tables using `LOAD DATA INFILE` commands, doing a `SHOW PROCESSLIST` command in a separate client session shows progress. There are two progress stages. The first will state something like `Inserted about 1000000 rows`. After all rows are processed like this, the next stage tracks progress by showing what fraction of the work is done (e.g. `Loading of data about 45% done`)
- Adding Indexes: When adding indexes via `ALTER TABLE` or `CREATE INDEX`, the command `SHOW PROCESSLIST` shows progress. When adding indexes via `ALTER TABLE` or `CREATE INDEX`, the command `SHOW PROCESSLIST` will include an estimation of the number of rows processed. Use this information to verify progress is being made. Similar to bulk loading, the first stage shows how many rows have been processed, and the second stage shows progress with a fraction.
- Commits and Aborts: When committing or aborting a transaction, the command `SHOW PROCESSLIST` will include an estimate of the transactional operations processed.

### Migrating to TokuDB

To convert an existing table to use the *TokuDB* engine, run `ALTER TABLE... ENGINE=TokuDB`. If you wish to load from a file, use `LOAD DATA INFILE` and not `mysqldump`. Using `mysqldump` will be much slower. To create a file that can be loaded with `LOAD DATA INFILE`, refer to the `INTO OUTFILE` option of the SELECT Syntax.

---

**Note:** Creating this file does not save the schema of your table, so you may want to create a copy of that as well.

---

## Getting Started with TokuDB

### System and Hardware Requirements

**Operating Systems**: *TokuDB* is currently supported on 64-bit Linux only.

**Memory**: *TokuDB* Requires at least 1GB of main memory but for best results, we recommend to run with at least 2GB of main memory.

**Disk space and configuration**: Please make sure to allocate enough disk space for data, indexes and logs. In our users' experience, *TokuDB* achieves up to 25x space savings on data and indexes over *InnoDB* due to high compression.

## Creating Tables and Loading Data

### Creating TokuDB Tables

*TokuDB* tables are created the same way as other tables in *MySQL* by specifying `ENGINE=TokuDB` in the table definition. For example, the following command creates a table with a single column and uses the *TokuDB* storage engine to store its data:

```
CREATE TABLE table (
  id INT(11) NOT NULL) ENGINE=TokuDB;
```

### Loading Data

Once *TokuDB* tables have been created, data can be inserted or loaded using standard *MySQL* insert or bulk load operations. For example, the following command loads data from a file into the table:

```
LOAD DATA INFILE file
  INTO TABLE table;
```

**Note:** For more information about loading data, see the *MySQL* 5.6 reference manual.

### Migrating Data from an Existing Database

Use the following command to convert an existing table for the *TokuDB* storage engine:

```
ALTER TABLE table
  ENGINE=TokuDB;
```

## Bulk Loading Data

The *TokuDB* bulk loader imports data much faster than regular *MySQL* with *InnoDB*. To make use of the loader you need flat files in either comma separated or tab separated format. The *MySQL* `LOAD DATA INFILE ...` statement will invoke the bulk loader if the table is empty. Keep in mind that while this is the most convenient and, in most cases, the fastest way to initialize a *TokuDB* table, it may not be replication safe if applied to the master

For more information, see the *MySQL* 5.6 Reference Manual: LOAD DATA INFILE.

To obtain the logical backup and then bulk load into *TokuDB*, follow these steps:

1. Create a logical backup of the original table. The easiest way to achieve this is using `SELECT ... INTO OUTFILE`. Keep in mind that the file will be created on the server.

   ```
   SELECT * FROM table
     INTO OUTFILE 'file.csv';
   ```

2. The output file should either be copied to the destination server or the client machine from which you plan to load it.
3. To load the data into the server use `LOAD DATA INFILE`. If loading from a machine other than the server use the keyword `LOCAL` to point to the file on local machine. Keep in mind that you will need enough disk space on the temporary directory on the server since the local file will be copied onto the server by the MySQL client utility.

```
LOAD DATA [LOCAL] INFILE 'file.csv';
```

It is possible to create the CSV file using either **mysqldump** or the *MySQL* client utility as well, in which case the resulting file will reside on a local directory. In these 2 cases you have to make sure to use the correct command line options to create a file compatible with `LOAD DATA INFILE`.

The bulk loader will use more space than normal for logs and temporary files while running, make sure that your file system has enough disk space to process your load. As a rule of thumb, it should be approximately 1.5 times the size of the raw data.

---

**Note:** Please read the original *MySQL* documentation to understand the needed privileges and replication issues needed around `LOAD DATA INFILE`.

---

## Considerations to Run TokuDB in Production

In most cases, the default options should be left in-place to run *TokuDB*, however it is a good idea to review some of the configuration parameters.

### Memory allocation

*TokuDB* will allocate 50% of the installed RAM for its own cache (global variable *tokudb_cache_size*). While this is optimal in most situations, there are cases where it may lead to memory over allocation. If the system tries to allocate more memory than is available, the machine will begin swapping and run much slower than normal.

It is necessary to set the *tokudb_cache_size* to a value other than the default in the following cases:

- **Running other memory heavy processes on the same server as TokuDB**: In many cases, the database process needs to share the system with other server processes like additional database instances, http server, application server, e-mail server, monitoring systems and others. In order to properly configure TokuDB's memory consumption, it's important to understand how much free memory will be left and assign a sensible value for *TokuDB*. There is no fixed rule, but a conservative choice would be 50% of available RAM while all the other processes are running. If the result is under 2 GB, you should consider moving some of the other processes to a different system or using a dedicated database server.

  *tokudb_cache_size* is a static variable, so it needs to be set before starting the server and cannot be changed while the server is running. For example, to set up TokuDB's cache to 4G, add the following line to your `my.cnf` file:

```
tokudb_cache_size = 4G
```

- **System using InnoDB and TokuDB**: When using both the *TokuDB* and *InnoDB* storage engines, you need to manage the cache size for each. For example, on a server with 16 GB of RAM you could use the following values in your configuration file:

```
innodb_buffer_pool_size = 2G
tokudb_cache_size = 8G
```

- **Using TokuDB with Federated or FederatedX tables**: The Federated engine in *MySQL* and FederatedX in *MariaDB* allow you to connect to a table on a remote server and query it as if it were a local table (please see the *MySQL* documentation: 14.11. The FEDERATED Storage Engine for details). When accessing the remote table, these engines could import the complete table contents to the local server to execute a query. In this case, you will have to make sure that there is enough free memory on the server to handle these remote tables. For example, if your remote table is 8 GB in size, the server has to have more than 8 GB of free RAM to process queries against that table without going into swapping or causing a kernel panic and crash the MySQL process. There are no parameters to limit the amount of memory that the Federated or FederatedX engine will allocate while importing the remote dataset.

### Specifying the Location for Files

As with *InnoDB*, it is possible to specify different locations than the default for TokuDB's data, log and temporary files. This way you may distribute the load and control the disk space. The following variables control file location:

- *tokudb_data_dir*: This variable defines the directory where the *TokuDB* tables are stored. The default location for TokuDB's data files is the *MySQL* data directory.
- *tokudb_log_dir*: This variable defines the directory where the *TokuDB* log files are stored. The default location for TokuDB's log files is the *MySQL* data directory. Configuring a separate log directory is somewhat involved and should be done only if absolutely necessary. We recommend to keep the data and log files under the same directory.
- *tokudb_tmp_dir*: This variable defines the directory where the *TokuDB* bulk loader stores temporary files. The bulk loader can create large temporary files while it is loading a table, so putting these temporary files on a disk separate from the data directory can be useful. For example, it can make sense to use a high-performance disk for the data directory and a very inexpensive disk for the temporary directory. The default location for TokuDB's temporary files is the MySQL data directory.

### Table Maintenance

#### Overview

The fractal tree provides fast performance by inserting small messages in the buffers in the fractal trees instead of requiring a potential IO for an update on every row in the table as required by a B-tree. Additional background information on how fractal trees operate can be found here. For tables whose workload pattern is a high number of sequential deletes, it may be beneficial to flush these delete messages down to the basement nodes in order to allow for faster access. The way to perform this operation is via the `OPTIMIZE` command.

The following extensions to the `OPTIMIZE` command have been added in *TokuDB* version 7.5.5:

- **Hot Optimize Throttling**

  By default, table optimization will run with all available resources. To limit the amount of resources, it is possible to limit the speed of table optimization. The *tokudb_optimize_throttle* session variable determines an upper bound on how many fractal tree leaf nodes per second are optimized. The default is 0 (no upper bound) with a valid range of [0,1000000]. For example, to limit the table optimization to 1 leaf node per second, use the following setting:

```
SET tokudb_optimize_throttle=1;
```

- **Optimize a Single Index of a Table**

  To optimize a single index in a table, the *tokudb_optimize_index_name* session variable can be set to select the index by name. For example, to optimize the primary key of a table:

```
SET tokudb_optimize_index_name='primary';
OPTIMIZE TABLE t;
```

- **Optimize a Subset of a Fractal Tree Index**

  For patterns where the left side of the tree has many deletions (a common pattern with increasing id or date values), it may be useful to delete a percentage of the tree. In this case, it is possible to optimize a subset of a fractal tree starting at the left side. The *tokudb_optimize_index_fraction* session variable controls the size of the sub tree. Valid values are in the range [0.0,1.0] with default 1.0 (optimize the whole tree). For example, to optimize the leftmost 10% of the primary key:

```
SET tokudb_optimize_index_name='primary';
SET tokudb_optimize_index_fraction=0.1;
OPTIMIZE TABLE t;
```

# TokuDB Variables

Like all storage engines, *TokuDB* has variables to tune performance and control behavior. Fractal Tree algorithms are designed for near optimal performance and TokuDB's default settings should work well in most situations, eliminating the need for complex and time consuming tuning in most cases.



## TokuDB Server Variables

| Name | Cmd-Line | Option File | Var Scope | Dynamic |
|---|---|---|---|---|
| *tokudb_alter_print_error* | Yes | Yes | Session, Global | Yes |
| *tokudb_analyze_delete_fraction* | Yes | Yes | Session, Global | Yes |
| *tokudb_analyze_in_background* | Yes | Yes | Session, Global | Yes |
| *tokudb_analyze_mode* | Yes | Yes | Session, Global | Yes |
| *tokudb_analyze_throttle* | Yes | Yes | Session, Global | Yes |
| *tokudb_analyze_time* | Yes | Yes | Session, Global | Yes |
| *tokudb_auto_analyze* | Yes | Yes | Session, Global | Yes |
| *tokudb_backup_allowed_prefix* | No | Yes | Global | No |
| *tokudb_backup_dir* | No | Yes | Session | No |
| *tokudb_backup_exclude* | Yes | Yes | Session, Global | Yes |
| *tokudb_backup_last_error* | Yes | Yes | Session, Global | Yes |

Continued on next page

Table 64.1 – continued from previous page

| Name | Cmd-Line | Option File | Var Scope | Dynamic |
|---|---|---|---|---|
| *tokudb_backup_last_error_string* | Yes | Yes | Session, Global | Yes |
| *tokudb_backup_plugin_version* | No | No | Global | No |
| *tokudb_backup_throttle* | Yes | Yes | Session, Global | Yes |
| *tokudb_backup_version* | No | No | Global | No |
| *tokudb_block_size* | Yes | Yes | Session, Global | Yes |
| *tokudb_bulk_fetch* | Yes | Yes | Session, Global | Yes |
| *tokudb_cachetable_pool_threads* | Yes | Yes | Global | No |
| *tokudb_cardinality_scale_percent* | Yes | Yes | Global | Yes |
| *tokudb_check_jemalloc* | Yes | Yes | Global | No |
| *tokudb_checkpoint_lock* | Yes | Yes | Global | No |
| *tokudb_checkpoint_on_flush_logs* | Yes | Yes | Global | Yes |
| *tokudb_checkpoint_pool_threads* | Yes | Yes | Global | Yes |
| *tokudb_checkpointing_period* | Yes | Yes | Global | Yes |
| *tokudb_cleaner_iterations* | Yes | Yes | Global | Yes |
| *tokudb_cleaner_period* | Yes | Yes | Global | Yes |
| *tokudb_client_pool_threads* | Yes | Yes | Global | No |
| *tokudb_commit_sync* | Yes | Yes | Session, Global | Yes |
| *tokudb_compress_buffers_before_eviction* | Yes | Yes | Global | No |
| *tokudb_create_index_online* | Yes | Yes | Session, Global | Yes |
| *tokudb_data_dir* | Yes | Yes | Global | No |
| *tokudb_debug* | Yes | Yes | Global | Yes |
| *tokudb_dir_per_db* | Yes | Yes | Global | Yes |
| *tokudb_directio* | Yes | Yes | Global | No |
| *tokudb_disable_hot_alter* | Yes | Yes | Session, Global | Yes |
| *tokudb_disable_prefetching* | Yes | Yes | Session, Global | Yes |
| *tokudb_disable_slow_alter* | Yes | Yes | Session, Global | Yes |
| *tokudb_empty_scan* | Yes | Yes | Session, Global | Yes |
| *tokudb_enable_fast_update* | Yes | Yes | Session, Global | Yes |
| *tokudb_enable_fast_upsert* | Yes | Yes | Session, Global | Yes |
| *tokudb_enable_partial_eviction* | Yes | Yes | Global | No |
| *tokudb_fanout* | Yes | Yes | Session, Global | Yes |
| *tokudb_fs_reserve_percent* | Yes | Yes | Global | No |
| *tokudb_fsync_log_period* | Yes | Yes | Global | Yes |
| *tokudb_hide_default_row_format* | Yes | Yes | Session, Global | Yes |

Continued on next page

Table 64.1 – continued from previous page

| Name | Cmd-Line | Option File | Var Scope | Dynamic |
|---|---|---|---|---|
| *tokudb_killed_time* | Yes | Yes | Session, Global | Yes |
| *tokudb_last_lock_timeout* | Yes | Yes | Session, Global | Yes |
| *tokudb_load_save_space* | Yes | Yes | Session, Global | Yes |
| *tokudb_loader_memory_size* | Yes | Yes | Session, Global | Yes |
| *tokudb_lock_timeout* | Yes | Yes | Session, Global | Yes |
| *tokudb_lock_timeout_debug* | Yes | Yes | Session, Global | Yes |
| *tokudb_log_dir* | Yes | Yes | Global | No |
| *tokudb_max_lock_memory* | Yes | Yes | Global | No |
| *tokudb_optimize_index_fraction* | Yes | Yes | Session, Global | Yes |
| *tokudb_optimize_index_name* | Yes | Yes | Session, Global | Yes |
| *tokudb_optimize_throttle* | Yes | Yes | Session, Global | Yes |
| *tokudb_pk_insert_mode* | Yes | Yes | Session, Global | Yes |
| *tokudb_prelock_empty* | Yes | Yes | Session, Global | Yes |
| *tokudb_read_block_size* | Yes | Yes | Session, Global | Yes |
| *tokudb_read_buf_size* | Yes | Yes | Session, Global | Yes |
| *tokudb_read_status_frequency* | Yes | Yes | Global | Yes |
| *tokudb_row_format* | Yes | Yes | Session, Global | Yes |
| *tokudb_rpl_check_readonly* | Yes | Yes | Session, Global | Yes |
| *tokudb_rpl_lookup_rows* | Yes | Yes | Session, Global | Yes |
| *tokudb_rpl_lookup_rows_delay* | Yes | Yes | Session, Global | Yes |
| *tokudb_rpl_unique_checks* | Yes | Yes | Session, Global | Yes |
| *tokudb_rpl_unique_checks_delay* | Yes | Yes | Session, Global | Yes |
| *tokudb_strip_frm_data* | Yes | Yes | Global | No |
| *tokudb_support_xa* | Yes | Yes | Session, Global | Yes |
| *tokudb_tmp_dir* | Yes | Yes | Global | No |
| *tokudb_version* | No | No | Global | No |
| *tokudb_write_status_frequency* | Yes | Yes | Global | Yes |

**variable tokudb_alter_print_error**

**Command Line** Yes

**Config File** Yes

**Scope** Global/Session

**Dynamic** Yes

**Variable Type** Boolean

**Default Value** OFF

When set to `ON` errors will be printed to the client during the `ALTER TABLE` operations on *TokuDB* tables.

**variable `tokudb_analyze_delete_fraction`**

**Command Line** Yes

**Config File** Yes

**Scope** Global/Session

**Dynamic** Yes

**Variable Type** Numeric

**Default Value** `1.000000`

**Range** `0.0 - 1.000000`

This variables controls whether or not deleted rows in the fractal tree are reported to the client and to the *MySQL* error log during an `ANALYZE TABLE` operation on a *TokuDB* table. When set to `1`, nothing is reported. When set to `0.1` and at least 10% of the rows scanned by `ANALYZE` were deleted rows that are not yet garbage collected, a report is returned to the client and the *MySQL* error log.

**variable `tokudb_backup_allowed_prefix`**

**Command Line** No

**Config File** Yes

**Scope** Global

**Dynamic** No

**Variable Type** String

**Default Value** NULL

This system-level variable restricts the location of the destination directory where the backups can be located. Attempts to backup to a location outside of the directory this variable points to or its children will result in an error.

The default is NULL, backups have no restricted locations. This read only variable can be set in the `my.cnf` configuration file and displayed with the `SHOW VARIABLES` command when *Percona TokuBackup* plugin is loaded.

```
mysql> SHOW VARIABLES LIKE 'tokudb_backup_allowed_prefix';
+-------------------------------+-----------+
| Variable_name                 | Value     |
+-------------------------------+-----------+
| tokudb_backup_allowed_prefix | /dumpdir  |
+-------------------------------+-----------+
```

**variable `tokudb_backup_dir`**

**Command Line** No

**Config File** No

**Scope** Session

**Dynamic** Yes

**Variable Type** String

**Default Value** NULL

When enabled, this session level variable serves two purposes, to point to the destination directory where the backups will be dumped and to kick off the backup as soon as it is set. For more information see *Percona TokuBackup*.

**variable `tokudb_backup_exclude`**

**Command Line** No

**Config File** No

**Scope** Session

**Dynamic** Yes

**Variable Type** String

**Default Value** `(mysqld_safe\.pid)+`

Use this variable to set a regular expression that defines source files excluded from backup. For example, to exclude all `lost+found` directories, use the following command:

```
mysql> set tokudb_backup_exclude='/lost\\+found($|/)';
```

For more information see *Percona TokuBackup*.

**variable `tokudb_backup_last_error`**

**Command Line** Yes

**Config File** Yes

**Scope** Session, Global

**Dynamic** Yes

**Variable Type** Numeric

**Default Value** 0

This session variable will contain the error number from the last backup. `0` indicates success. For more information see *Percona TokuBackup*.

**variable `tokudb_backup_last_error_string`**

**Command Line** Yes

**Config File** Yes

**Scope** Session, Global

**Dynamic** Yes

**Variable Type** String

**Default Value** NULL

This session variable will contain the error string from the last backup. For more information see *Percona TokuBackup*.

**variable `tokudb_backup_plugin_version`**

**Command Line** No

**Config File** No

**Scope** Global

**Dynamic** No

**Variable Type** String

This read-only server variable documents the version of the *TokuBackup* plugin. For more information see *Percona TokuBackup*.

**variable `tokudb_backup_throttle`**

**Command Line** Yes

**Config File** Yes

**Scope** Session, Global

**Dynamic** Yes

**Variable Type** Numeric

**Default Value** 18446744073709551615

This variable specifies the maximum number of bytes per second the copier of a hot backup process will consume. Lowering its value will cause the hot backup operation to take more time but consume less I/O on the server. The default value is `18446744073709551615` which means no throttling. For more information see *Percona TokuBackup*.

**variable `tokudb_backup_version`**

**Command Line** No

**Config File** No

**Scope** Global

**Dynamic** No

**Variable Type** String

This read-only server variable documents the version of the hot backup library. For more information see *Percona TokuBackup*.

**variable `tokudb_block_size`**

**Command Line** Yes

**Config File** Yes

**Scope** Session, Global

**Dynamic** Yes

**Variable Type** Numeric

**Default Value** 512 MB

**Range** 4096 - 4294967295

This variable controls the maximum size of node in memory before messages must be flushed or node must be split.

Changing the value of `tokudb_block_size` only affects subsequently created tables and indexes. The value of this variable cannot be changed for an existing table/index without a dump and reload.

**variable `tokudb_bulk_fetch`**

**Command Line** Yes

**Config File** Yes

**Scope** Session, Global

**Dynamic** Yes

**Variable Type** Boolean

**Default Value** ON

This variable determines if our bulk fetch algorithm is used for `SELECT` statements. `SELECT` statements include pure `SELECT ...` statements, as well as `INSERT INTO table-name ... SELECT ..`, `CREATE TABLE table-name ... SELECT ...`, `REPLACE INTO table-name ... SELECT ..`, `INSERT IGNORE INTO table-name ... SELECT ...`, and `INSERT INTO table-name ... SELECT ... ON DUPLICATE KEY UPDATE`.

**variable `tokudb_cache_size`**

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dynamic** No

**Variable Type** Numeric

This variable configures the size in bytes of the *TokuDB* cache table. The default cache table size is 1/2 of physical memory. Percona highly recommends using the default setting if using buffered I/O, if using direct I/O then consider setting this parameter to 80% of available memory.

Consider decreasing *`tokudb_cache_size`* if excessive swapping is causing performance problems. Swapping may occur when running multiple *MySQL* server instances or if other running applications use large amounts of physical memory.

**variable `tokudb_cachetable_pool_threads`**

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dynamic** No

**Variable Type** Numeric

**Range** 0 - 1024

**Default Value** 0

This variable defines the number of threads for the cachetable worker thread pool. This pool is used to perform node prefetches, and to serialize, compress, and write nodes during cachetable eviction. The default value of 0 calculates the pool size to be num_cpu_threads * 2.

**variable `tokudb_check_jemalloc`**

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dynamic** No

**Variable Type** Boolean

**Default Value** OFF

This variable enables/disables startup checking if jemalloc is linked and correct version and that transparent huge pages are disabled. Used for testing only.

**variable `tokudb_checkpoint_lock`**

**Command Line** Yes

**Config File** Yes

**Scope** Session, Global

**Dynamic** Yes

**Variable Type** Boolean

**Default Value** OFF

Disables checkpointing when true. Session variable but acts like a global, any session disabling checkpointing disables it globally. If a session sets this lock and disconnects or terminates for any reason, the lock will not be released. Special purpose only, do **not** use this in your application.

**variable `tokudb_checkpoint_on_flush_logs`**

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dynamic** Yes

**Variable Type** Boolean

**Default Value** OFF

When enabled forces a checkpoint if we get a flush logs command from the server.

**variable `tokudb_checkpoint_pool_threads`**

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dynamic** No

**Variable Type** Numeric

**Range** 0 - 1024

**Default Value** 0

This defines the number of threads for the checkpoint worker thread pool. This pool is used to serialize, compress and write nodes cloned during checkpoint. Default of `0` uses old algorithm to set pool size to `num_cpu_threads/4`.

**variable `tokudb_checkpointing_period`**

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dynamic** Yes

**Variable Type** Numeric

**Range** 0 - 4294967295

**Default Value** 60

This variable specifies the time in seconds between the beginning of one checkpoint and the beginning of the next. The default time between *TokuDB* checkpoints is 60 seconds. We recommend leaving this variable unchanged.

**variable `tokudb_cleaner_iterations`**

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dynamic** Yes

**Variable Type** Numeric

**Range** 0 - 18446744073709551615

**Default Value** 5

This variable specifies how many internal nodes get processed in each *`tokudb_cleaner_period`* period. The default value is `5`. Setting this variable to `0` turns off cleaner threads.

**variable `tokudb_cleaner_period`**

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dynamic** Yes

**Variable Type** Numeric

**Range** 0 - 18446744073709551615

**Default Value** 1

This variable specifies how often in seconds the cleaner thread runs. The default value is `1`. Setting this variable to `0` turns off cleaner threads.

**variable `tokudb_client_pool_threads`**

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dynamic** No

**Variable Type** Numeric

**Range** 0 - 1024

**Default Value** 0

This variable defines the number of threads for the client operations thread pool. This pool is used to perform node maintenance on over/undersized nodes such as message flushing down the tree, node splits, and node merges. Default of `0` uses old algorithm to set pool size to `1 * num_cpu_threads`.

**variable `tokudb_commit_sync`**

**Command Line** Yes

**Config File** Yes

**Scope** Session, Global

**Dynamic** Yes

**Variable Type** Boolean

**Default Value** ON

Session variable *tokudb_commit_sync* controls whether or not the transaction log is flushed when a transaction commits. The default behavior is that the transaction log is flushed by the commit. Flushing the transaction log requires a disk write and may adversely affect the performance of your application.

To disable synchronous flushing of the transaction log, disable the *tokudb_commit_sync* session variable as follows:

```
SET tokudb_commit_sync=OFF;
```

Disabling this variable may make the system run faster. However, transactions committed since the last checkpoint are not guaranteed to survive a crash.

**Warning:** By disabling this variable and/or setting the *tokudb_fsync_log_period* to non-zero value you have effectively downgraded the durability of the storage engine. If you were to have a crash in this same window, you would lose data. The same issue would also appear if you were using some kind of volume snapshot for backups.

**variable tokudb_compress_buffers_before_eviction**

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dynamic** No

**Variable Type** Boolean

**Default Value** ON

When this variable is enabled it allows the evictor to compress unused internal node partitions in order to reduce memory requirements as a first step of partial eviction before fully evicting the partition and eventually the entire node.

**variable tokudb_create_index_online**

**Command Line** Yes

**Config File** Yes

**Scope** Session, Global

**Dynamic** Yes

**Variable Type** Boolean

**Default Value** ON

This variable controls whether indexes created with the `CREATE INDEX` command are hot (if enabled), or offline (if disabled). Hot index creation means that the table is available for inserts and queries while the index is being created. Offline index creation means that the table is not available for inserts and queries while the index is being created.

**Note:** Hot index creation is slower than offline index creation.

**variable `tokudb_data_dir`**

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dynamic** No

**Variable Type** String

**Default Value** `NULL`

This variable configures the directory name where the *TokuDB* tables are stored. The default value is `NULL` which uses the location of the *MySQL* data directory. For more information check *TokuDB files and file types* and *TokuDB file management*.

**variable `tokudb_debug`**

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dynamic** Yes

**Variable Type** Numeric

**Range** 0 - 18446744073709551615

**Default Value** 0

This variable enables mysqld debug printing to `STDERR` for *TokuDB*. Produces tremendous amounts of output that is nearly useless to anyone but a *TokuDB* developer, not recommended for any production use at all. It is a mask value `ULONG`:

```
#define TOKUDB_DEBUG_INIT                   (1<<0)
#define TOKUDB_DEBUG_OPEN                   (1<<1)
#define TOKUDB_DEBUG_ENTER                  (1<<2)
#define TOKUDB_DEBUG_RETURN                 (1<<3)
#define TOKUDB_DEBUG_ERROR                  (1<<4)
#define TOKUDB_DEBUG_TXN                    (1<<5)
#define TOKUDB_DEBUG_AUTO_INCREMENT         (1<<6)
#define TOKUDB_DEBUG_INDEX_KEY              (1<<7)
#define TOKUDB_DEBUG_LOCK                   (1<<8)
#define TOKUDB_DEBUG_CHECK_KEY              (1<<9)
#define TOKUDB_DEBUG_HIDE_DDL_LOCK_ERRORS   (1<<10)
#define TOKUDB_DEBUG_ALTER_TABLE            (1<<11)
#define TOKUDB_DEBUG_UPSERT                 (1<<12)
#define TOKUDB_DEBUG_CHECK                  (1<<13)
#define TOKUDB_DEBUG_ANALYZE                (1<<14)
#define TOKUDB_DEBUG_XA                     (1<<15)
#define TOKUDB_DEBUG_SHARE                  (1<<16)
```

**variable `tokudb_dir_per_db`**

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dynamic** Yes

**Variable Type** Boolean

**Default Value** ON

When this variable is set to `ON` all new tables and indices will be placed within their corresponding database directory within the `tokudb_data_dir` or system datadir. Existing table files will not be automatically relocated to their corresponding database directory. If you rename a table, while this variable is enabled, the mapping in the *Percona FT* directory file will be updated and the files will be renamed on disk to reflect the new table name. For more information check *TokuDB files and file types* and *TokuDB file management*.

**variable `tokudb_directio`**

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dynamic** No

**Variable Type** Boolean

**Default Value** OFF

When enabled, *TokuDB* employs Direct I/O rather than Buffered I/O for writes. When using Direct I/O, consider increasing `tokudb_cache_size` from its default of 1/2 physical memory.

**variable `tokudb_disable_hot_alter`**

**Command Line** Yes

**Config File** Yes

**Scope** Session, Global

**Dynamic** Yes

**Variable Type** Boolean

**Default Value** OFF

This variable is used specifically for testing or to disable hot alter in case there are bugs. Not for use in production.

**variable `tokudb_disable_prefetching`**

**Command Line** Yes

**Config File** Yes

**Scope** Session, Global

**Dynamic** Yes

**Variable Type** Boolean

**Default Value** OFF

*TokuDB* attempts to aggressively prefetch additional blocks of rows, which is helpful for most range queries but may create unnecessary I/O for range queries with `LIMIT` clauses. Prefetching is `ON` by default, with a value of `0`, it can be disabled by setting this variable to `1`.

**variable `tokudb_disable_slow_alter`**

**Command Line** Yes

**Config File** Yes

**Scope** Session, Global

**Dynamic** Yes

**Variable Type** Boolean

**Default Value** OFF

This variable is used specifically for testing or to disable hot alter in case there are bugs. Not for use in production. It controls whether slow alter tables are allowed. For example, the following command is slow because `HCADER` does not allow a mixture of column additions, deletions, or expansions:

```
ALTER TABLE table
ADD COLUMN column_a INT,
DROP COLUMN column_b;
```

By default, *tokudb_disable_slow_alter* is disabled, and the engine reports back to *MySQL* that this is unsupported resulting in the following output:

```
ERROR 1112 (42000): Table 'test_slow' uses an extension that doesn't exist in this␣
↪MySQL version
```

**variable tokudb_empty_scan**

**Command Line** Yes

**Config File** Yes

**Scope** Global/Session

**Dynamic** Yes

**Variable Type** ENUM

**Default Value** `rl`

**Range** `disabled`, `rl` - right to left, `lr` - left to right

Defines direction to be used to perform table scan to check for empty tables for bulk loader.

**variable tokudb_enable_fast_update**

**Command Line** Yes

**Config File** Yes

**Scope** Global/Session

**Dynamic** Yes

**Variable Type** Boolean

**Default Value** OFF

Toggles the fast updates feature ON/OFF for the `UPDATE` statement. Fast update involves queries optimization to avoid random reads during their execution.

**variable tokudb_enable_fast_upsert**

**Command Line** Yes

**Config File** Yes

**Scope** Global/Session

**Dynamic** Yes

**Variable Type** Boolean

**Default Value** OFF

Toggles the fast updates feature ON/OFF for the `INSERT` statement. Fast update involves queries optimization to avoid random reads during their execution.

**variable `tokudb_enable_partial_eviction`**

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dynamic** No

**Variable Type** Boolean

**Default Value** OFF

This variable is used to control if partial eviction of nodes is enabled or disabled.

**variable `tokudb_fanout`**

**Command Line** Yes

**Config File** Yes

**Scope** Session, Global

**Dynamic** Yes

**Variable Type** Numeric

**Range** 2-16384

**Default Value** 16

This variable controls the Fractal Tree fanout. The fanout defines the number of pivot keys or child nodes for each internal tree node. Changing the value of *`tokudb_fanout`* only affects subsequently created tables and indexes. The value of this variable cannot be changed for an existing table/index without a dump and reload.

**variable `tokudb_fs_reserve_percent`**

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dynamic** No

**Variable Type** Numeric

**Range** 0-100

**Default Value** 5

This variable controls the percentage of the file system that must be available for inserts to be allowed. By default, this is set to `5`. We recommend that this reserve be at least half the size of your physical memory. See *Full Disks* for more information.

**variable `tokudb_fsync_log_period`**

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dynamic** Yes

**Variable Type** Numeric

**Range** 0-4294967295

**Default Value** 0

This variable controls the frequency, in milliseconds, for `fsync()` operations. If set to `0` then the `fsync()` behavior is only controlled by the *tokudb_commit_sync*, which can be `ON` or `OFF`.

**variable tokudb_hide_default_row_format**

**Command Line** Yes

**Config File** Yes

**Scope** Session, Global

**Dynamic** Yes

**Variable Type** Boolean

**Default Value** ON

This variable is used to hide the `ROW_FORMAT` in `SHOW CREATE TABLE`. If `zlib` compression is used, row format will show as `DEFAULT`.

**variable tokudb_killed_time**

**Command Line** Yes

**Config File** Yes

**Scope** Session, Global

**Dynamic** Yes

**Variable Type** Numeric

**Range** 0-18446744073709551615

**Default Value** 4000

This variable is used to specify frequency in milliseconds for lock wait to check to see if the lock was killed.

**variable tokudb_last_lock_timeout**

**Command Line** Yes

**Config File** Yes

**Scope** Session, Global

**Dynamic** Yes

**Variable Type** String

**Default Value** NULL

This variable contains a JSON document that describes the last lock conflict seen by the current *MySQL* client. It gets set when a blocked lock request times out or a lock deadlock is detected.

The *tokudb_lock_timeout_debug* session variable must have bit `0` set for this behavior, otherwise this session variable will be empty.

**variable tokudb_load_save_space**

**Command Line** Yes

**Config File** Yes

**Scope** Session, Global

**Dynamic** Yes

**Variable Type** Boolean

**Default Value** ON

This session variable changes the behavior of the bulk loader. When it is disabled the bulk loader stores intermediate data using uncompressed files (which consumes additional CPU), whereas `ON` compresses the intermediate files.

---

**Note:** The location of the temporary disk space used by the bulk loader may be specified with the *tokudb_tmp_dir* server variable.

---

If a `LOAD DATA INFILE` statement fails with the error message `ERROR 1030 (HY000): Got error 1` from storage engine, then there may not be enough disk space for the optimized loader, so disable *tokudb_prelock_empty* and try again. More information is available in *Known Issues*.

**variable `tokudb_loader_memory_size`**

**Command Line** Yes

**Config File** Yes

**Scope** Session, Global

**Dynamic** Yes

**Variable Type** Numeric

**Range** 0-18446744073709551615

**Default Value** 100000000

This variable limits the amount of memory (in bytes) that the *TokuDB* bulk loader will use for each loader instance. Increasing this value may provide a performance benefit when loading extremely large tables with several secondary indexes.

---

**Note:** Memory allocated to a loader is taken from the *TokuDB* cache, defined in *tokudb_cache_size*, and may impact the running workload's performance as existing cached data must be ejected for the loader to begin.

---

**variable `tokudb_lock_timeout`**

**Command Line** Yes

**Config File** Yes

**Scope** Session, Global

**Dynamic** Yes

**Variable Type** Numeric

**Range** 0-18446744073709551615

**Default Value** 4000

This variable controls the amount of time that a transaction will wait for a lock held by another transaction to be released. If the conflicting transaction does not release the lock within the lock timeout, the transaction that was waiting for the lock will get a lock timeout error. The units are milliseconds. A value of `0` disables lock waits. The default value is 4000 (four seconds).

If your application gets a `lock wait timeout` error (-30994), then you may find that increasing the *`tokudb_lock_timeout`* may help. If your application gets a `deadlock found` error (-30995), then you need to abort the current transaction and retry it.

**variable `tokudb_lock_timeout_debug`**

**Command Line** Yes

**Config File** Yes

**Scope** Session, Global

**Dynamic** Yes

**Variable Type** Numeric

**Range** 0-3

**Default Value** 1

The following values are available:

- `0`: No lock timeouts or lock deadlocks are reported.
- `1`: A JSON document that describes the lock conflict is stored in the *`tokudb_last_lock_timeout`* session variable
- `2`: A JSON document that describes the lock conflict is printed to the MySQL error log.

  In addition to the JSON document describing the lock conflict, the following lines are printed to the MySQL error log:

  - A line containing the blocked thread id and blocked SQL
  - A line containing the blocking thread id and the blocking SQL.
- `3`: A JSON document that describes the lock conflict is stored in the *`tokudb_last_lock_timeout`* session variable and is printed to the MySQL error log.

  In addition to the JSON document describing the lock conflict, the following lines are printed to the MySQL error log:

  - A line containing the blocked thread id and blocked SQL
  - A line containing the blocking thread id and the blocking SQL.

**variable `tokudb_log_dir`**

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dynamic** No

**Variable Type** String

**Default Value** NULL

This variable specifies the directory where the *TokuDB* log files are stored. The default value is `NULL` which uses the location of the *MySQL* data directory. Configuring a separate log directory is somewhat involved. Please contact Percona support for more details. For more information check *TokuDB files and file types* and *TokuDB file management*.

**Warning:** After changing *TokuDB* log directory path, the old *TokuDB* recovery log file should be moved to new directory prior to start of *MySQL* server and log file's owner must be the `mysql` user. Otherwise server will fail to initialize the *TokuDB* store engine restart.

**variable `tokudb_max_lock_memory`**

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dynamic** No

**Variable Type** Numeric

**Range** 0-18446744073709551615

**Default Value** 65560320

This variable specifies the maximum amount of memory for the PerconaFT lock table.

**variable `tokudb_optimize_index_fraction`**

**Command Line** Yes

**Config File** Yes

**Scope** Session, Global

**Dynamic** Yes

**Variable Type** Numeric

**Range** 0.000000 - 1.000000

**Default Value** 1.000000

For patterns where the left side of the tree has many deletions (a common pattern with increasing id or date values), it may be useful to delete a percentage of the tree. In this case, it's possible to optimize a subset of a fractal tree starting at the left side. The *tokudb_optimize_index_fraction* session variable controls the size of the sub tree. Valid values are in the range [0.0,1.0] with default 1.0 (optimize the whole tree).

**variable `tokudb_optimize_index_name`**

**Command Line** Yes

**Config File** Yes

**Scope** Session, Global

**Dynamic** Yes

**Variable Type** String

**Default Value** NULL

This variable can be used to optimize a single index in a table, it can be set to select the index by name.

**variable `tokudb_optimize_throttle`**

**Command Line** Yes

**Config File** Yes

**Scope** Session, Global

**Dynamic** Yes

**Variable Type** Numeric

**Range** 0-18446744073709551615

**Default Value** 0

By default, table optimization will run with all available resources. To limit the amount of resources, it is possible to limit the speed of table optimization. This determines an upper bound on how many fractal tree leaf nodes per second are optimized. The default `0` imposes no limit.

**variable `tokudb_pk_insert_mode`**

**Command Line** Yes

**Config File** Yes

**Scope** Session, Global

**Dynamic** Yes

**Variable Type** Numeric

**Range** 0-3

**Default Value** 1

---

**Note:** The *`tokudb_pk_insert_mode`* session variable was removed and the behavior is now that of the former *`tokudb_pk_insert_mode`* set to `1`. The optimization will be used where safe and not used where not safe.

---

**variable `tokudb_prelock_empty`**

**Command Line** Yes

**Config File** Yes

**Scope** Session, Global

**Dynamic** Yes

**Variable Type** Boolean

**Default Value** ON

By default *TokuDB* preemptively grabs an entire table lock on empty tables. If one transaction is doing the loading, such as when the user is doing a table load into an empty table, this default provides a considerable speedup.

However, if multiple transactions try to do concurrent operations on an empty table, all but one transaction will be locked out. Disabling *`tokudb_prelock_empty`* optimizes for this multi-transaction case by turning off preemptive pre-locking.

---

**Note:** If this variable is set to `OFF`, fast bulk loading is turned off as well.

---

**variable `tokudb_read_block_size`**

**Command Line** Yes

**Config File** Yes

**Scope** Session, Global

**Dynamic** Yes

**Variable Type** Numeric

**Range** 4096 - 4294967295

**Default Value** 16384 (16KB)

Fractal tree leaves are subdivided into read blocks, in order to speed up point queries. This variable controls the target uncompressed size of the read blocks. The units are bytes and the default is 64 KB. A smaller value favors read performance for point and small range scans over large range scans and higher compression. The minimum value of this variable is 4096 (4KB).

Changing the value of *tokudb_read_block_size* only affects subsequently created tables. The value of this variable cannot be changed for an existing table without a dump and reload.

**variable tokudb_read_buf_size**

**Command Line** Yes

**Config File** Yes

**Scope** Session, Global

**Dynamic** Yes

**Variable Type** Numeric

**Range** 0 - 1048576

**Default Value** 131072 (128KB)

This variable controls the size of the buffer used to store values that are bulk fetched as part of a large range query. Its unit is bytes and its default value is 131,072 (128 KB).

A value of `0` turns off bulk fetching. Each client keeps a thread of this size, so it should be lowered if situations where there are a large number of clients simultaneously querying a table.

**variable tokudb_read_status_frequency**

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dynamic** Yes

**Variable Type** Numeric

**Range** 0 - 4294967295

**Default Value** 10000

This variable controls in how many reads the progress is measured to display `SHOW PROCESSLIST`. Reads are defined as `SELECT` queries.

For slow queries, it can be helpful to set this variable and *tokudb_write_status_frequency* to `1`, and then run `SHOW PROCESSLIST` several times to understand what progress is being made.

**variable tokudb_row_format**

**Command Line** Yes

**Config File** Yes

**Scope** Session, Global

**Dynamic** Yes

**Variable Type** ENUM

**Range** TOKUDB_DEFAULT, TOKUDB_FAST, TOKUDB_SMALL, TOKUDB_ZLIB, TOKUDB_QUICKLZ, TOKUDB_LZMA, TOKUDB_SNAPPY, TOKUDB_UNCOMPRESSED

**Default Value** TOKUDB_QUICKLZ

This controls the default compression algorithm used to compress data. For more information on compression algorithms see *Compression Details*.

**variable tokudb_rpl_check_readonly**

**Command Line** Yes

**Config File** Yes

**Scope** Session, Global

**Dynamic** Yes

**Variable Type** Boolean

**Default Value** ON

The *TokuDB* replication code will run row events from the binary log with *Read Free Replication* when the slave is in read-only mode. This variable is used to disable the slave read only check in the *TokuDB* replication code.

This allows Read-Free-Replication to run when the slave is NOT read-only. By default, *tokudb_rpl_check_readonly* is enabled (check that slave is read-only). Do **NOT** change this value unless you completely understand the implications!

**variable tokudb_rpl_lookup_rows**

**Command Line** Yes

**Config File** Yes

**Scope** Session, Global

**Dynamic** Yes

**Variable Type** Boolean

**Default Value** ON

When disabled, *TokuDB* replication slaves skip row lookups for `delete row` log events and `update row` log events, which eliminates all associated read I/O for these operations.

**Warning:** *TokuDB Read Free Replication* will not propagate `UPDATE` and `DELETE` events reliably if *TokuDB* table is missing the primary key which will eventually lead to data inconsistency on the slave.

**Note:** Optimization is only enabled when `read_only` is set to `1` and `binlog_format` is `ROW`.

**variable tokudb_rpl_lookup_rows_delay**

**Command Line** Yes

**Config File** Yes

**Scope** Session, Global

**Dynamic** Yes

**Variable Type** Numeric

**Range** 0 - 18446744073709551615

**Default Value** 0

This variable allows for simulation of long disk reads by sleeping for the given number of microseconds prior to the row lookup query, it should only be set to a non-zero value for testing.

**variable `tokudb_rpl_unique_checks`**

**Command Line** Yes

**Config File** Yes

**Scope** Session, Global

**Dynamic** Yes

**Variable Type** Boolean

**Default Value** ON

When disabled, *TokuDB* replication slaves skip uniqueness checks on inserts and updates, which eliminates all associated read I/O for these operations.

---

**Note:** Optimization is only enabled when `read_only` is set to `1` and `binlog_format` is `ROW`.

---

**variable `tokudb_rpl_unique_checks_delay`**

**Command Line** Yes

**Config File** Yes

**Scope** Session, Global

**Dynamic** Yes

**Variable Type** Numeric

**Range** 0 - 18446744073709551615

**Default Value** 0

This variable allows for simulation of long disk reads by sleeping for the given number of microseconds prior to the row lookup query, it should only be set to a non-zero value for testing.

**variable `tokudb_strip_frm_data`**

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dynamic** No

**Variable Type** Boolean

**Default Value** OFF

When this variable is set to `ON` during the startup server will check all the status files and remove the embedded `.frm` metadata. This variable can be used to assist in *TokuDB* data recovery. **WARNING:** Use this variable only if you know what you're doing otherwise it could lead to data loss.

**variable `tokudb_support_xa`**

**Command Line** Yes

**Config File** Yes

**Scope** Session, Global

**Dynamic** Yes

**Variable Type** Boolean

**Default Value** ON

This variable defines whether or not the prepare phase of an XA transaction performs an `fsync()`.

**variable `tokudb_tmp_dir`**

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dynamic** No

**Variable Type** String

This variable specifies the directory where the *TokuDB* bulk loader stores temporary files. The bulk loader can create large temporary files while it is loading a table, so putting these temporary files on a disk separate from the data directory can be useful.

*`tokudb_load_save_space`* determines whether the data is compressed or not. The error message `ERROR 1030 (HY000): Got error 1 from storage engine` could indicate that the disk has run out of space.

For example, it can make sense to use a high-performance disk for the data directory and a very inexpensive disk for the temporary directory. The default location for temporary files is the *MySQL* data directory.

For more information check *TokuDB files and file types* and *TokuDB file management*.

**variable `tokudb_version`**

**Command Line** No

**Config File** No

**Scope** Global

**Dynamic** No

**Variable Type** String

This read-only variable documents the version of the *TokuDB* storage engine.

**variable `tokudb_write_status_frequency`**

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dynamic** Yes

**Variable Type** Numeric

**Range** 0 - 4294967295

**Default Value** 1000

This variable controls in how many writes the progress is measured to display `SHOW PROCESSLIST`. Writes are defined as `INSERT`, `UPDATE` and `DELETE` queries.

For slow queries, it can be helpful to set this variable and *`tokudb_read_status_frequency`* to 1, and then run `SHOW PROCESSLIST` several times to understand what progress is being made.

# Percona TokuBackup

Percona *TokuBackup* is an open-source hot backup utility for *MySQL* servers running the *TokuDB* storage engine (including *Percona Server for MySQL* and *MariaDB*). It does not lock your database during backup. The *TokuBackup* library intercepts system calls that write files and duplicates the writes to the backup directory.

**Note:** This feature is currently considered *Experimental*



## Installing From Binaries

The installation of *TokuBackup* can be performed with the **ps-admin** script.

To install *Percona TokuBackup* complete the following steps. Run the following commands as root or by using the **sudo** command.

1. Run **ps-admin --enable-tokubackup** to add the `preload-hotbackup` option into **[mysqld_safe]** section of `my.cnf`.

   **Output**

   ```
   Checking SELinux status...
   INFO: SELinux is disabled.

   Checking if preload-hotbackup option is already set in config file...
   INFO: Option preload-hotbackup is not set in the config file.

   Checking TokuBackup plugin status...
   INFO: TokuBackup plugin is not installed.

   Adding preload-hotbackup option into /etc/my.cnf
   INFO: Successfully added preload-hotbackup option into /etc/my.cnf
   PLEASE RESTART MYSQL SERVICE AND RUN THIS SCRIPT AGAIN TO FINISH INSTALLATION!
   ```

2. Restart mysql service: `service mysql restart`

3. Run **ps-admin --enable-tokubackup** again to finish the installation of the *TokuBackup* plugin.

**Output**

```
Checking SELinux status...
INFO: SELinux is disabled.

Checking if preload-hotbackup option is already set in config file...
INFO: Option preload-hotbackup is set in the config file.

Checking TokuBackup plugin status...
INFO: TokuBackup plugin is not installed.

Checking if Percona Server is running with libHotBackup.so preloaded...
INFO: Percona Server is running with libHotBackup.so preloaded.

Installing TokuBackup plugin...
INFO: Successfully installed TokuBackup plugin.
```

## Making a Backup

To run *Percona TokuBackup*, the backup destination directory must exist, be writable and owned by the same user under which *MySQL* server is running (usually `mysql`) and empty.

Once this directory is created, the backup can be run using the following command:

```
mysql> set tokudb_backup_dir='/path_to_empty_directory';
```

**Note:** Setting the *tokudb_backup_dir* variable automatically starts the backup process to the specified directory. Percona TokuBackup will take full backup each time, currently there is no incremental backup option

If you get any error on this step (e.g. caused by some misconfiguration), the *Reporting Errors* section explains how to find out the reason.

## Restoring From Backup

*Percona TokuBackup* does not have any functionality for restoring a backup. You can use **rsync** or **cp** to restore the files. You should check that the restored files have the correct ownership and permissions.

**Note:** Make sure that the datadir is empty and that *MySQL* server is shut down before restoring from backup. You can't restore to a datadir of a running mysqld instance (except when importing a partial backup).

The following example shows how you might use the **rsync** command to restore the backup:

```
$ rsync -avrP /data/backup/ /var/lib/mysql/
```

Since attributes of files are preserved, in most cases you will need to change their ownership to *mysql* before starting the database server. Otherwise, the files will be owned by the user who created the backup.

```
$ chown -R mysql:mysql /var/lib/mysql
```

If you have changed default *TokuDB* data directory (*tokudb_data_dir*) or *TokuDB* log directory (*tokudb_log_dir*) or both of them, you will see separate folders for each setting in backup directory after taking backup. You'll need to restore each folder separately:

```
$ rsync -avrP /data/backup/mysql_data_dir/ /var/lib/mysql/
$ rsync -avrP /data/backup/tokudb_data_dir/ /path/to/original/tokudb_data_dir/
$ rsync -avrP /data/backup/tokudb_log_dir/ /path/to/original/tokudb_log_dir/
$ chown -R mysql:mysql /var/lib/mysql
$ chown -R mysql:mysql /path/to/original/tokudb_data_dir
$ chown -R mysql:mysql /path/to/original/tokudb_log_dir
```

## Advanced Configuration



### Monitoring Progress

*TokuBackup* updates the *PROCESSLIST* state while the backup is in progress. You can see the output by running `SHOW PROCESSLIST` or `SHOW FULL PROCESSLIST`.

### Excluding Source Files

You can exclude certain files and directories based on a regular expression set in the *tokudb_backup_exclude* session variable. If the source file name matches the excluded regular expression, then the source file is excluded from backup.

For example, to exclude all `lost+found` directories from backup, use the following command:

```
mysql> SET tokudb_backup_exclude='/lost\\+found($|/)';
```

**Note:** The server `pid` file is excluded by default. If you're providing your own additions to the exclusions and have the `pid` file in the default location, you will need to add the mysqld_safe.pid entry.

### Throttling Backup Rate

You can throttle the backup rate using the *tokudb_backup_throttle* session-level variable. This variable throttles the write rate in bytes per second of the backup to prevent TokuBackup from crowding out other jobs in the system. The default and max value is 18446744073709551615.

```
mysql> SET tokudb_backup_throttle=1000000;
```

### Restricting Backup Target

You can restrict the location of the destination directory where the backups can be located using the *tokudb_backup_allowed_prefix* system-level variable. Attempts to backup to a location outside of the specified directory or its children will result in an error.

The default is `null`, backups have no restricted locations. This read-only variable can be set in the `my.cnf` configuration file and displayed with the `SHOW VARIABLES` command:

```
mysql> SHOW VARIABLES LIKE 'tokudb_backup_allowed_prefix';
+-------------------------------+-----------+
| Variable_name                 | Value     |
+-------------------------------+-----------+
| tokudb_backup_allowed_prefix | /dumpdir  |
+-------------------------------+-----------+
```

### Reporting Errors

*Percona TokuBackup* uses two variables to capture errors. They are *tokudb_backup_last_error* and *tokudb_backup_last_error_string*. When *TokuBackup* encounters an error, these will report on the error number and the error string respectively. For example, the following output shows these parameters following an attempted backup to a directory that was not empty:

```
mysql> SET tokudb_backup_dir='/tmp/backupdir';
ERROR 1231 (42000): Variable 'tokudb_backup_dir' can't be set to the value of '/tmp/
↪backupdir'

mysql> SELECT @@tokudb_backup_last_error;
+-----------------------------+
| @@tokudb_backup_last_error |
+-----------------------------+
|                          17 |
+-----------------------------+

mysql> SELECT @@tokudb_backup_last_error_string;
+-------------------------------------------------+
| @@tokudb_backup_last_error_string               |
+-------------------------------------------------+
| tokudb backup couldn't create needed directories. |
+-------------------------------------------------+
```

## Limitations and known issues

- You must disable *InnoDB* asynchronous IO if backing up *InnoDB* tables with *TokuBackup*. Otherwise you will have inconsistent, unrecoverable backups. The appropriate setting is `innodb_use_native_aio=0`.
- To be able to run Point-In-Time-Recovery you'll need to manually get the binary log position.
- Transactional storage engines (*TokuDB* and *InnoDB*) will perform recovery on the backup copy of the database when it is first started.

- Tables using non-transactional storage engines (*MyISAM*) are not locked during the copy and may report issues when starting up the backup. It is best to avoid operations that modify these tables at the end of a hot backup operation (adding/changing users, stored procedures, etc.).
- The database is copied locally to the path specified in `/path/to/backup`. This folder must exist, be writable, be empty, and contain enough space for a full copy of the database.
- *TokuBackup* always makes a backup of the *MySQL* `datadir` and optionally the *tokudb_data_dir*, *tokudb_log_dir*, and the binary log folder. The latter three are only backed up separately if they are not the same as or contained in the *MySQL* `datadir`. None of these three folders can be a parent of the *MySQL* `datadir`.
- No other directory structures are supported. All *InnoDB*, *MyISAM*, and other storage engine files must be within the *MySQL* `datadir`.
- *TokuBackup* does not follow symbolic links.
- *TokuBackup* does not backup *MySQL* configuration file(s).
- *TokuBackup* does not backup tablespaces if they are out of `datadir`.
- Due to upstream bug #80183, *TokuBackup* can't recover backed-up table data if backup was taken while running `OPTIMIZE TABLE` or `ALTER TABLE ... TABLESPACE`.
- *TokuBackup* doesn't support incremental backups.

# TokuDB Troubleshooting



## Known Issues

**Replication and binary logging**: *TokuDB* supports binary logging and replication, with one restriction. *TokuDB* does not implement a lock on the auto-increment function, so concurrent insert statements with one or more of the statements inserting multiple rows may result in a non-deterministic interleaving of the auto-increment values. When running replication with these concurrent inserts, the auto-increment values on the slave table may not match the auto-increment values on the master table. Note that this is only an issue with Statement Based Replication (SBR), and not Row Based Replication (RBR).

For more information about auto-increment and replication, see the *MySQL* Reference Manual: AUTO_INCREMENT handling in InnoDB.

In addition, when using the `REPLACE INTO` or `INSERT IGNORE` on tables with no secondary indexes or tables where secondary indexes are subsets of the primary, the session variable *tokudb_pk_insert_mode* controls whether row based replication will work.

**Uninformative error message: The `LOAD DATA INFILE` command can sometimes** produce `ERROR 1030 (HY000): Got error 1 from storage engine`. The message should say that the error is caused by insufficient disk space for the temporary files created by the loader.

**Transparent Huge Pages: *TokuDB* will refuse to start if transparent huge** pages are enabled. Transparent huge page support can be disabled by issuing the following as root:

```
# echo never > /sys/kernel/mm/redhat_transparent_hugepage/enabled
```

**Note:** The previous command needs to be executed after every reboot, because it defaults to `always`.

**XA behavior vs. InnoDB:** ***InnoDB*** **forces a deadlocked XA transaction to** abort, *TokuDB* does not.

**Disabling the unique checks**: For tables with unique keys, every insertion into the table causes a lookup by key followed by an insertion, if the key is not in the table. This greatly limits insertion performance. If one knows by design that the rows being inserted into the table have unique keys, then one can disable the key lookup prior to insertion.

If your primary key is an auto-increment key, and none of your secondary keys are declared to be unique, then setting `unique_checks=OFF` will provide limited performance gains. On the other hand, if your primary key has a lot of entropy (it looks random), or your secondary keys are declared unique and have a lot of entropy, then disabling unique checks can provide a significant performance boost.

If `unique_checks` is disabled when the primary key is not unique, secondary indexes may become corrupted. In this case, the indexes should be dropped and rebuilt. This behavior differs from that of *InnoDB*, in which uniqueness is always checked on the primary key, and setting `unique_checks` to off turns off uniqueness checking on secondary indexes only. Turning off uniqueness checking on the primary key can provide large performance boosts, but it should only be done when the primary key is known to be unique.

**Group Replication**: *TokuDB* storage engine doesn't support Group Replication.

As of 8.0.17, InnoDB supports multi-valued indexes. TokuDB does not support this feature.

As of 8.0.17, InnoDB supports the Clone Plugin and the Clone Plugin API. TokuDB tables do not support either of these features.

## Lock Visualization in TokuDB

*TokuDB* uses key range locks to implement serializable transactions, which are acquired as the transaction progresses. The locks are released when the transaction commits or aborts (this implements two phase locking).

*TokuDB* stores these locks in a data structure called the lock tree. The lock tree stores the set of range locks granted to each transaction. In addition, the lock tree stores the set of locks that are not granted due to a conflict with locks granted to some other transaction. When these other transactions are retired, these pending lock requests are retried. If a pending lock request is not granted before the lock timer expires, then the lock request is aborted.

Lock visualization in *TokuDB* exposes the state of the lock tree with tables in the information schema. We also provide a mechanism that may be used by a database client to retrieve details about lock conflicts that it encountered while executing a transaction.

### The `TOKUDB_TRX` table

The `TOKUDB_TRX` table in the `INFORMATION_SCHEMA` maps *TokuDB* transaction identifiers to *MySQL* client identifiers. This mapping allows one to associate a *TokuDB* transaction with a *MySQL* client operation.

The following query returns the *MySQL* clients that have a live *TokuDB* transaction:

```
SELECT * FROM INFORMATION_SCHEMA.TOKUDB_TRX,
INFORMATION_SCHEMA.PROCESSLIST
WHERE trx_mysql_thread_id = id;
```

### The TOKUDB_LOCKS table

The tokudb_locks table in the information schema contains the set of locks granted to *TokuDB* transactions.

The following query returns all of the locks granted to some *TokuDB* transaction:

```
SELECT * FROM INFORMATION_SCHEMA.TOKUDB_LOCKS;
```

The following query returns the locks granted to some *MySQL* client:

```
SELECT id FROM INFORMATION_SCHEMA.TOKUDB_LOCKS,
INFORMATION_SCHEMA.PROCESSLIST
WHERE locks_mysql_thread_id = id;
```

### The TOKUDB_LOCK_WAITS table

The tokudb_lock_waits table in the information schema contains the set of lock requests that are not granted due to a lock conflict with some other transaction.

The following query returns the locks that are waiting to be granted due to a lock conflict with some other transaction:

```
SELECT * FROM INFORMATION_SCHEMA.TOKUDB_LOCK_WAITS;
```

### Supporting explicit DEFAULT value expressions as of 8.0.13-3

TokuDB does not support explicit DEFAULT value expressions as of verion 8.0.13-3.

### The tokudb_lock_timeout_debug session variable

The *tokudb_lock_timeout_debug* session variable controls how lock timeouts and lock deadlocks seen by the database client are reported.

The following values are available:

**0** No lock timeouts or lock deadlocks are reported.

**1** A JSON document that describes the lock conflict is stored in the *tokudb_last_lock_timeout* session variable

**2** A JSON document that describes the lock conflict is printed to the *MySQL* error log.

*Supported since 7.5.5*: In addition to the JSON document describing the lock conflict, the following lines are printed to the MySQL error log:

- A line containing the blocked thread id and blocked SQL
- A line containing the blocking thread id and the blocking SQL.

**3** A JSON document that describes the lock conflict is stored in the *tokudb_last_lock_timeout* session variable and is printed to the *MySQL* error log.

*Supported since 7.5.5*: In addition to the JSON document describing the lock conflict, the following lines are printed to the *MySQL* error log:

- A line containing the blocked thread id and blocked SQL
- A line containing the blocking thread id and the blocking SQL.

### The `tokudb_last_lock_timeout` session variable

The *tokudb_last_lock_timeout* session variable contains a JSON document that describes the last lock conflict seen by the current *MySQL* client. It gets set when a blocked lock request times out or a lock deadlock is detected. The *tokudb_lock_timeout_debug* session variable should have bit `0` set (decimal `1`).

### Example

Suppose that we create a table with a single column that is the primary key.

```
mysql> SHOW CREATE TABLE table;

Create Table: CREATE TABLE `table` (
`id` int(11) NOT NULL,
PRIMARY KEY (`id`)) ENGINE=TokuDB DEFAULT CHARSET=latin1
```

Suppose that we have 2 *MySQL* clients with ID's 1 and 2 respectively. Suppose that *MySQL* client 1 inserts some values into `table`. *TokuDB* transaction 51 is created for the insert statement. Since autocommit is disabled, transaction 51 is still live after the insert statement completes, and we can query the `tokudb_locks` table in information schema to see the locks that are held by the transaction.

```
mysql> SET AUTOCOMMIT=OFF;
mysql> INSERT INTO table VALUES (1),(10),(100);
```

**Output**

```
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

```
mysql> SELECT * FROM INFORMATION_SCHEMA.TOKUDB_LOCKS;
```

**Output**

```
+--------------+-----------------------+---------------+----------------+-------------
↪----+--------------------+------------------+-----------------------------+
| locks_trx_id | locks_mysql_thread_id | locks_dname   | locks_key_left | locks_key_
↪right | locks_table_schema | locks_table_name | locks_table_dictionary_name |
+--------------+-----------------------+---------------+----------------+-------------
↪----+--------------------+------------------+-----------------------------+
|           51 |                     1 | ./test/t-main | 0001000000     | 0001000000 ␣
↪    | test               | t                | main                        |
|           51 |                     1 | ./test/t-main | 000a000000     | 000a000000 ␣
↪    | test               | t                | main                        |
|           51 |                     1 | ./test/t-main | 0064000000     | 0064000000 ␣
↪    | test               | t                | main                        |
+--------------+-----------------------+---------------+----------------+-------------
↪----+--------------------+------------------+-----------------------------+
```

```
mysql> SELECT * FROM INFORMATION_SCHEMA.TOKUDB_LOCK_WAITS;
```

**Output**

```
Empty set (0.00 sec)
```

The keys are currently hex dumped.

Now we switch to the other *MySQL* client with ID 2.

```
mysql> INSERT INTO table VALUES (2),(20),(100);
```

The insert gets blocked since there is a conflict on the primary key with value 100.

The granted *TokuDB* locks are:

```
SELECT * FROM INFORMATION_SCHEMA.TOKUDB_LOCKS;
```

**Output**

```
+--------------+----------------------+---------------+----------------+-------------
↪----+-------------------+------------------+-----------------------------+
| locks_trx_id | locks_mysql_thread_id | locks_dname   | locks_key_left | locks_key_
↪right | locks_table_schema | locks_table_name | locks_table_dictionary_name |
+--------------+----------------------+---------------+----------------+-------------
↪----+-------------------+------------------+-----------------------------+
|           51 |                    1 | ./test/t-main | 0001000000     | 0001000000 ␣
↪    | test               | t                | main                        |
|           51 |                    1 | ./test/t-main | 000a000000     | 000a000000 ␣
↪    | test               | t                | main                        |
|           51 |                    1 | ./test/t-main | 0064000000     | 0064000000 ␣
↪    | test               | t                | main                        |
|           51 |                    1 | ./test/t-main | 0002000000     | 0002000000 ␣
↪    | test               | t                | main                        |
|           51 |                    1 | ./test/t-main | 0014000000     | 0014000000 ␣
↪    | test               | t                | main                        |
+--------------+----------------------+---------------+----------------+-------------
↪----+-------------------+------------------+-----------------------------+
```

The locks that are pending due to a conflict are:

```
SELECT * FROM INFORMATION_SCHEMA.TOKUDB_LOCK_WAITS;

+-------------------+-----------------+-----------------+----------------------+------
↪---------------+-----------------------+-------------------+------------------+---
↪---------------------------+
| requesting_trx_id | blocking_trx_id | lock_waits_dname | lock_waits_key_left | lock_
↪waits_key_right | lock_waits_start_time | locks_table_schema | locks_table_name |␣
↪locks_table_dictionary_name |
+-------------------+-----------------+-----------------+----------------------+------
↪---------------+-----------------------+-------------------+------------------+---
↪---------------------------+
|                62 |              51 | ./test/t-main    | 0064000000           |␣
↪0064000000           |         1380656990910 | test               | t                ␣
↪ | main                      |
+-------------------+-----------------+-----------------+----------------------+------
↪---------------+-----------------------+-------------------+------------------+---
↪---------------------------+
```

Eventually, the lock for client 2 times out, and we can retrieve a JSON document that describes the conflict.

**Error**

ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction

```
SELECT @@TOKUDB_LAST_LOCK_TIMEOUT;
```

**Output**

```
+-----------------------------------------------------------------------------------
↪-------------------------+
| @@tokudb_last_lock_timeout
↪                         |
+-----------------------------------------------------------------------------------
↪-------------------------+
| "mysql_thread_id":2, "dbname":"./test/t-main", "requesting_txnid":62, "blocking_
↪txnid":51, "key":"0064000000" |
+-----------------------------------------------------------------------------------
↪-------------------------+
```

```
ROLLBACK;
```

Since transaction 62 was rolled back, all of the locks taken by it are released.

```
SELECT * FROM INFORMATION_SCHEMA.TOKUDB_LOCKS;
```

**Output**

```
+--------------+----------------------+---------------+----------------+-------------
↪----+--------------------+------------------+-----------------------------+
| locks_trx_id | locks_mysql_thread_id | locks_dname   | locks_key_left | locks_key_
↪right | locks_table_schema | locks_table_name | locks_table_dictionary_name |
+--------------+----------------------+---------------+----------------+-------------
↪----+--------------------+------------------+-----------------------------+
|           51 |                    1 | ./test/t-main | 0001000000     | 0001000000
↪    | test               | t                | main                        |
|           51 |                    1 | ./test/t-main | 000a000000     | 000a000000
↪    | test               | t                | main                        |
|           51 |                    1 | ./test/t-main | 0064000000     | 0064000000
↪    | test               | t                | main                        |
|           51 |                    2 | ./test/t-main | 0002000000     | 0002000000
↪    | test               | t                | main                        |
|           51 |                    2 | ./test/t-main | 0014000000     | 0014000000
↪    | test               | t                | main                        |
+--------------+----------------------+---------------+----------------+-------------
↪----+--------------------+------------------+-----------------------------+
```

### Engine Status

Engine status provides details about the inner workings of *TokuDB* and can be useful in tuning your particular environment. The engine status can be determined by running the following command: `SHOW ENGINE tokudb STATUS;`

The following is a reference of the table status statements:

| Table Status | Description |
|---|---|
| disk free space | This is a gross estimate of how much of your file system is available. Possible displays in this field are:<br>• More than twice the reserve ("more than 10 percent of total file system space")<br>• Less than twice the reserve<br>• Less than the reserve<br>• File system is completely full |
| time of environment creation | This is the time when the *TokuDB* storage engine was first started up. Normally, this is when `mysqld` was initially installed with *TokuDB*. If the environment was upgraded from *TokuDB* 4.x (4.2.0 or later), then this will be displayed as "Dec 31, 1969" on Linux hosts. |
| time of engine startup | This is the time when the *TokuDB* storage engine started up. Normally, this is when `mysqld` started. |
| time now | Current date/time on server. |
| db opens | This is the number of times an individual PerconaFT dictionary file was opened. This is a not a useful value for a regular user to use for any purpose due to layers of open/close caching on top. |
| db closes | This is the number of times an individual PerconaFT dictionary file was closed. This is a not a useful value for a regular user to use for any purpose due to layers of open/close caching on top. |
| num open dbs now | This is the number of currently open databases. |
| max open dbs | This is the maximum number of concurrently opened databases. |
| period, in ms, that recovery log is automatically fsynced | `fsync()` frequency in milliseconds. |
| dictionary inserts | This is the total number of rows that have been inserted into all primary and secondary indexes combined, when those inserts have been done with a separate recovery log entry per index. For example, inserting a row into a table with one primary and two secondary indexes will increase this count by three, if the inserts were done with separate recovery log entries. |
| dictionary inserts fail | This is the number of single-index insert operations that failed. |
| dictionary deletes | This is the total number of rows that have been deleted from all primary and secondary indexes combined, if those deletes have been done with a separate recovery log entry per index. |
| dictionary deletes fail | This is the number of single-index delete operations that failed. |

Continued on next page

Table 64.2 – continued from previous page

| Table Status | Description |
|---|---|
| dictionary updates | This is the total number of rows that have been updated in all primary and secondary indexes combined, if those updates have been done with a separate recovery log entry per index. |
| dictionary updates fail | This is the number of single-index update operations that failed. |
| dictionary broadcast updates‘‘: | This is the number of broadcast updates that have been successfully performed. A broadcast update is an update that affects all rows in a dictionary. |
| dictionary broadcast updates fail | This is the number of broadcast updates that have failed. |
| dictionary multi inserts | This is the total number of rows that have been inserted into all primary and secondary indexes combined, when those inserts have been done with a single recovery log entry for the entire row. (For example, inserting a row into a table with one primary and two secondary indexes will normally increase this count by three). |
| dictionary multi inserts fail | This is the number of multi-index insert operations that failed. |
| dictionary multi deletes | This is the total number of rows that have been deleted from all primary and secondary indexes combined, when those deletes have been done with a single recovery log entry for the entire row. |
| dictionary multi deletes fail | This is the number of multi-index delete operations that failed. |
| dictionary updates multi | This is the total number of rows that have been updated in all primary and secondary indexes combined, if those updates have been done with a single recovery log entry for the entire row. |
| dictionary updates fail multi | This is the number of multi-index update operations that failed. |
| le: max committed xr | This is the maximum number of committed transaction records that were stored on disk in a new or modified row. |
| le: max provisional xr | This is the maximum number of provisional transaction records that were stored on disk in a new or modified row. |
| le: expanded | This is the number of times that an expanded memory mechanism was used to store a new or modified row on disk. |
| le: max memsize | This is the maximum number of bytes that were stored on disk as a new or modified row. This is the maximum uncompressed size of any row stored in *TokuDB* that was created or modified since the server started. |
| le: size of leafentries before garbage collection (during message application) | Total number of bytes of leaf nodes data before performing garbage collection for non-flush events. |
| le: size of leafentries after garbage collection (during message application) | Total number of bytes of leaf nodes data after performing garbage collection for non-flush events. |
| le: size of leafentries before garbage collection (outside message application) | Total number of bytes of leaf nodes data before performing garbage collection for flush events. |

Continued on next page

Table 64.2 – continued from previous page

| Table Status | Description |
|---|---|
| le: size of leafentries after garbage collection (outside message application) | Total number of bytes of leaf nodes data after performing garbage collection for flush events. |
| checkpoint: period | This is the interval in seconds between the end of an automatic checkpoint and the beginning of the next automatic checkpoint. |
| checkpoint: footprint | Where the database is in the checkpoint process. |
| checkpoint: last checkpoint began | This is the time the last checkpoint began. If a checkpoint is currently in progress, then this time may be later than the time the last checkpoint completed.<br>**Note:** If no checkpoint has ever taken place, then this value will be `Dec 31, 1969` on Linux hosts. |
| checkpoint: last complete checkpoint began | This is the time the last complete checkpoint started. Any data that changed after this time will not be captured in the checkpoint. |
| checkpoint: last complete checkpoint ended | This is the time the last complete checkpoint ended. |
| checkpoint: time spent during checkpoint (begin and end phases) | Time (in seconds) required to complete all checkpoints. |
| checkpoint: time spent during last checkpoint (begin and end phases) | Time (in seconds) required to complete the last checkpoint. |
| checkpoint: last complete checkpoint LSN | This is the Log Sequence Number of the last complete checkpoint. |
| checkpoint: checkpoints taken | This is the number of complete checkpoints that have been taken. |
| checkpoint: checkpoints failed | This is the number of checkpoints that have failed for any reason. |
| checkpoint: waiters now | This is the current number of threads simultaneously waiting for the checkpoint-safe lock to perform a checkpoint. |
| checkpoint: waiters max | This is the maximum number of threads ever simultaneously waiting for the checkpoint-safe lock to perform a checkpoint. |
| checkpoint: non-checkpoint client wait on mo lock | The number of times a non-checkpoint client thread waited for the multi-operation lock. |
| checkpoint: non-checkpoint client wait on cs lock | The number of times a non-checkpoint client thread waited for the checkpoint-safe lock. |
| checkpoint: checkpoint begin time | Cumulative time (in microseconds) required to mark all dirty nodes as pending a checkpoint. |
| checkpoint: long checkpoint begin time | The total time, in microseconds, of long checkpoint begins. A long checkpoint begin is one taking more than 1 second. |
| checkpoint: long checkpoint begin count | The total number of times a checkpoint begin took more than 1 second. |
| checkpoint: checkpoint end time | The time spent in checkpoint end operation in seconds. |
| checkpoint: long checkpoint end time | The time spent in checkpoint end operation in seconds. |
| checkpoint: long checkpoint end count | This is the count of end_checkpoint operations that exceeded 1 minute. |

Continued on next page

Table 64.2 – continued from previous page

| Table Status | Description |
|---|---|
| cachetable: miss | This is a count of how many times the application was unable to access your data in the internal cache. |
| cachetable: miss time | This is the total time, in microseconds, of how long the database has had to wait for a disk read to complete. |
| cachetable: prefetches | This is the total number of times that a block of memory has been prefetched into the database's cache. Data is prefetched when the database's algorithms determine that a block of memory is likely to be accessed by the application. |
| cachetable: size current | This shows how much of the uncompressed data, in bytes, is currently in the database's internal cache. |
| cachetable: size limit | This shows how much of the uncompressed data, in bytes, will fit in the database's internal cache. |
| cachetable: size writing | This is the number of bytes that are currently queued up to be written to disk. |
| cachetable: size nonleaf | This shows the amount of memory, in bytes, the current set of non-leaf nodes occupy in the cache. |
| cachetable: size leaf | This shows the amount of memory, in bytes, the current set of (decompressed) leaf nodes occupy in the cache. |
| cachetable: size rollback | This shows the rollback nodes size, in bytes, in the cache. |
| cachetable: size cachepressure | This shows the number of bytes causing cache pressure (the sum of buffers and work done counters), helps to understand if cleaner threads are keeping up with workload. It should really be looked at as more of a value to use in a ratio of cache pressure / cache table size. The closer that ratio evaluates to 1, the higher the cache pressure. |
| cachetable: size currently cloned data for checkpoint | Amount of memory, in bytes, currently used for cloned nodes. During the checkpoint operation, dirty nodes are cloned prior to serialization/compression, then written to disk. After which, the memory for the cloned block is returned for re-use. |
| cachetable: evictions | Number of blocks evicted from cache. |
| cachetable: cleaner executions | Total number of times the cleaner thread loop has executed. |
| cachetable: cleaner period | *TokuDB* includes a cleaner thread that optimizes indexes in the background. This variable is the time, in seconds, between the completion of a group of cleaner operations and the beginning of the next group of cleaner operations. The cleaner operations run on a background thread performing work that does not need to be done on the client thread. |
| cachetable: cleaner iterations | This is the number of cleaner operations that are performed every cleaner period. |
| cachetable: number of waits on cache pressure | The number of times a thread was stalled due to cache pressure. |
| cachetable: time waiting on cache pressure | Total time, in microseconds, waiting on cache pressure to subside. |

Continued on next page

Table 64.2 – continued from previous page

| Table Status | Description |
|---|---|
| cachetable: number of long waits on cache pressure | The number of times a thread was stalled for more than 1 second due to cache pressure. |
| cachetable: long time waiting on cache pressure | Total time, in microseconds, waiting on cache pressure to subside for more than 1 second. |
| cachetable: client pool: number of threads in pool | The number of threads in the client thread pool. |
| cachetable: client pool: number of currently active threads in pool | The number of currently active threads in the client thread pool. |
| cachetable: client pool: number of currently queued work items | The number of currently queued work items in the client thread pool. |
| cachetable: client pool: largest number of queued work items | The largest number of queued work items in the client thread pool. |
| cachetable: client pool: total number of work items processed | The total number of work items processed in the client thread pool. |
| cachetable: client pool: total execution time of processing work items | The total execution time of processing work items in the client thread pool. |
| cachetable: cachetable pool: number of threads in pool | The number of threads in the cachetable thread pool. |
| cachetable: cachetable pool: number of currently active threads in pool | The number of currently active threads in the cachetable thread pool. |
| cachetable: cachetable pool: number of currently queued work items‘‘: | The number of currently queued work items in the cachetable thread pool. |
| cachetable: cachetable pool: largest number of queued work items‘‘: | The largest number of queued work items in the cachetable thread pool. |
| cachetable: cachetable pool: total number of work items processed‘‘: | The total number of work items processed in the cachetable thread pool. |
| cachetable: cachetable pool: total execution time of processing work items‘‘: | The total execution time of processing work items in the cachetable thread pool. |
| cachetable: checkpoint pool: number of threads in pool‘‘: | The number of threads in the checkpoint thread pool. |
| cachetable: checkpoint pool: number of currently active threads in pool | The number of currently active threads in the checkpoint thread pool. |
| cachetable: checkpoint pool: number of currently queued work items‘‘: | The number of currently queued work items in the checkpoint thread pool. |
| cachetable: checkpoint pool: largest number of queued work items‘‘: | The largest number of queued work items in the checkpoint thread pool. |
| cachetable: checkpoint pool: total number of work items processed‘‘: | The total number of work items processed in the checkpoint thread pool. |
| cachetable: checkpoint pool: total execution time of processing work items‘‘: | The total execution time of processing work items in the checkpoint thread pool. |
| locktree: memory size | The amount of memory, in bytes, that the locktree is currently using. |
| locktree: memory size limit | The maximum amount of memory, in bytes, that the locktree is allowed to use. |
| locktree: number of times lock escalation ran | Number of times the locktree needed to run lock escalation to reduce its memory footprint. |
| locktree: time spent running escalation (seconds) | Total number of seconds spent performing locktree escalation. |
| locktree: latest post-escalation memory size | Size of the locktree, in bytes, after most current locktree escalation. |
| locktree: number of locktrees open now | Number of locktrees currently open. |
| locktree: number of pending lock requests | Number of requests waiting for a lock grant. |

Continued on next page

Table 64.2 – continued from previous page

| Table Status | Description |
|---|---|
| locktree: number of locktrees eligible for the STO | Number of locktrees eligible for "Single Transaction Optimizations". `STO` optimization are behaviors that can happen within the locktree when there is exactly one transaction active within the locktree. This is a not a useful value for a regular user to use for any purpose. |
| locktree: number of times a locktree ended the STO early | Total number of times a "single transaction optimization" was ended early due to another trans- action starting. |
| locktree: time spent ending the STO early (seconds) | Total number of seconds ending "Single Transaction Optimizations". `STO` optimization are behaviors that can happen within the locktree when there is exactly one transaction active within the locktree. This is a not a useful value for a regular user to use for any purpose. |
| locktree: number of wait locks | Number of times that a lock request could not be acquired because of a conflict with some other transaction. |
| locktree: time waiting for locks | Total time, in microseconds, spend by some client waiting for a lock conflict to be resolved. |
| locktree: number of long wait locks | Number of lock waits greater than 1 second in duration. |
| locktree: long time waiting for locks | Total time, in microseconds, of the long waits. |
| locktree: number of lock timeouts | Count of the number of times that a lock request timed out. |
| locktree: number of waits on lock escalation | When the sum of the sizes of locks taken reaches the lock tree limit, we run lock escalation on a background thread. The clients threads need to wait for escalation to consolidate locks and free up memory. This counter counts the number of times a client thread has to wait on lock escalation. |
| locktree: time waiting on lock escalation | Total time, in microseconds, that a client thread spent waiting for lock escalation to free up memory. |
| locktree: number of long waits on lock escalation | Number of times that a client thread had to wait on lock escalation and the wait time was greater than 1 second. |
| locktree: long time waiting on lock escalation | Total time, in microseconds, of the long waits for lock escalation to free up memory. |
| ft: dictionary updates | This is the total number of rows that have been updated in all primary and secondary indexes combined, if those updates have been done with a separate recovery log entry per index. |
| ft: dictionary broadcast updates | This is the number of broadcast updates that have been successfully performed. A broadcast update is an update that affects all rows in a dictionary. |
| ft: descriptor set | This is the number of time a descriptor was updated when the entire dictionary was updated (for example, when the schema has been changed). |
| ft: messages ignored by leaf due to msn | The number of messages that were ignored by a leaf because it had already been applied. |
| ft: total search retries due to TRY AGAIN‘‘ | Total number of search retries due to TRY AGAIN. Internal value that is no use to anyone other than a developer debugging a specific query/search issue. |
| ft: searches requiring more tries than the height of the tree | Number of searches that required more tries than the height of the tree. |

Continued on next page

Table 64.2 – continued from previous page

| Table Status | Description |
|---|---|
| ft: searches requiring more tries than the height of the tree plus three‘‘ | Number of searches that required more tries than the height of the tree plus three. |
| ft: leaf nodes flushed to disk (not for checkpoint) | Number of leaf nodes flushed to disk, not for checkpoint. |
| ft: leaf nodes flushed to disk (not for checkpoint) (bytes) | Number of bytes of leaf nodes flushed to disk, not for checkpoint. |
| ft: leaf nodes flushed to disk (not for checkpoint) (uncompressed bytes) | Number of bytes of leaf nodes flushed to disk, not for checkpoint. |
| ft: leaf nodes flushed to disk (not for checkpoint) (seconds) | Number of seconds waiting for IO when writing leaf nodes flushed to disk, not for checkpoint. |
| ft: nonleaf nodes flushed to disk (not for checkpoint) | Number of non-leaf nodes flushed to disk, not for checkpoint. |
| ft: nonleaf nodes flushed to disk (not for checkpoint) (bytes) | Number of bytes of non-leaf nodes flushed to disk, not for checkpoint. |
| ft: nonleaf nodes flushed to disk (not for checkpoint) (uncompressed bytes) | Number of uncompressed bytes of non-leaf nodes flushed to disk, not for checkpoint. |
| ft: nonleaf nodes flushed to disk (not for checkpoint) (seconds) | Number of seconds waiting for I/O when writing non-leaf nodes flushed to disk, not for checkpoint. |
| ft: leaf nodes flushed to disk (for checkpoint) | Number of leaf nodes flushed to disk for checkpoint. |
| ft: leaf nodes flushed to disk (for checkpoint) (bytes) | Number of bytes of leaf nodes flushed to disk for checkpoint. |
| ft: leaf nodes flushed to disk (for checkpoint) (uncompressed bytes) | Number of uncompressed bytes of leaf nodes flushed to disk for checkpoint. |
| ft: leaf nodes flushed to disk (for checkpoint) (seconds)‘‘ | Number of seconds waiting for IO when writing leaf nodes flushed to disk for checkpoint. |
| ft: nonleaf nodes flushed to disk (for checkpoint) | Number of non-leaf nodes flushed to disk for checkpoint. |
| ft: nonleaf nodes flushed to disk (for checkpoint) (bytes) | Number of bytes of non-leaf nodes flushed to disk for checkpoint. |
| ft: nonleaf nodes flushed to disk (for checkpoint) (uncompressed bytes) | Number of uncompressed bytes of non-leaf nodes flushed to disk for checkpoint. |
| ft: nonleaf nodes flushed to disk (for checkpoint) (seconds) | Number of seconds waiting for IO when writing non-leaf nodes flushed to disk for checkpoint. |
| ft: uncompressed / compressed bytes written (leaf) | Ratio of uncompressed bytes (in-memory) to compressed bytes (on-disk) for leaf nodes. |
| ft: uncompressed / compressed bytes written (nonleaf) | Ratio of uncompressed bytes (in-memory) to compressed bytes (on-disk) for non-leaf nodes. |
| ft: uncompressed / compressed bytes written (overall) | Ratio of uncompressed bytes (in-memory) to compressed bytes (on-disk) for all nodes. |
| ft: nonleaf node partial evictions | The number of times a partition of a non-leaf node was evicted from the cache. |
| ft: nonleaf node partial evictions (bytes) | The number of bytes freed by evicting partitions of non-leaf nodes from the cache. |
| ft: leaf node partial evictions | The number of times a partition of a leaf node was evicted from the cache. |
| ft: leaf node partial evictions (bytes) | The number of bytes freed by evicting partitions of leaf nodes from the cache. |
| ft: leaf node full evictions‘‘ | The number of times a full leaf node was evicted from the cache. |

Continued on next page

Table 64.2 – continued from previous page

| Table Status | Description |
|---|---|
| ft: leaf node full evictions (bytes) | The number of bytes freed by evicting full leaf nodes from the cache. |
| ft: nonleaf node full evictions (bytes) | The number of bytes freed by evicting full non-leaf nodes from the cache. |
| ft: nonleaf node full evictions | The number of times a full non-leaf node was evicted from the cache. |
| ft: leaf nodes created | Number of created leaf nodes . |
| ft: nonleaf nodes created | Number of created non-leaf nodes. |
| ft: leaf nodes destroyed | Number of destroyed leaf nodes. |
| ft: nonleaf nodes destroyed | Number of destroyed non-leaf nodes. |
| ft: bytes of messages injected at root (all trees) | Amount of messages, in bytes, injected at root (for all trees). |
| ft: bytes of messages flushed from h1 nodes to leaves‘‘ | Amount of messages, in bytes, flushed from `h1` nodes to leaves. |
| ft: bytes of messages currently in trees (estimate) | Amount of messages, in bytes, currently in trees (estimate). |
| ft: messages injected at root | Number of messages injected at root node of a tree. |
| ft: broadcast messages injected at root | Number of broadcast messages injected at root node of a tree. |
| ft: basements decompressed as a target of a query | Number of basement nodes decompressed for queries. |
| ft: basements decompressed for prelocked range | Number of basement nodes decompressed by queries aggressively. |
| ft: basements decompressed for prefetch | Number of basement nodes decompressed by a prefetch thread. |
| ft: basements decompressed for write | Number of basement nodes decompressed for writes. |
| ft: buffers decompressed as a target of a query | Number of buffers decompressed for queries. |
| ft: buffers decompressed for prelocked range | Number of buffers decompressed by queries aggressively. |
| ft: buffers decompressed for prefetch | Number of buffers decompressed by a prefetch thread. |
| ft: buffers decompressed for write | Number of buffers decompressed for writes. |
| ft: pivots fetched for query | Number of pivot nodes fetched for queries. |
| ft: pivots fetched for query (bytes) | Number of bytes of pivot nodes fetched for queries. |
| ft: pivots fetched for query (seconds) | Number of seconds waiting for I/O when fetching pivot nodes for queries. |
| ft: pivots fetched for prefetch | Number of pivot nodes fetched by a prefetch thread. |
| ft: pivots fetched for prefetch (bytes) | Number of bytes of pivot nodes fetched by a prefetch thread. |
| ft: pivots fetched for prefetch (seconds) | Number seconds waiting for I/O when fetching pivot nodes by a prefetch thread. |
| ft: pivots fetched for write | Number of pivot nodes fetched for writes. |
| ft: pivots fetched for write (bytes) | Number of bytes of pivot nodes fetched for writes. |
| ft: pivots fetched for write (seconds) | Number of seconds waiting for I/O when fetching pivot nodes for writes. |
| ft: basements fetched as a target of a query | Number of basement nodes fetched from disk for queries. |
| ft: basements fetched as a target of a query (bytes) | Number of basement node bytes fetched from disk for queries. |
| ft: basements fetched as a target of a query (seconds) | Number of seconds waiting for IO when fetching basement nodes from disk for queries. |

Continued on next page

Table 64.2 – continued from previous page

| Table Status | Description |
|---|---|
| ft: basements fetched for prelocked range | Number of basement nodes fetched from disk aggressively. |
| ft: basements fetched for prelocked range (bytes) | Number of basement node bytes fetched from disk aggressively. |
| ft: basements fetched for prelocked range (seconds) | Number of seconds waiting for I/O when fetching basement nodes from disk aggressively. |
| ft: basements fetched for prefetch | Number of basement nodes fetched from disk by a prefetch thread. |
| ft: basements fetched for prefetch (bytes) | Number of basement node bytes fetched from disk by a prefetch thread. |
| ft: basements fetched for prefetch (seconds) | Number of seconds waiting for I/O when fetching basement nodes from disk by a prefetch thread. |
| ft: basements fetched for write | Number of basement nodes fetched from disk for writes. |
| ft: basements fetched for write (bytes) | Number of basement node bytes fetched from disk for writes. |
| ft: basements fetched for write (seconds) | Number of seconds waiting for I/O when fetching basement nodes from disk for writes. |
| ft: buffers fetched as a target of a query | Number of buffers fetched from disk for queries. |
| ft: buffers fetched as a target of a query (bytes) | Number of buffer bytes fetched from disk for queries. |
| ft: buffers fetched as a target of a query (seconds) | Number of seconds waiting for I/O when fetching buffers from disk for queries. |
| ft: buffers fetched for prelocked range | Number of buffers fetched from disk aggressively. |
| ft: buffers fetched for prelocked range (bytes) | Number of buffer bytes fetched from disk aggressively. |
| ft: buffers fetched for prelocked range (seconds) | Number of seconds waiting for I/O when fetching buffers from disk aggressively. |
| ft: buffers fetched for prefetch | Number of buffers fetched from disk by a prefetch thread. |
| ft: buffers fetched for prefetch (bytes) | Number of buffer bytes fetched from disk by a prefetch thread. |
| ft: buffers fetched for prefetch (seconds) | Number of seconds waiting for I/O when fetching buffers from disk by a prefetch thread. |
| ft: buffers fetched for write | Number of buffers fetched from disk for writes. |
| ft: buffers fetched for write (bytes) | Number of buffer bytes fetched from disk for writes. |
| ft: buffers fetched for write (seconds) | Number of seconds waiting for I/O when fetching buffers from disk for writes. |
| ft: leaf compression to memory (seconds) | Total time, in seconds, spent compressing leaf nodes. |
| ft: leaf serialization to memory (seconds) | Total time, in seconds, spent serializing leaf nodes. |
| ft: leaf decompression to memory (seconds) | Total time, in seconds, spent decompressing leaf nodes. |
| ft: leaf deserialization to memory (seconds) | Total time, in seconds, spent deserializing leaf nodes. |
| ft: nonleaf compression to memory (seconds) | Total time, in seconds, spent compressing non leaf nodes. |
| ft: nonleaf serialization to memory (seconds) | Total time, in seconds, spent serializing non leaf nodes. |
| ft: nonleaf decompression to memory (seconds) | Total time, in seconds, spent decompressing non leaf nodes. |
| ft: nonleaf deserialization to memory (seconds) | Total time, in seconds, spent deserializing non leaf nodes. |
| ft: promotion: roots split | Number of times the root split during promotion. |
| ft: promotion: leaf roots injected into | Number of times a message stopped at a root with height `0`. |

Continued on next page

Table 64.2 – continued from previous page

| Table Status | Description |
|---|---|
| ft: promotion: h1 roots injected into | Number of times a message stopped at a root with height 1. |
| ft: promotion: injections at depth 0 | Number of times a message stopped at depth 0. |
| ft: promotion: injections at depth 1 | Number of times a message stopped at depth 1. |
| ft: promotion: injections at depth 2 | Number of times a message stopped at depth 2. |
| ft: promotion: injections at depth 3 | Number of times a message stopped at depth 3. |
| ft: promotion: injections lower than depth 3 | Number of times a message was promoted past depth 3. |
| ft: promotion: stopped because of a nonempty buffer | Number of times a message stopped because it reached a nonempty buffer. |
| ft: promotion: stopped at height 1‘‘ | Number of times a message stopped because it had reached height 1. |
| ft: promotion: stopped because the child was locked or not at all in memory | Number of times promotion was stopped because the child node was locked or not at all in memory. This is a not a useful value for a regular user to use for any purpose. |
| ft: promotion: stopped because the child was not fully in memory | Number of times promotion was stopped because the child node was not at all in memory. This is a not a useful value for a normal user to use for any purpose. |
| ft: promotion: stopped anyway, after locking the child | Number of times a message stopped before a child which had been locked. |
| ft: basement nodes deserialized with fixed-keysize | The number of basement nodes deserialized where all keys had the same size, leaving the basement in a format that is optimal for in-memory workloads. |
| ft: basement nodes deserialized with variable-keysize | The number of basement nodes deserialized where all keys did not have the same size, and thus ineligible for an in-memory optimization. |
| ft: promotion: succeeded in using the rightmost leaf shortcut | Rightmost insertions used the rightmost-leaf pin path, meaning that the Fractal Tree index detected and properly optimized rightmost inserts. |
| ft: promotion: tried the rightmost leaf shortcut but failed (out-of-bounds) | Rightmost insertions did not use the rightmost-leaf pin path, due to the insert not actually being into the rightmost leaf node. |
| ft: promotion: tried the rightmost leaf shortcut but failed (child reactive) | Rightmost insertions did not use the rightmost-leaf pin path, due to the leaf being too large (needed to split). |
| ft: cursor skipped deleted leaf entries | Number of leaf entries skipped during search/scan because the result of message application and reconciliation of the leaf entry MVCC stack reveals that the leaf entry is deleted in the current transactions view. It is a good indicator that there might be excessive garbage in a tree if a range scan seems to take too long. |
| ft flusher: total nodes potentially flushed by cleaner thread | Total number of nodes whose buffers are potentially flushed by cleaner thread. |
| ft flusher: height-one nodes flushed by cleaner thread | Number of nodes of height one whose message buffers are flushed by cleaner thread. |
| ft flusher: height-greater-than-one nodes flushed by cleaner thread | Number of nodes of height > 1 whose message buffers are flushed by cleaner thread. |
| ft flusher: nodes cleaned which had empty buffers | Number of nodes that are selected by cleaner, but whose buffers are empty. |
| ft flusher: nodes dirtied by cleaner thread | Number of nodes that are made dirty by the cleaner thread. |

Continued on next page

Table 64.2 – continued from previous page

| Table Status | Description |
|---|---|
| ft flusher: max bytes in a buffer flushed by cleaner thread | Max number of bytes in message buffer flushed by cleaner thread. |
| ft flusher: min bytes in a buffer flushed by cleaner thread | Min number of bytes in message buffer flushed by cleaner thread. |
| ft flusher: total bytes in buffers flushed by cleaner thread | Total number of bytes in message buffers flushed by cleaner thread. |
| ft flusher: max workdone in a buffer flushed by cleaner thread | Max workdone value of any message buffer flushed by cleaner thread. |
| ft flusher: min workdone in a buffer flushed by cleaner thread | Min workdone value of any message buffer flushed by cleaner thread. |
| ft flusher: total workdone in buffers flushed by cleaner thread | Total workdone value of message buffers flushed by cleaner thread. |
| ft flusher: times cleaner thread tries to merge a leaf | The number of times the cleaner thread tries to merge a leaf. |
| ft flusher: cleaner thread leaf merges in progress | The number of cleaner thread leaf merges in progress. |
| ft flusher: cleaner thread leaf merges successful | The number of times the cleaner thread successfully merges a leaf. |
| ft flusher: nodes dirtied by cleaner thread leaf merges | The number of nodes dirtied by the “flush from root” process to merge a leaf node. |
| ft flusher: total number of flushes done by flusher threads or cleaner threads | Total number of flushes done by flusher threads or cleaner threads. |
| ft flusher: number of in memory flushes | Number of in-memory flushes. |
| ft flusher: number of flushes that read something off disk | Number of flushes that had to read a child (or part) off disk. |
| ft flusher: number of flushes that triggered another flush in child | Number of flushes that triggered another flush in the child. |
| ft flusher: number of flushes that triggered 1 cascading flush | Number of flushes that triggered 1 cascading flush. |
| ft flusher: number of flushes that triggered 2 cascading flushes | Number of flushes that triggered 2 cascading flushes. |
| ft flusher: number of flushes that triggered 3 cascading flushes:‘‘ | Number of flushes that triggered 3 cascading flushes. |
| ft flusher: number of flushes that triggered 4 cascading flushes | Number of flushes that triggered 4 cascading flushes. |
| ft flusher: number of flushes that triggered 5 cascading flushes | Number of flushes that triggered 5 cascading flushes. |
| ft flusher: number of flushes that triggered over 5 cascading flushes | Number of flushes that triggered more than 5 cascading flushes. |
| ft flusher: leaf node splits | Number of leaf nodes split. |
| ft flusher: nonleaf node splits | Number of non-leaf nodes split. |
| ft flusher: leaf node merges | Number of times leaf nodes are merged. |
| ft flusher: nonleaf node merges | Number of times non-leaf nodes are merged. |
| ft flusher: leaf node balances | Number of times a leaf node is balanced. |
| hot: operations ever started | This variable shows the number of hot operations started (`OPTIMIZE TABLE`). This is a not a useful value for a regular user to use for any purpose. |
| hot: operations successfully completed | The number of hot operations that have successfully completed (`OPTIMIZE TABLE`). This is a not a useful value for a regular user to use for any purpose. |

Continued on next page

Table 64.2 – continued from previous page

| Table Status | Description |
|---|---|
| hot: operations aborted | The number of hot operations that have been aborted (`OPTIMIZE TABLE`). This is a not a useful value for a regular user to use for any purpose. |
| hot: max number of flushes from root ever required to optimize a tree | The maximum number of flushes from the root ever required to optimize a tree. |
| txn: begin | This is the number of transactions that have been started. |
| txn: begin read only | Number of read only transactions started. |
| txn: successful commits | This is the total number of transactions that have been committed. |
| txn: aborts | This is the total number of transactions that have been aborted. |
| logger: next LSN | This is the next unassigned Log Sequence Number. It will be assigned to the next entry in the recovery log. |
| logger: writes | Number of times the logger has written to disk. |
| logger: writes (bytes) | Number of bytes the logger has written to disk. |
| logger: writes (uncompressed bytes) | Number of uncompressed the logger has written to disk. |
| logger: writes (seconds) | Number of seconds waiting for I/O when writing logs to disk. |
| logger: number of long logger write operations | Number of times a logger write operation required 100ms or more. |
| indexer: number of indexers successfully created | This is the number of times one of our internal objects, a indexer, has been created. |
| indexer: number of calls to toku_indexer_create_indexer() that failed | This is the number of times a indexer was requested but could not be created. |
| indexer: number of calls to indexer->build() succeeded | This is the total number of times that indexes were created using a indexer. |
| indexer: number of calls to indexer->build() failed | This is the total number of times that indexes were unable to be created using a indexer |
| indexer: number of calls to indexer->close() that succeeded | This is the number of indexers that successfully created the requested index(es). |
| indexer: number of calls to indexer->close() that failed | This is the number of indexers that were unable to create the requested index(es). |
| indexer: number of calls to indexer->abort() | This is the number of indexers that were aborted. |
| indexer: number of indexers currently in existence | This is the number of indexers that currently exist. |
| indexer: max number of indexers that ever existed simultaneously | This is the maximum number of indexers that ever existed simultaneously. |
| loader: number of loaders successfully created | This is the number of times one of our internal objects, a loader, has been created. |
| loader: number of calls to toku_loader_create_loader() that failed | This is the number of times a loader was requested but could not be created. |
| loader: number of calls to loader->put() succeeded | This is the total number of rows that were inserted using a loader. |
| loader: number of calls to loader->put() failed | This is the total number of rows that were unable to be inserted using a loader. |
| loader: number of calls to loader->close() that succeeded | This is the number of loaders that successfully created the requested table. |
| loader: number of calls to loader->close() that failed | This is the number of loaders that were unable to create the requested table. |
| loader: number of calls to loader->abort() | This is the number of loaders that were aborted. |
| loader: number of loaders currently in existence | This is the number of loaders that currently exist. |

Continued on next page

Table 64.2 – continued from previous page

| Table Status | Description |
|---|---|
| loader: max number of loaders that ever existed simultaneously | This is the maximum number of loaders that ever existed simultaneously. |
| memory: number of malloc operations | Number of calls to `malloc()`. |
| memory: number of free operations | Number of calls to `free()`. |
| memory: number of realloc operations | Number of calls to `realloc()`. |
| memory: number of malloc operations that failed | Number of failed calls to `malloc()`. |
| memory: number of realloc operations that failed | Number of failed calls to `realloc()`. |
| memory: number of bytes requested | Total number of bytes requested from memory allocator library. |
| memory: number of bytes freed | Total number of bytes allocated from memory allocation library that have been freed (used - freed = bytes in use). |
| memory: largest attempted allocation size | Largest number of bytes in a single successful `malloc()` operation. |
| memory: size of the last failed allocation attempt | Largest number of bytes in a single failed `malloc()` operation. |
| memory: number of bytes used (requested + overhead) | Total number of bytes allocated by memory allocator library. |
| memory: estimated maximum memory footprint | Maximum memory footprint of the storage engine, the max value of (used - freed). |
| memory: mallocator version | Version string from in-use memory allocator. |
| memory: mmap threshold | The threshold for malloc to use mmap. |
| filesystem: ENOSPC redzone state | The state of how much disk space exists with respect to the red zone value. Redzone is space greater than *tokudb_fs_reserve_percent* and less than full disk.<br>Valid values are:<br>**0** Space is available<br>**1** Warning, with 2x of redzone value. Operations are allowed, but engine status prints a warning.<br>**2** In red zone, insert operations are blocked<br>**3** All operations are blocked |
| filesystem: threads currently blocked by full disk | This is the number of threads that are currently blocked because they are attempting to write to a full disk. This is normally zero. If this value is non-zero, then a warning will appear in the "disk free space" field. |
| filesystem: number of operations rejected by enospc prevention (red zone) | This is the number of database inserts that have been rejected because the amount of disk free space was less than the reserve. |
| filesystem: most recent disk full | This is the most recent time when the disk file system was entirely full. If the disk has never been full, then this value will be `Dec 31, 1969` on Linux hosts. |
| filesystem: number of write operations that returned ENOSPC | This is the number of times that an attempt to write to disk failed because the disk was full. If the disk is full, this number will continue increasing until space is available. |
| filesystem: fsync time | This the total time, in microseconds, used to fsync to disk. |

Continued on next page

Table 64.2 – continued from previous page

| Table Status | Description |
|---|---|
| filesystem: fsync count | This is the total number of times the database has flushed the operating system's file buffers to disk. |
| filesystem: long fsync time | This the total time, in microseconds, used to fsync to disk when the operation required more than 1 second. |
| filesystem: long fsync count | This is the total number of times the database has flushed the operating system's file buffers to disk and this operation required more than 1 second. |
| context: tree traversals blocked by a full fetch | Number of times node `rwlock` contention was observed while pinning nodes from root to leaf because of a full fetch. |
| context: tree traversals blocked by a partial fetch | Number of times node `rwlock` contention was observed while pinning nodes from root to leaf because of a partial fetch. |
| context: tree traversals blocked by a full eviction‘‘ | Number of times node `rwlock` contention was observed while pinning nodes from root to leaf because of a full eviction. |
| context: tree traversals blocked by a partial eviction‘‘ | Number of times node `rwlock` contention was observed while pinning nodes from root to leaf because of a partial eviction. |
| context: tree traversals blocked by a message injection | Number of times node `rwlock` contention was observed while pinning nodes from root to leaf because of message injection. |
| context: tree traversals blocked by a message application‘‘ | Number of times node `rwlock` contention was observed while pinning nodes from root to leaf because of message application (applying fresh ancestors messages to a basement node). |
| context: tree traversals blocked by a flush | Number of times node `rwlock` contention was observed while pinning nodes from root to leaf because of a buffer flush from parent to child. |
| context: tree traversals blocked by a the cleaner thread | Number of times node `rwlock` contention was observed while pinning nodes from root to leaf because of a cleaner thread. |
| context: tree traversals blocked by something uninstrumented | Number of times node `rwlock` contention was observed while pinning nodes from root to leaf because of something uninstrumented. |
| context: promotion blocked by a full fetch (should never happen) | Number of times node `rwlock` contention was observed within promotion (pinning nodes from root to the buffer to receive the message) because of a full fetch. |
| context: promotion blocked by a partial fetch (should never happen) | Number of times node `rwlock` contention was observed within promotion (pinning nodes from root to the buffer to receive the message) because of a partial fetch. |
| context: promotion blocked by a full eviction (should never happen) | Number of times node `rwlock` contention was observed within promotion (pinning nodes from root to the buffer to receive the message) because of a full eviction. |
| context: promotion blocked by a partial eviction (should never happen) | Number of times node `rwlock` contention was observed within promotion (pinning nodes from root to the buffer to receive the message) because of a partial eviction. |

Continued on next page

Table 64.2 – continued from previous page

| Table Status | Description |
|---|---|
| context: promotion blocked by a message injection | Number of times node `rwlock` contention was observed within promotion (pinning nodes from root to the buffer to receive the message) because of message injection. |
| context: promotion blocked by a message application | Number of times node `rwlock` contention was observed within promotion (pinning nodes from root to the buffer to receive the message) because of message application (applying fresh ancestors messages to a basement node). |
| context: promotion blocked by a flush | Number of times node `rwlock` contention was observed within promotion (pinning nodes from root to the buffer to receive the message) because of a buffer flush from parent to child. |
| context: promotion blocked by the cleaner thread | Number of times node `rwlock` contention was observed within promotion (pinning nodes from root to the buffer to receive the message) because of a cleaner thread. |
| context: promotion blocked by something uninstrumented | Number of times node `rwlock` contention was observed within promotion (pinning nodes from root to the buffer to receive the message) because of something uninstrumented. |
| context: something uninstrumented blocked by something uninstrumented | Number of times node `rwlock` contention was observed for an uninstrumented process because of something uninstrumented. |
| handlerton: primary key bytes inserted | Total number of bytes inserted into all primary key indexes. |

## Frequently Asked Questions

This section contains frequently asked questions regarding *TokuDB* and related software.

- *Truncate Table vs Delete from Table*
- *Foreign Keys*
- *Dropping Indexes*

## Transactional Operations

**What transactional operations does TokuDB support?**

*TokuDB* supports `BEGIN TRANSACTION`, `END TRANSACTION`, `COMMIT`, `ROLLBACK`, `SAVEPOINT`, and `RELEASE SAVEPOINT`.

## TokuDB and the File System

**How can I determine which files belong to the various tables and indexes in my schemas?**

The `tokudb_file_map` plugin lists all Fractal Tree Indexes and their corresponding data files. The `internal_file_name` is the actual file name (in the data folder).

```
mysql> SELECT * FROM information_schema.tokudb_file_map;

+-------------------------+-----------------------------------------+---------------+--
↪----------+-----------------------+
| dictionary_name         | internal_file_name                      | table_schema  |␣
↪table_name  | table_dictionary_name  |
+-------------------------+-----------------------------------------+---------------+--
↪----------+-----------------------+
| ./test/tmc-key-idx_col2  | ./_test_tmc_key_idx_col2_a_14.tokudb  | test          |␣
↪tmc         | key_idx_col2           |
| ./test/tmc-main          | ./_test_tmc_main_9_14.tokudb          | test          |␣
↪tmc         | main                   |
| ./test/tmc-status        | ./_test_tmc_status_8_14.tokudb        | test          |␣
↪tmc         | status                 |
+-------------------------+-----------------------------------------+---------------+--
↪----------+-----------------------+
```

## Full Disks

**What happens when the disk system fills up?**

The disk system may fill up during bulk load operations, such as `LOAD DATA IN FILE` or `CREATE INDEX`, or during incremental operations like `INSERT`.

In the bulk case, running out of disk space will cause the statement to fail with `ERROR 1030 (HY000): Got error 1 from storage engine`. The temporary space used by the bulk loader will be released. If this happens, you can use a separate physical disk for the temporary files (for more information, see *tokudb_tmp_dir*). If server runs out of free space *TokuDB* will assert the server to prevent data corruption to existing data files.

Otherwise, disk space can run low during non-bulk operations. When available space is below a user-configurable reserve (5% by default) inserts are prevented and transactions that perform inserts are aborted. If the disk becomes completely full then *TokuDB* will freeze until some disk space is made available.

Details about the disk system:

- There is a free-space reserve requirement, which is a user-configurable parameter given as a percentage of the total space in the file system. The default reserve is five percent. This value is available in the global variable *tokudb_fs_reserve_percent*. We recommend that this reserve be at least half the size of your physical memory.

  *TokuDB* polls the file system every five seconds to determine how much free space is available. If the free space dips below the reserve, then further table inserts are prohibited. Any transaction that attempts to insert rows will be aborted. Inserts are re-enabled when twice the reserve is available in the file system (so freeing a small amount of disk storage will not be sufficient to resume inserts). Warning messages are sent to the system error log when free space dips below twice the reserve and again when free space dips below the reserve.

  Even with inserts prohibited it is still possible for the file system to become completely full. For example this can happen because another storage engine or another application consumes disk space.

- If the file system becomes completely full, then *TokuDB* will freeze. It will not crash, but it will not respond to most SQL commands until some disk space is made available. When *TokuDB* is frozen in this state, it will still respond to the following command:

```
SHOW ENGINE TokuDB STATUS;
```

  Make disk space available will allow the storage engine to continue running, but inserts will still be prohibited until twice the reserve is free.

  ---

  **Note:** Engine status displays a field indicating if disk free space is above twice the reserve, below twice the reserve, or below the reserve. It will also display a special warning if the disk is completely full.

  ---

- In order to make space available on this system you can:
  - Add some disk space to the filesystem.
  - Delete some non-TokuDB files manually.
  - If the disk is not completely full, you may be able to reclaim space by aborting any transactions that are very old. Old transactions can consume large volumes of disk space in the recovery log.
  - If the disk is not completely full, you can drop indexes or drop tables from your *TokuDB* databases.
  - Deleting large numbers of rows from an existing table and then closing the table may free some space, but it may not. Deleting rows may simply leave unused space (available for new inserts) inside *TokuDB* data files rather than shrink the files (internal fragmentation).

The fine print:

- The *TokuDB* storage engine can use up to three separate file systems simultaneously, one each for the data, the recovery log, and the error log. All three are monitored, and if any one of the three falls below the relevant threshold then a warning message will be issued and inserts may be prohibited.
- Warning messages to the error log are not repeated unless available disk space has been above the relevant threshold for at least one minute. This prevents excess messages in the error log if the disk free space is fluctuating around the limit.
- Even if there are no other storage engines or other applications running, it is still possible for *TokuDB* to consume more disk space when operations such as row delete and query are performed, or when checkpoints are taken. This can happen because *TokuDB* can write cached information when it is time-efficient rather than when inserts are issued by the application, because operations in addition to insert (such as delete) create log entries, and also because of internal fragmentation of *TokuDB* data files.
- The *tokudb_fs_reserve_percent* variable can not be changed once the system has started. It can only be set in `my.cnf` or on the mysqld command line.

## Backup

**How do I back up a system with TokuDB tables?**

### Taking backups with Percona TokuBackup

*TokuDB* is capable of performing online backups with *Percona TokuBackup*. To perform a backup, execute `backup to '/path/to/backup';`. This will create backup of the server and return when complete. The backup can be used by another server using a copy of the binaries on the source server. You can view the progress of the backup by executing `SHOW PROCESSLIST;`. *TokuBackup* produces a copy of your running *MySQL* server that is consistent at the end time of the backup process. The thread copying files from source to destination can be throttled by setting the `tokudb_backup_throttle` server variable. For more information check *Percona TokuBackup*.

The following conditions apply:

- Currently, *TokuBackup* only supports tables using the *TokuDB* storage engine and the *MyISAM* tables in the `mysql` database.

> **Warning:** You must disable *InnoDB* asynchronous IO if backing up *InnoDB* tables via *TokuBackup* utility. Otherwise you will have inconsistent, unrecoverable backups. The appropriate setting is `innodb_use_native_aio` to `0`.

- Transactional storage engines (*TokuDB* and *InnoDB*) will perform recovery on the backup copy of the database when it is first started.
- Tables using non-transactional storage engines (*MyISAM*) are not locked during the copy and may report issues when starting up the backup. It is best to avoid operations that modify these tables at the end of a hot backup operation (adding/changing users, stored procedures, etc.).
- The database is copied locally to the path specified in `/path/to/backup`. This folder must exist, be writable, be empty, and contain enough space for a full copy of the database.
- *TokuBackup* always makes a backup of the *MySQL* `datadir` and optionally the `tokudb_data_dir`, `tokudb_log_dir`, and the binary log folder. The latter three are only backed up separately if they are not the same as or contained in the *MySQL* `datadir`. None of these three folders can be a parent of the *MySQL* `datadir`.
- A folder is created in the given backup destination for each of the source folders.
- No other directory structures are supported. All *InnoDB*, *MyISAM*, and other storage engine files must be within the *MySQL* `datadir`.
- *TokuBackup* does not follow symbolic links.

### Other options for taking backups

*TokuDB* tables are represented in the file system with dictionary files, log files, and metadata files. A consistent copy of all of these files must be made during a backup. Copying the files while they may be modified by a running *MySQL* may result in an inconsistent copy of the database.

LVM snapshots may be used to get a consistent snapshot of all of the *TokuDB* files. The LVM snapshot may then be backed up at leisure.

The `SELECT INTO OUTFILE` statement or **`mysqldump`** application may also be used to get a logical backup of the database.

### References

The MySQL 5.5 reference manual describes several backup methods and strategies. In addition, we recommend reading the backup and recovery chapter in the following book:

*High Performance MySQL, 3rd Edition*, by Baron Schwartz, Peter Zaitsev, and Vadim Tkachenko, Copyright 2012, O'Reilly Media.

### Cold Backup

When *MySQL* is shut down, a copy of the *MySQL* data directory, the *TokuDB* data directory, and the *TokuDB* log directory can be made. In the simplest configuration, the *TokuDB* files are stored in the *MySQL* data directory with all of other *MySQL* files. One merely has to back up this directory.

### Hot Backup using mylvmbackup

The **`mylvmbackup`** utility, located on Launchpad, works with *TokuDB*. It does all of the magic required to get consistent copies of all of the *MySQL* tables, including *MyISAM* tables, *InnoDB* tables, etc., creates the LVM snapshots, and backs up the snapshots.

### Logical Snapshots

A logical snapshot of the databases uses a SQL statements to retrieve table rows and restore them. When used within a transaction, a consistent snapshot of the database can be taken. This method can be used to export tables from one database server and import them into another server.

The `SELECT INTO OUTFILE` statement is used to take a logical snapshot of a database. The `LOAD DATA INFILE` statement is used to load the table data. Please see the *MySQL* 5.6 reference manual for details.

---

**Note:** Please do not use the :program‘mysqlhotcopy‘ to back up *TokuDB* tables. This script is incompatible with *TokuDB*.

---

## Missing Log Files

**What do I do if I delete my logs files or they are otherwise missing?**

You'll need to recover from a backup. It is essential that the log files be present in order to restart the database.

## Isolation Levels

**What is the default isolation level for TokuDB?**

It is repeatable-read (MVCC).

**How can I change the isolation level?**

*TokuDB* supports repeatable-read, serializable, read-uncommitted and read-committed isolation levels (other levels are not supported). *TokuDB* employs pessimistic locking, and aborts a transaction when a lock conflict is detected.

To guarantee that lock conflicts do not occur, use repeatable-read, read-uncommitted or read- committed isolation level.

## Lock Wait Timeout Exceeded

**Why do my |MySQL| clients get lock timeout errors for my update queries? And what should my application do when it gets these errors?**

Updates can get lock timeouts if some other transaction is holding a lock on the rows being updated for longer than the *TokuDB* lock timeout. You may want to increase the this timeout.

If an update deadlocks, then the transaction should abort and retry.

For more information on diagnosing locking issues, see *Lock Visualization in TokuDB*.

## Row Size

**What is the maximum row size?**

The maximum row size is 32 MiB.

## NFS & CIFS

**Can the data directories reside on a disk that is NFS or CIFS mounted?**

Yes, we do have customers in production with NFS & CIFS volumes today. However, both of these disk types can pose a challenge to performance and data integrity due to their complexity. If you're seeking performance, the switching infrastructure and protocols of a traditional network were not conceptualized for low response times and can be very difficult to troubleshoot. If you're concerned with data integrity, the possible data caching at the NFS level can cause inconsistencies between the logs and data files that may never be detected in the event of a crash. If you are thinking of using a NFS or CIFS mount, we would recommend that you use synchronous mount options, which are available from the NFS mount man page, but these settings may decrease performance. For further discussion please look here.

## Using Other Storage Engines

**Can the MyISAM and InnoDB Storage Engines be used?**

*MyISAM* and *InnoDB* can be used directly in conjunction with *TokuDB*. Please note that you should not overcommit memory between *InnoDB* and *TokuDB*. The total memory assigned to both caches must be less than physical memory.

**Can the Federated Storage Engines be used?**

The Federated Storage Engine can also be used, however it is disabled by default in *MySQL*. It can be enabled by either running mysqld with `--federated` as a command line parameter, or by putting `federated` in the `[mysqld]` section of the `my.cnf` file.

For more information see the *MySQL* 5.6 Reference Manual: FEDERATED Storage Engine.

## Using MySQL Patches with TokuDB

**Can I use MySQL source code patches with TokuDB?**

Yes, but you need to apply Percona patches as well as your patches to *MySQL* to build a binary that works with the Percona Fractal Tree library.

### Truncate Table vs Delete from Table

**Which is faster, TRUNCATE TABLE or DELETE FROM TABLE?**

Please use `TRUNCATE TABLE` whenever possible. A table truncation runs in constant time, whereas a `DELETE FROM TABLE` requires a row-by-row deletion and thus runs in time linear to the table size.

### Foreign Keys

**Does TokuDB enforce foreign key constraints?**

No, *TokuDB* ignores foreign key declarations.

### Dropping Indexes

**Is dropping an index in TokuDB hot?**

No, the table is locked for the amount of time it takes the file system to delete the file associated with the index.

## Removing TokuDB storage engine

In case you want remove the TokuDB storage engine from *Percona Server for MySQL* without causing any errors following is the recommended procedure:

### Change the tables from TokuDB to InnoDB

If you still need the data in the TokuDB tables you'll need to alter the tables to other supported storage engine i.e., *InnoDB*: `ALTER TABLE City ENGINE=InnoDB;`

**Note:** In case you remove the TokuDB storage engine before you've changed your tables to other supported storage engine you won't be able to access that data without re-installing the TokuDB storage engine.

### Removing the plugins

To remove the *TokuDB* storage engine with all installed plugins you can use the **`ps-admin`** script:

```
$ ps-admin --disable-tokudb -uroot -pPassw0rd
```

Script output should look like this:

**Output**

```
Checking if Percona server is running with jemalloc enabled...
>> Percona server is running with jemalloc enabled.

Checking transparent huge pages status on the system...
>> Transparent huge pages are currently disabled on the system.

Checking if thp-setting=never option is already set in config file...
```

```
>> Option thp-setting=never is set in the config file.

Checking TokuDB plugin status...
>> TokuDB plugin is installed.

Removing thp-setting=never option from /etc/mysql/my.cnf
>> Successfuly removed thp-setting=never option from /etc/mysql/my.cnf

Uninstalling TokuDB plugin...
>> Successfuly uninstalled TokuDB plugin.
```

Another option is to manually remove the *TokuDB* storage engine with all installed plugins:

```
UNINSTALL PLUGIN tokudb;
UNINSTALL PLUGIN tokudb_file_map;
UNINSTALL PLUGIN tokudb_fractal_tree_info;
UNINSTALL PLUGIN tokudb_fractal_tree_block_map;
UNINSTALL PLUGIN tokudb_trx;
UNINSTALL PLUGIN tokudb_locks;
UNINSTALL PLUGIN tokudb_lock_waits;
UNINSTALL PLUGIN tokudb_background_job_status;
```

After the engine and the plugins have been uninstalled you can remove the TokuDB package by using the apt/yum commands:

```
[root@centos ~]# yum remove Percona-Server-tokudb-80.x86_64
```

or `apt remove percona-server-tokudb-8.0`

**Note:** Make sure you've removed all the TokuDB specific variables from your configuration file (`my.cnf`) before you restart the server, otherwise server could show errors or warnings and won't be able to start.

## Getting the Most from TokuDB

**Compression** *TokuDB* compresses all data on disk, including indexes. Compression lowers cost by reducing the amount of storage required and frees up disk space for additional indexes to achieve improved query performance. Depending on the compressibility of the data, we have seen compression ratios up to 25x for high compression. Compression can also lead to improved performance since less data needs to be read from and written to disk.

**Fast Insertions and Deletions** TokuDB's Fractal Tree technology enables fast indexed insertions and deletions. Fractal Trees match B-trees in their indexing sweet spot (sequential data) and are up to two orders of magnitude faster for random data with high cardinality.

**Eliminates Slave Lag** *TokuDB* replication slaves can be configured to process the replication stream with virtually no read IO. Uniqueness checking is performed on the *TokuDB* master and can be skipped on all *TokuDB* slaves. Also, row based replication ensures that all before and after row images are captured in the binary logs, so the *TokuDB* slaves can harness the power of Fractal Tree indexes and bypass traditional read-modify-write behavior. This "Read Free Replication" ensures that replication slaves do not fall behind the master and can be used for read scaling, backups, and disaster recovery, without sharding, expensive hardware, or limits on what can be replicated.

**Hot Index Creation** *TokuDB* allows the addition of indexes to an existing table while inserts and queries are being performed on that table. This means that *MySQL* can be run continuously with no blocking of queries or insertions while indexes are added and eliminates the down-time that index changes would otherwise require.

**Hot Column Addition, Deletion, Expansion and Rename** *TokuDB* allows the addition of new columns to an existing table, the deletion of existing columns from an existing table, the expansion of `char`, `varchar`, `varbinary`, and `integer` type columns in an existing table, and the renaming of an existing column while inserts and queries are being performed on that table.

**Online (Hot) Backup** The *TokuDB* can create backups of online database servers without downtime.

**Fast Indexing** In practice, slow indexing often leads users to choose a smaller number of sub-optimal indexes in order to keep up with incoming data rates. These sub-optimal indexes result in disproportionately slower queries, since the difference in speed between a query with an index and the same query when no index is available can be many orders of magnitude. Thus, fast indexing means fast queries.

**Clustering Keys and Other Indexing Improvements** *TokuDB* tables are clustered on the primary key. *TokuDB* also supports clustering secondary keys, providing better performance on a broader range of queries. A clustering key includes (or clusters) all of the columns in a table along with the key. As a result, one can efficiently retrieve any column when doing a range query on a clustering key. Also, with *TokuDB*, an auto-increment column can be used in any index and in any position within an index. Lastly, *TokuDB* indexes can include up to 32 columns.

**Less Aging/Fragmentation** *TokuDB* can run much longer, likely indefinitely, without the need to perform the customary practice of dump/reload or `OPTIMIZE TABLE` to restore database performance. The key is the fundamental difference with which the Fractal Tree stores data on disk. Since, by default, the Fractal Tree will store data in 4MB chunks (pre-compression), as compared to InnoDB's 16KB, *TokuDB* has the ability to avoid "database disorder" up to 250x better than InnoDB.

**Bulk Loader** *TokuDB* uses a parallel loader to create tables and offline indexes. This parallel loader will use multiple cores for fast offline table and index creation.

**Full-Featured Database** *TokuDB* supports fully ACID-compliant transactions, MVCC (Multi-Version Concurrency Control), serialized isolation levels, row-level locking, and XA. *TokuDB* scales with high number of client connections, even for large tables.

**Lock Diagnostics** *TokuDB* provides users with the tools to diagnose locking and deadlock issues. For more information, see *Lock Visualization in TokuDB*.

**Progress Tracking** Running `SHOW PROCESSLIST` when adding indexes provides status on how many rows have been processed. Running `SHOW PROCESSLIST` also shows progress on queries, as well as insertions, deletions and updates. This information is helpful for estimating how long operations will take to complete.

**Fast Recovery** *TokuDB* supports very fast recovery, typically less than a minute.

CHAPTER

SIXTYFIVE

# FAST UPDATES WITH TOKUDB

## Introduction

Update intensive applications can have their throughput limited by the random read capacity of the storage system. The cause of the throughput limit is the read-modify-write algorithm that *MySQL* uses to process update statements (read a row from the storage engine, apply the updates to it, write the new row back to the storage engine).

To address this throughput limit, *TokuDB* provides an experimental fast update feature, which uses a different update algorithm. Update expressions of the SQL statement are encoded into tiny programs that are stored in an update Fractal Tree message. This update message is injected into the root of the Fractal Tree index. Eventually, these update messages reach a leaf node, where the update programs are applied to the row. Since messages are moved between Fractal Tree levels in batches, the cost of reading in the leaf node is amortized over many update messages.

This feature is available for `UPDATE` and `INSERT` statements, and can be turned ON/OFF separately for them with use of two variables. Variable *`tokudb_enable_fast_update`* variable toggles fast updates for the `UPDATE`, and *`tokudb_enable_fast_upsert`* does the same for `INSERT`.

## Limitations

Fast updates are activated instead of normal MySQL read-modify-write updates if the executed expression meets the number of conditions.

- fast updates can be activated for a statement or a mixed replication,
- a primary key must be defined for the involved table,
- both simple and compound primary keys are supported, and `int`, `char` or `varchar` are the allowed data types for them,
- updated fields should have `Integer` or `char` data type,
- fields that are part of any key should be not updated,
- clustering keys are not allowed,
- triggers should be not involved,
- supported update expressions should belong to one of the following types:
  - `x = constant`
  - `x = x + constant`
  - `x = x - constant`
  - `x = if (x=0,0,x-1)`

- `x = x + values`

## Usage Specifics and Examples

Following example creates a table that associates event identifiers with their count:

```
CREATE TABLE t (
    event_id bigint unsigned NOT NULL PRIMARY KEY,
    event_count bigint unsigned NOT NULL
);
```

Many graph applications that map onto relational tables can use duplicate key inserts and updates to maintain the graph. For example, one can update the meta-data associated with a link in the graph using duplicate key insertions. If the affected rows is not used by the application, then the insertion or update can be marked and executed as a fast insertion or a fast update.

### Insertion example

If it is not known if the event identifier (represented by *event_id*) already exists in the table, then `INSERT ... ON DUPLICATE KEY UPDATE ...` statement can insert it if not existing, or increment its *event_count* otherwise. Here is an example with duplicate key insertion statement, where `%id` is some specific *event_id* value:

```
INSERT INTO t VALUES (%id, 1)
  ON DUPLICATE KEY UPDATE event_count=event_count+1;
```

#### Explanation

If the event id's are random, then the throughput of this application would be limited by the random read capacity of the storage system since each `INSERT` statement has to determine if this *event_id* exists in the table.

*TokuDB* replaces the primary key existence check with an insertion of an "upsert" message into the Fractal Tree index. This "upsert" message contains a copy of the row and a program that increments event_count. As the Fractal Tree buffer's get filled, this "upsert" message is flushed down the tree. Eventually, the message reaches a leaf node and gets executed there. If the key exists in the leaf node, then the event_count is incremented. Otherwise, the new row is inserted into the leaf node.

### Update example

If *event_id* is known to exist in the table, then `UPDATE` statement can be used to increment its *event_count* (once again, specific *event_id* value is written here as `%id`):

```
UPDATE t SET event_count=event_count+1
WHERE event_id=%id;
```

#### Explanation

TokuDB generates an "update" message from the `UPDATE` statement and its update expression trees, and inserts this message into the Fractal Tree index. When the message eventually reaches the leaf node, the increment program is extracted from the message and executed.

CHAPTER

SIXTYSIX

# TOKUDB FILES AND FILE TYPES

The *TokuDB* file set consists of many different files that all serve various purposes.

If you have any *TokuDB* data your data directory should look similar to this:

```
root@server:/var/lib/mysql# ls -lah
...
-rw-rw----  1 mysql mysql  76M Oct 13 18:45 ibdata1
...
-rw-rw----  1 mysql mysql  16K Oct 13 15:52 tokudb.directory
-rw-rw----  1 mysql mysql  16K Oct 13 15:52 tokudb.environment
-rw-------  1 mysql mysql    0 Oct 13 15:52 __tokudb_lock_dont_delete_me_data
-rw-------  1 mysql mysql    0 Oct 13 15:52 __tokudb_lock_dont_delete_me_environment
-rw-------  1 mysql mysql    0 Oct 13 15:52 __tokudb_lock_dont_delete_me_logs
-rw-------  1 mysql mysql    0 Oct 13 15:52 __tokudb_lock_dont_delete_me_recovery
-rw-------  1 mysql mysql    0 Oct 13 15:52 __tokudb_lock_dont_delete_me_temp
-rw-rw----  1 mysql mysql  16K Oct 13 15:52 tokudb.rollback
...
```

This document lists the different types of *TokuDB* and *Percona Fractal Tree* files, explains their purpose, shows their location and how to move them around.

## tokudb.environment

This file is the root of the *Percona FT* file set and contains various bits of metadata about the system, such as creation times, current file format versions, etc.

*Percona FT* will create/expect this file in the directory specified by the *MySQL* datadir.

## tokudb.rollback

Every transaction within *Percona FT* maintains its own transaction rollback log. These logs are stored together within a single *Percona FT* dictionary file and take up space within the *Percona FT* cachetable (just like any other *Percona FT* dictionary).

The transaction rollback logs will `undo` any changes made by a transaction if the transaction is explicitly rolled back, or rolled back via recovery as a result of an uncommitted transaction when a crash occurs.

*Percona FT* will create/expect this file in the directory specified by the *MySQL* datadir.

## tokudb.directory

*Percona FT* maintains a mapping of a dictionary name (example: `sbtest.sbtest1.main`) to an internal file name (example: `_sbtest_sbtest1_main_xx_x_xx.tokudb`). This mapping is stored within this single *Percona FT* dictionary file and takes up space within the *Percona FT* cachetable just like any other *Percona FT* dictionary.

*Percona FT* will create/expect this file in the directory specified by the *MySQL* datadir.

## Dictionary files

*TokuDB* dictionary (data) files store actual user data. For each *MySQL* table there will be:

- One `status` dictionary that contains metadata about the table.
- One `main` dictionary that stores the full primary key (an imaginary key is used if one was not explicitly specified) and full row data.
- One `key` dictionary for each additional key/index on the table.

These are typically named: `_<database>_<table>_<key>_<internal_txn_id>.tokudb`

*Percona FT* creates/expects these files in the directory specified by *tokudb_data_dir* if set, otherwise the *MySQL* `datadir` is used.

## Recovery log files

The *Percona FT* recovery log records every operation that modifies a *Percona FT* dictionary. Periodically, the system will take a snapshot of the system called a checkpoint. This checkpoint ensures that the modifications recorded within the *Percona FT* recovery logs have been applied to the appropriate dictionary files up to a known point in time and synced to disk.

These files have a rolling naming convention, but use: `log<log_file_number>.tokulog<log_file_format_version>`.

*Percona FT* creates/expects these files in the directory specified by *tokudb_log_dir* if set, otherwise the *MySQL* datadir is used.

*Percona FT* does not track what log files should or shouldn't be present. Upon startup, it discovers the logs in the log directory, and replays them in order. If the wrong logs are present, the recovery aborts and possibly damages the dictionaries.

## Temporary files

*Percona FT* might need to create some temporary files in order to perform some operations. When the bulk loader is active, these temporary files might grow to be quite large.

As different operations start and finish, the files will come and go.

There are no temporary files left behind upon a clean shutdown,

*Percona FT* creates/expects these files in the directory specified by *tokudb_tmp_dir* if set. If not, the *tokudb_data_dir* is used if set, otherwise the *MySQL* datadir is used.

## Lock files

*Percona FT* uses lock files to prevent multiple processes from accessing and writing to the files in the assorted *Percona FT* functionality areas. Each lock file will be in the same directory as the file(s) that it is protecting.

These empty files are only used as semaphores across processes. They are safe to delete/ignore as long as no server instances are currently running and using the data set.

`__tokudb_lock_dont_delete_me_environment`

`__tokudb_lock_dont_delete_me_recovery`

`__tokudb_lock_dont_delete_me_logs`

`__tokudb_lock_dont_delete_me_data`

`__tokudb_lock_dont_delete_me_temp`

*Percona FT* is extremely pedantic about validating its data set. If a file goes missing or unfound, or seems to contain some nonsensical data, it will assert, abort or fail to start. It does this not to annoy you, but to try to protect you from doing any further damage to your data.

CHAPTER

SIXTYSEVEN

# TOKUDB FILE MANAGEMENT

As mentioned in the *TokuDB files and file types* *Percona FT* is extremely pedantic about validating its data set. If a file goes missing or can't be accessed, or seems to contain some nonsensical data, it will assert, abort or fail to start. It does this not to annoy you, but to try to protect you from doing any further damage to your data.

This document contains examples of common file maintenance operations and instructions on how to safely execute these operations.

The `tokudb_dir_per_db` option addressed two shortcomings the *renaming of data files* on table/index rename, and the ability to *group data files together* within a directory that represents a single database. This feature is enabled by default.

The `tokudb_dir_cmd` variable can be used to edit the contents of the TokuDB/PerconaFT directory map.

## Moving TokuDB data files to a location outside of the default MySQL datadir

*TokuDB* uses the location specified by the `tokudb_data_dir` variable for all of its data files. If the `tokudb_data_dir` variable is not explicitly set, *TokuDB* will use the location specified by the servers datadir for these files.

The *TokuDB* data files are protected from concurrent process access by the `__tokudb_lock_dont_delete_me_data` file that is located in the same directory as the *TokuDB* data files.

*TokuDB* data files may be moved to other locations with symlinks left behind in their place. If those symlinks refer to files on other physical data volumes, the `tokudb_fs_reserve_percent` monitor will not traverse the symlink and monitor the real location for adequate space in the file system.

To safely move your TokuDB data files:

1. Shut the server down cleanly.
2. Change the `tokudb_data_dir` in your `my.cnf` configuration file to the location where you wish to store your *TokuDB* data files.
3. Create your new target directory.
4. Move your `*.tokudb` files and your `__tokudb_lock_dont_delete_me_data` from the current location to the new location.
5. Restart your server.

## Moving TokuDB temporary files to a location outside of the default MySQL datadir

*TokuDB* will use the location specified by the `tokudb_tmp_dir` variable for all of its temporary files. If `tokudb_tmp_dir` variable is not explicitly set, *TokuDB* will use the location specified by the `tokudb_data_dir` variable. If the `tokudb_data_dir` variable is also not explicitly set, *TokuDB* will use the location specified by the servers datadir for these files.

*TokuDB* temporary files are protected from concurrent process access by the `__tokudb_lock_dont_delete_me_temp` file that is located in the same directory as the *TokuDB* temporary files.

If you locate your *TokuDB* temporary files on a physical volume that is different from where your *TokuDB* data files or recovery log files are located, the `tokudb_fs_reserve_percent` monitor will not monitor their location for adequate space in the file system.

To safely move your *TokuDB* temporary files:

1. Shut the server down cleanly. A clean shutdown will ensure that there are no temporary files that need to be relocated.
2. Change the `tokudb_tmp_dir` variable in your `my.cnf` configuration file to the location where you wish to store your new *TokuDB* temporary files.
3. Create your new target directory.
4. Move your `__tokudb_lock_dont_delete_me_temp` file from the current location to the new location.
5. Restart your server.

## Moving TokuDB recovery log files to a location outside of the default MySQL datadir

*TokuDB* will use the location specified by the `tokudb_log_dir` variable for all of its recovery log files. If the `tokudb_log_dir` variable is not explicitly set, *TokuDB* will use the location specified by the servers datadir for these files.

The *TokuDB* recovery log files are protected from concurrent process access by the `__tokudb_lock_dont_delete_me_logs` file that is located in the same directory as the *TokuDB* recovery log files.

*TokuDB* recovery log files may be moved to another location with symlinks left behind in place of the `tokudb_log_dir`. If that symlink refers to a directory on another physical data volume, the `tokudb_fs_reserve_percent` monitor will not traverse the symlink and monitor the real location for adequate space in the file system.

To safely move your *TokuDB* recovery log files:

1. Shut the server down cleanly.
2. Change the `tokudb_log_dir` in your `my.cnf` configuration file to the location where you wish to store your *TokuDB* recovery log files.
3. Create your new target directory.
4. Move your `log*.tokulog*` files and your `__tokudb_lock_dont_delete_me_logs` file from the current location to the new location.
5. Restart your server.

## Improved table renaming functionality

When you rename a *TokuDB* table via SQL, the data files on disk keep their original names and only the mapping in the *Percona FT* directory file is changed to map the new dictionary name to the original internal file names. This makes it difficult to quickly match database/table/index names to their actual files on disk, requiring you to use the `INFORMATION_SCHEMA.TOKUDB_FILE_MAP` table to cross reference.

The *tokudb_dir_per_db* variable is implemented to address this issue.

When *tokudb_dir_per_db* is enabled (`ON` by default), this is no longer the case. When you rename a table, the mapping in the *Percona FT* directory file will be updated and the files will be renamed on disk to reflect the new table name.

## Improved directory layout functionality

Many users have had issues with managing the huge volume of individual files that *TokuDB* and *Percona FT* use. The *tokudb_dir_per_db* variable addresses this issue.

When *tokudb_dir_per_db* variable is enabled (`ON` by default), all new tables and indices will be placed within their corresponding database directory within the `tokudb_data_dir` or server datadir.

If you have *tokudb_data_dir* variable set to something other than the server datadir, *TokuDB* will create a directory matching the name of the database, but upon dropping of the database, this directory will remain behind.

Existing table files will not be automatically relocated to their corresponding database directory.

You can easily move a tables data files into the new scheme and proper database directory with a few steps:

```
mysql> SET GLOBAL tokudb_dir_per_db=true;
mysql> RENAME TABLE <table> TO <tmp_table>;
mysql> RENAME TABLE <tmp_table> TO <table>;
```

**Note:** Two renames are needed because *MySQL* doesn't allow you to rename a table to itself. The first rename, renames the table to the temporary name and moves the table files into the owning database directory. The second rename sets the table name back to the original name. Tables can also be renamed/moved across databases and will be placed correctly into the corresponding database directory.

**Warning:** You must be careful with renaming tables in case you have used any tricks to create symlinks of the database directories on different storage volumes, the move is not a simple directory move on the same volume but a physical copy across volumes. This can take quite some time and prevent access to the table being moved during the copy.

## Editing *TokuDB* directory map with `tokudb_dir_cmd`

**Note:** This feature is currently considered *Experimental*.

The *tokudb_dir_cmd* variable can be used to edit the *TokuDB* directory map. **WARNING:** Use this variable only if you know what you're doing otherwise it **WILL** lead to data loss.

This method can be used if any kind of system issue causes the loss of specific `.tokudb` files for a given table, because the *TokuDB* tablespace file mapping will then contain invalid (nonexistent) entries, visible in `INFORMATION_SCHEMA.TokuDB_file_map` table.

This variable is used to send commands to edit directory file. The format of the command line is the following:

```
command arg1 arg2 .. argn
```

I.e, if we want to execute some command the following statement can be used:

```
SET tokudb_dir_cmd = "command arg1 ... argn"
```

Currently the following commands are available:

- `attach dictionary_name internal_file_name` - attach internal_file_name to a dictionary_name, if the dictionary_name exists override the previous value, add new record otherwise
- `detach dictionary_name` - remove record with corresponding dictionary_name, the corresponding internal_file_name file stays untouched
- `move old_dictionary_name new_dictionary_name` - rename (only) dictionary_name from old_dictionary_name to new_dictionary_name

Information about the dictionary_name and internal_file_name can be found in the `TokuDB_file_map` table:

```
mysql> SELECT dictionary_name, internal_file_name FROM INFORMATION_SCHEMA.TokuDB_file_
↪map;
+-----------------------------+-------------------------------------------------------
↪---+
| dictionary_name             | internal_file_name                                    ␣
↪    |
+-----------------------------+-------------------------------------------------------
↪---+
| ./world/City-key-CountryCode | ./_world_sql_340a_39_key_CountryCode_12_1_1d_B_1.
↪tokudb |
| ./world/City-main           | ./_world_sql_340a_39_main_12_1_1d_B_0.tokudb          ␣
↪    |
| ./world/City-status         | ./_world_sql_340a_39_status_f_1_1d.tokudb             ␣
↪    |
+-----------------------------+-------------------------------------------------------
↪---+
```

## System Variables

**variable `tokudb_dir_cmd`**

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dynamic** Yes

**Variable Type** String

This variable is used to send commands to edit *TokuDB* directory map.

**Warning:** Use this variable only if you know what you're doing otherwise it **WILL** lead to data loss.

## Status Variables

**variable `tokudb_dir_cmd_last_error`**

**Variable Type** Numeric

**Scope** Global

This variable contains the error number of the last executed command by using the *`tokudb_dir_cmd`* variable.

**variable `tokudb_dir_cmd_last_error_string`**

**Variable Type** Numeric

**Scope** Global

This variable contains the error string of the last executed command by using the *`tokudb_dir_cmd`* variable.

CHAPTER

SIXTYEIGHT

# TOKUDB BACKGROUND ANALYZE TABLE

*Percona Server for MySQL* has an option to automatically analyze tables in the background based on a measured change in data. This has been done by implementing the background job manager that can perform operations on a background thread.

## Background Jobs

Background jobs and schedule are transient in nature and are not persisted anywhere. Any currently running job will be terminated on shutdown and all scheduled jobs will be forgotten about on server restart. There can't be two jobs on the same table scheduled or running at any one point in time. If you manually invoke an `ANALYZE TABLE` that conflicts with either a pending or running job, the running job will be canceled and the users task will run immediately in the foreground. All the scheduled and running background jobs can be viewed by querying the *`TOKUDB_BACKGROUND_JOB_STATUS`* table.

New *`tokudb_analyze_in_background`* variable has been implemented in order to control if the `ANALYZE TABLE` will be dispatched to the background process or if it will be running in the foreground. To control the function of `ANALYZE TABLE` a new *`tokudb_analyze_mode`* variable has been implemented. This variable offers options to cancel any running or scheduled job on the specified table (`TOKUDB_ANALYZE_CANCEL`), use existing analysis algorithm (`TOKUDB_ANALYZE_STANDARD`), or to recount the logical rows in table and update persistent count (`TOKUDB_ANALYZE_RECOUNT_ROWS`).

`TOKUDB_ANALYZE_RECOUNT_ROWS` is a new mechanism that is used to perform a logical recount of all rows in a table and persist that as the basis value for the table row estimate. This mode was added for tables that have been upgraded from an older version of *TokuDB* that only reported physical row counts and never had a proper logical row count. Newly created tables/partitions will begin counting logical rows correctly from their creation and should not need to be recounted unless some odd edge condition causes the logical count to become inaccurate over time. This analysis mode has no effect on the table cardinality counts. It will take the currently set session values for *`tokudb_analyze_in_background`*, and *`tokudb_analyze_throttle`*. Changing the global or session instances of these values after scheduling will have no effect on the job.

Any background job, both pending and running, can be canceled by setting the *`tokudb_analyze_mode`* to `TOKUDB_ANALYZE_CANCEL` and issuing the `ANALYZE TABLE` on the table for which you want to cancel all the jobs for.

## Auto analysis

To implement the background analysis and gathering of cardinality statistics on a *TokuDB* tables new `delta` value is now maintained in memory for each *TokuDB* table. This value is not persisted anywhere and it is reset to `0` on a server start. It is incremented for each `INSERT/UPDATE/DELETE` command and ignores the impact of transactions (rollback specifically). When this delta value exceeds the *`tokudb_auto_analyze`* percentage of rows in the table

an analysis is performed according to the current session's settings. Other analysis for this table will be disabled until this analysis completes. When this analysis completes, the delta is reset to `0` to begin recalculating table changes for the next potential analysis.

Status values are now reported to server immediately upon completion of any analysis (previously new status values were not used until the table has been closed and re-opened). Half-time direction reversal of analysis has been implemented, meaning that if a *tokudb_analyze_time* is in effect and the analysis has not reached the half way point of the index by the time *tokudb_analyze_time*/2 has been reached: it will stop the forward progress and restart the analysis from the last/rightmost row in the table, progressing leftwards and keeping/adding to the status information accumulated from the first half of the scan.

For small ratios of `table_rows` / *tokudb_auto_analyze*, auto analysis will be run for almost every change. The trigger formula is: `if (table_delta >= ((table_rows * tokudb_auto_analyze) / 100))` then run `ANALYZE TABLE`. If a user manually invokes an `ANALYZE TABLE` and *tokudb_auto_analyze* is enabled and there are no conflicting background jobs, the users `ANALYZE TABLE` will behave exactly as if the delta level has been exceeded in that the analysis is executed and delta reset to `0` upon completion.

## System Variables

**variable `tokudb_analyze_in_background`**

**Command Line** Yes

**Config File** Yes

**Scope** Global/Session

**Dynamic** Yes

**Variable Type** Boolean

**Default Value** `ON`

When this variable is set to `ON` it will dispatch any `ANALYZE TABLE` job to a background process and return immediately, otherwise `ANALYZE TABLE` will run in foreground/client context.

**variable `tokudb_analyze_mode`**

**Command Line** Yes

**Config File** Yes

**Scope** Global/Session

**Dynamic** Yes

**Variable Type** ENUM

**Default Value** `TOKUDB_ANALYZE_STANDARD`

**Range** `TOKUDB_ANALYZE_CANCEL`, `TOKUDB_ANALYZE_STANDARD`, `TOKUDB_ANALYZE_RECOUNT_ROWS`

This variable is used to control the function of `ANALYZE TABLE`. Possible values are:

- `TOKUDB_ANALYZE_CANCEL` - Cancel any running or scheduled job on the specified table.
- `TOKUDB_ANALYZE_STANDARD` - Use existing analysis algorithm. This is the standard table cardinality analysis mode used to obtain cardinality statistics for a tables and its indexes. It will take the currently set session values for *tokudb_analyze_time*, *tokudb_analyze_in_background*, and *tokudb_analyze_throttle* at the time of its scheduling, either via a user invoked `ANALYZE TABLE`

or an auto schedule as a result of *tokudb_auto_analyze* threshold being hit. Changing the global or session instances of these values after scheduling will have no effect on the scheduled job.

- TOKUDB_ANALYZE_RECOUNT_ROWS - Recount logical rows in table and update persistent count. This is a new mechanism that is used to perform a logical recount of all rows in a table and persist that as the basis value for the table row estimate. This mode was added for tables that have been upgraded from an older version of *TokuDB*/PerconaFT that only reported physical row counts and never had a proper logical row count. Newly created tables/partitions will begin counting logical rows correctly from their creation and should not need to be recounted unless some odd edge condition causes the logical count to become inaccurate over time. This analysis mode has no effect on the table cardinality counts. It will take the currently set session values for *tokudb_analyze_in_background*, and *tokudb_analyze_throttle*. Changing the global or session instances of these values after scheduling will have no effect on the job.

**variable tokudb_analyze_throttle**

**Command Line** Yes

**Config File** Yes

**Scope** Global/Session

**Dynamic** Yes

**Variable Type** Numeric

**Default Value** 0

This variable is used to define maximum number of keys to visit per second when performing ANALYZE TABLE with either a TOKUDB_ANALYZE_STANDARD or TOKUDB_ANALYZE_RECOUNT_ROWS.

**variable tokudb_analyze_time**

**Command Line** Yes

**Config File** Yes

**Scope** Global/Session

**Dynamic** Yes

**Variable Type** Numeric

**Default Value** 5

This session variable controls the number of seconds an analyze operation will spend on each index when calculating cardinality. Cardinality is shown by executing the following command:

```
SHOW INDEXES FROM table_name;
```

If an analyze is never performed on a table then the cardinality is 1 for primary key indexes and unique secondary indexes, and NULL (unknown) for all other indexes. Proper cardinality can lead to improved performance of complex SQL statements.

**variable tokudb_auto_analyze**

**Command Line** Yes

**Config File** Yes

**Scope** Global/Session

**Dynamic** Yes

**Variable Type** Numeric

**Default Value** 30

Percentage of table change as INSERT/UPDATE/DELETE commands to trigger an ANALYZE TABLE using the current session *tokudb_analyze_in_background*, *tokudb_analyze_mode*, *tokudb_analyze_throttle*, and *tokudb_analyze_time* settings. If this variable is enabled and *tokudb_analyze_in_background* variable is set to OFF, analysis will be performed directly within the client thread context that triggered the analysis. **NOTE:** *InnoDB* enabled this functionality by default when they introduced it. Due to the potential unexpected new load it might place on a server, it is disabled by default in *TokuDB*.

**variable tokudb_cardinality_scale_percent**

**Command Line** Yes

**Config File** Yes

**Scope** Global

**Dynamic** Yes

**Variable Type** Numeric

**Default Value** 100

**Range** 0-100

Percentage to scale table/index statistics when sending to the server to make an index appear to be either more or less unique than it actually is. *InnoDB* has a hard coded scaling factor of 50%. So if a table of 200 rows had an index with 40 unique values, InnoDB would return 200/40/2 or 2 for the index. The new TokuDB formula is the same but factored differently to use percent, for the same table.index (200/40 * tokudb_cardinality_scale) / 100, for a scale of 50% the result would also be 2 for the index.

## INFORMATION_SCHEMA Tables

**table** INFORMATION_SCHEMA.**TOKUDB_BACKGROUND_JOB_STATUS**

**Columns**

- **id** – Simple monotonically incrementing job id, resets to 0 on server start.
- **database_name** – Database name
- **table_name** – Table name
- **job_type** – Type of job, either TOKUDB_ANALYZE_STANDARD or TOKUDB_ANALYZE_RECOUNT_ROWS
- **job_params** – Param values used by this job in string format. For example: TOKUDB_ANALYZE_DELETE_TIME=1.0; TOKUDB_ANALYZE_TIME=5; TOKUDB_ANALYZE_THROTTLE=2048;
- **scheduler** – Either USER or AUTO to indicate if the job was explicitly scheduled by a user or if it was scheduled as an automatic trigger
- **scheduled_time** – The time the job was scheduled
- **started_time** – The time the job was started
- **status** – Current job status if running. For example: ANALYZE TABLE standard db.tbl.idx 3 of 5 50% rows 10% time scanning forward

This table holds the information on scheduled and running background ANALYZE TABLE jobs for *TokuDB* tables.

CHAPTER

SIXTYNINE

# TOKUDB STATUS VARIABLES

*TokuDB* status variables provide details about the inner workings of *TokuDB* storage engine and they can be useful in tuning the storage engine to a particular environment.

You can view these variables and their values by running:

```
mysql> SHOW STATUS LIKE 'tokudb%';
```

## TokuDB Status Variables Summary

The following global status variables are available:

| Name | Var Type |
|---|---|
| *Tokudb_DB_OPENS* | integer |
| *Tokudb_DB_CLOSES* | integer |
| *Tokudb_DB_OPEN_CURRENT* | integer |
| *Tokudb_DB_OPEN_MAX* | integer |
| *Tokudb_LEAF_ENTRY_MAX_COMMITTED_XR* | integer |
| *Tokudb_LEAF_ENTRY_MAX_PROVISIONAL_XR* | integer |
| *Tokudb_LEAF_ENTRY_EXPANDED* | integer |
| *Tokudb_LEAF_ENTRY_MAX_MEMSIZE* | integer |
| *Tokudb_LEAF_ENTRY_APPLY_GC_BYTES_IN* | integer |
| *Tokudb_LEAF_ENTRY_APPLY_GC_BYTES_OUT* | integer |
| *Tokudb_LEAF_ENTRY_NORMAL_GC_BYTES_IN* | integer |
| *Tokudb_LEAF_ENTRY_NORMAL_GC_BYTES_OUT* | integer |
| *Tokudb_CHECKPOINT_PERIOD* | integer |
| *Tokudb_CHECKPOINT_FOOTPRINT* | integer |
| *Tokudb_CHECKPOINT_LAST_BEGAN* | datetime |
| *Tokudb_CHECKPOINT_LAST_COMPLETE_BEGAN* | datetime |
| *Tokudb_CHECKPOINT_LAST_COMPLETE_ENDED* | datetime |
| *Tokudb_CHECKPOINT_DURATION* | integer |
| *Tokudb_CHECKPOINT_DURATION_LAST* | integer |
| *Tokudb_CHECKPOINT_LAST_LSN* | integer |
| *Tokudb_CHECKPOINT_TAKEN* | integer |
| *Tokudb_CHECKPOINT_FAILED* | integer |
| *Tokudb_CHECKPOINT_WAITERS_NOW* | integer |
| *Tokudb_CHECKPOINT_WAITERS_MAX* | integer |
| *Tokudb_CHECKPOINT_CLIENT_WAIT_ON_MO* | integer |
| *Tokudb_CHECKPOINT_CLIENT_WAIT_ON_CS* | integer |

Continued on next page

Table 69.1 – continued from previous page

| Name | Var Type |
|---|---|
| *Tokudb_CHECKPOINT_BEGIN_TIME* | integer |
| *Tokudb_CHECKPOINT_LONG_BEGIN_TIME* | integer |
| *Tokudb_CHECKPOINT_LONG_BEGIN_COUNT* | integer |
| *Tokudb_CHECKPOINT_END_TIME* | integer |
| *Tokudb_CHECKPOINT_LONG_END_TIME* | integer |
| *Tokudb_CHECKPOINT_LONG_END_COUNT* | integer |
| *Tokudb_CACHETABLE_MISS* | integer |
| *Tokudb_CACHETABLE_MISS_TIME* | integer |
| *Tokudb_CACHETABLE_PREFETCHES* | integer |
| *Tokudb_CACHETABLE_SIZE_CURRENT* | integer |
| *Tokudb_CACHETABLE_SIZE_LIMIT* | integer |
| *Tokudb_CACHETABLE_SIZE_WRITING* | integer |
| *Tokudb_CACHETABLE_SIZE_NONLEAF* | integer |
| *Tokudb_CACHETABLE_SIZE_LEAF* | integer |
| *Tokudb_CACHETABLE_SIZE_ROLLBACK* | integer |
| *Tokudb_CACHETABLE_SIZE_CACHEPRESSURE* | integer |
| *Tokudb_CACHETABLE_SIZE_CLONED* | integer |
| *Tokudb_CACHETABLE_EVICTIONS* | integer |
| *Tokudb_CACHETABLE_CLEANER_EXECUTIONS* | integer |
| *Tokudb_CACHETABLE_CLEANER_PERIOD* | integer |
| *Tokudb_CACHETABLE_CLEANER_ITERATIONS* | integer |
| *Tokudb_CACHETABLE_WAIT_PRESSURE_COUNT* | integer |
| *Tokudb_CACHETABLE_WAIT_PRESSURE_TIME* | integer |
| *Tokudb_CACHETABLE_LONG_WAIT_PRESSURE_COUNT* | integer |
| *Tokudb_CACHETABLE_LONG_WAIT_PRESSURE_TIME* | integer |
| *Tokudb_CACHETABLE_POOL_CLIENT_NUM_THREADS* | integer |
| *Tokudb_CACHETABLE_POOL_CLIENT_NUM_THREADS_ACTIVE* | integer |
| *Tokudb_CACHETABLE_POOL_CLIENT_QUEUE_SIZE* | integer |
| *Tokudb_CACHETABLE_POOL_CLIENT_MAX_QUEUE_SIZE* | integer |
| *Tokudb_CACHETABLE_POOL_CLIENT_TOTAL_ITEMS_PROCESSED* | integer |
| *Tokudb_CACHETABLE_POOL_CLIENT_TOTAL_EXECUTION_TIME* | integer |
| *Tokudb_CACHETABLE_POOL_CACHETABLE_NUM_THREADS* | integer |
| *Tokudb_CACHETABLE_POOL_CACHETABLE_NUM_THREADS_ACTIVE* | integer |
| *Tokudb_CACHETABLE_POOL_CACHETABLE_QUEUE_SIZE* | integer |
| *Tokudb_CACHETABLE_POOL_CACHETABLE_MAX_QUEUE_SIZE* | integer |
| *Tokudb_CACHETABLE_POOL_CACHETABLE_TOTAL_ITEMS_PROCESSED* | integer |
| *Tokudb_CACHETABLE_POOL_CACHETABLE_TOTAL_EXECUTION_TIME* | integer |
| *Tokudb_CACHETABLE_POOL_CHECKPOINT_NUM_THREADS* | integer |
| *Tokudb_CACHETABLE_POOL_CHECKPOINT_NUM_THREADS_ACTIVE* | integer |
| *Tokudb_CACHETABLE_POOL_CHECKPOINT_QUEUE_SIZE* | integer |
| *Tokudb_CACHETABLE_POOL_CHECKPOINT_MAX_QUEUE_SIZE* | integer |
| *Tokudb_CACHETABLE_POOL_CHECKPOINT_TOTAL_ITEMS_PROCESSED* | integer |
| *Tokudb_CACHETABLE_POOL_CHECKPOINT_TOTAL_EXECUTION_TIME* | integer |
| *Tokudb_LOCKTREE_MEMORY_SIZE* | integer |
| *Tokudb_LOCKTREE_MEMORY_SIZE_LIMIT* | integer |
| *Tokudb_LOCKTREE_ESCALATION_NUM* | integer |
| *Tokudb_LOCKTREE_ESCALATION_SECONDS* | numeric |
| *Tokudb_LOCKTREE_LATEST_POST_ESCALATION_MEMORY_SIZE* | integer |
| *Tokudb_LOCKTREE_OPEN_CURRENT* | integer |

Continued on next page

Table 69.1 – continued from previous page

| Name | Var Type |
|---|---|
| *Tokudb_LOCKTREE_PENDING_LOCK_REQUESTS* | integer |
| *Tokudb_LOCKTREE_STO_ELIGIBLE_NUM* | integer |
| *Tokudb_LOCKTREE_STO_ENDED_NUM* | integer |
| *Tokudb_LOCKTREE_STO_ENDED_SECONDS* | numeric |
| *Tokudb_LOCKTREE_WAIT_COUNT* | integer |
| *Tokudb_LOCKTREE_WAIT_TIME* | integer |
| *Tokudb_LOCKTREE_LONG_WAIT_COUNT* | integer |
| *Tokudb_LOCKTREE_LONG_WAIT_TIME* | integer |
| *Tokudb_LOCKTREE_TIMEOUT_COUNT* | integer |
| *Tokudb_LOCKTREE_WAIT_ESCALATION_COUNT* | integer |
| *Tokudb_LOCKTREE_WAIT_ESCALATION_TIME* | integer |
| *Tokudb_LOCKTREE_LONG_WAIT_ESCALATION_COUNT* | integer |
| *Tokudb_LOCKTREE_LONG_WAIT_ESCALATION_TIME* | integer |
| *Tokudb_DICTIONARY_UPDATES* | integer |
| *Tokudb_DICTIONARY_BROADCAST_UPDATES* | integer |
| *Tokudb_DESCRIPTOR_SET* | integer |
| *Tokudb_MESSAGES_IGNORED_BY_LEAF_DUE_TO_MSN* | integer |
| *Tokudb_TOTAL_SEARCH_RETRIES* | integer |
| *Tokudb_SEARCH_TRIES_GT_HEIGHT* | integer |
| *Tokudb_SEARCH_TRIES_GT_HEIGHTPLUS3* | integer |
| *Tokudb_LEAF_NODES_FLUSHED_NOT_CHECKPOINT* | integer |
| *Tokudb_LEAF_NODES_FLUSHED_NOT_CHECKPOINT_BYTES* | integer |
| *Tokudb_LEAF_NODES_FLUSHED_NOT_CHECKPOINT_UNCOMPRESSED_BYTES* | integer |
| *Tokudb_LEAF_NODES_FLUSHED_NOT_CHECKPOINT_SECONDS* | numeric |
| *Tokudb_NONLEAF_NODES_FLUSHED_TO_DISK_NOT_CHECKPOINT* | integer |
| *Tokudb_NONLEAF_NODES_FLUSHED_TO_DISK_NOT_CHECKPOINT_BYTES* | integer |
| *Tokudb_NONLEAF_NODES_FLUSHED_TO_DISK_NOT_CHECKPOINT_UNCOMPRESSE* | integer |
| *Tokudb_NONLEAF_NODES_FLUSHED_TO_DISK_NOT_CHECKPOINT_SECONDS* | numeric |
| *Tokudb_LEAF_NODES_FLUSHED_CHECKPOINT* | integer |
| *Tokudb_LEAF_NODES_FLUSHED_CHECKPOINT_BYTES* | integer |
| *Tokudb_LEAF_NODES_FLUSHED_CHECKPOINT_UNCOMPRESSED_BYTES* | integer |
| *Tokudb_LEAF_NODES_FLUSHED_CHECKPOINT_SECONDS* | numeric |
| *Tokudb_NONLEAF_NODES_FLUSHED_TO_DISK_CHECKPOINT* | integer |
| *Tokudb_NONLEAF_NODES_FLUSHED_TO_DISK_CHECKPOINT_BYTES* | integer |
| *Tokudb_NONLEAF_NODES_FLUSHED_TO_DISK_CHECKPOINT_UNCOMPRESSED_BY* | integer |
| *Tokudb_NONLEAF_NODES_FLUSHED_TO_DISK_CHECKPOINT_SECONDS* | numeric |
| *Tokudb_LEAF_NODE_COMPRESSION_RATIO* | numeric |
| *Tokudb_NONLEAF_NODE_COMPRESSION_RATIO* | numeric |
| *Tokudb_OVERALL_NODE_COMPRESSION_RATIO* | numeric |
| *Tokudb_NONLEAF_NODE_PARTIAL_EVICTIONS* | numeric |
| *Tokudb_NONLEAF_NODE_PARTIAL_EVICTIONS_BYTES* | integer |
| *Tokudb_LEAF_NODE_PARTIAL_EVICTIONS* | integer |
| *Tokudb_LEAF_NODE_PARTIAL_EVICTIONS_BYTES* | integer |
| *Tokudb_LEAF_NODE_FULL_EVICTIONS* | integer |
| *Tokudb_LEAF_NODE_FULL_EVICTIONS_BYTES* | integer |
| *Tokudb_NONLEAF_NODE_FULL_EVICTIONS* | integer |
| *Tokudb_NONLEAF_NODE_FULL_EVICTIONS_BYTES* | integer |
| *Tokudb_LEAF_NODES_CREATED* | integer |
| *Tokudb_NONLEAF_NODES_CREATED* | integer |

Continued on next page

Table 69.1 – continued from previous page

| Name | Var Type |
|---|---|
| *Tokudb_LEAF_NODES_DESTROYED* | integer |
| *Tokudb_NONLEAF_NODES_DESTROYED* | integer |
| *Tokudb_MESSAGES_INJECTED_AT_ROOT_BYTES* | integer |
| *Tokudb_MESSAGES_FLUSHED_FROM_H1_TO_LEAVES_BYTES* | integer |
| *Tokudb_MESSAGES_IN_TREES_ESTIMATE_BYTES* | integer |
| *Tokudb_MESSAGES_INJECTED_AT_ROOT* | integer |
| *Tokudb_BROADCASE_MESSAGES_INJECTED_AT_ROOT* | integer |
| *Tokudb_BASEMENTS_DECOMPRESSED_TARGET_QUERY* | integer |
| *Tokudb_BASEMENTS_DECOMPRESSED_PRELOCKED_RANGE* | integer |
| *Tokudb_BASEMENTS_DECOMPRESSED_PREFETCH* | integer |
| *Tokudb_BASEMENTS_DECOMPRESSED_FOR_WRITE* | integer |
| *Tokudb_BUFFERS_DECOMPRESSED_TARGET_QUERY* | integer |
| *Tokudb_BUFFERS_DECOMPRESSED_PRELOCKED_RANGE* | integer |
| *Tokudb_BUFFERS_DECOMPRESSED_PREFETCH* | integer |
| *Tokudb_BUFFERS_DECOMPRESSED_FOR_WRITE* | integer |
| *Tokudb_PIVOTS_FETCHED_FOR_QUERY* | integer |
| *Tokudb_PIVOTS_FETCHED_FOR_QUERY_BYTES* | integer |
| *Tokudb_PIVOTS_FETCHED_FOR_QUERY_SECONDS* | numeric |
| *Tokudb_PIVOTS_FETCHED_FOR_PREFETCH* | integer |
| *Tokudb_PIVOTS_FETCHED_FOR_PREFETCH_BYTES* | integer |
| *Tokudb_PIVOTS_FETCHED_FOR_PREFETCH_SECONDS* | numeric |
| *Tokudb_PIVOTS_FETCHED_FOR_WRITE* | integer |
| *Tokudb_PIVOTS_FETCHED_FOR_WRITE_BYTES* | integer |
| *Tokudb_PIVOTS_FETCHED_FOR_WRITE_SECONDS* | numeric |
| *Tokudb_BASEMENTS_FETCHED_TARGET_QUERY* | integer |
| *Tokudb_BASEMENTS_FETCHED_TARGET_QUERY_BYTES* | integer |
| *Tokudb_BASEMENTS_FETCHED_TARGET_QUERY_SECONDS* | numeric |
| *Tokudb_BASEMENTS_FETCHED_PRELOCKED_RANGE* | integer |
| *Tokudb_BASEMENTS_FETCHED_PRELOCKED_RANGE_BYTES* | integer |
| *Tokudb_BASEMENTS_FETCHED_PRELOCKED_RANGE_SECONDS* | numeric |
| *Tokudb_BASEMENTS_FETCHED_PREFETCH* | integer |
| *Tokudb_BASEMENTS_FETCHED_PREFETCH_BYTES* | integer |
| *Tokudb_BASEMENTS_FETCHED_PREFETCH_SECONDS* | numeric |
| *Tokudb_BASEMENTS_FETCHED_FOR_WRITE* | integer |
| *Tokudb_BASEMENTS_FETCHED_FOR_WRITE_BYTES* | integer |
| *Tokudb_BASEMENTS_FETCHED_FOR_WRITE_SECONDS* | numeric |
| *Tokudb_BUFFERS_FETCHED_TARGET_QUERY* | integer |
| *Tokudb_BUFFERS_FETCHED_TARGET_QUERY_BYTES* | integer |
| *Tokudb_BUFFERS_FETCHED_TARGET_QUERY_SECONDS* | numeric |
| *Tokudb_BUFFERS_FETCHED_PRELOCKED_RANGE* | integer |
| *Tokudb_BUFFERS_FETCHED_PRELOCKED_RANGE_BYTES* | integer |
| *Tokudb_BUFFERS_FETCHED_PRELOCKED_RANGE_SECONDS* | numeric |
| *Tokudb_BUFFERS_FETCHED_PREFETCH* | integer |
| *Tokudb_BUFFERS_FETCHED_PREFETCH_BYTES* | integer |
| *Tokudb_BUFFERS_FETCHED_PREFETCH_SECONDS* | numeric |
| *Tokudb_BUFFERS_FETCHED_FOR_WRITE* | integer |
| *Tokudb_BUFFERS_FETCHED_FOR_WRITE_BYTES* | integer |
| *Tokudb_BUFFERS_FETCHED_FOR_WRITE_SECONDS* | integer |
| *Tokudb_LEAF_COMPRESSION_TO_MEMORY_SECONDS* | numeric |

Continued on next page

Table 69.1 – continued from previous page

| Name | Var Type |
|---|---|
| *Tokudb_LEAF_SERIALIZATION_TO_MEMORY_SECONDS* | numeric |
| *Tokudb_LEAF_DECOMPRESSION_TO_MEMORY_SECONDS* | numeric |
| *Tokudb_LEAF_DESERIALIZATION_TO_MEMORY_SECONDS* | numeric |
| *Tokudb_NONLEAF_COMPRESSION_TO_MEMORY_SECONDS* | numeric |
| *Tokudb_NONLEAF_SERIALIZATION_TO_MEMORY_SECONDS* | numeric |
| *Tokudb_NONLEAF_DECOMPRESSION_TO_MEMORY_SECONDS* | numeric |
| *Tokudb_NONLEAF_DESERIALIZATION_TO_MEMORY_SECONDS* | numeric |
| *Tokudb_PROMOTION_ROOTS_SPLIT* | integer |
| *Tokudb_PROMOTION_LEAF_ROOTS_INJECTED_INTO* | integer |
| *Tokudb_PROMOTION_H1_ROOTS_INJECTED_INTO* | integer |
| *Tokudb_PROMOTION_INJECTIONS_AT_DEPTH_0* | integer |
| *Tokudb_PROMOTION_INJECTIONS_AT_DEPTH_1* | integer |
| *Tokudb_PROMOTION_INJECTIONS_AT_DEPTH_2* | integer |
| *Tokudb_PROMOTION_INJECTIONS_AT_DEPTH_3* | integer |
| *Tokudb_PROMOTION_INJECTIONS_LOWER_THAN_DEPTH_3* | integer |
| *Tokudb_PROMOTION_STOPPED_NONEMPTY_BUFFER* | integer |
| *Tokudb_PROMOTION_STOPPED_AT_HEIGHT_1* | integer |
| *Tokudb_PROMOTION_STOPPED_CHILD_LOCKED_OR_NOT_IN_MEMORY* | integer |
| *Tokudb_PROMOTION_STOPPED_CHILD_NOT_FULLY_IN_MEMORY* | integer |
| *Tokudb_PROMOTION_STOPPED_AFTER_LOCKING_CHILD* | integer |
| *Tokudb_BASEMENT_DESERIALIZATION_FIXED_KEY* | integer |
| *Tokudb_BASEMENT_DESERIALIZATION_VARIABLE_KEY* | integer |
| *Tokudb_PRO_RIGHTMOST_LEAF_SHORTCUT_SUCCESS* | integer |
| *Tokudb_PRO_RIGHTMOST_LEAF_SHORTCUT_FAIL_POS* | integer |
| *Tokudb_RIGHTMOST_LEAF_SHORTCUT_FAIL_REACTIVE* | integer |
| *Tokudb_CURSOR_SKIP_DELETED_LEAF_ENTRY* | integer |
| *Tokudb_FLUSHER_CLEANER_TOTAL_NODES* | integer |
| *Tokudb_FLUSHER_CLEANER_H1_NODES* | integer |
| *Tokudb_FLUSHER_CLEANER_HGT1_NODES* | integer |
| *Tokudb_FLUSHER_CLEANER_EMPTY_NODES* | integer |
| *Tokudb_FLUSHER_CLEANER_NODES_DIRTIED* | integer |
| *Tokudb_FLUSHER_CLEANER_MAX_BUFFER_SIZE* | integer |
| *Tokudb_FLUSHER_CLEANER_MIN_BUFFER_SIZE* | integer |
| *Tokudb_FLUSHER_CLEANER_TOTAL_BUFFER_SIZE* | integer |
| *Tokudb_FLUSHER_CLEANER_MAX_BUFFER_WORKDONE* | integer |
| *Tokudb_FLUSHER_CLEANER_MIN_BUFFER_WORKDONE* | integer |
| *Tokudb_FLUSHER_CLEANER_TOTAL_BUFFER_WORKDONE* | integer |
| *Tokudb_FLUSHER_CLEANER_NUM_LEAF_MERGES_STARTED* | integer |
| *Tokudb_FLUSHER_CLEANER_NUM_LEAF_MERGES_RUNNING* | integer |
| *Tokudb_FLUSHER_CLEANER_NUM_LEAF_MERGES_COMPLETED* | integer |
| *Tokudb_FLUSHER_CLEANER_NUM_DIRTIED_FOR_LEAF_MERGES* | integer |
| *Tokudb_FLUSHER_FLUSH_TOTAL* | integer |
| *Tokudb_FLUSHER_FLUSH_IN_MEMORY* | integer |
| *Tokudb_FLUSHER_FLUSH_NEEDED_IO* | integer |
| *Tokudb_FLUSHER_FLUSH_CASCADES* | integer |
| *Tokudb_FLUSHER_FLUSH_CASCADES_1* | integer |
| *Tokudb_FLUSHER_FLUSH_CASCADES_2* | integer |
| *Tokudb_FLUSHER_FLUSH_CASCADES_3* | integer |
| *Tokudb_FLUSHER_FLUSH_CASCADES_4* | integer |

Continued on next page

Table 69.1 – continued from previous page

| Name | Var Type |
|---|---|
| *Tokudb_FLUSHER_FLUSH_CASCADES_5* | integer |
| *Tokudb_FLUSHER_FLUSH_CASCADES_GT_5* | integer |
| *Tokudb_FLUSHER_SPLIT_LEAF* | integer |
| *Tokudb_FLUSHER_SPLIT_NONLEAF* | integer |
| *Tokudb_FLUSHER_MERGE_LEAF* | integer |
| *Tokudb_FLUSHER_MERGE_NONLEAF* | integer |
| *Tokudb_FLUSHER_BALANCE_LEAF* | integer |
| *Tokudb_HOT_NUM_STARTED* | integer |
| *Tokudb_HOT_NUM_COMPLETED* | integer |
| *Tokudb_HOT_NUM_ABORTED* | integer |
| *Tokudb_HOT_MAX_ROOT_FLUSH_COUNT* | integer |
| *Tokudb_TXN_BEGIN* | integer |
| *Tokudb_TXN_BEGIN_READ_ONLY* | integer |
| *Tokudb_TXN_COMMITS* | integer |
| *Tokudb_TXN_ABORTS* | integer |
| *Tokudb_LOGGER_NEXT_LSN* | integer |
| *Tokudb_LOGGER_WRITES* | integer |
| *Tokudb_LOGGER_WRITES_BYTES* | integer |
| *Tokudb_LOGGER_WRITES_UNCOMPRESSED_BYTES* | integer |
| *Tokudb_LOGGER_WRITES_SECONDS* | numeric |
| *Tokudb_LOGGER_WAIT_LONG* | integer |
| *Tokudb_LOADER_NUM_CREATED* | integer |
| *Tokudb_LOADER_NUM_CURRENT* | integer |
| *Tokudb_LOADER_NUM_MAX* | integer |
| *Tokudb_MEMORY_MALLOC_COUNT* | integer |
| *Tokudb_MEMORY_FREE_COUNT* | integer |
| *Tokudb_MEMORY_REALLOC_COUNT* | integer |
| *Tokudb_MEMORY_MALLOC_FAIL* | integer |
| *Tokudb_MEMORY_REALLOC_FAIL* | integer |
| *Tokudb_MEMORY_REQUESTED* | integer |
| *Tokudb_MEMORY_USED* | integer |
| *Tokudb_MEMORY_FREED* | integer |
| *Tokudb_MEMORY_MAX_REQUESTED_SIZE* | integer |
| *Tokudb_MEMORY_LAST_FAILED_SIZE* | integer |
| *Tokudb_MEM_ESTIMATED_MAXIMUM_MEMORY_FOOTPRINT* | integer |
| *Tokudb_MEMORY_MALLOCATOR_VERSION* | string |
| *Tokudb_MEMORY_MMAP_THRESHOLD* | integer |
| *Tokudb_FILESYSTEM_THREADS_BLOCKED_BY_FULL_DISK* | integer |
| *Tokudb_FILESYSTEM_FSYNC_TIME* | integer |
| *Tokudb_FILESYSTEM_FSYNC_NUM* | integer |
| *Tokudb_FILESYSTEM_LONG_FSYNC_TIME* | integer |
| *Tokudb_FILESYSTEM_LONG_FSYNC_NUM* | integer |

**variable `Tokudb_DB_OPENS`**

This variable shows the number of times an individual PerconaFT dictionary file was opened. This is a not a useful value for a regular user to use for any purpose due to layers of open/close caching on top.

**variable `Tokudb_DB_CLOSES`**

This variable shows the number of times an individual PerconaFT dictionary file was closed. This is a not a useful value for a regular user to use for any purpose due to layers of open/close caching on top.

**variable Tokudb_DB_OPEN_CURRENT**

This variable shows the number of currently opened databases.

**variable Tokudb_DB_OPEN_MAX**

This variable shows the maximum number of concurrently opened databases.

**variable Tokudb_LEAF_ENTRY_MAX_COMMITTED_XR**

This variable shows the maximum number of committed transaction records that were stored on disk in a new or modified row.

**variable Tokudb_LEAF_ENTRY_MAX_PROVISIONAL_XR**

This variable shows the maximum number of provisional transaction records that were stored on disk in a new or modified row.

**variable Tokudb_LEAF_ENTRY_EXPANDED**

This variable shows the number of times that an expanded memory mechanism was used to store a new or modified row on disk.

**variable Tokudb_LEAF_ENTRY_MAX_MEMSIZE**

This variable shows the maximum number of bytes that were stored on disk as a new or modified row. This is the maximum uncompressed size of any row stored in *TokuDB* that was created or modified since the server started.

**variable Tokudb_LEAF_ENTRY_APPLY_GC_BYTES_IN**

This variable shows the total number of bytes of leaf nodes data before performing garbage collection for non-flush events.

**variable Tokudb_LEAF_ENTRY_APPLY_GC_BYTES_OUT**

This variable shows the total number of bytes of leaf nodes data after performing garbage collection for non-flush events.

**variable Tokudb_LEAF_ENTRY_NORMAL_GC_BYTES_IN**

This variable shows the total number of bytes of leaf nodes data before performing garbage collection for flush events.

**variable Tokudb_LEAF_ENTRY_NORMAL_GC_BYTES_OUT**

This variable shows the total number of bytes of leaf nodes data after performing garbage collection for flush events.

**variable Tokudb_CHECKPOINT_PERIOD**

This variable shows the interval in seconds between the end of an automatic checkpoint and the beginning of the next automatic checkpoint.

**variable Tokudb_CHECKPOINT_FOOTPRINT**

This variable shows at what stage the checkpointer is at. It's used for debugging purposes only and not a useful value for a normal user.

**variable Tokudb_CHECKPOINT_LAST_BEGAN**

This variable shows the time the last checkpoint began. If a checkpoint is currently in progress, then this time may be later than the time the last checkpoint completed. If no checkpoint has ever taken place, then this value will be `Dec 31, 1969` on Linux hosts.

**variable Tokudb_CHECKPOINT_LAST_COMPLETE_BEGAN**

This variable shows the time the last complete checkpoint started. Any data that changed after this time will not be captured in the checkpoint.

**variable Tokudb_CHECKPOINT_LAST_COMPLETE_ENDED**

This variable shows the time the last complete checkpoint ended.

**variable Tokudb_CHECKPOINT_DURATION**

This variable shows time (in seconds) required to complete all checkpoints.

**variable Tokudb_CHECKPOINT_DURATION_LAST**

This variable shows time (in seconds) required to complete the last checkpoint.

**variable Tokudb_CHECKPOINT_LAST_LSN**

This variable shows the last successful checkpoint LSN. Each checkpoint from the time the PerconaFT environment is created has a monotonically incrementing LSN. This is not a useful value for a normal user to use for any purpose other than having some idea of how many checkpoints have occurred since the system was first created.

**variable Tokudb_CHECKPOINT_TAKEN**

This variable shows the number of complete checkpoints that have been taken.

**variable Tokudb_CHECKPOINT_FAILED**

This variable shows the number of checkpoints that have failed for any reason.

**variable Tokudb_CHECKPOINT_WAITERS_NOW**

This variable shows the current number of threads waiting for the `checkpoint safe` lock. This is a not a useful value for a regular user to use for any purpose.

**variable Tokudb_CHECKPOINT_WAITERS_MAX**

This variable shows the maximum number of threads that concurrently waited for the `checkpoint safe` lock. This is a not a useful value for a regular user to use for any purpose.

**variable Tokudb_CHECKPOINT_CLIENT_WAIT_ON_MO**

This variable shows the number of times a non-checkpoint client thread waited for the multi-operation lock. It is an internal `rwlock` that is similar in nature to the *InnoDB* kernel mutex, it effectively halts all access to the PerconaFT API when write locked. The `begin` phase of the checkpoint takes this lock for a brief period.

**variable Tokudb_CHECKPOINT_CLIENT_WAIT_ON_CS**

This variable shows the number of times a non-checkpoint client thread waited for the checkpoint-safe lock. This is the lock taken when you `SET tokudb_checkpoint_lock=1`. If a client trying to lock/postpone the checkpointer has to wait for the currently running checkpoint to complete, that wait time will be reflected here and summed. This is not a useful metric as regular users should never be manipulating the checkpoint lock.

**variable Tokudb_CHECKPOINT_BEGIN_TIME**

This variable shows the cumulative time (in microseconds) required to mark all dirty nodes as pending a checkpoint.

**variable Tokudb_CHECKPOINT_LONG_BEGIN_TIME**

This variable shows the cumulative actual time (in microseconds) of checkpoint `begin` stages that took longer than 1 second.

**variable Tokudb_CHECKPOINT_LONG_BEGIN_COUNT**

This variable shows the number of checkpoints whose `begin` stage took longer than 1 second.

**variable Tokudb_CHECKPOINT_END_TIME**

This variable shows the time spent in checkpoint end operation in seconds.

**variable Tokudb_CHECKPOINT_LONG_END_TIME**

This variable shows the total time of long checkpoints in seconds.

**variable Tokudb_CHECKPOINT_LONG_END_COUNT**

This variable shows the number of checkpoints whose `end_checkpoint` operations exceeded 1 minute.

**variable Tokudb_CACHETABLE_MISS**

This variable shows the number of times the application was unable to access the data in the internal cache. A cache miss means that date will need to be read from disk.

**variable Tokudb_CACHETABLE_MISS_TIME**

This variable shows the total time, in microseconds, of how long the database has had to wait for a disk read to complete.

**variable Tokudb_CACHETABLE_PREFETCHES**

This variable shows the total number of times that a block of memory has been prefetched into the database's cache. Data is prefetched when the database's algorithms determine that a block of memory is likely to be accessed by the application.

**variable Tokudb_CACHETABLE_SIZE_CURRENT**

This variable shows how much of the uncompressed data, in bytes, is currently in the database's internal cache.

**variable Tokudb_CACHETABLE_SIZE_LIMIT**

This variable shows how much of the uncompressed data, in bytes, will fit in the database's internal cache.

**variable Tokudb_CACHETABLE_SIZE_WRITING**

This variable shows the number of bytes that are currently queued up to be written to disk.

**variable Tokudb_CACHETABLE_SIZE_NONLEAF**

This variable shows the amount of memory, in bytes, the current set of non-leaf nodes occupy in the cache.

**variable Tokudb_CACHETABLE_SIZE_LEAF**

This variable shows the amount of memory, in bytes, the current set of (decompressed) leaf nodes occupy in the cache.

**variable Tokudb_CACHETABLE_SIZE_ROLLBACK**

This variable shows the rollback nodes size, in bytes, in the cache.

**variable Tokudb_CACHETABLE_SIZE_CACHEPRESSURE**

This variable shows the number of bytes causing cache pressure (the sum of buffers and work done counters), helps to understand if cleaner threads are keeping up with workload. It should really be looked at as more of a value to use in a ratio of cache pressure / cache table size. The closer that ratio evaluates to 1, the higher the cache pressure.

**variable Tokudb_CACHETABLE_SIZE_CLONED**

This variable shows the amount of memory, in bytes, currently used for cloned nodes. During the checkpoint operation, dirty nodes are cloned prior to serialization/compression, then written to disk. After which, the memory for the cloned block is returned for re-use.

**variable Tokudb_CACHETABLE_EVICTIONS**

This variable shows the number of blocks evicted from cache. On its own this is not a useful number as its impact on performance depends entirely on the hardware and workload in use. For example, two workloads, one random, one linear for the same starting data set will have two wildly different eviction patterns.

**variable Tokudb_CACHETABLE_CLEANER_EXECUTIONS**

This variable shows the total number of times the cleaner thread loop has executed.

**variable Tokudb_CACHETABLE_CLEANER_PERIOD**

*TokuDB* includes a cleaner thread that optimizes indexes in the background. This variable is the time, in seconds, between the completion of a group of cleaner operations and the beginning of the next group of cleaner operations. The cleaner operations run on a background thread performing work that does not need to be done on the client thread.

**variable Tokudb_CACHETABLE_CLEANER_ITERATIONS**

This variable shows the number of cleaner operations that are performed every cleaner period.

**variable Tokudb_CACHETABLE_WAIT_PRESSURE_COUNT**

This variable shows the number of times a thread was stalled due to cache pressure.

**variable Tokudb_CACHETABLE_WAIT_PRESSURE_TIME**

This variable shows the total time, in microseconds, waiting on cache pressure to subside.

**variable Tokudb_CACHETABLE_LONG_WAIT_PRESSURE_COUNT**

This variable shows the number of times a thread was stalled for more than one second due to cache pressure.

**variable Tokudb_CACHETABLE_LONG_WAIT_PRESSURE_TIME**

This variable shows the total time, in microseconds, waiting on cache pressure to subside for more than one second.

**variable Tokudb_CACHETABLE_POOL_CLIENT_NUM_THREADS**

This variable shows the number of threads in the client thread pool.

**variable Tokudb_CACHETABLE_POOL_CLIENT_NUM_THREADS_ACTIVE**

This variable shows the number of currently active threads in the client thread pool.

**variable Tokudb_CACHETABLE_POOL_CLIENT_QUEUE_SIZE**

This variable shows the number of currently queued work items in the client thread pool.

**variable Tokudb_CACHETABLE_POOL_CLIENT_MAX_QUEUE_SIZE**

This variable shows the largest number of queued work items in the client thread pool.

**variable Tokudb_CACHETABLE_POOL_CLIENT_TOTAL_ITEMS_PROCESSED**

This variable shows the total number of work items processed in the client thread pool.

**variable Tokudb_CACHETABLE_POOL_CLIENT_TOTAL_EXECUTION_TIME**

This variable shows the total execution time of processing work items in the client thread pool.

**variable Tokudb_CACHETABLE_POOL_CACHETABLE_NUM_THREADS**

This variable shows the number of threads in the cachetable threadpool.

**variable Tokudb_CACHETABLE_POOL_CACHETABLE_NUM_THREADS_ACTIVE**

This variable shows the number of currently active threads in the cachetable thread pool.

**variable Tokudb_CACHETABLE_POOL_CACHETABLE_QUEUE_SIZE**

This variable shows the number of currently queued work items in the cachetable thread pool.

**variable Tokudb_CACHETABLE_POOL_CACHETABLE_MAX_QUEUE_SIZE**

This variable shows the largest number of queued work items in the cachetable thread pool.

**variable Tokudb_CACHETABLE_POOL_CACHETABLE_TOTAL_ITEMS_PROCESSED**

This variable shows the total number of work items processed in the cachetable thread pool.

**variable Tokudb_CACHETABLE_POOL_CACHETABLE_TOTAL_EXECUTION_TIME**

This variable shows the total execution time of processing work items in the cachetable thread pool.

**variable Tokudb_CACHETABLE_POOL_CHECKPOINT_NUM_THREADS**

This variable shows the number of threads in the checkpoint threadpool.

**variable Tokudb_CACHETABLE_POOL_CHECKPOINT_NUM_THREADS_ACTIVE**

This variable shows the number of currently active threads in the checkpoint thread pool.

**variable Tokudb_CACHETABLE_POOL_CHECKPOINT_QUEUE_SIZE**

This variable shows the number of currently queued work items in the checkpoint thread pool.

**variable Tokudb_CACHETABLE_POOL_CHECKPOINT_MAX_QUEUE_SIZE**

This variable shows the largest number of queued work items in the checkpoint thread pool.

**variable Tokudb_CACHETABLE_POOL_CHECKPOINT_TOTAL_ITEMS_PROCESSED**

This variable shows the total number of work items processed in the checkpoint thread pool.

**variable Tokudb_CACHETABLE_POOL_CHECKPOINT_TOTAL_EXECUTION_TIME**

This variable shows the total execution time of processing work items in the checkpoint thread pool.

**variable Tokudb_LOCKTREE_MEMORY_SIZE**

This variable shows the amount of memory, in bytes, that the locktree is currently using.

**variable Tokudb_LOCKTREE_MEMORY_SIZE_LIMIT**

This variable shows the maximum amount of memory, in bytes, that the locktree is allowed to use.

**variable Tokudb_LOCKTREE_ESCALATION_NUM**

This variable shows the number of times the locktree needed to run lock escalation to reduce its memory footprint.

**variable Tokudb_LOCKTREE_ESCALATION_SECONDS**

This variable shows the total number of seconds spent performing locktree escalation.

**variable Tokudb_LOCKTREE_LATEST_POST_ESCALATION_MEMORY_SIZE**

This variable shows the locktree size, in bytes, after most current locktree escalation.

**variable Tokudb_LOCKTREE_OPEN_CURRENT**

This variable shows the number of locktrees that are currently opened.

**variable Tokudb_LOCKTREE_PENDING_LOCK_REQUESTS**

This variable shows the number of requests waiting for a lock grant.

**variable Tokudb_LOCKTREE_STO_ELIGIBLE_NUM**

This variable shows the number of locktrees eligible for `Single Transaction optimizations`. STO optimization are behaviors that can happen within the locktree when there is exactly one transaction active within the locktree. This is a not a useful value for a regular user to use for any purpose.

**variable Tokudb_LOCKTREE_STO_ENDED_NUM**

This variable shows the total number of times a `Single Transaction Optimization` was ended early due to another transaction starting. STO optimization are behaviors that can happen within the locktree when there is exactly one transaction active within the locktree. This is a not a useful value for a regular user to use for any purpose.

**variable Tokudb_LOCKTREE_STO_ENDED_SECONDS**

This variable shows the total number of seconds ending the `Single Transaction Optimizations`. STO optimization are behaviors that can happen within the locktree when there is exactly one transaction active within the locktree. This is a not a useful value for a regular user to use for any purpose.

**variable `Tokudb_LOCKTREE_WAIT_COUNT`**

This variable shows the number of times that a lock request could not be acquired because of a conflict with some other transaction. PerconaFT lock request cycles to try to obtain a lock, if it can not get a lock, it sleeps/waits and times out, checks to get the lock again, repeat. This value indicates the number of cycles it needed to execute before it obtained the lock.

**variable `Tokudb_LOCKTREE_WAIT_TIME`**

This variable shows the total time, in microseconds, spent by client waiting for a lock conflict to be resolved.

**variable `Tokudb_LOCKTREE_LONG_WAIT_COUNT`**

This variable shows number of lock waits greater than one second in duration.

**variable `Tokudb_LOCKTREE_LONG_WAIT_TIME`**

This variable shows the total time, in microseconds, of the long waits.

**variable `Tokudb_LOCKTREE_TIMEOUT_COUNT`**

This variable shows the number of times that a lock request timed out.

**variable `Tokudb_LOCKTREE_WAIT_ESCALATION_COUNT`**

When the sum of the sizes of locks taken reaches the lock tree limit, we run lock escalation on a background thread. The clients threads need to wait for escalation to consolidate locks and free up memory. This variables shows the number of times a client thread had to wait on lock escalation.

**variable `Tokudb_LOCKTREE_WAIT_ESCALATION_TIME`**

This variable shows the total time, in microseconds, that a client thread spent waiting for lock escalation to free up memory.

**variable `Tokudb_LOCKTREE_LONG_WAIT_ESCALATION_COUNT`**

This variable shows number of times that a client thread had to wait on lock escalation and the wait time was greater than one second.

**variable `Tokudb_LOCKTREE_LONG_WAIT_ESCALATION_TIME`**

This variable shows the total time, in microseconds, of the long waits for lock escalation to free up memory.

**variable `Tokudb_DICTIONARY_UPDATES`**

This variable shows the total number of rows that have been updated in all primary and secondary indexes combined, if those updates have been done with a separate recovery log entry per index.

**variable `Tokudb_DICTIONARY_BROADCAST_UPDATES`**

This variable shows the number of broadcast updates that have been successfully performed. A broadcast update is an update that affects all rows in a dictionary.

**variable `Tokudb_DESCRIPTOR_SET`**

This variable shows the number of time a descriptor was updated when the entire dictionary was updated (for example, when the schema has been changed).

**variable `Tokudb_MESSAGES_IGNORED_BY_LEAF_DUE_TO_MSN`**

This variable shows the number of messages that were ignored by a leaf because it had already been applied.

**variable `Tokudb_TOTAL_SEARCH_RETRIES`**

Internal value that is no use to anyone other than a developer debugging a specific query/search issue.

**variable Tokudb_SEARCH_TRIES_GT_HEIGHT**

Internal value that is no use to anyone other than a developer debugging a specific query/search issue.

**variable Tokudb_SEARCH_TRIES_GT_HEIGHTPLUS3**

Internal value that is no use to anyone other than a developer debugging a specific query/search issue.

**variable Tokudb_LEAF_NODES_FLUSHED_NOT_CHECKPOINT**

This variable shows the number of leaf nodes flushed to disk, not for checkpoint.

**variable Tokudb_LEAF_NODES_FLUSHED_NOT_CHECKPOINT_BYTES**

This variable shows the size, in bytes, of leaf nodes flushed to disk, not for checkpoint.

**variable Tokudb_LEAF_NODES_FLUSHED_NOT_CHECKPOINT_UNCOMPRESSED_BYTES**

This variable shows the size, in bytes, of uncompressed leaf nodes flushed to disk not for checkpoint.

**variable Tokudb_LEAF_NODES_FLUSHED_NOT_CHECKPOINT_SECONDS**

This variable shows the number of seconds waiting for I/O when writing leaf nodes flushed to disk, not for checkpoint

**variable Tokudb_NONLEAF_NODES_FLUSHED_TO_DISK_NOT_CHECKPOINT**

This variable shows the number of non-leaf nodes flushed to disk, not for checkpoint.

**variable Tokudb_NONLEAF_NODES_FLUSHED_TO_DISK_NOT_CHECKPOINT_BYTES**

This variable shows the size, in bytes, of non-leaf nodes flushed to disk, not for checkpoint.

**variable Tokudb_NONLEAF_NODES_FLUSHED_TO_DISK_NOT_CHECKPOINT_UNCOMPRESSE**

This variable shows the size, in bytes, of uncompressed non-leaf nodes flushed to disk not for checkpoint.

**variable Tokudb_NONLEAF_NODES_FLUSHED_TO_DISK_NOT_CHECKPOINT_SECONDS**

This variable shows the number of seconds waiting for I/O when writing non-leaf nodes flushed to disk, not for checkpoint

**variable Tokudb_LEAF_NODES_FLUSHED_CHECKPOINT**

This variable shows the number of leaf nodes flushed to disk, for checkpoint.

**variable Tokudb_LEAF_NODES_FLUSHED_CHECKPOINT_BYTES**

This variable shows the size, in bytes, of leaf nodes flushed to disk, for checkpoint.

**variable Tokudb_LEAF_NODES_FLUSHED_CHECKPOINT_UNCOMPRESSED_BYTES**

This variable shows the size, in bytes, of uncompressed leaf nodes flushed to disk for checkpoint.

**variable Tokudb_LEAF_NODES_FLUSHED_CHECKPOINT_SECONDS**

This variable shows the number of seconds waiting for I/O when writing leaf nodes flushed to disk for checkpoint

**variable Tokudb_NONLEAF_NODES_FLUSHED_TO_DISK_CHECKPOINT**

This variable shows the number of non-leaf nodes flushed to disk, for checkpoint.

**variable Tokudb_NONLEAF_NODES_FLUSHED_TO_DISK_CHECKPOINT_BYTES**

This variable shows the size, in bytes, of non-leaf nodes flushed to disk, for checkpoint.

**variable Tokudb_NONLEAF_NODES_FLUSHED_TO_DISK_CHECKPOINT_UNCOMPRESSED_BY**

This variable shows the size, in bytes, of uncompressed non-leaf nodes flushed to disk for checkpoint.

**variable Tokudb_NONLEAF_NODES_FLUSHED_TO_DISK_CHECKPOINT_SECONDS**

This variable shows the number of seconds waiting for I/O when writing non-leaf nodes flushed to disk for checkpoint

**variable Tokudb_LEAF_NODE_COMPRESSION_RATIO**

This variable shows the ratio of uncompressed bytes (in-memory) to compressed bytes (on-disk) for leaf nodes.

**variable Tokudb_NONLEAF_NODE_COMPRESSION_RATIO**

This variable shows the ratio of uncompressed bytes (in-memory) to compressed bytes (on-disk) for non-leaf nodes.

**variable Tokudb_OVERALL_NODE_COMPRESSION_RATIO**

This variable shows the ratio of uncompressed bytes (in-memory) to compressed bytes (on-disk) for all nodes.

**variable Tokudb_NONLEAF_NODE_PARTIAL_EVICTIONS**

This variable shows the number of times a partition of a non-leaf node was evicted from the cache.

**variable Tokudb_NONLEAF_NODE_PARTIAL_EVICTIONS_BYTES**

This variable shows the amount, in bytes, of memory freed by evicting partitions of non-leaf nodes from the cache.

**variable Tokudb_LEAF_NODE_PARTIAL_EVICTIONS**

This variable shows the number of times a partition of a leaf node was evicted from the cache.

**variable Tokudb_LEAF_NODE_PARTIAL_EVICTIONS_BYTES**

This variable shows the amount, in bytes, of memory freed by evicting partitions of leaf nodes from the cache.

**variable Tokudb_LEAF_NODE_FULL_EVICTIONS**

This variable shows the number of times a full leaf node was evicted from the cache.

**variable Tokudb_LEAF_NODE_FULL_EVICTIONS_BYTES**

This variable shows the amount, in bytes, of memory freed by evicting full leaf nodes from the cache.

**variable Tokudb_NONLEAF_NODE_FULL_EVICTIONS**

This variable shows the number of times a full non-leaf node was evicted from the cache.

**variable Tokudb_NONLEAF_NODE_FULL_EVICTIONS_BYTES**

This variable shows the amount, in bytes, of memory freed by evicting full non-leaf nodes from the cache.

**variable Tokudb_LEAF_NODES_CREATED**

This variable shows the number of created leaf nodes.

**variable Tokudb_NONLEAF_NODES_CREATED**

This variable shows the number of created non-leaf nodes.

**variable Tokudb_LEAF_NODES_DESTROYED**

This variable shows the number of destroyed leaf nodes.

**variable Tokudb_NONLEAF_NODES_DESTROYED**

This variable shows the number of destroyed non-leaf nodes.

**variable Tokudb_MESSAGES_INJECTED_AT_ROOT_BYTES**

This variable shows the size, in bytes, of messages injected at root (for all trees).

**variable Tokudb_MESSAGES_FLUSHED_FROM_H1_TO_LEAVES_BYTES**

This variable shows the size, in bytes, of messages flushed from `h1` nodes to leaves.

**variable Tokudb_MESSAGES_IN_TREES_ESTIMATE_BYTES**

This variable shows the estimated size, in bytes, of messages currently in trees.

**variable Tokudb_MESSAGES_INJECTED_AT_ROOT**

This variables shows the number of messages that were injected at root node of a tree.

**variable Tokudb_BROADCASE_MESSAGES_INJECTED_AT_ROOT**

This variable shows the number of broadcast messages dropped into the root node of a tree. These are things such as the result of `OPTIMIZE TABLE` and a few other operations. This is not a useful metric for a regular user to use for any purpose.

**variable Tokudb_BASEMENTS_DECOMPRESSED_TARGET_QUERY**

This variable shows the number of basement nodes decompressed for queries.

**variable Tokudb_BASEMENTS_DECOMPRESSED_PRELOCKED_RANGE**

This variable shows the number of basement nodes aggressively decompressed by queries.

**variable Tokudb_BASEMENTS_DECOMPRESSED_PREFETCH**

This variable shows the number of basement nodes decompressed by a prefetch thread.

**variable Tokudb_BASEMENTS_DECOMPRESSED_FOR_WRITE**

This variable shows the number of basement nodes decompressed for writes.

**variable Tokudb_BUFFERS_DECOMPRESSED_TARGET_QUERY**

This variable shows the number of buffers decompressed for queries.

**variable Tokudb_BUFFERS_DECOMPRESSED_PRELOCKED_RANGE**

This variable shows the number of buffers decompressed by queries aggressively.

**variable Tokudb_BUFFERS_DECOMPRESSED_PREFETCH**

This variable shows the number of buffers decompressed by a prefetch thread.

**variable Tokudb_BUFFERS_DECOMPRESSED_FOR_WRITE**

This variable shows the number of buffers decompressed for writes.

**variable Tokudb_PIVOTS_FETCHED_FOR_QUERY**

This variable shows the number of pivot nodes fetched for queries.

**variable Tokudb_PIVOTS_FETCHED_FOR_QUERY_BYTES**

This variable shows the number of bytes of pivot nodes fetched for queries.

**variable Tokudb_PIVOTS_FETCHED_FOR_QUERY_SECONDS**

This variable shows the number of seconds waiting for I/O when fetching pivot nodes for queries.

**variable Tokudb_PIVOTS_FETCHED_FOR_PREFETCH**

This variable shows the number of pivot nodes fetched by a prefetch thread.

**variable Tokudb_PIVOTS_FETCHED_FOR_PREFETCH_BYTES**

This variable shows the number of bytes of pivot nodes fetched for queries.

**variable Tokudb_PIVOTS_FETCHED_FOR_PREFETCH_SECONDS**

This variable shows the number seconds waiting for I/O when fetching pivot nodes by a prefetch thread.

**variable Tokudb_PIVOTS_FETCHED_FOR_WRITE**

This variable shows the number of pivot nodes fetched for writes.

**variable Tokudb_PIVOTS_FETCHED_FOR_WRITE_BYTES**

This variable shows the number of bytes of pivot nodes fetched for writes.

**variable Tokudb_PIVOTS_FETCHED_FOR_WRITE_SECONDS**

This variable shows the number of seconds waiting for I/O when fetching pivot nodes for writes.

**variable Tokudb_BASEMENTS_FETCHED_TARGET_QUERY**

This variable shows the number of basement nodes fetched from disk for queries.

**variable Tokudb_BASEMENTS_FETCHED_TARGET_QUERY_BYTES**

This variable shows the number of basement node bytes fetched from disk for queries.

**variable Tokudb_BASEMENTS_FETCHED_TARGET_QUERY_SECONDS**

This variable shows the number of seconds waiting for I/O when fetching basement nodes from disk for queries.

**variable Tokudb_BASEMENTS_FETCHED_PRELOCKED_RANGE**

This variable shows the number of basement nodes fetched from disk aggressively.

**variable Tokudb_BASEMENTS_FETCHED_PRELOCKED_RANGE_BYTES**

This variable shows the number of basement node bytes fetched from disk aggressively.

**variable Tokudb_BASEMENTS_FETCHED_PRELOCKED_RANGE_SECONDS**

This variable shows the number of seconds waiting for I/O when fetching basement nodes from disk aggressively.

**variable Tokudb_BASEMENTS_FETCHED_PREFETCH**

This variable shows the number of basement nodes fetched from disk by a prefetch thread.

**variable Tokudb_BASEMENTS_FETCHED_PREFETCH_BYTES**

This variable shows the number of basement node bytes fetched from disk by a prefetch thread.

**variable Tokudb_BASEMENTS_FETCHED_PREFETCH_SECONDS**

This variable shows the number of seconds waiting for I/O when fetching basement nodes from disk by a prefetch thread.

**variable Tokudb_BASEMENTS_FETCHED_FOR_WRITE**

This variable shows the number of buffers fetched from disk for writes.

**variable Tokudb_BASEMENTS_FETCHED_FOR_WRITE_BYTES**

This variable shows the number of buffer bytes fetched from disk for writes.

**variable Tokudb_BASEMENTS_FETCHED_FOR_WRITE_SECONDS**

This variable shows the number of seconds waiting for I/O when fetching buffers from disk for writes.

**variable Tokudb_BUFFERS_FETCHED_TARGET_QUERY**

This variable shows the number of buffers fetched from disk for queries.

**variable Tokudb_BUFFERS_FETCHED_TARGET_QUERY_BYTES**

This variable shows the number of buffer bytes fetched from disk for queries.

**variable Tokudb_BUFFERS_FETCHED_TARGET_QUERY_SECONDS**

This variable shows the number of seconds waiting for I/O when fetching buffers from disk for queries.

**variable Tokudb_BUFFERS_FETCHED_PRELOCKED_RANGE**

This variable shows the number of buffers fetched from disk aggressively.

**variable Tokudb_BUFFERS_FETCHED_PRELOCKED_RANGE_BYTES**

This variable shows the number of buffer bytes fetched from disk aggressively.

**variable Tokudb_BUFFERS_FETCHED_PRELOCKED_RANGE_SECONDS**

This variable shows the number of seconds waiting for I/O when fetching buffers from disk aggressively.

**variable Tokudb_BUFFERS_FETCHED_PREFETCH**

This variable shows the number of buffers fetched from disk aggressively.

**variable Tokudb_BUFFERS_FETCHED_PREFETCH_BYTES**

This variable shows the number of buffer bytes fetched from disk by a prefetch thread.

**variable Tokudb_BUFFERS_FETCHED_PREFETCH_SECONDS**

This variable shows the number of seconds waiting for I/O when fetching buffers from disk by a prefetch thread.

**variable Tokudb_BUFFERS_FETCHED_FOR_WRITE**

This variable shows the number of buffers fetched from disk for writes.

**variable Tokudb_BUFFERS_FETCHED_FOR_WRITE_BYTES**

This variable shows the number of buffer bytes fetched from disk for writes.

**variable Tokudb_BUFFERS_FETCHED_FOR_WRITE_SECONDS**

This variable shows the number of seconds waiting for I/O when fetching buffers from disk for writes.

**variable Tokudb_LEAF_COMPRESSION_TO_MEMORY_SECONDS**

This variable shows the total time, in seconds, spent compressing leaf nodes.

**variable Tokudb_LEAF_SERIALIZATION_TO_MEMORY_SECONDS**

This variable shows the total time, in seconds, spent serializing leaf nodes.

**variable Tokudb_LEAF_DECOMPRESSION_TO_MEMORY_SECONDS**

This variable shows the total time, in seconds, spent decompressing leaf nodes.

**variable Tokudb_LEAF_DESERIALIZATION_TO_MEMORY_SECONDS**

This variable shows the total time, in seconds, spent deserializing leaf nodes.

**variable Tokudb_NONLEAF_COMPRESSION_TO_MEMORY_SECONDS**

This variable shows the total time, in seconds, spent compressing non leaf nodes.

**variable Tokudb_NONLEAF_SERIALIZATION_TO_MEMORY_SECONDS**

This variable shows the total time, in seconds, spent serializing non leaf nodes.

**variable Tokudb_NONLEAF_DECOMPRESSION_TO_MEMORY_SECONDS**

This variable shows the total time, in seconds, spent decompressing non leaf nodes.

**variable Tokudb_NONLEAF_DESERIALIZATION_TO_MEMORY_SECONDS**

This variable shows the total time, in seconds, spent deserializing non leaf nodes.

**variable Tokudb_PROMOTION_ROOTS_SPLIT**

This variable shows the number of times the root split during promotion.

**variable Tokudb_PROMOTION_LEAF_ROOTS_INJECTED_INTO**

This variable shows the number of times a message stopped at a root with height `0`.

**variable Tokudb_PROMOTION_H1_ROOTS_INJECTED_INTO**

This variable shows the number of times a message stopped at a root with height `1`.

**variable Tokudb_PROMOTION_INJECTIONS_AT_DEPTH_0**

This variable shows the number of times a message stopped at depth `0`.

**variable Tokudb_PROMOTION_INJECTIONS_AT_DEPTH_1**

This variable shows the number of times a message stopped at depth `1`.

**variable Tokudb_PROMOTION_INJECTIONS_AT_DEPTH_2**

This variable shows the number of times a message stopped at depth `2`.

**variable Tokudb_PROMOTION_INJECTIONS_AT_DEPTH_3**

This variable shows the number of times a message stopped at depth `3`.

**variable Tokudb_PROMOTION_INJECTIONS_LOWER_THAN_DEPTH_3**

This variable shows the number of times a message was promoted past depth `3`.

**variable Tokudb_PROMOTION_STOPPED_NONEMPTY_BUFFER**

This variable shows the number of times a message stopped because it reached a nonempty buffer.

**variable Tokudb_PROMOTION_STOPPED_AT_HEIGHT_1**

This variable shows the number of times a message stopped because it had reached height `1`.

**variable Tokudb_PROMOTION_STOPPED_CHILD_LOCKED_OR_NOT_IN_MEMORY**

This variable shows the number of times a message stopped because it could not cheaply get access to a child.

**variable Tokudb_PROMOTION_STOPPED_CHILD_NOT_FULLY_IN_MEMORY**

This variable shows the number of times a message stopped because it could not cheaply get access to a child.

**variable Tokudb_PROMOTION_STOPPED_AFTER_LOCKING_CHILD**

This variable shows the number of times a message stopped before a child which had been locked.

**variable Tokudb_BASEMENT_DESERIALIZATION_FIXED_KEY**

This variable shows the number of basement nodes deserialized where all keys had the same size, leaving the basement in a format that is optimal for in-memory workloads.

**variable Tokudb_BASEMENT_DESERIALIZATION_VARIABLE_KEY**

This variable shows the number of basement nodes deserialized where all keys did not have the same size, and thus ineligible for an in-memory optimization.

**variable Tokudb_PRO_RIGHTMOST_LEAF_SHORTCUT_SUCCESS**

This variable shows the number of times a message injection detected a series of sequential inserts to the rightmost side of the tree and successfully applied an insert message directly to the rightmost leaf node. This is a not a useful value for a regular user to use for any purpose.

**variable Tokudb_PRO_RIGHTMOST_LEAF_SHORTCUT_FAIL_POS**

This variable shows the number of times a message injection detected a series of sequential inserts to the rightmost side of the tree and was unable to follow the pattern of directly applying an insert message directly to the rightmost leaf node because the key does not continue the sequence. This is a not a useful value for a regular user to use for any purpose.

**variable Tokudb_RIGHTMOST_LEAF_SHORTCUT_FAIL_REACTIVE**

This variable shows the number of times a message injection detected a series of sequential inserts to the rightmost side of the tree and was unable to follow the pattern of directly applying an insert message directly to the rightmost leaf node because the leaf is full. This is a not a useful value for a regular user to use for any purpose.

**variable Tokudb_CURSOR_SKIP_DELETED_LEAF_ENTRY**

This variable shows the number of leaf entries skipped during search/scan because the result of message application and reconciliation of the leaf entry MVCC stack reveals that the leaf entry is `deleted` in the current transactions view. It is a good indicator that there might be excessive garbage in a tree if a range scan seems to take too long.

**variable Tokudb_FLUSHER_CLEANER_TOTAL_NODES**

This variable shows the total number of nodes potentially flushed by flusher or cleaner threads. This is a not a useful value for a regular user to use for any purpose.

**variable Tokudb_FLUSHER_CLEANER_H1_NODES**

This variable shows the number of height `1` nodes that had messages flushed by flusher or cleaner threads, i.e., internal nodes immediately above leaf nodes. This is a not a useful value for a regular user to use for any purpose.

**variable Tokudb_FLUSHER_CLEANER_HGT1_NODES**

This variable shows the number of nodes with height greater than `1` that had messages flushed by flusher or cleaner threads. This is a not a useful value for a regular user to use for any purpose.

**variable Tokudb_FLUSHER_CLEANER_EMPTY_NODES**

This variable shows the number of nodes cleaned by flusher or cleaner threads which had empty message buffers. This is a not a useful value for a regular user to use for any purpose.

**variable Tokudb_FLUSHER_CLEANER_NODES_DIRTIED**

This variable shows the number of nodes dirtied by flusher or cleaner threads as a result of flushing messages downward. This is a not a useful value for a regular user to use for any purpose.

**variable Tokudb_FLUSHER_CLEANER_MAX_BUFFER_SIZE**

This variable shows the maximum bytes in a message buffer flushed by flusher or cleaner threads. This is a not a useful value for a regular user to use for any purpose.

**variable Tokudb_FLUSHER_CLEANER_MIN_BUFFER_SIZE**

This variable shows the minimum bytes in a message buffer flushed by flusher or cleaner threads. This is a not a useful value for a regular user to use for any purpose.

**variable Tokudb_FLUSHER_CLEANER_TOTAL_BUFFER_SIZE**

This variable shows the total bytes in buffers flushed by flusher and cleaner threads. This is a not a useful value for a regular user to use for any purpose.

**variable Tokudb_FLUSHER_CLEANER_MAX_BUFFER_WORKDONE**

This variable shows the maximum bytes worth of work done in a message buffer flushed by flusher or cleaner threads. This is a not a useful value for a regular user to use for any purpose.

**variable Tokudb_FLUSHER_CLEANER_MIN_BUFFER_WORKDONE**

This variable shows the minimum bytes worth of work done in a message buffer flushed by flusher or cleaner threads. This is a not a useful value for a regular user to use for any purpose.

**variable Tokudb_FLUSHER_CLEANER_TOTAL_BUFFER_WORKDONE**

This variable shows the total bytes worth of work done in buffers flushed by flusher or cleaner threads. This is a not a useful value for a regular user to use for any purpose.

**variable Tokudb_FLUSHER_CLEANER_NUM_LEAF_MERGES_STARTED**

This variable shows the number of times flusher and cleaner threads tried to merge two leafs. This is a not a useful value for a regular user to use for any purpose.

**variable Tokudb_FLUSHER_CLEANER_NUM_LEAF_MERGES_RUNNING**

This variable shows the number of flusher and cleaner threads leaf merges in progress. This is a not a useful value for a regular user to use for any purpose.

**variable Tokudb_FLUSHER_CLEANER_NUM_LEAF_MERGES_COMPLETED**

This variable shows the number of successful flusher and cleaner threads leaf merges. This is a not a useful value for a regular user to use for any purpose.

**variable Tokudb_FLUSHER_CLEANER_NUM_DIRTIED_FOR_LEAF_MERGE**

This variable shows the number of nodes dirtied by flusher or cleaner threads performing leaf node merges. This is a not a useful value for a regular user to use for any purpose.

**variable Tokudb_FLUSHER_FLUSH_TOTAL**

This variable shows the total number of flushes done by flusher threads or cleaner threads. This is a not a useful value for a regular user to use for any purpose.

**variable Tokudb_FLUSHER_FLUSH_IN_MEMORY**

This variable shows the number of in memory flushes (required no disk reads) by flusher or cleaner threads. This is a not a useful value for a regular user to use for any purpose.

**variable Tokudb_FLUSHER_FLUSH_NEEDED_IO**

This variable shows the number of flushes that read something off disk by flusher or cleaner threads. This is a not a useful value for a regular user to use for any purpose.

**variable Tokudb_FLUSHER_FLUSH_CASCADES**

This variable shows the number of flushes that triggered a flush in child node by flusher or cleaner threads. This is a not a useful value for a regular user to use for any purpose.

**variable Tokudb_FLUSHER_FLUSH_CASCADES_1**

This variable shows the number of flushes that triggered one cascading flush by flusher or cleaner threads. This is a not a useful value for a regular user to use for any purpose.

**variable Tokudb_FLUSHER_FLUSH_CASCADES_2**

This variable shows the number of flushes that triggered two cascading flushes by flusher or cleaner threads. This is a not a useful value for a regular user to use for any purpose.

**variable Tokudb_FLUSHER_FLUSH_CASCADES_3**

This variable shows the number of flushes that triggered three cascading flushes by flusher or cleaner threads. This is a not a useful value for a regular user to use for any purpose.

**variable Tokudb_FLUSHER_FLUSH_CASCADES_4**

This variable shows the number of flushes that triggered four cascading flushes by flusher or cleaner threads. This is a not a useful value for a regular user to use for any purpose.

**variable Tokudb_FLUSHER_FLUSH_CASCADES_5**

This variable shows the number of flushes that triggered five cascading flushes by flusher or cleaner threads. This is a not a useful value for a regular user to use for any purpose.

**variable Tokudb_FLUSHER_FLUSH_CASCADES_GT_5**

This variable shows the number of flushes that triggered more than five cascading flushes by flusher or cleaner threads. This is a not a useful value for a regular user to use for any purpose.

**variable Tokudb_FLUSHER_SPLIT_LEAF**

This variable shows the total number of leaf node splits done by flusher threads or cleaner threads. This is a not a useful value for a regular user to use for any purpose.

**variable Tokudb_FLUSHER_SPLIT_NONLEAF**

This variable shows the total number of non-leaf node splits done by flusher threads or cleaner threads. This is a not a useful value for a regular user to use for any purpose.

**variable Tokudb_FLUSHER_MERGE_LEAF**

This variable shows the total number of leaf node merges done by flusher threads or cleaner threads. This is a not a useful value for a regular user to use for any purpose.

**variable Tokudb_FLUSHER_MERGE_NONLEAF**

This variable shows the total number of non-leaf node merges done by flusher threads or cleaner threads. This is a not a useful value for a regular user to use for any purpose.

**variable Tokudb_FLUSHER_BALANCE_LEAF**

This variable shows the number of times two adjacent leaf nodes were rebalanced or had their content redistributed evenly by flusher or cleaner threads. This is a not a useful value for a regular user to use for any purpose.

**variable Tokudb_HOT_NUM_STARTED**

This variable shows the number of hot operations started (`OPTIMIZE TABLE`). This is a not a useful value for a regular user to use for any purpose.

**variable Tokudb_HOT_NUM_COMPLETED**

This variable shows the number of hot operations completed (`OPTIMIZE TABLE`). This is a not a useful value for a regular user to use for any purpose.

**variable Tokudb_HOT_NUM_ABORTED**

This variable shows the number of hot operations aborted (`OPTIMIZE TABLE`). This is a not a useful value for a regular user to use for any purpose.

**variable Tokudb_HOT_MAX_ROOT_FLUSH_COUNT**

This variable shows the maximum number of flushes from root ever required to optimize trees. This is a not a useful value for a regular user to use for any purpose.

**variable Tokudb_TXN_BEGIN**

This variable shows the number of transactions that have been started.

**variable Tokudb_TXN_BEGIN_READ_ONLY**

This variable shows the number of read-only transactions started.

**variable Tokudb_TXN_COMMITS**

This variable shows the total number of transactions that have been committed.

**variable Tokudb_TXN_ABORTS**

This variable shows the total number of transactions that have been aborted.

**variable Tokudb_LOGGER_NEXT_LSN**

This variable shows the recovery logger next LSN. This is a not a useful value for a regular user to use for any purpose.

**variable Tokudb_LOGGER_WRITES**

This variable shows the number of times the logger has written to disk.

**variable Tokudb_LOGGER_WRITES_BYTES**

This variable shows the number of bytes the logger has written to disk.

**variable Tokudb_LOGGER_WRITES_UNCOMPRESSED_BYTES**

This variable shows the number of uncompressed bytes the logger has written to disk.

**variable Tokudb_LOGGER_WRITES_SECONDS**

This variable shows the number of seconds waiting for IO when writing logs to disk.

**variable Tokudb_LOGGER_WAIT_LONG**

This variable shows the number of times a logger write operation required 100ms or more.

**variable Tokudb_LOADER_NUM_CREATED**

This variable shows the number of times one of our internal objects, a loader, has been created.

**variable Tokudb_LOADER_NUM_CURRENT**

This variable shows the number of loaders that currently exist.

**variable Tokudb_LOADER_NUM_MAX**

This variable shows the maximum number of loaders that ever existed simultaneously.

**variable Tokudb_MEMORY_MALLOC_COUNT**

This variable shows the number of `malloc` operations by PerconaFT.

**variable Tokudb_MEMORY_FREE_COUNT**

This variable shows the number of `free` operations by PerconaFT.

**variable Tokudb_MEMORY_REALLOC_COUNT**

This variable shows the number of `realloc` operations by PerconaFT.

**variable Tokudb_MEMORY_MALLOC_FAIL**

This variable shows the number of `malloc` operations that failed by PerconaFT.

**variable Tokudb_MEMORY_REALLOC_FAIL**

This variable shows the number of `realloc` operations that failed by PerconaFT.

**variable Tokudb_MEMORY_REQUESTED**

This variable shows the number of bytes requested by PerconaFT.

**variable Tokudb_MEMORY_USED**

This variable shows the number of bytes used (requested + overhead) by PerconaFT.

**variable Tokudb_MEMORY_FREED**

This variable shows the number of bytes freed by PerconaFT.

**variable Tokudb_MEMORY_MAX_REQUESTED_SIZE**

This variable shows the largest attempted allocation size by PerconaFT.

**variable Tokudb_MEMORY_LAST_FAILED_SIZE**

This variable shows the size of the last failed allocation attempt by PerconaFT.

**variable Tokudb_MEM_ESTIMATED_MAXIMUM_MEMORY_FOOTPRINT**

This variable shows the maximum memory footprint of the storage engine, the max value of (used - freed).

**variable Tokudb_MEMORY_MALLOCATOR_VERSION**

This variable shows the version of the memory allocator library detected by PerconaFT.

**variable Tokudb_MEMORY_MMAP_THRESHOLD**

This variable shows the `mmap` threshold in PerconaFT, anything larger than this gets `mmap'ed`.

**variable Tokudb_FILESYSTEM_THREADS_BLOCKED_BY_FULL_DISK**

This variable shows the number of threads that are currently blocked because they are attempting to write to a full disk. This is normally zero. If this value is non-zero, then a warning will appear in the `disk free space` field.

**variable Tokudb_FILESYSTEM_FSYNC_TIME**

This variable shows the total time, in microseconds, used to `fsync` to disk.

**variable Tokudb_FILESYSTEM_FSYNC_NUM**

This variable shows the total number of times the database has flushed the operating system's file buffers to disk.

**variable Tokudb_FILESYSTEM_LONG_FSYNC_TIME**

This variable shows the total time, in microseconds, used to `fsync` to dis k when the operation required more than one second.

**variable Tokudb_FILESYSTEM_LONG_FSYNC_NUM**

This variable shows the total number of times the database has flushed the operating system's file buffers to disk and this operation required more than one second.

CHAPTER

SEVENTY

# *TOKUDB* PERFORMANCE SCHEMA INTEGRATION

*TokuDB* is integrated with Performance Schema

This integration can be used for profiling additional *TokuDB* operations.

*TokuDB* instruments available in Performance Schema can be seen in `PERFORMANCE_SCHEMA.SETUP_INSTRUMENTS` table:

```
mysql> SELECT * FROM performance_schema.setup_instruments WHERE NAME LIKE "%/fti/%";
+-----------------------------------------------------------+---------+-------+
| NAME                                                      | ENABLED | TIMED |
+-----------------------------------------------------------+---------+-------+
| wait/synch/mutex/fti/kibbutz_mutex                        | NO      | NO    |
| wait/synch/mutex/fti/minicron_p_mutex                     | NO      | NO    |
| wait/synch/mutex/fti/queue_result_mutex                   | NO      | NO    |
| wait/synch/mutex/fti/tpool_lock_mutex                     | NO      | NO    |
| wait/synch/mutex/fti/workset_lock_mutex                   | NO      | NO    |
| wait/synch/mutex/fti/bjm_jobs_lock_mutex                  | NO      | NO    |
| wait/synch/mutex/fti/log_internal_lock_mutex              | NO      | NO    |
| wait/synch/mutex/fti/cachetable_ev_thread_lock_mutex      | NO      | NO    |
| wait/synch/mutex/fti/cachetable_disk_nb_mutex             | NO      | NO    |
| wait/synch/mutex/fti/safe_file_size_lock_mutex            | NO      | NO    |
| wait/synch/mutex/fti/cachetable_m_mutex_key               | NO      | NO    |
| wait/synch/mutex/fti/checkpoint_safe_mutex                | NO      | NO    |
| wait/synch/mutex/fti/ft_ref_lock_mutex                    | NO      | NO    |
| wait/synch/mutex/fti/ft_open_close_lock_mutex             | NO      | NO    |
| wait/synch/mutex/fti/loader_error_mutex                   | NO      | NO    |
| wait/synch/mutex/fti/bfs_mutex                            | NO      | NO    |
| wait/synch/mutex/fti/loader_bl_mutex                      | NO      | NO    |
| wait/synch/mutex/fti/loader_fi_lock_mutex                 | NO      | NO    |
| wait/synch/mutex/fti/loader_out_mutex                     | NO      | NO    |
| wait/synch/mutex/fti/result_output_condition_lock_mutex   | NO      | NO    |
| wait/synch/mutex/fti/block_table_mutex                    | NO      | NO    |
| wait/synch/mutex/fti/rollback_log_node_cache_mutex        | NO      | NO    |
| wait/synch/mutex/fti/txn_lock_mutex                       | NO      | NO    |
| wait/synch/mutex/fti/txn_state_lock_mutex                 | NO      | NO    |
| wait/synch/mutex/fti/txn_child_manager_mutex              | NO      | NO    |
| wait/synch/mutex/fti/txn_manager_lock_mutex               | NO      | NO    |
| wait/synch/mutex/fti/treenode_mutex                       | NO      | NO    |
| wait/synch/mutex/fti/locktree_request_info_mutex          | NO      | NO    |
| wait/synch/mutex/fti/locktree_request_info_retry_mutex_key | NO      | NO    |
| wait/synch/mutex/fti/manager_mutex                        | NO      | NO    |
| wait/synch/mutex/fti/manager_escalation_mutex             | NO      | NO    |
| wait/synch/mutex/fti/db_txn_struct_i_txn_mutex            | NO      | NO    |
| wait/synch/mutex/fti/manager_escalator_mutex              | NO      | NO    |
| wait/synch/mutex/fti/indexer_i_indexer_lock_mutex         | NO      | NO    |
```

```
| wait/synch/mutex/fti/indexer_i_indexer_estimate_lock_mutex | NO       | NO    |
| wait/synch/mutex/fti/fti_probe_1                           | NO       | NO    |
| wait/synch/rwlock/fti/multi_operation_lock                 | NO       | NO    |
| wait/synch/rwlock/fti/low_priority_multi_operation_lock    | NO       | NO    |
| wait/synch/rwlock/fti/cachetable_m_list_lock               | NO       | NO    |
| wait/synch/rwlock/fti/cachetable_m_pending_lock_expensive  | NO       | NO    |
| wait/synch/rwlock/fti/cachetable_m_pending_lock_cheap      | NO       | NO    |
| wait/synch/rwlock/fti/cachetable_m_lock                    | NO       | NO    |
| wait/synch/rwlock/fti/result_i_open_dbs_rwlock             | NO       | NO    |
| wait/synch/rwlock/fti/checkpoint_safe_rwlock               | NO       | NO    |
| wait/synch/rwlock/fti/cachetable_value                     | NO       | NO    |
| wait/synch/rwlock/fti/safe_file_size_lock_rwlock           | NO       | NO    |
| wait/synch/rwlock/fti/cachetable_disk_nb_rwlock            | NO       | NO    |
| wait/synch/cond/fti/result_state_cond                      | NO       | NO    |
| wait/synch/cond/fti/bjm_jobs_wait                          | NO       | NO    |
| wait/synch/cond/fti/cachetable_p_refcount_wait             | NO       | NO    |
| wait/synch/cond/fti/cachetable_m_flow_control_cond         | NO       | NO    |
| wait/synch/cond/fti/cachetable_m_ev_thread_cond            | NO       | NO    |
| wait/synch/cond/fti/bfs_cond                               | NO       | NO    |
| wait/synch/cond/fti/result_output_condition                | NO       | NO    |
| wait/synch/cond/fti/manager_m_escalator_done               | NO       | NO    |
| wait/synch/cond/fti/lock_request_m_wait_cond               | NO       | NO    |
| wait/synch/cond/fti/queue_result_cond                      | NO       | NO    |
| wait/synch/cond/fti/ws_worker_wait                         | NO       | NO    |
| wait/synch/cond/fti/rwlock_wait_read                       | NO       | NO    |
| wait/synch/cond/fti/rwlock_wait_write                      | NO       | NO    |
| wait/synch/cond/fti/rwlock_cond                            | NO       | NO    |
| wait/synch/cond/fti/tp_thread_wait                         | NO       | NO    |
| wait/synch/cond/fti/tp_pool_wait_free                      | NO       | NO    |
| wait/synch/cond/fti/frwlock_m_wait_read                    | NO       | NO    |
| wait/synch/cond/fti/kibbutz_k_cond                         | NO       | NO    |
| wait/synch/cond/fti/minicron_p_condvar                     | NO       | NO    |
| wait/synch/cond/fti/locktree_request_info_retry_cv_key     | NO       | NO    |
| wait/io/file/fti/tokudb_data_file                          | YES      | YES   |
| wait/io/file/fti/tokudb_load_file                          | YES      | YES   |
| wait/io/file/fti/tokudb_tmp_file                           | YES      | YES   |
| wait/io/file/fti/tokudb_log_file                           | YES      | YES   |
+------------------------------------------------------------+----------+-------+
```

For *TokuDB*-related objects, following clauses can be used when querying Performance Schema tables:

- `WHERE EVENT_NAME LIKE '%fti%'` or
- `WHERE NAME LIKE '%fti%'`

For example, to get the information about *TokuDB* related events you can query `PERFORMANCE_SCHEMA.events_waits_summary_global_by_event_name` like:

```
mysql> SELECT * FROM performance_schema.events_waits_summary_global_by_event_name␣
↪WHERE EVENT_NAME LIKE '%fti%';

+----------------------------------------+------------+----------------+-------------
↪---+----------------+----------------+
| EVENT_NAME                             | COUNT_STAR | SUM_TIMER_WAIT | MIN_TIMER_
↪WAIT | AVG_TIMER_WAIT | MAX_TIMER_WAIT |
+----------------------------------------+------------+----------------+-------------
↪---+----------------+----------------+
| wait/synch/mutex/fti/kibbutz_mutex     |          0 |              0 |           ␣
↪ 0 |              0 |              0 |
```

```
| wait/synch/mutex/fti/minicron_p_mutex    |          0 |               0 |
↪ 0 |              0 |              0 |
| wait/synch/mutex/fti/queue_result_mutex |          0 |               0 |
↪ 0 |              0 |              0 |
| wait/synch/mutex/fti/tpool_lock_mutex    |          0 |               0 |
↪ 0 |              0 |              0 |
| wait/synch/mutex/fti/workset_lock_mutex |          0 |               0 |
↪ 0 |              0 |              0 |
...
| wait/io/file/fti/tokudb_data_file        |         30 |       179862410 |
↪ 0 |        5995080 |       68488420 |
| wait/io/file/fti/tokudb_load_file        |          0 |               0 |
↪ 0 |              0 |              0 |
| wait/io/file/fti/tokudb_tmp_file         |          0 |               0 |
↪ 0 |              0 |              0 |
| wait/io/file/fti/tokudb_log_file         |       1367 |  2925647870145 |
↪ 0 |     2140195785 |    12013357720 |
+------------------------------------------+------------+-----------------+-------------
↪---+----------------+----------------+
71 rows in set (0.02 sec)
```

# Part XI

# Percona MyRocks

CHAPTER

SEVENTYONE

# PERCONA MYROCKS INTRODUCTION

MyRocks is a storage engine for MySQL based on RocksDB, an embeddable, persistent key-value store. *Percona MyRocks* is an implementation for Percona Server for MySQL.

The RocksDB store is based on the log-structured merge-tree (or LSM tree). It is optimized for fast storage and combines outstanding space and write efficiency with acceptable read performance. As a result, MyRocks has the following advantages compared to other storage engines, if your workload uses fast storage, such as SSD:

- Requires less storage space
- Provides more storage endurance
- Ensures better IO capacity

## Percona MyRocks Installation Guide

Percona MyRocks is distributed as a separate package that can be enabled as a plugin for *Percona Server for MySQL* 8.0 and later versions.

**Note:** File formats across different MyRocks variants may not be compatible. *Percona Server for MySQL* supports only *Percona MyRocks*. Migrating from one variant to another requires a logical data dump and reload.



### Installing Percona MyRocks

It is recommended to install Percona software from official repositories:

1. Configure Percona repositories as described in Percona Software Repositories Documentation.
2. Install Percona MyRocks using the corresponding package manager:
   - For Debian or Ubuntu:

     ```
     $ sudo apt-get install percona-server-rocksdb
     ```

   - For RHEL or CentOS:

```
$ sudo yum install percona-server-rocksdb
```

After you install the Percona MyRocks package, you should see the following output:

```
* This release of |Percona Server| is distributed with RocksDB storage engine.
* Run the following script to enable the RocksDB storage engine in Percona Server:
```

```
$ ps-admin --enable-rocksdb -u <mysql_admin_user> -p[mysql_admin_pass] [-S
↪<socket>] [-h <host> -P <port>]
```

Run the `ps-admin` script as system root user or with **sudo** and provide the MySQL root user credentials to properly enable the RocksDB (MyRocks) storage engine:

```
$ sudo ps-admin --enable-rocksdb -u root -pPassw0rd

Checking if RocksDB plugin is available for installation ...
INFO: ha_rocksdb.so library for RocksDB found at /usr/lib64/mysql/plugin/ha_rocksdb.
↪so.

Checking RocksDB engine plugin status...
INFO: RocksDB engine plugin is not installed.

Installing RocksDB engine...
INFO: Successfully installed RocksDB engine plugin.
```

**Note:** When you use the `ps-admin` script to enable Percona MyRocks, it performs the following:

- Disables Transparent huge pages
- Installs and enables the RocksDB plugin

If the script returns no errors, Percona MyRocks should be successfully enabled on the server. You can verify it as follows:

```
mysql> SHOW ENGINES;
+--------+--------+----------------------------------------------------------------
↪-----------+--------------+------+------------+
| Engine  | Support | Comment
↪            | Transactions | XA   | Savepoints |
+--------+--------+----------------------------------------------------------------
↪-----------+--------------+------+------------+
| ROCKSDB | YES     | RocksDB storage engine
↪            | YES          | YES  | YES        |
...
| InnoDB  | DEFAULT | Percona-XtraDB, Supports transactions, row-level locking, and␣
↪foreign keys | YES          | YES  | YES        |
+--------+--------+----------------------------------------------------------------
↪-----------+--------------+------+------------+
10 rows in set (0.00 sec)
```

Note that the RocksDB engine is not set to be default, new tables will still be created using the InnoDB (XtraDB) storage engine. To make RocksDB storage engine default, set `default-storage-engine=rocksdb` in the `[mysqld]` section of `my.cnf` and restart *Percona Server for MySQL*.

Alternatively, you can add `ENGINE=RocksDB` after the `CREATE TABLE` statement for every table that you create.

### Removing Percona MyRocks

It will not be possible to access tables created using the RocksDB engine with another storage engine after you remove Percona MyRocks. If you need this data, alter the tables to another storage engine. For example, to alter the `City` table to InnoDB, run the following:

```
mysql> ALTER TABLE City ENGINE=InnoDB;
```

To disable and uninstall the RocksDB engine plugins, use the `ps-admin` script as follows:

```
$ sudo ps-admin --disable-rocksdb -u root -pPassw0rd

Checking RocksDB engine plugin status...
INFO: RocksDB engine plugin is installed.

Uninstalling RocksDB engine plugin...
INFO: Successfully uninstalled RocksDB engine plugin.
```

After the engine plugins have been uninstalled, remove the Percona MyRocks package:

- For Debian or Ubuntu:

  ```
  $ sudo apt-get remove percona-server-rocksdb-8.0
  ```

- For RHEL or CentOS:

  ```
  $ sudo yum remove percona-server-rocksdb-80.x86_64
  ```

Finally, remove all the *MyRocks Server Variables* from the configuration file (`my.cnf`) and restart *Percona Server for MySQL*.

## MyRocks Limitations

The MyRocks storage engine lacks the following features compared to InnoDB:

- Online DDL
- ALTER TABLE ... EXCHANGE PARTITION
- SAVEPOINT
- Transportable tablespace
- Foreign keys
- Spatial indexes
- Fulltext indexes
- Gap locks
- Group Replication
- Partial Update of LOB in InnoDB

You should also consider the following:

- `*_bin` (e.g. `latin1_bin`) or binary collation should be used on `CHAR` and `VARCHAR` indexed columns. By default, MyRocks prevents creating indexes with non-binary collations (including `latin1`). You can optionally use it by setting *`rocksdb_strict_collation_exceptions`* to `t1` (table names with regex format), but

non-binary covering indexes other than `latin1` (excluding `german1`) still require a primary key lookup to return the `CHAR` or `VARCHAR` column.

- Either `ORDER BY DESC` or `ORDER BY ASC` is slow. This is because of "Prefix Key Encoding" feature in RocksDB. See http://www.slideshare.net/matsunobu/myrocks-deep-dive/58 for details. By default, ascending scan is faster and descending scan is slower. If the "reverse column family" is configured, then descending scan will be faster and ascending scan will be slower. Note that InnoDB also imposes a cost when the index is scanned in the opposite order.
- MyRocks does not support operating as either a master or a slave in any replication topology that is not exclusively row-based. Statement-based and mixed-format binary logging is not supported. For more information, see Replication Formats.
- As of 8.0.17, InnoDB supports multi-valued indexes. MyRocks does not support this feature.
- As of 8.0.17, InnoDB supports the use of the Clone Plugin and the Clone Plugin API. MyRocks tables do not support either these features.
- When converting from large MyISAM/InnoDB tables, either by using the `ALTER` or `INSERT INTO SELECT` statements it's recommended that you check the *Data loading* documentation and create MyRocks tables as below (in case the table is sufficiently big it will cause the server to consume all the memory and then be terminated by the OOM killer):

```
SET session sql_log_bin=0;
SET session rocksdb_bulk_load=1;
ALTER TABLE large_myisam_table ENGINE=RocksDB;
SET session rocksdb_bulk_load=0;

.. warning::

If you are loading large data without enabling :variable:`rocksdb_bulk_load`
or :variable:`rocksdb_commit_in_the_middle`, please make sure transaction
size is small enough. All modifications of the ongoing transactions are
kept in memory.
```

- The'XA protocol <https://dev.mysql.com/doc/refman/8.0/en/xa.html>'_ support, which allows distributed transactions combining multiple separate transactional resources, is an experimental feature in MyRocks: the implementation is less tested, it may lack some functionality and be not as stable as in case of InnoDB.
- With partitioned tables that use the *TokuDB* or *MyRocks* storage engine, the upgrade only works with native partitioning.

  **See also:**

  ***MySQL* Documentation: Preparing Your Installation for Upgrade** https://dev.mysql.com/doc/refman/8.0/en/upgrade-prerequisites.html
- *Percona Server for MySQL* 8.0 and Unicode 9.0.0 standards have defined a change in the handling of binary collations. These collations are handled as NO PAD, trailing spaces are included in key comparisons. A binary collation comparison may result in two unique rows inserted and does not generate a'DUP_ENTRY' error. MyRocks key encoding and comparison does not account for this character set attribute.
- In version 8.0.13-3 and later, MyRocks does not support explict DEFAULT value expressions.
- *Percona Server for MySQL* 8.0.16 does not support encryption for the MyRocks storage engine. At this time, during an `ALTER TABLE` operation, MyRocks mistakenly detects all InnoDB tables as encrypted. Therefore, any attempt to `ALTER` an InnoDB table to MyRocks fails.

  As a workaround, we recommend a manual move of the table. The following steps are the same as the `ALTER TABLE ... ENGINE=...` process:
- Use `SHOW CREATE TABLE ...` to return the InnoDB table definition.

- With the table definition as the source, perform a `CREATE TABLE ... ENGINE=RocksDB`.
- In the new table, use `INSERT INTO <new table> SELECT * FROM <old table>`.

**Note:** With MyRocks and with large tables, it is recommended to set the session variable `rocksdb_bulk_load=1` during the load to prevent running out of memory. This recommendation is because of the MyRocks large transaction limitation.

**See also:**

MyRocks Data Loading https://www.percona.com/doc/percona-server/8.0/myrocks/data_loading.html

## Differences between Percona MyRocks and Facebook MyRocks

The original MyRocks was developed by Facebook and works with their implementation of MySQL. *Percona MyRocks* is a branch of MyRocks for *Percona Server for MySQL* and includes the following differences from the original implementation:

- The behavior of the `START TRANSACTION WITH CONSISTENT SNAPSHOT` statement depends on the transaction isolation level.

| Storage Engine | Transaction isolation level | |
|---|---|---|
| | `READ COMMITTED` | `REPEATABLE READ` |
| InnoDB | Success | Success |
| Facebook MyRocks | Fail | Success (MyRocks engine only; read-only, as all MyRocks engine snapshots) |
| Percona MyRocks | Fail with any DML which would violate the read-only snapshot constraint | Success (read-only snapshots independent of the engines in use) |

- Percona MyRocks includes the `lz4` and `zstd` statically linked libraries.

## *MyRocks* Column Families

*MyRocks* stores all data in a single server instance as a collection of key-value pairs within the *log structured merge tree* data structure. This is a flat data structure that requires that keys be unique throughout the whole data structure. *MyRocks* incorporates table IDs and index IDs into the keys.

Each key-value pair belongs to a column family. It is a data structure similar in concept to tablespaces. Each column family has distinct attributes, such as block size, compression, sort order, and MemTable. Utilizing these attributes, *MyRocks* effectively uses column families to store indexes.

On system initialization, *MyRocks* creates two column families. The `__system__` column family is reserved by *MyRocks*; no user created tables or indexes belong to this column family. The `default` column family is the location for the indexes created by the user when you a column family is not explicitly specified.

To be able to apply a custom block size, compression, or sort order you need to create an index in its own column family using the `COMMENT` clause.

The following example demonstrates how to place the `PRIMARY KEY` into the *cf1* column family and the index `kb` — into the *cf2* column family.

```
CREATE TABLE t1 (a INT, b INT,
PRIMARY KEY(a) COMMENT 'cfname=cf1',
KEY kb(b) COMMENT 'cf_name=cf2')
ENGINE=ROCKSDB;
```

The column family name is specified as the value of the *cf_name* attribute at the beginning of the `COMMENT` clause. The name is case sensitive and may not contain leading or trailing whitespace characters.

The `COMMENT` clause may contain other information following the semicolon character (;) after the column family name: 'cfname=foo; special column family'. If the column family cannot be created, *MyRocks* uses the default column family.

**Important:** The *cf_name* attribute must be all lowercase. Place the equals sign (=) in front of the column family name without any whitespace on both sides of it.

```
COMMENT 'cfname=Foo; Creating the Foo family name'
```

**See also:**

**Using `COMMENT` to Specify Column Family Names with Multiple Table Partitions** https://github.com/facebook/mysql-5.6/wiki/Column-Families-on-Partitioned-Tables.

### Controlling the number of column families to reduce memory consumption

Each column family has its own MemTable. It is an in-memory data structure where data are written to before they are flushed to SST files. The queries also use MemTables first. To reduce the overall memory consumption, the number of active column families should stay low.

With the option *`rocksdb_no_create_column_family`* set to *true*, the `COMMENT` clause will not treat *cf_name* as a special token; it will not be possible to create column families using the `COMMENT` clause.

## Column Family Options

On startup, the server applies the *`rocksdb_default_cf_options`* option to all existing column families. You may use the *`rocksdb_override_cf_options`* option to override the value of any attribute of a chosen column family.

Note that the options *`rocksdb_default_cf_options`* and *`rocksdb_override_cf_options`* are read-only at runtime.

At runtime, use the the *`rocksdb_update_cf_options`* option to update some column family attributes.

**Important:** Changes made to a column families using the *`rocksdb_update_cf_options`* option only persist until the server is restarted.

# MyRocks Server Variables

The MyRocks server variables expose configuration of the underlying RocksDB engine. There several ways to set these variables:

- For production deployments, you should have all variables defined in the configuration file.
- *Dynamic* variables can be changed at runtime using the `SET` statement.

- If you want to test things out, you can set some of the variables when starting `mysqld` using corresponding command-line options.

If a variable was not set in either the configuration file or as a command-line option, the default value is used.

Also, all variables can exist in one or both of the following scopes:

- *Global* scope defines how the variable affects overall server operation.
- *Session* scope defines how the variable affects operation for individual client connections.

| Name | Command Line | Dynamic | Scope |
|---|---|---|---|
| *rocksdb_access_hint_on_compaction_start* | Yes | No | Global |
| *rocksdb_advise_random_on_open* | Yes | No | Global |
| *rocksdb_allow_concurrent_memtable_write* | Yes | No | Global |
| *rocksdb_allow_to_start_after_corruption* | Yes | No | Global |
| *rocksdb_allow_mmap_reads* | Yes | No | Global |
| *rocksdb_allow_mmap_writes* | Yes | No | Global |
| *rocksdb_base_background_compactions* | Yes | No | Global |
| *rocksdb_block_cache_size* | Yes | Yes | Global |
| *rocksdb_block_restart_interval* | Yes | No | Global |
| *rocksdb_block_size* | Yes | No | Global |
| *rocksdb_block_size_deviation* | Yes | No | Global |
| *rocksdb_bulk_load* | Yes | Yes | Global, Session |
| *rocksdb_bulk_load_allow_unsorted* | Yes | Yes | Global, Session |
| *rocksdb_bulk_load_size* | Yes | Yes | Global |
| *rocksdb_bytes_per_sync* | Yes | Yes | Global |
| *rocksdb_cache_index_and_filter_blocks* | Yes | No | Global |
| *rocksdb_checksums_pct* | Yes | Yes | Global, Session |
| *rocksdb_collect_sst_properties* | Yes | No | Global |
| *rocksdb_commit_in_the_middle* | Yes | Yes | Global |
| *rocksdb_compact_cf* | Yes | Yes | Global |
| *rocksdb_compaction_readahead_size* | Yes | Yes | Global |
| *rocksdb_compaction_sequential_deletes* | Yes | Yes | Global |
| *rocksdb_compaction_sequential_deletes_count_sd* | Yes | Yes | Global |
| *rocksdb_compaction_sequential_deletes_file_size* | Yes | Yes | Global |
| *rocksdb_compaction_sequential_deletes_window* | Yes | Yes | Global |
| *rocksdb_concurrent_prepare* | Yes | No | Global |
| *rocksdb_create_checkpoint* | Yes | Yes | Global |
| *rocksdb_create_if_missing* | Yes | No | Global |
| *rocksdb_create_missing_column_families* | Yes | No | Global |
| *rocksdb_create_temporary_checkpoint* | Yes | Yes | Session |
| *rocksdb_datadir* | Yes | No | Global |
| *rocksdb_db_write_buffer_size* | Yes | No | Global |
| *rocksdb_deadlock_detect* | Yes | Yes | Global, Session |
| *rocksdb_deadlock_detect_depth* | Yes | Yes | Global, Session |
| *rocksdb_debug_optimizer_no_zero_cardinality* | Yes | Yes | Global, Session |

Continued on next page

Table 71.1 – continued from previous page

| Name | Command Line | Dynamic | Scope |
|---|---|---|---|
| *rocksdb_debug_ttl_ignore_pk* | Yes | Yes | Global |
| *rocksdb_debug_ttl_read_filter_ts* | Yes | Yes | Global |
| *rocksdb_debug_ttl_rec_ts* | Yes | Yes | Global |
| *rocksdb_debug_ttl_snapshot_ts* | Yes | Yes | Global |
| *rocksdb_default_cf_options* | Yes | No | Global |
| *rocksdb_delayed_write_rate* | Yes | Yes | Global |
| *rocksdb_delete_obsolete_files_period_micros* | Yes | No | Global |
| *rocksdb_disable_file_deletions* | Yes | Yes | Session |
| *rocksdb_enable_bulk_load_api* | Yes | No | Global |
| *rocksdb_enable_ttl* | Yes | No | Global |
| *rocksdb_enable_ttl_read_filtering* | Yes | Yes | Global |
| *rocksdb_enable_thread_tracking* | Yes | No | Global |
| *rocksdb_enable_write_thread_adaptive_yield* | Yes | No | Global |
| *rocksdb_error_if_exists* | Yes | No | Global |
| *rocksdb_flush_log_at_trx_commit* | Yes | Yes | Global, Session |
| *rocksdb_flush_memtable_on_analyze* | Yes | Yes | Global, Session |
| *rocksdb_force_compute_memtable_stats* | Yes | Yes | Global |
| *rocksdb_force_compute_memtable_stats_cachetime* | Yes | Yes | Global |
| *rocksdb_force_flush_memtable_and_lzero_now* | Yes | Yes | Global |
| *rocksdb_force_flush_memtable_now* | Yes | Yes | Global |
| *rocksdb_force_index_records_in_range* | Yes | Yes | Global, Session |
| *rocksdb_hash_index_allow_collision* | Yes | No | Global |
| *rocksdb_ignore_unknown_options* | Yes | No | Global |
| *rocksdb_index_type* | Yes | No | Global |
| *rocksdb_info_log_level* | Yes | Yes | Global |
| *rocksdb_is_fd_close_on_exec* | Yes | No | Global |
| *rocksdb_keep_log_file_num* | Yes | No | Global |
| *rocksdb_large_prefix* | Yes | Yes | Global |
| *rocksdb_lock_scanned_rows* | Yes | Yes | Global, Session |
| *rocksdb_lock_wait_timeout* | Yes | Yes | Global, Session |
| *rocksdb_log_file_time_to_roll* | Yes | No | Global |
| *rocksdb_manifest_preallocation_size* | Yes | No | Global |
| *rocksdb_manual_wal_flush* | Yes | No | Global |
| *rocksdb_max_background_compactions* | Yes | Yes | Global |
| *rocksdb_max_background_flushes* | Yes | No | Global |
| *rocksdb_max_background_jobs* | Yes | Yes | Global |
| *rocksdb_max_latest_deadlocks* | Yes | Yes | Global |
| *rocksdb_max_log_file_size* | Yes | No | Global |
| *rocksdb_max_manifest_file_size* | Yes | No | Global |
| *rocksdb_max_open_files* | Yes | No | Global |
| *rocksdb_max_row_locks* | Yes | Yes | Global, Session |
| *rocksdb_max_subcompactions* | Yes | No | Global |

Continued on next page

Table 71.1 – continued from previous page

| Name | Command Line | Dynamic | Scope |
|---|---|---|---|
| *rocksdb_max_total_wal_size* | Yes | No | Global |
| *rocksdb_merge_buf_size* | Yes | Yes | Global, Session |
| *rocksdb_merge_combine_read_size* | Yes | Yes | Global, Session |
| *rocksdb_merge_tmp_file_removal_delay_ms* | Yes | Yes | Global, Session |
| *rocksdb_new_table_reader_for_compaction_inputs* | Yes | No | Global |
| *rocksdb_no_block_cache* | Yes | No | Global |
| *rocksdb_no_create_column_family* | Yes | No | Global |
| *rocksdb_override_cf_options* | Yes | No | Global |
| *rocksdb_paranoid_checks* | Yes | No | Global |
| *rocksdb_pause_background_work* | Yes | Yes | Global |
| *rocksdb_perf_context_level* | Yes | Yes | Global, Session |
| *rocksdb_persistent_cache_path* | Yes | No | Global |
| *rocksdb_persistent_cache_size_mb* | Yes | No | Global, Session |
| *rocksdb_pin_l0_filter_and_index_blocks_in_cache* | Yes | No | Global |
| *rocksdb_print_snapshot_conflict_queries* | Yes | Yes | Global |
| *rocksdb_rate_limiter_bytes_per_sec* | Yes | Yes | Global |
| *rocksdb_read_free_rpl_tables* | Yes | Yes | Global, Session |
| *rocksdb_records_in_range* | Yes | Yes | Global, Session |
| *rocksdb_reset_stats* | Yes | Yes | Global |
| *rocksdb_rpl_skip_tx_api* | Yes | Yes | Global |
| *rocksdb_seconds_between_stat_computes* | Yes | Yes | Global |
| *rocksdb_signal_drop_index_thread* | Yes | Yes | Global |
| *rocksdb_sim_cache_size* | Yes | Yes | Global |
| *rocksdb_skip_bloom_filter_on_read* | Yes | Yes | Global, Session |
| *rocksdb_skip_fill_cache* | Yes | Yes | Global, Session |
| *rocksdb_sst_mgr_rate_bytes_per_sec* | Yes | No | Global |
| *rocksdb_stats_dump_period_sec* | Yes | No | Global |
| *rocksdb_store_row_debug_checksums* | Yes | Yes | Global, Session |
| *rocksdb_strict_collation_check* | Yes | Yes | Global |
| *rocksdb_strict_collation_exceptions* | Yes | Yes | Global |
| *rocksdb_table_cache_numshardbits* | Yes | No | Global |
| *rocksdb_table_stats_sampling_pct* | Yes | Yes | Global |
| *rocksdb_tmpdir* | Yes | Yes | Global, Session |
| *rocksdb_two_write_queues* | Yes | No | Global |
| *rocksdb_trace_sst_api* | Yes | Yes | Global, Session |

Continued on next page

Table 71.1 – continued from previous page

| Name | Command Line | Dynamic | Scope |
|---|---|---|---|
| *rocksdb_unsafe_for_binlog* | Yes | Yes | Global, Session |
| *rocksdb_update_cf_options* | Yes | Yes | Global |
| *rocksdb_use_adaptive_mutex* | Yes | No | Global |
| *rocksdb_use_direct_io_for_flush_and_compaction* | Yes | No | Global |
| *rocksdb_use_direct_reads* | Yes | No | Global |
| *rocksdb_use_fsync* | Yes | No | Global |
| *rocksdb_validate_tables* | Yes | No | Global |
| *rocksdb_verify_row_debug_checksums* | Yes | Yes | Global, Session |
| *rocksdb_wal_bytes_per_sync* | Yes | Yes | Global |
| *rocksdb_wal_dir* | Yes | No | Global |
| *rocksdb_wal_recovery_mode* | Yes | Yes | Global |
| *rocksdb_wal_size_limit_mb* | Yes | No | Global |
| *rocksdb_wal_ttl_seconds* | Yes | No | Global |
| *rocksdb_whole_key_filtering* | Yes | No | Global |
| *rocksdb_write_batch_max_bytes* | Yes | Yes | Global, Session |
| *rocksdb_write_disable_wal* | Yes | Yes | Global, Session |
| *rocksdb_write_ignore_missing_column_families* | Yes | Yes | Global, Session |

**variable `rocksdb_access_hint_on_compaction_start`**

**Command Line** `--rocksdb-access-hint-on-compaction-start`

**Dynamic** No

**Scope** Global

**Variable Type** String or Numeric

**Default Value** `NORMAL` or `1`

Specifies the file access pattern once a compaction is started, applied to all input files of a compaction. Possible values are:

- `0` = `NONE`
- `1` = `NORMAL` (default)
- `2` = `SEQUENTIAL`
- `3` = `WILLNEED`

**variable `rocksdb_advise_random_on_open`**

**Command Line** `--rocksdb-advise-random-on-open`

**Dynamic** No

**Scope** Global

**Variable Type** Boolean

**Default Value** `ON`

Specifies whether to hint the underlying file system that the file access pattern is random, when a data file is opened. Enabled by default.

**variable rocksdb_allow_concurrent_memtable_write**

**Command Line** `--rocksdb-allow-concurrent-memtable-write`

**Dynamic** No

**Scope** Global

**Variable Type** Boolean

**Default Value** `OFF`

Specifies whether to allow multiple writers to update memtables in parallel. Disabled by default.

**variable rocksdb_allow_to_start_after_corruption**

**Command Line** `--rocksdb_allow_to_start_after_corruption`

**Dynamic** No

**Scope** Global

**Variable Type** Boolean

**Default Value** `OFF`

Specifies whether to allow server to restart once MyRocks reported data corruption. Disabled by default.

Once corruption is detected server writes marker file (named ROCKSDB_CORRUPTED) in the data directory and aborts. If marker file exists, then mysqld exits on startup with an error message. The restart failure will continue until the problem is solved or until mysqld is started with this variable turned on in the command line.

---

**Note:** Not all memtables support concurrent writes.

---

**variable rocksdb_allow_mmap_reads**

**Command Line** `--rocksdb-allow-mmap-reads`

**Dynamic** No

**Scope** Global

**Variable Type** Boolean

**Default Value** `OFF`

Specifies whether to allow the OS to map a data file into memory for reads. Disabled by default. If you enable this, make sure that *rocksdb_use_direct_reads* is disabled.

**variable rocksdb_allow_mmap_writes**

**Command Line** `--rocksdb-allow-mmap-writes`

**Dynamic** No

**Scope** Global

**Variable Type** Boolean

**Default Value** `OFF`

Specifies whether to allow the OS to map a data file into memory for writes. Disabled by default.

**variable rocksdb_base_background_compactions**

**Command Line** `--rocksdb-base-background-compactions`

**Dynamic** No

**Scope** Global

**Variable Type** Numeric

**Default Value** `2`

Specifies the suggested number of concurrent background compaction jobs, submitted to the default LOW priority thread pool in RocksDB. Default is `1`. Allowed range of values is from `-1` to `64`. Maximum depends on the *`rocksdb_max_background_compactions`* variable. This variable was replaced with *`rocksdb_max_background_jobs`*, which automatically decides how many threads to allocate towards flush/compaction.

**variable `rocksdb_block_cache_size`**

**Command Line** `--rocksdb-block-cache-size`

**Dynamic** Yes

**Scope** Global

**Variable Type** Numeric

**Default Value** `536870912`

Specifies the size of the LRU block cache for RocksDB. This memory is reserved for the block cache, which is in addition to any filesystem caching that may occur.

Minimum value is `1024`, because that's the size of one block.

Default value is `536870912`.

Maximum value is `9223372036854775807`.

**variable `rocksdb_block_restart_interval`**

**Command Line** `--rocksdb-block-restart-interval`

**Dynamic** No

**Scope** Global

**Variable Type** Numeric

**Default Value** `16`

Specifies the number of keys for each set of delta encoded data. Default value is `16`. Allowed range is from `1` to `2147483647`.

**variable `rocksdb_block_size`**

**Command Line** `--rocksdb-block-size`

**Dynamic** No

**Scope** Global

**Variable Type** Numeric

**Default Value** 16 KB

Specifies the size of the data block for reading RocksDB data files. The default value is `16 KB`. The allowed range is from `1024` to `18446744073709551615` bytes.

**variable `rocksdb_block_size_deviation`**

**Command Line** `--rocksdb-block-size-deviation`

**Dynamic** No

**Scope** Global

**Variable Type** Numeric

**Default Value** `10`

Specifies the threshold for free space allowed in a data block (see *rocksdb_block_size*). If there is less space remaining, close the block (and write to new block). Default value is `10`, meaning that the block is not closed until there is less than 10 bits of free space remaining.

Allowed range is from `1` to `2147483647`.

**variable rocksdb_bulk_load_allow_unsorted**

**Command Line** `--rocksdb-bulk-load-allow-unsorted`

**Dynamic** Yes

**Scope** Global, Session

**Variable Type** Boolean

**Default Value** `OFF`

By default, the bulk loader requires its input to be sorted in the primary key order. If enabled, unsorted inputs are allowed too, which are then sorted by the bulkloader itself, at a performance penalty.

**variable rocksdb_bulk_load**

**Command Line** `--rocksdb-bulk-load`

**Dynamic** Yes

**Scope** Global, Session

**Variable Type** Boolean

**Default Value** `OFF`

Specifies whether to use bulk load: MyRocks will ignore checking keys for uniqueness or acquiring locks during transactions. Disabled by default. Enable this only if you are certain that there are no row conflicts, for example, when setting up a new MyRocks instance from a MySQL dump.

Enabling this variable will also enable the *rocksdb_commit_in_the_middle* variable.

**variable rocksdb_bulk_load_size**

**Command Line** `--rocksdb-bulk-load-size`

**Dynamic** Yes

**Scope** Global. Session

**Variable Type** Numeric

**Default Value** `1000`

Specifies the number of keys to accumulate before committing them to the storage engine when bulk load is enabled (see *rocksdb_bulk_load*). Default value is `1000`, which means that a batch can contain up to 1000 records before they are implicitly committed. Allowed range is from `1` to `1073741824`.

**variable rocksdb_bytes_per_sync**

**Command Line** `--rocksdb-bytes-per-sync`

**Dynamic** Yes

**Scope** Global

**Variable Type** Numeric

**Default Value** `0`

Specifies how often should the OS sync files to disk as they are being written, asynchronously, in the background. This operation can be used to smooth out write I/O over time. Default value is `0` meaning that files are never synced. Allowed range is up to `18446744073709551615`.

**variable `rocksdb_cache_index_and_filter_blocks`**

**Command Line** `--rocksdb-cache-index-and-filter-blocks`

**Dynamic** No

**Scope** Global

**Variable Type** Boolean

**Default Value** `ON`

Specifies whether RocksDB should use the block cache for caching the index and bloomfilter data blocks from each data file. Enabled by default. If you disable this feature, RocksDB will allocate additional memory to maintain these data blocks.

**variable `rocksdb_checksums_pct`**

**Command Line** `--rocksdb-checksums-pct`

**Dynamic** Yes

**Scope** Global, Session

**Variable Type** Numeric

**Default Value** `100`

Specifies the percentage of rows to be checksummed. Default value is `100` (checksum all rows). Allowed range is from `0` to `100`.

**variable `rocksdb_collect_sst_properties`**

**Command Line** `--rocksdb-collect-sst-properties`

**Dynamic** No

**Scope** Global

**Variable Type** Boolean

**Default Value** `ON`

Specifies whether to collect statistics on each data file to improve optimizer behavior. Enabled by default.

**variable `rocksdb_commit_in_the_middle`**

**Command Line** `--rocksdb-commit-in-the-middle`

**Dynamic** Yes

**Scope** Global

**Variable Type** Boolean

**Default Value** `OFF`

Specifies whether to commit rows implicitly when a batch contains more than the value of *rocksdb_bulk_load_size*. This is disabled by default and will be enabled if *rocksdb_bulk_load* is enabled.

**variable rocksdb_compact_cf**

**Command Line** `--rocksdb-compact-cf`

**Dynamic** Yes

**Scope** Global

**Variable Type** String

**Default Value**

Specifies the name of the column family to compact.

**variable rocksdb_compaction_readahead_size**

**Command Line** `--rocksdb-compaction-readahead-size`

**Dynamic** Yes

**Scope** Global

**Variable Type** Numeric

**Default Value** `0`

Specifies the size of reads to perform ahead of compaction. Default value is `0`. Set this to at least 2 megabytes (`16777216`) when using MyRocks with spinning disks to ensure sequential reads instead of random. Maximum allowed value is `18446744073709551615`.

---

**Note:** If you set this variable to a non-zero value, *rocksdb_new_table_reader_for_compaction_inputs* is enabled.

---

**variable rocksdb_compaction_sequential_deletes**

**Command Line** `--rocksdb-compaction-sequential-deletes`

**Dynamic** Yes

**Scope** Global

**Variable Type** Numeric

**Default Value** `0`

Specifies the threshold to trigger compaction on a file if it has more than this number of sequential delete markers. Default value is `0` meaning that compaction is not triggered regardless of the number of delete markers. Maximum allowed value is `2000000` (two million delete markers).

---

**Note:** Depending on workload patterns, MyRocks can potentially maintain large numbers of delete markers, which increases latency of queries. This compaction feature will reduce latency, but may also increase the MyRocks write rate. Use this variable together with *rocksdb_compaction_sequential_deletes_file_size* to only perform compaction on large files.

---

**variable rocksdb_compaction_sequential_deletes_count_sd**

**Command Line** `--rocksdb-compaction-sequential-deletes-count-sd`

**Dynamic** Yes

**Scope** Global

**Variable Type** Boolean

**Default Value** `OFF`

Specifies whether to count single deletes as delete markers recognized by *`rocksdb_compaction_sequential_deletes`*. Disabled by default.

**variable `rocksdb_compaction_sequential_deletes_file_size`**

**Command Line** `--rocksdb-compaction-sequential-deletes-file-size`

**Dynamic** Yes

**Scope** Global

**Variable Type** Numeric

**Default Value** `0`

Specifies the minimum file size required to trigger compaction on it by *`rocksdb_compaction_sequential_deletes`*. Default value is `0`, meaning that compaction is triggered regardless of file size. Allowed range is from `-1` to `9223372036854775807`.

**variable `rocksdb_compaction_sequential_deletes_window`**

**Command Line** `--rocksdb-compaction-sequential-deletes-window`

**Dynamic** Yes

**Scope** Global

**Variable Type** Numeric

**Default Value** `0`

Specifies the size of the window for counting delete markers by *`rocksdb_compaction_sequential_deletes`*. Default value is `0`. Allowed range is up to `2000000` (two million).

**variable `rocksdb_concurrent_prepare`**

**Command Line** `--rocksdb-concurrent_prepare`

**Dynamic** No

**Scope** Global

**Variable Type** Boolean

**Default Value** `ON`

When enabled this variable allows/encourages threads that are using two-phase commit to `prepare` in parallel. This variable was renamed in upstream to *`rocksdb_two_write_queues`*.

**variable `rocksdb_create_checkpoint`**

**Command Line** `--rocksdb-create-checkpoint`

**Dynamic** Yes

**Scope** Global

**Variable Type** String

**Default Value**

Specifies the directory where MyRocks should create a checkpoint. Empty by default.

**variable `rocksdb_create_if_missing`**

**Command Line** `--rocksdb-create-if-missing`

**Dynamic** No

**Scope** Global

**Variable Type** Boolean

**Default Value** `ON`

Specifies whether MyRocks should create its database if it does not exist. Enabled by default.

**variable `rocksdb_create_missing_column_families`**

**Command Line** `--rocksdb-create-missing-column-families`

**Dynamic** No

**Scope** Global

**Variable Type** Boolean

**Default Value** `OFF`

Specifies whether MyRocks should create new column families if they do not exist. Disabled by default.

**variable `rocksdb_create_temporary_checkpoint`**

**Command Line** `--rocksdb-create-temporary-checkpoint`

**Dynamic** Yes

**Scope** Session

**Variable Type** String

This variable has been implemented in *Percona Server for MySQL `8.0.15-6`*. When specified it will create a temporary RocksDB 'checkpoint' or 'snapshot' in the datadir. If the session ends with an existing checkpoint, or if the variable is reset to another value, the checkpoint will get removed. This variable should be used by backup tools. Prolonged use or other misuse can have serious side effects to the server instance.

**variable `rocksdb_datadir`**

**Command Line** `--rocksdb-datadir`

**Dynamic** No

**Scope** Global

**Variable Type** String

**Default Value** `./.rocksdb`

Specifies the location of the MyRocks data directory. By default, it is created in the current working directory.

**variable `rocksdb_db_write_buffer_size`**

**Command Line** `--rocksdb-db-write-buffer-size`

**Dynamic** No

**Scope** Global

**Variable Type** Numeric

**Default Value** `0`

Specifies the size of the memtable used to store writes in MyRocks. This is the size per column family. When this size is reached, the memtable is flushed to persistent media. Default value is `0`. Allowed range is up to `18446744073709551615`.

**variable rocksdb_deadlock_detect**

**Command Line** `--rocksdb-deadlock-detect`

**Dynamic** Yes

**Scope** Global, Session

**Variable Type** Boolean

**Default Value** `OFF`

Specifies whether MyRocks should detect deadlocks. Disabled by default.

**variable rocksdb_deadlock_detect_depth**

**Command Line** `--rocksdb-deadlock-detect-depth`

**Dynamic** Yes

**Scope** Global, Session

**Variable Type** Numeric

**Default Value** `50`

Specifies the number of transactions deadlock detection will traverse through before assuming deadlock.

**variable rocksdb_debug_optimizer_no_zero_cardinality**

**Command Line** `--rocksdb-debug-optimizer-no-zero-cardinality`

**Dynamic** Yes

**Scope** Global

**Variable Type** Boolean

**Default Value** `ON`

Specifies whether MyRocks should prevent zero cardinality by always overriding it with some value.

**variable rocksdb_debug_ttl_ignore_pk**

**Command Line** `--rocksdb-debug-ttl-ignore-pk`

**Dynamic** Yes

**Scope** Global

**Variable Type** Boolean

**Default Value** `OFF`

For debugging purposes only. If true, compaction filtering will not occur on Primary Key TTL data. This variable is a no-op in non-debug builds.

**variable rocksdb_debug_ttl_read_filter_ts**

**Command Line** `--rocksdb_debug-ttl-read-filter-ts`

**Dynamic** Yes

**Scope** Global

**Variable Type** Numeric

**Default Value** `0`

For debugging purposes only. Overrides the TTL read filtering time to time + `debug_ttl_read_filter_ts`. A value of `0` denotes that the variable is not set. This variable is a no-op in non-debug builds.

**variable `rocksdb_debug_ttl_rec_ts`**

**Command Line** `--rocksdb-debug-ttl-rec-ts`

**Dynamic** Yes

**Scope** Global

**Variable Type** Numeric

**Default Value** `0`

For debugging purposes only. Overrides the TTL of records to `now()` + `debug_ttl_rec_ts`. The value can be +/- to simulate a record inserted in the past vs a record inserted in the "future". A value of `0` denotes that the variable is not set. This variable is a no-op in non-debug builds.

**variable `rocksdb_debug_ttl_snapshot_ts`**

**Command Line** `--rocksdb_debug_ttl_ignore_pk`

**Dynamic** Yes

**Scope** Global

**Variable Type** Numeric

**Default Value** `0`

For debugging purposes only. Sets the snapshot during compaction to `now()` + `rocksdb_debug_set_ttl_snapshot_ts`. The value can be +/- to simulate a snapshot in the past vs a snapshot created in the "future". A value of `0` denotes that the variable is not set. This variable is a no-op in non-debug builds.

**variable `rocksdb_default_cf_options`**

**Command Line** `--rocksdb-default-cf-options`

**Dynamic** No

**Scope** Global

**Variable Type** String

**Default Value**

```
block_based_table_factory= { cache_index_and_filter_blocks=1;
      filter_policy=bloomfilter:10:false;
      whole_key_filtering=1};
level_compaction_dynamic_level_bytes=true;
optimize_filters_for_hits=true;
compaction_pri=kMinOverlappingRatio;
compression=kLZ4Compression;
bottommost_compression=kLZ4Compression;
```

Specifies the default column family options for *MyRocks*. On startup, the server applies this option to all existing column families. This option is read-only at runtime.

**variable `rocksdb_delayed_write_rate`**

**Command Line** `--rocksdb-delayed-write-rate`

**Dynamic** Yes

**Scope** Global

**Variable Type** Numeric

**Default Value** `16777216`

Specifies the write rate in bytes per second, which should be used if MyRocks hits a soft limit or threshold for writes. Default value is `16777216` (16 MB/sec). Allowed range is from `0` to `18446744073709551615`.

**variable `rocksdb_delete_obsolete_files_period_micros`**

**Command Line** `--rocksdb-delete-obsolete-files-period-micros`

**Dynamic** No

**Scope** Global

**Variable Type** Numeric

**Default Value** `21600000000`

Specifies the period in microseconds to delete obsolete files regardless of files removed during compaction. Default value is `21600000000` (6 hours). Allowed range is up to `9223372036854775807`.

**variable `rocksdb_disable_file_deletions`**

**Command Line** `--rocksdb-disable-file-deletions`

**Dynamic** Yes

**Scope** Session

**Variable Type** Boolean

**Default Value** `OFF`

This variable has been implemented in *Percona Server for MySQL `8.0.15-6`*. It allows a client to temporarily disable RocksDB deletion of old `WAL` and `.sst` files for the purposes of making a consistent backup. If the client session terminates for any reason after disabling deletions and has not re-enabled deletions, they will be explicitly re-enabled. This variable should be used by backup tools. Prolonged use or other misuse can have serious side effects to the server instance.

**variable `rocksdb_enable_bulk_load_api`**

**Command Line** `--rocksdb-enable-bulk-load-api`

**Dynamic** No

**Scope** Global

**Variable Type** Boolean

**Default Value** `ON`

Specifies whether to use the `SSTFileWriter` feature for bulk loading, This feature bypasses the memtable, but requires keys to be inserted into the table in either ascending or descending order. Enabled by default. If disabled, bulk loading uses the normal write path via the memtable and does not require keys to be inserted in any order.

**variable `rocksdb_enable_ttl`**

**Command Line** `--rocksdb-enable-ttl`

**Dynamic** No

**Scope** Global

**Variable Type** Boolean

**Default Value** `ON`

Specifies whether to keep expired TTL records during compaction. Enabled by default. If disabled, expired TTL records will be dropped during compaction.

**variable rocksdb_enable_ttl_read_filtering**

**Command Line** `--rocksdb-enable-ttl-read-filtering`

**Dynamic** Yes

**Scope** Global

**Variable Type** Boolean

**Default Value** `ON`

For tables with TTL, expired records are skipped/filtered out during processing and in query results. Disabling this will allow these records to be seen, but as a result rows may disappear in the middle of transactions as they are dropped during compaction. **Use with caution.**

**variable rocksdb_enable_thread_tracking**

**Command Line** `--rocksdb-enable-thread-tracking`

**Dynamic** No

**Scope** Global

**Variable Type** Boolean

**Default Value** `OFF`

Specifies whether to enable tracking the status of threads accessing the database. Disabled by default. If enabled, thread status will be available via `GetThreadList()`.

**variable rocksdb_enable_write_thread_adaptive_yield**

**Command Line** `--rocksdb-enable-write-thread-adaptive-yield`

**Dynamic** No

**Scope** Global

**Variable Type** Boolean

**Default Value** `OFF`

Specifies whether the MyRocks write batch group leader should wait up to the maximum allowed time before blocking on a mutex. Disabled by default. Enable it to increase throughput for concurrent workloads.

**variable rocksdb_error_if_exists**

**Command Line** `--rocksdb-error-if-exists`

**Dynamic** No

**Scope** Global

**Variable Type** Boolean

**Default Value** `OFF`

Specifies whether to report an error when a database already exists. Disabled by default.

**variable rocksdb_flush_log_at_trx_commit**

**Command Line** `--rocksdb-flush-log-at-trx-commit`

**Dynamic** Yes

**Scope** Global, Session

**Variable Type** Numeric

**Default Value** `1`

Specifies whether to sync on every transaction commit, similar to `innodb_flush_log_at_trx_commit`. Enabled by default, which ensures ACID compliance.

Possible values:

- `0`: Do not sync on transaction commit. This provides better performance, but may lead to data inconsistency in case of a crash.
- `1`: Sync on every transaction commit. This is set by default and recommended as it ensures data consistency, but reduces performance.
- `2`: Sync every second.

**variable `rocksdb_flush_memtable_on_analyze`**

**Command Line** `--rocksdb-flush-memtable-on-analyze`

**Dynamic** Yes

**Scope** Global, Session

**Variable Type** Boolean

**Default Value** `ON`

Specifies whether to flush the memtable when running `ANALYZE` on a table. Enabled by default. This ensures accurate cardinality by including data in the memtable for calculating stats.

**variable `rocksdb_force_compute_memtable_stats`**

**Command Line** `--rocksdb-force-compute-memtable-stats`

**Dynamic** Yes

**Scope** Global

**Variable Type** Boolean

**Default Value** `ON`

Specifies whether data in the memtables should be included for calculating index statistics used by the query optimizer. Enabled by default. This provides better accuracy, but may reduce performance.

**variable `rocksdb_force_compute_memtable_stats_cachetime`**

**Command Line** `--rocksdb-force-compute-memtable-stats-cachetime`

**Dynamic** Yes

**Scope** Global

**Variable Type** Numeric

**Default Value** 60000000

Specifies for how long the cached value of memtable statistics should be used instead of computing it every time during the query plan analysis.

**variable `rocksdb_force_flush_memtable_and_lzero_now`**

**Command Line** `--rocksdb-force-flush-memtable-and-lzero-now`

**Dynamic** Yes

**Scope** Global

**Variable Type** Boolean

**Default Value** `OFF`

Works similar to `force_flush_memtable_now` but also flushes all L0 files.

**variable `rocksdb_force_flush_memtable_now`**

**Command Line** `--rocksdb-force-flush-memtable-now`

**Dynamic** Yes

**Scope** Global

**Variable Type** Boolean

**Default Value** `OFF`

Forces MyRocks to immediately flush all memtables out to data files.

**Warning:** Use with caution! Write requests will be blocked until all memtables are flushed.

**variable `rocksdb_force_index_records_in_range`**

**Command Line** `--rocksdb-force-index-records-in-range`

**Dynamic** Yes

**Scope** Global, Session

**Variable Type** Numeric

**Default Value** `1`

Specifies the value used to override the number of rows returned to query optimizer when `FORCE INDEX` is used. Default value is `1`. Allowed range is from `0` to `2147483647`. Set to `0` if you do not want to override the returned value.

**variable `rocksdb_hash_index_allow_collision`**

**Command Line** `--rocksdb-hash-index-allow-collision`

**Dynamic** No

**Scope** Global

**Variable Type** Boolean

**Default Value** `ON`

Specifies whether hash collisions are allowed. Enabled by default, which uses less memory. If disabled, full prefix is stored to prevent hash collisions.

**variable `rocksdb_ignore_unknown_options`**

**Command Line** `--rocksdb-ignore-unknown-options`

**Dynamic** No

**Scope** Global

**Variable Type** Boolean

**Default Value** `ON`

When enabled, it allows RocksDB to receive unknown options and not exit.

**variable `rocksdb_index_type`**

**Command Line** `--rocksdb-index-type`

**Dynamic** No

**Scope** Global

**Variable Type** Enum

**Default Value** `kBinarySearch`

Specifies the type of indexing used by MyRocks:

- `kBinarySearch`: Binary search (default).
- `kHashSearch`: Hash search.

**variable `rocksdb_info_log_level`**

**Command Line** `--rocksdb-info-log-level`

**Dynamic** Yes

**Scope** Global

**Variable Type** Enum

**Default Value** `error_level`

Specifies the level for filtering messages written by MyRocks to the `mysqld` log.

- `debug_level`: Maximum logging (everything including debugging log messages)
- `info_level`
- `warn_level`
- `error_level` (default)
- `fatal_level`: Minimum logging (only fatal error messages logged)

**variable `rocksdb_is_fd_close_on_exec`**

**Command Line** `--rocksdb-is-fd-close-on-exec`

**Dynamic** No

**Scope** Global

**Variable Type** Boolean

**Default Value** `ON`

Specifies whether child processes should inherit open file jandles. Enabled by default.

**variable `rocksdb_large_prefix`**

**Command Line** `--rocksdb-large-prefix`

**Dynamic** Yes

**Scope** Global

**Variable Type** Boolean

**Default Value** `TRUE`

When enabled, this option allows index key prefixes longer than 767 bytes (up to 3072 bytes). This option mirrors the innodb_large_prefix The values for *`rocksdb_large_prefix`* should be the same between master and slave.

**Note:** In version 8.0.16-7 and later, the default value is changed to `TRUE`.

**variable `rocksdb_keep_log_file_num`**

**Command Line** `--rocksdb-keep-log-file-num`

**Dynamic** No

**Scope** Global

**Variable Type** Numeric

**Default Value** `1000`

Specifies the maximum number of info log files to keep. Default value is `1000`. Allowed range is from `1` to `18446744073709551615`.

**variable `rocksdb_lock_scanned_rows`**

**Command Line** `--rocksdb-lock-scanned-rows`

**Dynamic** Yes

**Scope** Global, Session

**Variable Type** Boolean

**Default Value** `OFF`

Specifies whether to hold the lock on rows that are scanned during `UPDATE` and not actually updated. Disabled by default.

**variable `rocksdb_lock_wait_timeout`**

**Command Line** `--rocksdb-lock-wait-timeout`

**Dynamic** Yes

**Scope** Global, Session

**Variable Type** Numeric

**Default Value** `1`

Specifies the number of seconds MyRocks should wait to acquire a row lock before aborting the request. Default value is `1`. Allowed range is up to `1073741824`.

**variable `rocksdb_log_file_time_to_roll`**

**Command Line** `--rocksdb-log-file-time-to-roll`

**Dynamic** No

**Scope** Global

**Variable Type** Numeric

**Default Value** `0`

Specifies the period (in seconds) for rotating the info log files. Default value is `0`, meaning that the log file is not rotated. Allowed range is up to `18446744073709551615`.

**variable `rocksdb_manifest_preallocation_size`**

**Command Line** `--rocksdb-manifest-preallocation-size`

**Dynamic** No

**Scope** Global

**Variable Type** Numeric

**Default Value** `0`

Specifies the number of bytes to preallocate for the MANIFEST file used by MyRocks to store information about column families, levels, active files, etc. Default value is `0`. Allowed range is up to `18446744073709551615`.

---

**Note:** A value of `4194304` (4 MB) is reasonable to reduce random I/O on XFS.

---

**variable `rocksdb_manual_wal_flush`**

**Command Line** `--rocksdb-manual-wal-flush`

**Dynamic** No

**Scope** Global

**Variable Type** Boolean

**Default Value** `ON`

This variable can be used to disable automatic/timed WAL flushing and instead rely on the application to do the flushing.

**variable `rocksdb_max_background_compactions`**

**Command Line** `--rocksdb-max-background-compactions`

**Dynamic** Yes

**Scope** Global

**Variable Type** Numeric

**Default Value** `1`

Specifies the maximum number of concurrent background compaction threads, submitted to the low-priority thread pool. Default value is `1`. Allowed range is up to `64`. This variable has been replaced with *`rocksdb_max_background_jobs`*, which automatically decides how many threads to allocate towards flush/compaction.

Replaced with *`rocksdb_max_background_jobs`*

**variable `rocksdb_max_background_flushes`**

**Command Line** `--rocksdb-max-background-flushes`

**Dynamic** No

**Scope** Global

**Variable Type** Numeric

**Default Value** `1`

Specifies the maximum number of concurrent background memtable flush threads, submitted to the high-priority thread-pool. Default value is `1`. Allowed range is up to `64`. This variable has been replaced with *`rocksdb_max_background_jobs`*, which automatically decides how many threads to allocate towards flush/compaction.

Replaced with *`rocksdb_max_background_jobs`*

**variable `rocksdb_max_background_jobs`**

**Command Line** `--rocksdb-max-background-jobs`

**Dynamic** Yes

**Scope** Global

**Variable Type** Numeric

**Default Value** `2`

This variable replaced *rocksdb_base_background_compactions*, *rocksdb_max_background_compactions*, and *rocksdb_max_background_flushes* variables. This variable specifies the maximum number of background jobs. It automatically decides how many threads to allocate towards flush/compaction. It was implemented to reduce the number of (confusing) options users and can tweak and push the responsibility down to RocksDB level.

**variable `rocksdb_max_latest_deadlocks`**

**Command Line** `--rocksdb-max-latest-deadlocks`

**Dynamic** Yes

**Scope** Global

**Variable Type** Numeric

**Default Value** `5`

Specifies the maximum number of recent deadlocks to store.

**variable `rocksdb_max_log_file_size`**

**Command Line** `--rocksdb-max-log-file-size`

**Dynamic** No

**Scope** Global

**Variable Type** Numeric

**Default Value** `0`

Specifies the maximum size for info log files, after which the log is rotated. Default value is `0`, meaning that only one log file is used. Allowed range is up to `18446744073709551615`.

Also see *rocksdb_log_file_time_to_roll*.

**variable `rocksdb_max_manifest_file_size`**

**Command Line** `--rocksdb-manifest-log-file-size`

**Dynamic** No

**Scope** Global

**Variable Type** Numeric

**Default Value** `18446744073709551615`

Specifies the maximum size of the MANIFEST data file, after which it is rotated. Default value is also the maximum, making it practically unlimited: only one manifest file is used.

**variable `rocksdb_max_open_files`**

**Command Line** `--rocksdb-max-open-files`

**Dynamic** No

**Scope** Global

**Variable Type** Numeric

**Default Value** 1000

Specifies the maximum number of file handles opened by MyRocks. Values in the range between 0 and open_files_limit are taken as they are. If *rocksdb_max_open_files* value is greater than open_files_limit, it will be reset to 1/2 of open_files_limit, and a warning will be emitted to the mysqld error log. A value of -2 denotes auto tuning: just sets *rocksdb_max_open_files* value to 1/2 of open_files_limit. Finally, -1 means no limit, i.e. an infinite number of file handles.

**Warning:** Setting *rocksdb_max_open_files* to -1 is dangerous, as server may quickly run out of file handles in this case.

**variable rocksdb_max_row_locks**

**Command Line** --rocksdb-max-row-locks

**Dynamic** Yes

**Scope** Global, Session

**Variable Type** Numeric

**Default Value** 1048576

Specifies the limit on the maximum number of row locks a transaction can have before it fails. Default value is also the maximum, making it practically unlimited: transactions never fail due to row locks.

**variable rocksdb_max_subcompactions**

**Command Line** --rocksdb-max-subcompactions

**Dynamic** No

**Scope** Global

**Variable Type** Numeric

**Default Value** 1

Specifies the maximum number of threads allowed for each compaction job. Default value of 1 means no subcompactions (one thread per compaction job). Allowed range is up to 64.

**variable rocksdb_max_total_wal_size**

**Command Line** --rocksdb-max-total-wal-size

**Dynamic** No

**Scope** Global

**Variable Type** Numeric

**Default Value** 2 GB

Specifies the maximum total size of WAL (write-ahead log) files, after which memtables are flushed. Default value is 2 GB The allowed range is up to 9223372036854775807.

**variable rocksdb_merge_buf_size**

**Command Line** --rocksdb-merge-buf-size

**Dynamic** Yes

**Scope** Global, Session

**Variable Type** Numeric

**Default Value** 67108864

Specifies the size (in bytes) of the merge-sort buffers used to accumulate data during secondary key creation. New entries are written directly to the lowest level in the database, instead of updating indexes through the memtable and L0. These values are sorted using merge-sort, with buffers set to 64 MB by default (67108864). Allowed range is from 100 to 18446744073709551615.

**variable rocksdb_merge_combine_read_size**

**Command Line** --rocksdb-merge-combine-read-size

**Dynamic** Yes

**Scope** Global, Session

**Variable Type** Numeric

**Default Value** 1073741824

Specifies the size (in bytes) of the merge-combine buffer used for the merge-sort algorithm as described in *rocksdb_merge_buf_size*. Default size is 1 GB (1073741824). Allowed range is from 100 to 18446744073709551615.

**variable rocksdb_merge_tmp_file_removal_delay_ms**

**Command Line** --rocksdb_merge_tmp_file_removal_delay_ms

**Dynamic** Yes

**Scope** Global, Session

**Variable Type** Numeric

**Default Value** 0

Fast secondary index creation creates merge files when needed. After finishing secondary index creation, merge files are removed. By default, the file removal is done without any sleep, so removing GBs of merge files within <1s may happen, which will cause trim stalls on Flash. This variable can be used to rate limit the delay in milliseconds.

**variable rocksdb_new_table_reader_for_compaction_inputs**

**Command Line** --rocksdb-new-table-reader-for-compaction-inputs

**Dynamic** No

**Scope** Global

**Variable Type** Boolean

**Default Value** OFF

Specifies whether MyRocks should create a new file descriptor and table reader for each compaction input. Disabled by default. Enabling this may increase memory consumption, but will also allow pre-fetch options to be specified for compaction input files without impacting table readers used for user queries.

**variable rocksdb_no_block_cache**

**Command Line** --rocksdb-no-block-cache

**Dynamic** No

**Scope** Global

**Variable Type** Boolean

**Default Value** OFF

Specifies whether to disable the block cache for column families. Variable is disabled by default, meaning that using the block cache is allowed.

**variable rocksdb_no_create_column_family**

**Command Line** `--rocksdb-no-create-column-family`

**Dynamic** No

**Scope** Global

**Variable Type** Boolean

**Default Value** `ON`

Controls the processing of the column family name given in the `COMMENT` clause in the `CREATE TABLE` or `ALTER TABLE` statement in case the column family name does not refer to an existing column family.

If *rocksdb_no_create_column_family* is set to *NO*, a new column family will be created and the new index will be placed into it.

If *rocksdb_no_create_column_family* is set to *YES*, no new column family will be created and the index will be placed into the *default* column family. A warning is issued in this case informing that the specified column family does not exist and cannot be created.

**See also:**

**More information about column families** *MyRocks Column Families*

**variable rocksdb_override_cf_options**

**Command Line** `--rocksdb-override-cf-options`

**Dynamic** No

**Scope** Global

**Variable Type** String

**Default Value**

Specifies option overrides for each column family. Empty by default.

**variable rocksdb_paranoid_checks**

**Command Line** `--rocksdb-paranoid-checks`

**Dynamic** No

**Scope** Global

**Variable Type** Boolean

**Default Value** `ON`

Specifies whether MyRocks should re-read the data file as soon as it is created to verify correctness. Enabled by default.

**variable rocksdb_pause_background_work**

**Command Line** `--rocksdb-pause-background-work`

**Dynamic** Yes

**Scope** Global

**Variable Type** Boolean

**Default Value** `OFF`

Specifies whether MyRocks should pause all background operations. Disabled by default. There is no practical reason for a user to ever use this variable because it is intended as a test synchronization tool for the MyRocks MTR test suites.

**Warning:** If someone were to set a *rocksdb_force_flush_memtable_now* to 1 while *rocksdb_pause_background_work* is set to 1, the client that issued the rocksdb_force_flush_memtable_now=1 will be blocked indefinitely until *rocksdb_pause_background_work* is set to 0.

**variable rocksdb_perf_context_level**

**Command Line** --rocksdb-perf-context-level

**Dynamic** Yes

**Scope** Global, Session

**Variable Type** Numeric

**Default Value** 0

Specifies the level of information to capture with the Perf Context plugins. Default value is 0. Allowed range is up to 4.

**variable rocksdb_persistent_cache_path**

**Command Line** --rocksdb-persistent-cache-path

**Dynamic** No

**Scope** Global

**Variable Type** String

**Default Value**

Specifies the path to the persistent cache. Set this together with *rocksdb_persistent_cache_size_mb*.

**variable rocksdb_persistent_cache_size_mb**

**Command Line** --rocksdb-persistent-cache-size-mb

**Dynamic** No

**Scope** Global

**Variable Type** Numeric

**Default Value** 0

Specifies the size of the persisten cache in megabytes. Default is 0 (persistent cache disabled). Allowed range is up to 18446744073709551615. Set this together with *rocksdb_persistent_cache_path*.

**variable rocksdb_pin_l0_filter_and_index_blocks_in_cache**

**Command Line** --rocksdb-pin-l0-filter-and-index-blocks-in-cache

**Dynamic** No

**Scope** Global

**Variable Type** Boolean

**Default Value** ON

Specifies whether MyRocks pins the filter and index blocks in the cache if *rocksdb_cache_index_and_filter_blocks* is enabled. Enabled by default.

**variable rocksdb_print_snapshot_conflict_queries**

**Command Line** `--rocksdb-print-snapshot-conflict-queries`

**Dynamic** Yes

**Scope** Global

**Variable Type** Boolean

**Default Value** `OFF`

Specifies whether queries that generate snapshot conflicts should be logged to the error log. Disabled by default.

**variable rocksdb_rate_limiter_bytes_per_sec**

**Command Line** `--rocksdb-rate-limiter-bytes-per-sec`

**Dynamic** Yes

**Scope** Global

**Variable Type** Numeric

**Default Value** `0`

Specifies the maximum rate at which MyRocks can write to media via memtable flushes and compaction. Default value is `0` (write rate is not limited). Allowed range is up to `9223372036854775807`.

**variable rocksdb_read_free_rpl_tables**

**Command Line** `--rocksdb-read-free-rpl-tables`

**Dynamic** Yes

**Scope** Global, Session

**Variable Type** String

**Default Value**

Lists tables (as a regular expression) that should use read-free replication on the slave (that is, replication without row lookups). Empty by default.

**variable rocksdb_records_in_range**

**Command Line** `--rocksdb-records-in-range`

**Dynamic** Yes

**Scope** Global, Session

**Variable Type** Numeric

**Default Value** `0`

Specifies the value to override the result of `records_in_range()`. Default value is `0`. Allowed range is up to `2147483647`.

**variable rocksdb_reset_stats**

**Command Line** `--rocksdb-reset-stats`

**Dynamic** Yes

**Scope** Global

**Variable Type** Boolean

**Default Value** `OFF`

Resets MyRocks internal statistics dynamically (without restarting the server).

**variable `rocksdb_rpl_skip_tx_api`**

**Command Line** `--rocksdb-rpl-skip-tx-api`

**Dynamic** No

**Scope** Global

**Variable Type** Boolean

**Default Value** `OFF`

Specifies whether write batches should be used for replication thread instead of the transaction API. Disabled by default.

There are two conditions which are necessary to use it: row replication format and slave operating in super read only mode.

**variable `rocksdb_seconds_between_stat_computes`**

**Command Line** `--rocksdb-seconds-between-stat-computes`

**Dynamic** Yes

**Scope** Global

**Variable Type** Numeric

**Default Value** `3600`

Specifies the number of seconds to wait between recomputation of table statistics for the optimizer. During that time, only changed indexes are updated. Default value is `3600`. Allowed is from `0` to `4294967295`.

**variable `rocksdb_signal_drop_index_thread`**

**Command Line** `--rocksdb-signal-drop-index-thread`

**Dynamic** Yes

**Scope** Global

**Variable Type** Boolean

**Default Value** `OFF`

Signals the MyRocks drop index thread to wake up.

**variable `rocksdb_sim_cache_size`**

**Command Line** `--rocksdb-sim-cache-size`

**Dynamic** No

**Scope** Global

**Variable Type** Numeric

**Default Value** `0`

Enables the simulated cache, which allows us to figure out the hit/miss rate with a specific cache size without changing the real block cache.

**variable `rocksdb_skip_bloom_filter_on_read`**

**Command Line** `--rocksdb-skip-bloom-filter-on_read`

**Dynamic** Yes

**Scope** Global, Session

**Variable Type** Boolean

**Default Value** `OFF`

Specifies whether bloom filters should be skipped on reads. Disabled by default (bloom filters are not skipped).

**variable `rocksdb_skip_fill_cache`**

**Command Line** `--rocksdb-skip-fill-cache`

**Dynamic** Yes

**Scope** Global, Session

**Variable Type** Boolean

**Default Value** `OFF`

Specifies whether to skip caching data on read requests. Disabled by default (caching is not skipped).

**variable `rocksdb_sst_mgr_rate_bytes_per_sec`**

**Command Line** `--rocksdb-sst-mgr-rate-bytes-per-sec`

**Dynamic** Yes

**Scope** Global, Session

**Variable Type** Numeric

**Default Value** `0`

Specifies the maximum rate for writing to data files. Default value is `0`. This option is not effective on HDD. Allowed range is from `0` to `18446744073709551615`.

**variable `rocksdb_stats_dump_period_sec`**

**Command Line** `--rocksdb-stats-dump-period-sec`

**Dynamic** No

**Scope** Global

**Variable Type** Numeric

**Default Value** `600`

Specifies the period in seconds for performing a dump of the MyRocks statistics to the info log. Default value is `600`. Allowed range is up to `2147483647`.

**variable `rocksdb_store_row_debug_checksums`**

**Command Line** `--rocksdb-store-row-debug-checksums`

**Dynamic** Yes

**Scope** Global, Session

**Variable Type** Boolean

**Default Value** `OFF`

Specifies whether to include checksums when writing index or table records. Disabled by default.

**variable `rocksdb_strict_collation_check`**

**Command Line** `--rocksdb-strict-collation-check`

**Dynamic** Yes

**Scope** Global

**Variable Type** Boolean

**Default Value** `ON`

Specifies whether to check and verify that table indexes have proper collation settings. Enabled by default.

**variable `rocksdb_strict_collation_exceptions`**

**Command Line** `--rocksdb-strict-collation-exceptions`

**Dynamic** Yes

**Scope** Global

**Variable Type** String

**Default Value**

Lists tables (as a regular expression) that should be excluded from verifying case-sensitive collation enforced by *`rocksdb_strict_collation_check`*. Empty by default.

**variable `rocksdb_table_cache_numshardbits`**

**Command Line** `--rocksdb-table-cache-numshardbits`

**Dynamic** No

**Scope** Global

**Variable Type** Numeric

**Default Value** `6`

Specifies the number if table caches. The default value is `6`. The allowed range is from `0` to `19`.

**variable `rocksdb_table_stats_sampling_pct`**

**Command Line** `--rocksdb-table-stats-sampling-pct`

**Dynamic** Yes

**Scope** Global

**Variable Type** Numeric

**Default Value** `10`

Specifies the percentage of entries to sample when collecting statistics about table properties. Default value is `10`. Allowed range is from `0` to `100`.

**variable `rocksdb_tmpdir`**

**Command Line** `--rocksdb-tmpdir`

**Dynamic** Yes

**Scope** Global, Session

**Variable Type** String

**Default Value**

Specifies the path to the directory for temporary files during DDL operations.

**variable `rocksdb_trace_sst_api`**

**Command Line** `--rocksdb-trace-sst-api`

**Dynamic** Yes

**Scope** Global, Session

**Variable Type** Boolean

**Default Value** `OFF`

Specifies whether to generate trace output in the log for each call to `SstFileWriter`. Disabled by default.

**variable `rocksdb_two_write_queues`**

**Command Line** `--rocksdb-two_write_queues`

**Dynamic** No

**Scope** Global

**Variable Type** Boolean

**Default Value** `ON`

When enabled this variable allows/encourages threads that are using two-phase commit to `prepare` in parallel.

**variable `rocksdb_unsafe_for_binlog`**

**Command Line** `--rocksdb-unsafe-for-binlog`

**Dynamic** Yes

**Scope** Global, Session

**Variable Type** Boolean

**Default Value** `OFF`

Specifies whether to allow statement-based binary logging which may break consistency. Disabled by default.

**variable `rocksdb_update_cf_options`**

**Command Line** `--rocksdb-update-cf-options`

**Dynamic** No

**Scope** Global

**Variable Type** String

**Default Value**

Specifies option updates for each column family. Empty by default.

**variable `rocksdb_use_adaptive_mutex`**

**Command Line** `--rocksdb-use-adaptive-mutex`

**Dynamic** No

**Scope** Global

**Variable Type** Boolean

**Default Value** `OFF`

Specifies whether to use adaptive mutex which spins in user space before resorting to the kernel. Disabled by default.

**variable `rocksdb_use_direct_io_for_flush_and_compaction`**

**Command Line** `--rocksdb-use-direct-io-for-flush-and-compaction`

**Dynamic** No

**Scope** Global

**Variable Type** Boolean

**Default Value** `OFF`

Specifies whether to write to data files directly, without caches or buffers. Disabled by default.

**variable `rocksdb_use_direct_reads`**

**Command Line** `--rocksdb-use-direct-reads`

**Dynamic** No

**Scope** Global

**Variable Type** Boolean

**Default Value** `OFF`

Specifies whether to read data files directly, without caches or buffers. Disabled by default. If you enable this, make sure that *`rocksdb_allow_mmap_reads`* is disabled.

**variable `rocksdb_use_fsync`**

**Command Line** `--rocksdb-use-fsync`

**Dynamic** No

**Scope** Global

**Variable Type** Boolean

**Default Value** `OFF`

Specifies whether MyRocks should use `fsync` instead of `fdatasync` when requesting a sync of a data file. Disabled by default.

**variable `rocksdb_validate_tables`**

**Command Line** `--rocksdb-validate-tables`

**Dynamic** No

**Scope** Global

**Variable Type** Numeric

**Default Value** `1`

Specifies whether to verify that MySQL `.frm` files match MyRocks tables.

- `0`: do not verify.
- `1`: verify and fail on error (default).
- `2`: verify and continue with error.

**variable `rocksdb_verify_row_debug_checksums`**

**Command Line** `--rocksdb-verify-row-debug-checksums`

**Dynamic** Yes

**Scope** Global, Session

**Variable Type** Boolean

**Default Value** `OFF`

Specifies whether to verify checksums when reading index or table records. Disabled by default.

**variable rocksdb_wal_bytes_per_sync**

**Command Line** `--rocksdb-wal-bytes-per-sync`

**Dynamic** Yes

**Scope** Global

**Variable Type** Numeric

**Default Value** `0`

Specifies how often should the OS sync WAL (write-ahead log) files to disk as they are being written, asynchronously, in the background. This operation can be used to smooth out write I/O over time. Default value is `0`, meaning that files are never synced. Allowed range is up to `18446744073709551615`.

**variable rocksdb_wal_dir**

**Command Line** `--rocksdb-wal-dir`

**Dynamic** No

**Scope** Global

**Variable Type** String

**Default Value**

Specifies the path to the directory where MyRocks stores WAL files.

**variable rocksdb_wal_recovery_mode**

**Command Line** `--rocksdb-wal-recovery-mode`

**Dynamic** Yes

**Scope** Global

**Variable Type** Numeric

**Default Value** `1`

Specifies the level of tolerance when recovering write-ahead logs (WAL) files after a system crash.

The following are the options:

- `0`: if the last WAL entry is corrupted, truncate the entry and either start the server normally or refuse to start.
- `1` (default): if a WAL entry is corrupted, the server fails to start and does not recover from the crash.
- `2`: if a corrupted WAL entry is detected, truncate all entries after the detected corrupted entry. You can select this setting for replication slaves.
- `3`: If a corrupted WAL entry is detected, skip only the corrupted entry and continue the apply WAL entries. This option can be dangerous.

**variable rocksdb_wal_size_limit_mb**

**Command Line** `--rocksdb-wal-size-limit-mb`

**Dynamic** No

**Scope** Global

**Variable Type** Numeric

**Default Value** `0`

Specifies the maximum size of all WAL files in megabytes before attempting to flush memtables and delete the oldest files. Default value is `0` (never rotated). Allowed range is up to `9223372036854775807`.

**variable `rocksdb_wal_ttl_seconds`**

**Command Line** `--rocksdb-wal-ttl-seconds`

**Dynamic** No

**Scope** Global

**Variable Type** Numeric

**Default Value** `0`

Specifies the timeout in seconds before deleting archived WAL files. Default is `0` (archived WAL files are never deleted). Allowed range is up to `9223372036854775807`.

**variable `rocksdb_whole_key_filtering`**

**Command Line** `--rocksdb-whole-key-filtering`

**Dynamic** No

**Scope** Global

**Variable Type** Boolean

**Default Value** `ON`

Specifies whether the bloomfilter should use the whole key for filtering instead of just the prefix. Enabled by default. Make sure that lookups use the whole key for matching.

**variable `rocksdb_write_batch_max_bytes`**

**Command Line** `--rocksdb-write-batch-max-bytes`

**Dynamic** Yes

**Scope** Global, Session

**Variable Type** Numeric

**Default Value** `0`

Specifies the maximum size of a RocksDB write batch in bytes. `0` means no limit. In case user exceeds the limit following error will be shown: `ERROR HY000: Status error 10 received from RocksDB: Operation aborted: Memory limit reached`.

**variable `rocksdb_write_disable_wal`**

**Command Line** `--rocksdb-write-disable-wal`

**Dynamic** Yes

**Scope** Global, Session

**Variable Type** Boolean

**Default Value** `OFF`

Lets you temporarily disable writes to WAL files, which can be useful for bulk loading.

**variable `rocksdb_write_ignore_missing_column_families`**

**Command Line** `--rocksdb-write-ignore-missing-column-families`

**Dynamic** Yes

**Scope** Global, Session

**Variable Type** Boolean

**Default Value** `OFF`

Specifies whether to ignore writes to column families that do not exist. Disabled by default (writes to non-existent column families are not ignored).

# MyRocks Information Schema Tables

When you install the MyRocks plugin for *MySQL*, the Information Schema is extended to include the following tables:



## ROCKSDB_GLOBAL_INFO

### Columns

| Column Name | Type |
|---|---|
| TYPE | varchar(513) |
| NAME | varchar(513) |
| VALUE | varchar(513) |

## ROCKSDB_CFSTATS

### Columns

| Column Name | Type |
|---|---|
| CF_NAME | varchar(193) |
| STAT_TYPE | varchar(193) |
| VALUE | bigint(8) |

## ROCKSDB_TRX

This table stores mappings of RocksDB transaction identifiers to *MySQL* client identifiers to enable associating a RocksDB transaction with a *MySQL* client operation.

### Columns

| Column Name | Type |
|---|---|
| TRANSACTION_ID | bigint(8) |
| STATE | varchar(193) |
| NAME | varchar(193) |
| WRITE_COUNT | bigint(8) |
| LOCK_COUNT | bigint(8) |
| TIMEOUT_SEC | int(4) |
| WAITING_KEY | varchar(513) |
| WAITING_COLUMN_FAMILY_ID | int(4) |
| IS_REPLICATION | int(4) |
| SKIP_TRX_API | int(4) |
| READ_ONLY | int(4) |
| HAS_DEADLOCK_DETECTION | int(4) |
| NUM_ONGOING_BULKLOAD | int(4) |
| THREAD_ID | int(8) |
| QUERY | varchar(193) |

## ROCKSDB_CF_OPTIONS

### Columns

| Column Name | Type |
|---|---|
| CF_NAME | varchar(193) |
| OPTION_TYPE | varchar(193) |
| VALUE | varchar(193) |

## ROCKSDB_COMPACTION_STATS

### Columns

| Column Name | Type |
|---|---|
| CF_NAME | varchar(193) |
| LEVEL | varchar(513) |
| TYPE | varchar(513) |
| VALUE | double |

## ROCKSDB_DBSTATS

### Columns

| Column Name | Type |
|---|---|
| STAT_TYPE | varchar(193) |
| VALUE | bigint(8) |

## ROCKSDB_DDL

### Columns

| Column Name | Type |
|---|---|
| TABLE_SCHEMA | varchar(193) |
| TABLE_NAME | varchar(193) |
| PARTITION_NAME | varchar(193) |
| INDEX_NAME | varchar(193) |
| COLUMN_FAMILY | int(4) |
| INDEX_NUMBER | int(4) |
| INDEX_TYPE | smallint(2) |
| KV_FORMAT_VERSION | smallint(2) |
| TTL_DURATION | bigint(8) |
| INDEX_FLAGS | bigint(8) |
| CF | varchar(193) |
| AUTO_INCREMENT | bigint(8) unsigned |

## ROCKSDB_INDEX_FILE_MAP

### Columns

| Column Name | Type |
|---|---|
| COLUMN_FAMILY | int(4) |
| INDEX_NUMBER | int(4) |
| SST_NAME | varchar(193) |
| NUM_ROWS | bigint(8) |
| DATA_SIZE | bigint(8) |
| ENTRY_DELETES | bigint(8) |
| ENTRY_SINGLEDELETES | bigint(8) |
| ENTRY_MERGES | bigint(8) |
| ENTRY_OTHERS | bigint(8) |
| DISTINCT_KEYS_PREFIX | varchar(400) |

## ROCKSDB_LOCKS

This table contains the set of locks granted to MyRocks transactions.

### Columns

| Column Name | Type |
|---|---|
| COLUMN_FAMILY_ID | int(4) |
| TRANSACTION_ID | int(4) |
| KEY | varchar(513) |
| MODE | varchar(32) |

## ROCKSDB_PERF_CONTEXT

### Columns

| Column Name | Type |
|---|---|
| TABLE_SCHEMA | varchar(193) |
| TABLE_NAME | varchar(193) |
| PARTITION_NAME | varchar(193) |
| STAT_TYPE | varchar(193) |
| VALUE | bigint(8) |

## ROCKSDB_PERF_CONTEXT_GLOBAL

### Columns

| Column Name | Type |
|---|---|
| STAT_TYPE | varchar(193) |
| VALUE | bigint(8) |

## ROCKSDB_DEADLOCK

This table records information about deadlocks.

### Columns

| Column Name | Type |
|---|---|
| DEADLOCK_ID | bigint(8) |
| TRANSACTION_ID | bigint(8) |
| CF_NAME | varchar(193) |
| WAITING_KEY | varchar(513) |
| LOCK_TYPE | varchar(193) |
| INDEX_NAME | varchar(193) |
| TABLE_NAME | varchar(193) |
| ROLLED_BACK | bigint(8) |

# Performance Schema MyRocks changes

RocksDB WAL file information can be seen in the performance_schema.log_status table in the `STORAGE ENGINE` column.

This feature has been implemented in Percona Server *8.0.15-6* release.

## Example

```
mysql> select * from performance_schema.log_status\G

*************************** 1. row ***************************

SERVER_UUID: f593b4f8-6fde-11e9-ad90-080027c2be11
    LOCAL: {"gtid_executed": "", "binary_log_file": "binlog.000004", "binary_log_
↪position": 1698222}
REPLICATION: {"channels": []}
STORAGE_ENGINES: {"InnoDB": {"LSN": 36810235, "LSN_checkpoint": 36810235}, "RocksDB":
↪{"wal_files": [{"path_name": "/000026.log", "log_number": 26, "size_file_bytes":␣
↪371869}]}}
1 row in set (0.00 sec)
```

CHAPTER

SEVENTYTWO

# MYROCKS STATUS VARIABLES

MyRocks status variables provide details about the inner workings of the storage engine and they can be useful in tuning the storage engine to a particular environment.

You can view these variables and their values by running:

```
mysql> SHOW STATUS LIKE 'rocksdb%';
```

The following global status variables are available:

| Name | Var Type |
|---|---|
| *rocksdb_rows_deleted* | Numeric |
| *rocksdb_rows_inserted* | Numeric |
| *rocksdb_rows_read* | Numeric |
| *rocksdb_rows_updated* | Numeric |
| *rocksdb_rows_expired* | Numeric |
| *rocksdb_system_rows_deleted* | Numeric |
| *rocksdb_system_rows_inserted* | Numeric |
| *rocksdb_system_rows_read* | Numeric |
| *rocksdb_system_rows_updated* | Numeric |
| *rocksdb_memtable_total* | Numeric |
| *rocksdb_memtable_unflushed* | Numeric |
| *rocksdb_queries_point* | Numeric |
| *rocksdb_queries_range* | Numeric |
| *rocksdb_covered_secondary_key_lookups* | Numeric |
| *rocksdb_block_cache_add* | Numeric |
| *rocksdb_block_cache_add_failures* | Numeric |
| *rocksdb_block_cache_bytes_read* | Numeric |
| *rocksdb_block_cache_bytes_write* | Numeric |
| *rocksdb_block_cache_data_add* | Numeric |
| *rocksdb_block_cache_data_bytes_insert* | Numeric |
| *rocksdb_block_cache_data_hit* | Numeric |
| *rocksdb_block_cache_data_miss* | Numeric |
| *rocksdb_block_cache_filter_add* | Numeric |
| *rocksdb_block_cache_filter_bytes_evict* | Numeric |
| *rocksdb_block_cache_filter_bytes_insert* | Numeric |
| *rocksdb_block_cache_filter_hit* | Numeric |
| *rocksdb_block_cache_filter_miss* | Numeric |
| *rocksdb_block_cache_hit* | Numeric |
| *rocksdb_block_cache_index_add* | Numeric |
| *rocksdb_block_cache_index_bytes_evict* | Numeric |

Continued on next page

Table 72.1 – continued from previous page

| Name | Var Type |
|---|---|
| rocksdb_block_cache_index_bytes_insert | Numeric |
| rocksdb_block_cache_index_hit | Numeric |
| rocksdb_block_cache_index_miss | Numeric |
| rocksdb_block_cache_miss | Numeric |
| rocksdb_block_cache_compressed_hit | Numeric |
| rocksdb_block_cache_compressed_miss | Numeric |
| rocksdb_bloom_filter_prefix_checked | Numeric |
| rocksdb_bloom_filter_prefix_useful | Numeric |
| rocksdb_bloom_filter_useful | Numeric |
| rocksdb_bytes_read | Numeric |
| rocksdb_bytes_written | Numeric |
| rocksdb_compact_read_bytes | Numeric |
| rocksdb_compact_write_bytes | Numeric |
| rocksdb_compaction_key_drop_new | Numeric |
| rocksdb_compaction_key_drop_obsolete | Numeric |
| rocksdb_compaction_key_drop_user | Numeric |
| rocksdb_flush_write_bytes | Numeric |
| rocksdb_get_hit_l0 | Numeric |
| rocksdb_get_hit_l1 | Numeric |
| rocksdb_get_hit_l2_and_up | Numeric |
| rocksdb_get_updates_since_calls | Numeric |
| rocksdb_iter_bytes_read | Numeric |
| rocksdb_memtable_hit | Numeric |
| rocksdb_memtable_miss | Numeric |
| rocksdb_no_file_closes | Numeric |
| rocksdb_no_file_errors | Numeric |
| rocksdb_no_file_opens | Numeric |
| rocksdb_num_iterators | Numeric |
| rocksdb_number_block_not_compressed | Numeric |
| rocksdb_number_db_next | Numeric |
| rocksdb_number_db_next_found | Numeric |
| rocksdb_number_db_prev | Numeric |
| rocksdb_number_db_prev_found | Numeric |
| rocksdb_number_db_seek | Numeric |
| rocksdb_number_db_seek_found | Numeric |
| rocksdb_number_deletes_filtered | Numeric |
| rocksdb_number_keys_read | Numeric |
| rocksdb_number_keys_updated | Numeric |
| rocksdb_number_keys_written | Numeric |
| rocksdb_number_merge_failures | Numeric |
| rocksdb_number_multiget_bytes_read | Numeric |
| rocksdb_number_multiget_get | Numeric |
| rocksdb_number_multiget_keys_read | Numeric |
| rocksdb_number_reseeks_iteration | Numeric |
| rocksdb_number_sst_entry_delete | Numeric |
| rocksdb_number_sst_entry_merge | Numeric |
| rocksdb_number_sst_entry_other | Numeric |
| rocksdb_number_sst_entry_put | Numeric |
| rocksdb_number_sst_entry_singledelete | Numeric |

Continued on next page

Table 72.1 – continued from previous page

| Name | Var Type |
|---|---|
| *rocksdb_number_stat_computes* | Numeric |
| *rocksdb_number_superversion_acquires* | Numeric |
| *rocksdb_number_superversion_cleanups* | Numeric |
| *rocksdb_number_superversion_releases* | Numeric |
| *rocksdb_rate_limit_delay_millis* | Numeric |
| *rocksdb_row_lock_deadlocks* | Numeric |
| *rocksdb_row_lock_wait_timeouts* | Numeric |
| *rocksdb_snapshot_conflict_errors* | Numeric |
| *rocksdb_stall_l0_file_count_limit_slowdowns* | Numeric |
| *rocksdb_stall_locked_l0_file_count_limit_slowdowns* | Numeric |
| *rocksdb_stall_l0_file_count_limit_stops* | Numeric |
| *rocksdb_stall_locked_l0_file_count_limit_stops* | Numeric |
| *rocksdb_stall_pending_compaction_limit_stops* | Numeric |
| *rocksdb_stall_pending_compaction_limit_slowdowns* | Numeric |
| *rocksdb_stall_memtable_limit_stops* | Numeric |
| *rocksdb_stall_memtable_limit_slowdowns* | Numeric |
| *rocksdb_stall_total_stops* | Numeric |
| *rocksdb_stall_total_slowdowns* | Numeric |
| *rocksdb_stall_micros* | Numeric |
| *rocksdb_wal_bytes* | Numeric |
| *rocksdb_wal_group_syncs* | Numeric |
| *rocksdb_wal_synced* | Numeric |
| *rocksdb_write_other* | Numeric |
| *rocksdb_write_self* | Numeric |
| *rocksdb_write_timedout* | Numeric |
| *rocksdb_write_wal* | Numeric |

**variable `rocksdb_rows_deleted`**

This variable shows the number of rows that were deleted from MyRocks tables.

**variable `rocksdb_rows_inserted`**

This variable shows the number of rows that were inserted into MyRocks tables.

**variable `rocksdb_rows_read`**

This variable shows the number of rows that were read from MyRocks tables.

**variable `rocksdb_rows_updated`**

This variable shows the number of rows that were updated in MyRocks tables.

**variable `rocksdb_rows_expired`**

This variable shows the number of expired rows in MyRocks tables.

**variable `rocksdb_system_rows_deleted`**

This variable shows the number of rows that were deleted from MyRocks system tables.

**variable `rocksdb_system_rows_inserted`**

This variable shows the number of rows that were inserted into MyRocks system tables.

**variable `rocksdb_system_rows_read`**

This variable shows the number of rows that were read from MyRocks system tables.

**variable `rocksdb_system_rows_updated`**

This variable shows the number of rows that were updated in MyRocks system tables.

**variable `rocksdb_memtable_total`**

This variable shows the memory usage, in bytes, of all memtables.

**variable `rocksdb_memtable_unflushed`**

This variable shows the memory usage, in bytes, of all unflushed memtables.

**variable `rocksdb_queries_point`**

This variable shows the number of single row queries.

**variable `rocksdb_queries_range`**

This variable shows the number of multi/range row queries.

**variable `rocksdb_covered_secondary_key_lookups`**

This variable shows the number of lookups via secondary index that were able to return all fields requested directly from the secondary index when the secondary index contained a field that is only a prefix of the `varchar` column.

**variable `rocksdb_block_cache_add`**

This variable shows the number of blocks added to block cache.

**variable `rocksdb_block_cache_add_failures`**

This variable shows the number of failures when adding blocks to block cache.

**variable `rocksdb_block_cache_bytes_read`**

This variable shows the number of bytes read from cache.

**variable `rocksdb_block_cache_bytes_write`**

This variable shows the number of bytes written into cache.

**variable `rocksdb_block_cache_data_add`**

This variable shows the number of data blocks added to block cache.

**variable `rocksdb_block_cache_data_bytes_insert`**

This variable shows the number of bytes of data blocks inserted into cache.

**variable `rocksdb_block_cache_data_hit`**

This variable shows the number of cache hits when accessing the data block from the block cache.

**variable `rocksdb_block_cache_data_miss`**

This variable shows the number of cache misses when accessing the data block from the block cache.

**variable `rocksdb_block_cache_filter_add`**

This variable shows the number of filter blocks added to block cache.

**variable `rocksdb_block_cache_filter_bytes_evict`**

This variable shows the number of bytes of bloom filter blocks removed from cache.

**variable `rocksdb_block_cache_filter_bytes_insert`**

This variable shows the number of bytes of bloom filter blocks inserted into cache.

**variable `rocksdb_block_cache_filter_hit`**

This variable shows the number of times cache hit when accessing filter block from block cache.

**variable `rocksdb_block_cache_filter_miss`**

This variable shows the number of times cache miss when accessing filter block from block cache.

**variable `rocksdb_block_cache_hit`**

This variable shows the total number of block cache hits.

**variable `rocksdb_block_cache_index_add`**

This variable shows the number of index blocks added to block cache.

**variable `rocksdb_block_cache_index_bytes_evict`**

This variable shows the number of bytes of index block erased from cache.

**variable `rocksdb_block_cache_index_bytes_insert`**

This variable shows the number of bytes of index blocks inserted into cache.

**variable `rocksdb_block_cache_index_hit`**

This variable shows the total number of block cache index hits.

**variable `rocksdb_block_cache_index_miss`**

This variable shows the number of times cache hit when accessing index block from block cache.

**variable `rocksdb_block_cache_miss`**

This variable shows the total number of block cache misses.

**variable `rocksdb_block_cache_compressed_hit`**

This variable shows the number of hits in the compressed block cache.

**variable `rocksdb_block_cache_compressed_miss`**

This variable shows the number of misses in the compressed block cache.

**variable `rocksdb_bloom_filter_prefix_checked`**

This variable shows the number of times bloom was checked before creating iterator on a file.

**variable `rocksdb_bloom_filter_prefix_useful`**

This variable shows the number of times the check was useful in avoiding iterator creation (and thus likely IOPs).

**variable `rocksdb_bloom_filter_useful`**

This variable shows the number of times bloom filter has avoided file reads.

**variable `rocksdb_bytes_read`**

This variable shows the total number of uncompressed bytes read. It could be either from memtables, cache, or table files.

**variable `rocksdb_bytes_written`**

This variable shows the total number of uncompressed bytes written.

**variable `rocksdb_compact_read_bytes`**

This variable shows the number of bytes read during compaction

**variable `rocksdb_compact_write_bytes`**

This variable shows the number of bytes written during compaction.

**variable rocksdb_compaction_key_drop_new**

This variable shows the number of key drops during compaction because it was overwritten with a newer value.

**variable rocksdb_compaction_key_drop_obsolete**

This variable shows the number of key drops during compaction because it was obsolete.

**variable rocksdb_compaction_key_drop_user**

This variable shows the number of key drops during compaction because user compaction function has dropped the key.

**variable rocksdb_flush_write_bytes**

This variable shows the number of bytes written during flush.

**variable rocksdb_get_hit_l0**

This variable shows the number of `Get()` queries served by L0.

**variable rocksdb_get_hit_l1**

This variable shows the number of `Get()` queries served by L1.

**variable rocksdb_get_hit_l2_and_up**

This variable shows the number of `Get()` queries served by L2 and up.

**variable rocksdb_get_updates_since_calls**

This variable shows the number of calls to `GetUpdatesSince` function. Useful to keep track of transaction log iterator refreshes

**variable rocksdb_iter_bytes_read**

This variable shows the number of uncompressed bytes read from an iterator. It includes size of key and value.

**variable rocksdb_memtable_hit**

This variable shows the number of memtable hits.

**variable rocksdb_memtable_miss**

This variable shows the number of memtable misses.

**variable rocksdb_no_file_closes**

This variable shows the number of time file were closed.

**variable rocksdb_no_file_errors**

This variable shows number of errors trying to read in data from an sst file.

**variable rocksdb_no_file_opens**

This variable shows the number of time file were opened.

**variable rocksdb_num_iterators**

This variable shows the number of currently open iterators.

**variable rocksdb_number_block_not_compressed**

This variable shows the number of uncompressed blocks.

**variable rocksdb_number_db_next**

This variable shows the number of calls to `next`.

**variable rocksdb_number_db_next_found**

This variable shows the number of calls to `next` that returned data.

**variable `rocksdb_number_db_prev`**

This variable shows the number of calls to `prev`.

**variable `rocksdb_number_db_prev_found`**

This variable shows the number of calls to `prev` that returned data.

**variable `rocksdb_number_db_seek`**

This variable shows the number of calls to `seek`.

**variable `rocksdb_number_db_seek_found`**

This variable shows the number of calls to `seek` that returned data.

**variable `rocksdb_number_deletes_filtered`**

This variable shows the number of deleted records that were not required to be written to storage because key did not exist.

**variable `rocksdb_number_keys_read`**

This variable shows the number of keys read.

**variable `rocksdb_number_keys_updated`**

This variable shows the number of keys updated, if inplace update is enabled.

**variable `rocksdb_number_keys_written`**

This variable shows the number of keys written to the database.

**variable `rocksdb_number_merge_failures`**

This variable shows the number of failures performing merge operator actions in RocksDB.

**variable `rocksdb_number_multiget_bytes_read`**

This variable shows the number of bytes read during RocksDB `MultiGet()` calls.

**variable `rocksdb_number_multiget_get`**

This variable shows the number `MultiGet()` requests to RocksDB.

**variable `rocksdb_number_multiget_keys_read`**

This variable shows the keys read via `MultiGet()`.

**variable `rocksdb_number_reseeks_iteration`**

This variable shows the number of times reseek happened inside an iteration to skip over large number of keys with same userkey.

**variable `rocksdb_number_sst_entry_delete`**

This variable shows the total number of delete markers written by MyRocks.

**variable `rocksdb_number_sst_entry_merge`**

This variable shows the total number of merge keys written by MyRocks.

**variable `rocksdb_number_sst_entry_other`**

This variable shows the total number of non-delete, non-merge, non-put keys written by MyRocks.

**variable `rocksdb_number_sst_entry_put`**

This variable shows the total number of put keys written by MyRocks.

**variable `rocksdb_number_sst_entry_singledelete`**

This variable shows the total number of single delete keys written by MyRocks.

**variable `rocksdb_number_stat_computes`**

This variable isn't used anymore and will be removed in future releases.

**variable `rocksdb_number_superversion_acquires`**

This variable shows the number of times the superversion structure has been acquired in RocksDB, this is used for tracking all of the files for the database.

**variable `rocksdb_number_superversion_cleanups`**

**variable `rocksdb_number_superversion_releases`**

**variable `rocksdb_rate_limit_delay_millis`**

This variable isn't used anymore and will be removed in future releases.

**variable `rocksdb_row_lock_deadlocks`**

This variable shows the total number of deadlocks that have been detected since the instance was started.

**variable `rocksdb_row_lock_wait_timeouts`**

This variable shows the total number of row lock wait timeouts that have been detected since the instance was started.

**variable `rocksdb_snapshot_conflict_errors`**

This variable shows the number of snapshot conflict errors occurring during write transactions that forces the transaction to rollback.

**variable `rocksdb_stall_l0_file_count_limit_slowdowns`**

This variable shows the slowdowns in write due to L0 being close to full.

**variable `rocksdb_stall_locked_l0_file_count_limit_slowdowns`**

This variable shows the slowdowns in write due to L0 being close to full and compaction for L0 is already in progress.

**variable `rocksdb_stall_l0_file_count_limit_stops`**

This variable shows the stalls in write due to L0 being full.

**variable `rocksdb_stall_locked_l0_file_count_limit_stops`**

This variable shows the stalls in write due to L0 being full and compaction for L0 is already in progress.

**variable `rocksdb_stall_pending_compaction_limit_stops`**

This variable shows the stalls in write due to hitting limits set for max number of pending compaction bytes.

**variable `rocksdb_stall_pending_compaction_limit_slowdowns`**

This variable shows the slowdowns in write due to getting close to limits set for max number of pending compaction bytes.

**variable `rocksdb_stall_memtable_limit_stops`**

This variable shows the stalls in write due to hitting max number of `memTables` allowed.

**variable `rocksdb_stall_memtable_limit_slowdowns`**

This variable shows the slowdowns in writes due to getting close to max number of memtables allowed.

**variable `rocksdb_stall_total_stops`**

This variable shows the total number of write stalls.

**variable `rocksdb_stall_total_slowdowns`**

This variable shows the total number of write slowdowns.

**variable `rocksdb_stall_micros`**

This variable shows how long (in microseconds) the writer had to wait for compaction or flush to finish.

**variable `rocksdb_wal_bytes`**

This variables shows the number of bytes written to WAL.

**variable `rocksdb_wal_group_syncs`**

This variable shows the number of group commit WAL file syncs that have occurred.

**variable `rocksdb_wal_synced`**

This variable shows the number of times WAL sync was done.

**variable `rocksdb_write_other`**

This variable shows the number of writes processed by another thread.

**variable `rocksdb_write_self`**

This variable shows the number of writes that were processed by a requesting thread.

**variable `rocksdb_write_timedout`**

This variable shows the number of writes ending up with timed-out.

**variable `rocksdb_write_wal`**

This variable shows the number of Write calls that request WAL.

CHAPTER

# SEVENTYTHREE

# GAP LOCKS DETECTION

The Gap locks detection is based on a Facebook *MySQL* patch.

If a transactional storage engine does not support gap locks (for example MyRocks) and a gap lock is being attempted while the transaction isolation level is either `REPEATABLE READ` or `SERIALIZABLE`, the following SQL error will be returned to the client and no actual gap lock will be taken on the effected rows.

```
ERROR HY000: Using Gap Lock without full unique key in multi-table or multi-statement␣
↪transactions is not allowed. You need to either rewrite queries to use all unique␣
↪key columns in WHERE equal conditions, or rewrite to single-table, single-statement␣
↪transaction.
```

CHAPTER

SEVENTYFOUR

# DATA LOADING

By default, MyRocks configurations are optimized for short transactions, and not for data loading. MyRocks has a couple of special session variables to speed up data loading dramatically.

## Sorted bulk loading

If your data is guaranteed to be loaded in primary key order, then this method is recommended. This method works by dropping any secondary keys first, loading data into your table in primary key order, and then restoring the secondary keys via Fast Secondary Index Creation.

### Creating Secondary Indexes

When loading data into empty tables, it is highly recommended to drop all secondary indexes first, then loading data, and adding all secondary indexes after finishing loading data. MyRocks has a feature called `Fast Secondary Index Creation`. Fast Secondary Index Creation is automatically used when executing `CREATE INDEX` or `ALTER TABLE ... ADD INDEX`. With Fast Secondary Index Creation, the secondary index entries are directly written to bottommost RocksDB levels and bypassing compaction. This significantly reduces total write volume and CPU time for decompressing and compressing data on higher levels.

### Loading Data

As described above, loading data is highly recommended for tables with primary key only (no secondary keys), with all secondary indexes added after loading data.

When loading data into MyRocks tables, there are two recommended session variables:

```
SET session sql_log_bin=0;
SET session rocksdb_bulk_load=1;
```

When converting from large MyISAM/InnoDB tables, either by using the `ALTER` or `INSERT INTO SELECT` statements it's recommended that you create MyRocks tables as below (in case the table is sufficiently big it will cause the server to consume all the memory and then be terminated by the OOM killer):

```
SET session sql_log_bin=0;
SET session rocksdb_bulk_load=1;
ALTER TABLE large_myisam_table ENGINE=RocksDB;
SET session rocksdb_bulk_load=0;
```

Using sql_log_bin=0 avoids writing to binary logs.

With *rocksdb_bulk_load* set to 1, MyRocks enters special mode to write all inserts into bottommost RocksDB levels, and skips writing data into MemTable and the following compactions. This is very efficient way to load data.

The *rocksdb_bulk_load* mode operates with a few conditions:

- None of the data being bulk loaded can overlap with existing data in the table. The easiest way to ensure this is to always bulk load into an empty table, but the mode will allow loading some data into the table, doing other operations, and then returning and bulk loading addition data if there is no overlap between what is being loaded and what already exists.
- The data may not be visible until bulk load mode is ended (i.e. the *rocksdb_bulk_load* is set to zero again). The method that is used is building up SST files which will later be added as-is to the database. Until a particular SST has been added the data will not be visible to the rest of the system, thus issuing a SELECT on the table currently being bulk loaded will only show older data and will likely not show the most recently added rows. Ending the bulk load mode will cause the most recent SST file to be added. When bulk loading multiple tables, starting a new table will trigger the code to add the most recent SST file to the system – as a result, it is inadvisable to interleave INSERT statements to two or more tables during bulk load mode.

By default, the *rocksdb_bulk_load* mode expects all data be inserted in primary key order (or reversed order). If the data is in the reverse order (i.e. the data is descending on a normally ordered primary key or is ascending on a reverse ordered primary key), the rows are cached in chunks to switch the order to match the expected order.

Inserting one or more rows out of order will result in an error and may result in some of the data being inserted in the table and some not. To resolve the problem, one can either fix the data order of the insert, truncate the table, and restart.

## Unsorted bulk loading

If your data is not ordered in primary key order, then this method is recommended. With this method, secondary keys do not need to be dropped and restored. However, writing to the primary key no longer goes directly to SST files, and are written to temporary files for sorted first, so there is extra cost to this method.

To allow for loading unsorted data:

```
SET session sql_log_bin=0;
SET session rocksdb_bulk_load_allow_unsorted=1;
SET session rocksdb_bulk_load=1;
...
SET session rocksdb_bulk_load=0;
SET session rocksdb_bulk_load_allow_unsorted=0;
```

Note that *rocksdb_bulk_load_allow_unsorted* can only be changed when *rocksdb_bulk_load* is disabled (set to 0). In this case, all input data will go through an intermediate step that writes the rows to temporary SST files, sorts them rows in the primary key order, and then writes to final SST files in the correct order.

## Other Approaches

If *rocksdb_commit_in_the_middle* is enabled, MyRocks implicitly commits every rocksdb_bulk_load_size records (default is 1,000) in the middle of your transaction. If your data loading fails in the middle of the statement (LOAD DATA or bulk INSERT), rows are not entirely rolled back, but some of rows are stored in the table. To restart data loading, you'll need to truncate the table and loading data again.

**Warning:** If you are loading large data without enabling *rocksdb_bulk_load* or *rocksdb_commit_in_the_middle*, please make sure transaction size is small enough. All modifications of the ongoing transactions are kept in memory.

## Other Reading

- Data Loading - this document has been used as a source for writing this documentation
- ALTER TABLE ... ENGINE=ROCKSDB uses too much memory

# Part XII

# Reference

CHAPTER

SEVENTYFIVE

# LIST OF UPSTREAM *MYSQL* BUGS FIXED IN *PERCONA SERVER FOR MYSQL* 8.0

**Upstream Bug** #93788 - main.mysqldump is failing because of dropped event
**JIRA bug** #5268
**Upstream State** Duplicate (checked on 2019-01-16)
**Fix Released** 8.0.13-4
**Upstream Fix** N/A

**Upstream Bug** #93708 - Page Cleaner will sleep for long time if clock changes
**JIRA bug** #5221
**Upstream State** Verified (checked on 2019-03-11)
**Fix Released** 8.0.15-5
**Upstream Fix** N/A

**Upstream Bug** #93703 - EXPLAIN SELECT returns inconsistent number of ROWS in main.group_by
**JIRA bug** #5306
**Upstream State** Need Feedback (checked on 2019-01-16)
**Fix Released** 8.0.13-4
**Upstream Fix** N/A

**Upstream Bug** #93686 - innodb.upgrade_orphan fails because of left files
**JIRA bug** #5209
**Upstream State** Verified (checked on 2019-01-16)
**Fix Released** 8.0.13-4
**Upstream Fix** N/A

**Upstream Bug** #93544 - SHOW BINLOG EVENTS FROM <bad offset> is not diagnosed
**JIRA bug** #5126
**Upstream State** Verified (checked on 2019-01-16)
**Fix Released** 8.0.13-4
**Upstream Fix** N/A

**Upstream Bug** #89840 - 60-80k connections causing empty reply for select
**JIRA bug** #314
**Upstream State** Verified (checked on 2018-11-20)
**Fix Released** 8.0.12-2rc1
**Upstream Fix** N/A

**Upstream Bug** #89607 - MySQL crash in debug, PFS thread not handling singals.
**JIRA bug** #311
**Upstream State** Verified (checked on 2018-11-20)
**Fix Released** 8.0.12-2rc1
**Upstream Fix** N/A

CHAPTER

SEVENTYSIX

# LIST OF VARIABLES INTRODUCED IN *PERCONA SERVER FOR MYSQL* 8.0

## System Variables

| Name | Cmd-Line | Option File | Var Scope | Dynamic |
|---|---|---|---|---|
| *audit_log_buffer_size* | Yes | Yes | Global | No |
| *audit_log_file* | Yes | Yes | Global | No |
| *audit_log_flush* | Yes | Yes | Global | Yes |
| *audit_log_format* | Yes | Yes | Global | No |
| *audit_log_handler* | Yes | Yes | Global | No |
| *audit_log_policy* | Yes | Yes | Global | Yes |
| *audit_log_rotate_on_size* | Yes | Yes | Global | No |
| *audit_log_rotations* | Yes | Yes | Global | No |
| *audit_log_strategy* | Yes | Yes | Global | No |
| *audit_log_syslog_facility* | Yes | Yes | Global | No |
| *audit_log_syslog_ident* | Yes | Yes | Global | No |
| *audit_log_syslog_priority* | Yes | Yes | Global | No |
| csv_mode | Yes | Yes | Both | Yes |
| enforce_storage_engine | Yes | Yes | Global | No |
| *expand_fast_index_creation* | Yes | No | Both | Yes |
| *extra_max_connections* | Yes | Yes | Global | Yes |
| *extra_port* | Yes | Yes | Global | No |
| *have_backup_locks* | Yes | No | Global | No |
| have_backup_safe_binlog_info | Yes | No | Global | No |
| *have_snapshot_cloning* | Yes | No | Global | No |
| innodb_cleaner_lsn_age_factor | Yes | Yes | Global | Yes |
| *innodb_corrupt_table_action* | Yes | Yes | Global | Yes |
| *innodb_empty_free_list_algorithm* | Yes | Yes | Global | Yes |
| *innodb_encrypt_online_alter_logs* | Yes | Yes | Global | Yes |
| *innodb_encrypt_tables* | Yes | Yes | Global | Yes |
| innodb_kill_idle_transaction | Yes | Yes | Global | Yes |
| *innodb_max_bitmap_file_size* | Yes | Yes | Global | Yes |
| *innodb_max_changed_pages* | Yes | Yes | Global | Yes |
| *innodb_print_lock_wait_timeout_info* | Yes | Yes | Global | Yes |
| *innodb_show_locks_held* | Yes | Yes | Global | Yes |
| *innodb_temp_tablespace_encrypt* | Yes | Yes | Global | No |
| *innodb_track_changed_pages* | Yes | Yes | Global | No |

Continued on next page

Table 76.1 – continued from previous page

| Name | Cmd-Line | Option File | Var Scope | Dynamic |
|---|---|---|---|---|
| *keyring_vault_config* | Yes | Yes | Global | Yes |
| *keyring_vault_timeout* | Yes | Yes | Global | Yes |
| *log_slow_filter* | Yes | Yes | Both | Yes |
| *log_slow_rate_limit* | Yes | Yes | Both | Yes |
| *log_slow_rate_type* | Yes | Yes | Global | Yes |
| *log_slow_sp_statements* | Yes | Yes | Global | Yes |
| *log_slow_verbosity* | Yes | Yes | Both | Yes |
| *log_warnings_suppress* | Yes | Yes | Global | Yes |
| *proxy_protocol_networks* | Yes | Yes | Global | No |
| query_response_time_flush | Yes | No | Global | No |
| query_response_time_range_base | Yes | Yes | Global | Yes |
| query_response_time_stats | Yes | Yes | Global | Yes |
| *slow_query_log_always_write_time* | Yes | Yes | Global | Yes |
| *slow_query_log_use_global_control* | Yes | Yes | Global | Yes |
| *thread_pool_high_prio_mode* | Yes | Yes | Both | Yes |
| *thread_pool_high_prio_tickets* | Yes | Yes | Both | Yes |
| *thread_pool_idle_timeout* | Yes | Yes | Global | Yes |
| *thread_pool_max_threads* | Yes | Yes | Global | Yes |
| *thread_pool_oversubscribe* | Yes | Yes | Global | Yes |
| *thread_pool_size* | Yes | Yes | Global | Yes |
| *thread_pool_stall_limit* | Yes | Yes | Global | No |
| *thread_statistics* | Yes | Yes | Global | Yes |
| *tokudb_alter_print_error* | | | | |
| *tokudb_analyze_delete_fraction* | | | | |
| *tokudb_analyze_in_background* | Yes | Yes | Both | Yes |
| *tokudb_analyze_mode* | Yes | Yes | Both | Yes |
| *tokudb_analyze_throttle* | Yes | Yes | Both | Yes |
| *tokudb_analyze_time* | Yes | Yes | Both | Yes |
| *tokudb_auto_analyze* | Yes | Yes | Both | Yes |
| *tokudb_block_size* | | | | |
| *tokudb_bulk_fetch* | | | | |
| *tokudb_cache_size* | | | | |
| *tokudb_cachetable_pool_threads* | Yes | Yes | Global | No |
| *tokudb_cardinality_scale_percent* | | | | |
| *tokudb_check_jemalloc* | | | | |
| *tokudb_checkpoint_lock* | | | | |
| *tokudb_checkpoint_on_flush_logs* | | | | |
| *tokudb_checkpoint_pool_threads* | Yes | Yes | Global | No |
| *tokudb_checkpointing_period* | | | | |
| *tokudb_cleaner_iterations* | | | | |
| *tokudb_cleaner_period* | | | | |
| *tokudb_client_pool_threads* | Yes | Yes | Global | No |
| *tokudb_commit_sync* | | | | |
| *tokudb_compress_buffers_before_eviction* | Yes | Yes | Global | No |
| *tokudb_create_index_online* | | | | |
| *tokudb_data_dir* | | | | |
| *tokudb_debug* | | | | |
| *tokudb_directio* | | | | |

Continued on next page

Table 76.1 – continued from previous page

| Name | Cmd-Line | Option File | Var Scope | Dynamic |
|---|---|---|---|---|
| *tokudb_disable_hot_alter* | | | | |
| *tokudb_disable_prefetching* | | | | |
| *tokudb_disable_slow_alter* | | | | |
| *tokudb_empty_scan* | | | | |
| *tokudb_enable_partial_eviction* | Yes | Yes | Global | No |
| *tokudb_fanout* | Yes | Yes | Both | Yes |
| *tokudb_fs_reserve_percent* | | | | |
| *tokudb_fsync_log_period* | | | | |
| *tokudb_hide_default_row_format* | | | | |
| *tokudb_killed_time* | | | | |
| *tokudb_last_lock_timeout* | | | | |
| *tokudb_load_save_space* | | | | |
| *tokudb_loader_memory_size* | | | | |
| *tokudb_lock_timeout* | | | | |
| *tokudb_lock_timeout_debug* | | | | |
| *tokudb_log_dir* | | | | |
| *tokudb_max_lock_memory* | | | | |
| *tokudb_optimize_index_fraction* | | | | |
| *tokudb_optimize_index_name* | | | | |
| *tokudb_optimize_throttle* | | | | |
| *tokudb_pk_insert_mode* | | | | |
| *tokudb_prelock_empty* | | | | |
| *tokudb_read_block_size* | | | | |
| *tokudb_read_buf_size* | | | | |
| *tokudb_read_status_frequency* | | | | |
| *tokudb_row_format* | | | | |
| *tokudb_rpl_check_readonly* | | | | |
| *tokudb_rpl_lookup_rows* | | | | |
| *tokudb_rpl_lookup_rows_delay* | | | | |
| *tokudb_rpl_unique_checks* | | | | |
| *tokudb_rpl_unique_checks_delay* | | | | |
| *tokudb_strip_frm_data* | Yes | Yes | Global | No |
| *tokudb_support_xa* | | | | |
| *tokudb_tmp_dir* | | | | |
| *tokudb_version* | | | | |
| *tokudb_write_status_frequency* | | | | |
| *userstat* | Yes | Yes | Global | Yes |
| version_comment | Yes | Yes | Global | Yes |
| version_suffix | Yes | Yes | Global | Yes |

## Status Variables

| Name | Var Type | Var Scope |
|---|---|---|
| *Binlog_snapshot_file* | String | Global |
| *Binlog_snapshot_position* | Numeric | Global |

Continued on next page

Table 76.2 – continued from previous page

| Name | Var Type | Var Scope |
|---|---|---|
| Com_lock_binlog_for_backup | Numeric | Both |
| *Com_lock_tables_for_backup* | Numeric | Both |
| *Com_show_client_statistics* | Numeric | Both |
| *Com_show_index_statistics* | Numeric | Both |
| *Com_show_table_statistics* | Numeric | Both |
| *Com_show_thread_statistics* | Numeric | Both |
| *Com_show_user_statistics* | Numeric | Both |
| Com_unlock_binlog | Numeric | Both |
| *Innodb_background_log_sync* | Numeric | Global |
| *Innodb_buffer_pool_pages_LRU_flushed* | Numeric | Global |
| *Innodb_buffer_pool_pages_made_not_young* | Numeric | Global |
| *Innodb_buffer_pool_pages_made_young* | Numeric | Global |
| *Innodb_buffer_pool_pages_old* | Numeric | Global |
| *Innodb_checkpoint_age* | Numeric | Global |
| *Innodb_checkpoint_max_age* | Numeric | Global |
| *Innodb_ibuf_free_list* | Numeric | Global |
| *Innodb_ibuf_segment_size* | Numeric | Global |
| *Innodb_lsn_current* | Numeric | Global |
| *Innodb_lsn_flushed* | Numeric | Global |
| *Innodb_lsn_last_checkpoint* | Numeric | Global |
| *Innodb_master_thread_active_loops* | Numeric | Global |
| *Innodb_master_thread_idle_loops* | Numeric | Global |
| *Innodb_max_trx_id* | Numeric | Global |
| *Innodb_mem_adaptive_hash* | Numeric | Global |
| *Innodb_mem_dictionary* | Numeric | Global |
| *Innodb_oldest_view_low_limit_trx_id* | Numeric | Global |
| *Innodb_purge_trx_id* | Numeric | Global |
| *Innodb_purge_undo_no* | Numeric | Global |
| *Threadpool_idle_threads* | Numeric | Global |
| *Threadpool_threads* | Numeric | Global |
| *Tokudb_DB_OPENS* | | |
| *Tokudb_DB_CLOSES* | | |
| *Tokudb_DB_OPEN_CURRENT* | | |
| *Tokudb_DB_OPEN_MAX* | | |
| *Tokudb_LEAF_ENTRY_MAX_COMMITTED_XR* | | |
| *Tokudb_LEAF_ENTRY_MAX_PROVISIONAL_XR* | | |
| *Tokudb_LEAF_ENTRY_EXPANDED* | | |
| *Tokudb_LEAF_ENTRY_MAX_MEMSIZE* | | |
| *Tokudb_LEAF_ENTRY_APPLY_GC_BYTES_IN* | | |
| *Tokudb_LEAF_ENTRY_APPLY_GC_BYTES_OUT* | | |
| *Tokudb_LEAF_ENTRY_NORMAL_GC_BYTES_IN* | | |
| *Tokudb_LEAF_ENTRY_NORMAL_GC_BYTES_OUT* | | |
| *Tokudb_CHECKPOINT_PERIOD* | | |
| *Tokudb_CHECKPOINT_FOOTPRINT* | | |
| *Tokudb_CHECKPOINT_LAST_BEGAN* | | |
| *Tokudb_CHECKPOINT_LAST_COMPLETE_BEGAN* | | |
| *Tokudb_CHECKPOINT_LAST_COMPLETE_ENDED* | | |
| *Tokudb_CHECKPOINT_DURATION* | | |

Continued on next page

Table 76.2 – continued from previous page

| Name | Var Type | Var Scope |
|---|---|---|
| *Tokudb_CHECKPOINT_DURATION_LAST* | | |
| *Tokudb_CHECKPOINT_LAST_LSN* | | |
| *Tokudb_CHECKPOINT_TAKEN* | | |
| *Tokudb_CHECKPOINT_FAILED* | | |
| *Tokudb_CHECKPOINT_WAITERS_NOW* | | |
| *Tokudb_CHECKPOINT_WAITERS_MAX* | | |
| *Tokudb_CHECKPOINT_CLIENT_WAIT_ON_MO* | | |
| *Tokudb_CHECKPOINT_CLIENT_WAIT_ON_CS* | | |
| *Tokudb_CHECKPOINT_BEGIN_TIME* | | |
| *Tokudb_CHECKPOINT_LONG_BEGIN_TIME* | | |
| *Tokudb_CHECKPOINT_LONG_BEGIN_COUNT* | | |
| *Tokudb_CHECKPOINT_END_TIME* | | |
| *Tokudb_CHECKPOINT_LONG_END_TIME* | | |
| *Tokudb_CHECKPOINT_LONG_END_COUNT* | | |
| *Tokudb_CACHETABLE_MISS* | | |
| *Tokudb_CACHETABLE_MISS_TIME* | | |
| *Tokudb_CACHETABLE_PREFETCHES* | | |
| *Tokudb_CACHETABLE_SIZE_CURRENT* | | |
| *Tokudb_CACHETABLE_SIZE_LIMIT* | | |
| *Tokudb_CACHETABLE_SIZE_WRITING* | | |
| *Tokudb_CACHETABLE_SIZE_NONLEAF* | | |
| *Tokudb_CACHETABLE_SIZE_LEAF* | | |
| *Tokudb_CACHETABLE_SIZE_ROLLBACK* | | |
| *Tokudb_CACHETABLE_SIZE_CACHEPRESSURE* | | |
| *Tokudb_CACHETABLE_SIZE_CLONED* | | |
| *Tokudb_CACHETABLE_EVICTIONS* | | |
| *Tokudb_CACHETABLE_CLEANER_EXECUTIONS* | | |
| *Tokudb_CACHETABLE_CLEANER_PERIOD* | | |
| *Tokudb_CACHETABLE_CLEANER_ITERATIONS* | | |
| *Tokudb_CACHETABLE_WAIT_PRESSURE_COUNT* | | |
| *Tokudb_CACHETABLE_WAIT_PRESSURE_TIME* | | |
| *Tokudb_CACHETABLE_LONG_WAIT_PRESSURE_COUNT* | | |
| *Tokudb_CACHETABLE_LONG_WAIT_PRESSURE_TIME* | | |
| *Tokudb_CACHETABLE_POOL_CLIENT_NUM_THREADS* | | |
| *Tokudb_CACHETABLE_POOL_CLIENT_NUM_THREADS_ACTIVE* | | |
| *Tokudb_CACHETABLE_POOL_CLIENT_QUEUE_SIZE* | | |
| *Tokudb_CACHETABLE_POOL_CLIENT_MAX_QUEUE_SIZE* | | |
| *Tokudb_CACHETABLE_POOL_CLIENT_TOTAL_ITEMS_PROCESSED* | | |
| *Tokudb_CACHETABLE_POOL_CLIENT_TOTAL_EXECUTION_TIME* | | |
| *Tokudb_CACHETABLE_POOL_CACHETABLE_NUM_THREADS* | | |
| *Tokudb_CACHETABLE_POOL_CACHETABLE_NUM_THREADS_ACTIVE* | | |
| *Tokudb_CACHETABLE_POOL_CACHETABLE_QUEUE_SIZE* | | |
| *Tokudb_CACHETABLE_POOL_CACHETABLE_MAX_QUEUE_SIZE* | | |
| *Tokudb_CACHETABLE_POOL_CACHETABLE_TOTAL_ITEMS_PROCESSED* | | |
| *Tokudb_CACHETABLE_POOL_CACHETABLE_TOTAL_EXECUTION_TIME* | | |
| *Tokudb_CACHETABLE_POOL_CHECKPOINT_NUM_THREADS* | | |
| *Tokudb_CACHETABLE_POOL_CHECKPOINT_NUM_THREADS_ACTIVE* | | |
| *Tokudb_CACHETABLE_POOL_CHECKPOINT_QUEUE_SIZE* | | |

Continued on next page

Table 76.2 – continued from previous page

| Name | Var Type | Var Scope |
|---|---|---|
| *Tokudb_CACHETABLE_POOL_CHECKPOINT_MAX_QUEUE_SIZE* | | |
| *Tokudb_CACHETABLE_POOL_CHECKPOINT_TOTAL_ITEMS_PROCESSED* | | |
| *Tokudb_CACHETABLE_POOL_CHECKPOINT_TOTAL_EXECUTION_TIME* | | |
| *Tokudb_LOCKTREE_MEMORY_SIZE* | | |
| *Tokudb_LOCKTREE_MEMORY_SIZE_LIMIT* | | |
| *Tokudb_LOCKTREE_ESCALATION_NUM* | | |
| *Tokudb_LOCKTREE_ESCALATION_SECONDS* | | |
| *Tokudb_LOCKTREE_LATEST_POST_ESCALATION_MEMORY_SIZE* | | |
| *Tokudb_LOCKTREE_OPEN_CURRENT* | | |
| *Tokudb_LOCKTREE_PENDING_LOCK_REQUESTS* | | |
| *Tokudb_LOCKTREE_STO_ELIGIBLE_NUM* | | |
| *Tokudb_LOCKTREE_STO_ENDED_NUM* | | |
| *Tokudb_LOCKTREE_STO_ENDED_SECONDS* | | |
| *Tokudb_LOCKTREE_WAIT_COUNT* | | |
| *Tokudb_LOCKTREE_WAIT_TIME* | | |
| *Tokudb_LOCKTREE_LONG_WAIT_COUNT* | | |
| *Tokudb_LOCKTREE_LONG_WAIT_TIME* | | |
| *Tokudb_LOCKTREE_TIMEOUT_COUNT* | | |
| *Tokudb_LOCKTREE_WAIT_ESCALATION_COUNT* | | |
| *Tokudb_LOCKTREE_WAIT_ESCALATION_TIME* | | |
| *Tokudb_LOCKTREE_LONG_WAIT_ESCALATION_COUNT* | | |
| *Tokudb_LOCKTREE_LONG_WAIT_ESCALATION_TIME* | | |
| *Tokudb_DICTIONARY_UPDATES* | | |
| *Tokudb_DICTIONARY_BROADCAST_UPDATES* | | |
| *Tokudb_DESCRIPTOR_SET* | | |
| *Tokudb_MESSAGES_IGNORED_BY_LEAF_DUE_TO_MSN* | | |
| *Tokudb_TOTAL_SEARCH_RETRIES* | | |
| *Tokudb_SEARCH_TRIES_GT_HEIGHT* | | |
| *Tokudb_SEARCH_TRIES_GT_HEIGHTPLUS3* | | |
| *Tokudb_LEAF_NODES_FLUSHED_NOT_CHECKPOINT* | | |
| *Tokudb_LEAF_NODES_FLUSHED_NOT_CHECKPOINT_BYTES* | | |
| *Tokudb_LEAF_NODES_FLUSHED_NOT_CHECKPOINT_UNCOMPRESSED_BYTES* | | |
| *Tokudb_LEAF_NODES_FLUSHED_NOT_CHECKPOINT_SECONDS* | | |
| *Tokudb_NONLEAF_NODES_FLUSHED_TO_DISK_NOT_CHECKPOINT* | | |
| *Tokudb_NONLEAF_NODES_FLUSHED_TO_DISK_NOT_CHECKPOINT_BYTES* | | |
| *Tokudb_NONLEAF_NODES_FLUSHED_TO_DISK_NOT_CHECKPOINT_UNCOMPRESSE* | | |
| *Tokudb_NONLEAF_NODES_FLUSHED_TO_DISK_NOT_CHECKPOINT_SECONDS* | | |
| *Tokudb_LEAF_NODES_FLUSHED_CHECKPOINT* | | |
| *Tokudb_LEAF_NODES_FLUSHED_CHECKPOINT_BYTES* | | |
| *Tokudb_LEAF_NODES_FLUSHED_CHECKPOINT_UNCOMPRESSED_BYTES* | | |
| *Tokudb_LEAF_NODES_FLUSHED_CHECKPOINT_SECONDS* | | |
| *Tokudb_NONLEAF_NODES_FLUSHED_TO_DISK_CHECKPOINT* | | |
| *Tokudb_NONLEAF_NODES_FLUSHED_TO_DISK_CHECKPOINT_BYTES* | | |
| *Tokudb_NONLEAF_NODES_FLUSHED_TO_DISK_CHECKPOINT_UNCOMPRESSED_BY* | | |
| *Tokudb_NONLEAF_NODES_FLUSHED_TO_DISK_CHECKPOINT_SECONDS* | | |
| *Tokudb_LEAF_NODE_COMPRESSION_RATIO* | | |
| *Tokudb_NONLEAF_NODE_COMPRESSION_RATIO* | | |
| *Tokudb_OVERALL_NODE_COMPRESSION_RATIO* | | |

Continued on next page

Table 76.2 – continued from previous page

| Name | Var Type | Var Scope |
|---|---|---|
| *Tokudb_NONLEAF_NODE_PARTIAL_EVICTIONS* | | |
| *Tokudb_NONLEAF_NODE_PARTIAL_EVICTIONS_BYTES* | | |
| *Tokudb_LEAF_NODE_PARTIAL_EVICTIONS* | | |
| *Tokudb_LEAF_NODE_PARTIAL_EVICTIONS_BYTES* | | |
| *Tokudb_LEAF_NODE_FULL_EVICTIONS* | | |
| *Tokudb_LEAF_NODE_FULL_EVICTIONS_BYTES* | | |
| *Tokudb_NONLEAF_NODE_FULL_EVICTIONS* | | |
| *Tokudb_NONLEAF_NODE_FULL_EVICTIONS_BYTES* | | |
| *Tokudb_LEAF_NODES_CREATED* | | |
| *Tokudb_NONLEAF_NODES_CREATED* | | |
| *Tokudb_LEAF_NODES_DESTROYED* | | |
| *Tokudb_NONLEAF_NODES_DESTROYED* | | |
| *Tokudb_MESSAGES_INJECTED_AT_ROOT_BYTES* | | |
| *Tokudb_MESSAGES_FLUSHED_FROM_H1_TO_LEAVES_BYTES* | | |
| *Tokudb_MESSAGES_IN_TREES_ESTIMATE_BYTES* | | |
| *Tokudb_MESSAGES_INJECTED_AT_ROOT* | | |
| *Tokudb_BROADCASE_MESSAGES_INJECTED_AT_ROOT* | | |
| *Tokudb_BASEMENTS_DECOMPRESSED_TARGET_QUERY* | | |
| *Tokudb_BASEMENTS_DECOMPRESSED_PRELOCKED_RANGE* | | |
| *Tokudb_BASEMENTS_DECOMPRESSED_PREFETCH* | | |
| *Tokudb_BASEMENTS_DECOMPRESSED_FOR_WRITE* | | |
| *Tokudb_BUFFERS_DECOMPRESSED_TARGET_QUERY* | | |
| *Tokudb_BUFFERS_DECOMPRESSED_PRELOCKED_RANGE* | | |
| *Tokudb_BUFFERS_DECOMPRESSED_PREFETCH* | | |
| *Tokudb_BUFFERS_DECOMPRESSED_FOR_WRITE* | | |
| *Tokudb_PIVOTS_FETCHED_FOR_QUERY* | | |
| *Tokudb_PIVOTS_FETCHED_FOR_QUERY_BYTES* | | |
| *Tokudb_PIVOTS_FETCHED_FOR_QUERY_SECONDS* | | |
| *Tokudb_PIVOTS_FETCHED_FOR_PREFETCH* | | |
| *Tokudb_PIVOTS_FETCHED_FOR_PREFETCH_BYTES* | | |
| *Tokudb_PIVOTS_FETCHED_FOR_PREFETCH_SECONDS* | | |
| *Tokudb_PIVOTS_FETCHED_FOR_WRITE* | | |
| *Tokudb_PIVOTS_FETCHED_FOR_WRITE_BYTES* | | |
| *Tokudb_PIVOTS_FETCHED_FOR_WRITE_SECONDS* | | |
| *Tokudb_BASEMENTS_FETCHED_TARGET_QUERY* | | |
| *Tokudb_BASEMENTS_FETCHED_TARGET_QUERY_BYTES* | | |
| *Tokudb_BASEMENTS_FETCHED_TARGET_QUERY_SECONDS* | | |
| *Tokudb_BASEMENTS_FETCHED_PRELOCKED_RANGE* | | |
| *Tokudb_BASEMENTS_FETCHED_PRELOCKED_RANGE_BYTES* | | |
| *Tokudb_BASEMENTS_FETCHED_PRELOCKED_RANGE_SECONDS* | | |
| *Tokudb_BASEMENTS_FETCHED_PREFETCH* | | |
| *Tokudb_BASEMENTS_FETCHED_PREFETCH_BYTES* | | |
| *Tokudb_BASEMENTS_FETCHED_PREFETCH_SECONDS* | | |
| *Tokudb_BASEMENTS_FETCHED_FOR_WRITE* | | |
| *Tokudb_BASEMENTS_FETCHED_FOR_WRITE_BYTES* | | |
| *Tokudb_BASEMENTS_FETCHED_FOR_WRITE_SECONDS* | | |
| *Tokudb_BUFFERS_FETCHED_TARGET_QUERY* | | |
| *Tokudb_BUFFERS_FETCHED_TARGET_QUERY_BYTES* | | |

Continued on next page

Table 76.2 – continued from previous page

| Name | Var Type | Var Scope |
|---|---|---|
| *Tokudb_BUFFERS_FETCHED_TARGET_QUERY_SECONDS* | | |
| *Tokudb_BUFFERS_FETCHED_PRELOCKED_RANGE* | | |
| *Tokudb_BUFFERS_FETCHED_PRELOCKED_RANGE_BYTES* | | |
| *Tokudb_BUFFERS_FETCHED_PRELOCKED_RANGE_SECONDS* | | |
| *Tokudb_BUFFERS_FETCHED_PREFETCH* | | |
| *Tokudb_BUFFERS_FETCHED_PREFETCH_BYTES* | | |
| *Tokudb_BUFFERS_FETCHED_PREFETCH_SECONDS* | | |
| *Tokudb_BUFFERS_FETCHED_FOR_WRITE* | | |
| *Tokudb_BUFFERS_FETCHED_FOR_WRITE_BYTES* | | |
| *Tokudb_BUFFERS_FETCHED_FOR_WRITE_SECONDS* | | |
| *Tokudb_LEAF_COMPRESSION_TO_MEMORY_SECONDS* | | |
| *Tokudb_LEAF_SERIALIZATION_TO_MEMORY_SECONDS* | | |
| *Tokudb_LEAF_DECOMPRESSION_TO_MEMORY_SECONDS* | | |
| *Tokudb_LEAF_DESERIALIZATION_TO_MEMORY_SECONDS* | | |
| *Tokudb_NONLEAF_COMPRESSION_TO_MEMORY_SECONDS* | | |
| *Tokudb_NONLEAF_SERIALIZATION_TO_MEMORY_SECONDS* | | |
| *Tokudb_NONLEAF_DECOMPRESSION_TO_MEMORY_SECONDS* | | |
| *Tokudb_NONLEAF_DESERIALIZATION_TO_MEMORY_SECONDS* | | |
| *Tokudb_PROMOTION_ROOTS_SPLIT* | | |
| *Tokudb_PROMOTION_LEAF_ROOTS_INJECTED_INTO* | | |
| *Tokudb_PROMOTION_H1_ROOTS_INJECTED_INTO* | | |
| *Tokudb_PROMOTION_INJECTIONS_AT_DEPTH_0* | | |
| *Tokudb_PROMOTION_INJECTIONS_AT_DEPTH_1* | | |
| *Tokudb_PROMOTION_INJECTIONS_AT_DEPTH_2* | | |
| *Tokudb_PROMOTION_INJECTIONS_AT_DEPTH_3* | | |
| *Tokudb_PROMOTION_INJECTIONS_LOWER_THAN_DEPTH_3* | | |
| *Tokudb_PROMOTION_STOPPED_NONEMPTY_BUFFER* | | |
| *Tokudb_PROMOTION_STOPPED_AT_HEIGHT_1* | | |
| *Tokudb_PROMOTION_STOPPED_CHILD_LOCKED_OR_NOT_IN_MEMORY* | | |
| *Tokudb_PROMOTION_STOPPED_CHILD_NOT_FULLY_IN_MEMORY* | | |
| *Tokudb_PROMOTION_STOPPED_AFTER_LOCKING_CHILD* | | |
| *Tokudb_BASEMENT_DESERIALIZATION_FIXED_KEY* | | |
| *Tokudb_BASEMENT_DESERIALIZATION_VARIABLE_KEY* | | |
| *Tokudb_PRO_RIGHTMOST_LEAF_SHORTCUT_SUCCESS* | | |
| *Tokudb_PRO_RIGHTMOST_LEAF_SHORTCUT_FAIL_POS* | | |
| *Tokudb_RIGHTMOST_LEAF_SHORTCUT_FAIL_REACTIVE* | | |
| *Tokudb_CURSOR_SKIP_DELETED_LEAF_ENTRY* | | |
| *Tokudb_FLUSHER_CLEANER_TOTAL_NODES* | | |
| *Tokudb_FLUSHER_CLEANER_H1_NODES* | | |
| *Tokudb_FLUSHER_CLEANER_HGT1_NODES* | | |
| *Tokudb_FLUSHER_CLEANER_EMPTY_NODES* | | |
| *Tokudb_FLUSHER_CLEANER_NODES_DIRTIED* | | |
| *Tokudb_FLUSHER_CLEANER_MAX_BUFFER_SIZE* | | |
| *Tokudb_FLUSHER_CLEANER_MIN_BUFFER_SIZE* | | |
| *Tokudb_FLUSHER_CLEANER_TOTAL_BUFFER_SIZE* | | |
| *Tokudb_FLUSHER_CLEANER_MAX_BUFFER_WORKDONE* | | |
| *Tokudb_FLUSHER_CLEANER_MIN_BUFFER_WORKDONE* | | |
| *Tokudb_FLUSHER_CLEANER_TOTAL_BUFFER_WORKDONE* | | |

Continued on next page

Table 76.2 – continued from previous page

| Name | Var Type | Var Scope |
|---|---|---|
| *Tokudb_FLUSHER_CLEANER_NUM_LEAF_MERGES_STARTED* | | |
| *Tokudb_FLUSHER_CLEANER_NUM_LEAF_MERGES_RUNNING* | | |
| *Tokudb_FLUSHER_CLEANER_NUM_LEAF_MERGES_COMPLETED* | | |
| *Tokudb_FLUSHER_CLEANER_NUM_DIRTIED_FOR_LEAF_MERGE* | | |
| *Tokudb_FLUSHER_FLUSH_TOTAL* | | |
| *Tokudb_FLUSHER_FLUSH_IN_MEMORY* | | |
| *Tokudb_FLUSHER_FLUSH_NEEDED_IO* | | |
| *Tokudb_FLUSHER_FLUSH_CASCADES* | | |
| *Tokudb_FLUSHER_FLUSH_CASCADES_1* | | |
| *Tokudb_FLUSHER_FLUSH_CASCADES_2* | | |
| *Tokudb_FLUSHER_FLUSH_CASCADES_3* | | |
| *Tokudb_FLUSHER_FLUSH_CASCADES_4* | | |
| *Tokudb_FLUSHER_FLUSH_CASCADES_5* | | |
| *Tokudb_FLUSHER_FLUSH_CASCADES_GT_5* | | |
| *Tokudb_FLUSHER_SPLIT_LEAF* | | |
| *Tokudb_FLUSHER_SPLIT_NONLEAF* | | |
| *Tokudb_FLUSHER_MERGE_LEAF* | | |
| *Tokudb_FLUSHER_MERGE_NONLEAF* | | |
| *Tokudb_FLUSHER_BALANCE_LEAF* | | |
| *Tokudb_HOT_NUM_STARTED* | | |
| *Tokudb_HOT_NUM_COMPLETED* | | |
| *Tokudb_HOT_NUM_ABORTED* | | |
| *Tokudb_HOT_MAX_ROOT_FLUSH_COUNT* | | |
| *Tokudb_TXN_BEGIN* | | |
| *Tokudb_TXN_BEGIN_READ_ONLY* | | |
| *Tokudb_TXN_COMMITS* | | |
| *Tokudb_TXN_ABORTS* | | |
| *Tokudb_LOGGER_NEXT_LSN* | | |
| *Tokudb_LOGGER_WRITES* | | |
| *Tokudb_LOGGER_WRITES_BYTES* | | |
| *Tokudb_LOGGER_WRITES_UNCOMPRESSED_BYTES* | | |
| *Tokudb_LOGGER_WRITES_SECONDS* | | |
| *Tokudb_LOGGER_WAIT_LONG* | | |
| *Tokudb_LOADER_NUM_CREATED* | | |
| *Tokudb_LOADER_NUM_CURRENT* | | |
| *Tokudb_LOADER_NUM_MAX* | | |
| *Tokudb_MEMORY_MALLOC_COUNT* | | |
| *Tokudb_MEMORY_FREE_COUNT* | | |
| *Tokudb_MEMORY_REALLOC_COUNT* | | |
| *Tokudb_MEMORY_MALLOC_FAIL* | | |
| *Tokudb_MEMORY_REALLOC_FAIL* | | |
| *Tokudb_MEMORY_REQUESTED* | | |
| *Tokudb_MEMORY_USED* | | |
| *Tokudb_MEMORY_FREED* | | |
| *Tokudb_MEMORY_MAX_REQUESTED_SIZE* | | |
| *Tokudb_MEMORY_LAST_FAILED_SIZE* | | |
| *Tokudb_MEM_ESTIMATED_MAXIMUM_MEMORY_FOOTPRINT* | | |
| *Tokudb_MEMORY_MALLOCATOR_VERSION* | | |

Continued on next page

Table 76.2 – continued from previous page

| Name | Var Type | Var Scope |
|---|---|---|
| *Tokudb_MEMORY_MMAP_THRESHOLD* | | |
| *Tokudb_FILESYSTEM_THREADS_BLOCKED_BY_FULL_DISK* | | |
| *Tokudb_FILESYSTEM_FSYNC_TIME* | | |
| *Tokudb_FILESYSTEM_FSYNC_NUM* | | |
| *Tokudb_FILESYSTEM_LONG_FSYNC_TIME* | | |
| *Tokudb_FILESYSTEM_LONG_FSYNC_NUM* | | |

CHAPTER

SEVENTYSEVEN

# DEVELOPMENT OF *PERCONA SERVER FOR MYSQL*

*Percona Server for MySQL* is an open source project to produce a distribution of the *MySQL* Server with improved performance, scalability and diagnostics.

## Submitting Changes

We keep trunk in a constant state of stability to allow for a release at any time and to minimize wasted time by developers due to broken code.

### Overview

At Percona we use Git for source control, GitHub for code hosting, and Jira for release management.

We change our software to implement new features or to fix bugs. Refactoring could be classed either as a new feature or a bug depending on the scope of work.

New features and bugs are targeted to specific milestones (releases). A milestone is part of a series. For example, 1.6 is a series in Percona XtraBackup and 1.6.1, 1.6.2 and 1.6.3 are milestones in this series.

Code is proposed for merging in the form of pull requests on GitHub.

For some software (such as Percona Xtrabackup), we maintain both a development branch and a stable branch. For example: Xtrabackup 1.6 is the current stable series. Changes that should make it into bugfix releases of 1.6 should be proposed for the 1.6 tree. However, most new features or more invasive (or smaller) bug fixes should be targeted to the next release, in this example - 1.7. If submitting something to the stable release, you should also propose a branch that has these changes merged to the development release. This way somebody else doesn't have to attempt to merge your code and we get to run any extra tests that may be in the tree (and check compatibility with all platforms).

For *Percona Server for MySQL*, we have several Git branches on which development occurs: 5.5, 5.6, 5.7, and 8.0. As *Percona Server for MySQL* is not a traditional project, instead of being a set of patches against an existing product, these branches are not related. In other words, we do not merge from one release branch to another. To have your changes in several branches, you must propose branches to each release branch.

### Making a Change to a Project

In this case, we are going to use `percona-xtrabackup` as an example. The workflow is similar for *Percona Server for MySQL*, but the patch will need to be modified in all release branches of *Percona Server for MySQL*.

- `git branch https://github.com/percona/percona-xtrabackup/featureX` (where 'featureX' is a sensible name for the task at hand)
- (developer makes changes in featureX, testing locally)

- The Developer pushes to `https://github.com/percona/username/percona-xtrabackup/featureX`
- The developer can submit a pull request to https://github.com/percona/percona-xtrabackup,
- Code undergoes a review
- Once code is accepted, it can be merged

If the change also applies to a stable release (e.g. 1.6) then changes should be made on a branch of 1.6 and merged to a branch of trunk. In this case there should be two branches run through the param build and two merge proposals (one for the stable release and one with the changes merged to trunk). This prevents somebody else having to guess how to merge your changes.

### *Percona Server for MySQL*

The same process for *Percona Server for MySQL*, but we have several different branches (and merge requests).

CHAPTER

SEVENTYEIGHT

# TRADEMARK POLICY

This Trademark Policy is to ensure that users of Percona-branded products or services know that what they receive has really been developed, approved, tested and maintained by Percona. Trademarks help to prevent confusion in the marketplace, by distinguishing one company's or person's products and services from another's.

Percona owns a number of marks, including but not limited to Percona, XtraDB, Percona XtraDB, XtraBackup, Percona XtraBackup, *Percona Server for MySQL*, and Percona Live, plus the distinctive visual icons and logos associated with these marks. Both the unregistered and registered marks of Percona are protected.

Use of any Percona trademark in the name, URL, or other identifying characteristic of any product, service, website, or other use is not permitted without Percona's written permission with the following three limited exceptions.

*First*, you may use the appropriate Percona mark when making a nominative fair use reference to a bona fide Percona product.

*Second*, when Percona has released a product under a version of the GNU General Public License ("GPL"), you may use the appropriate Percona mark when distributing a verbatim copy of that product in accordance with the terms and conditions of the GPL.

*Third*, you may use the appropriate Percona mark to refer to a distribution of GPL-released Percona software that has been modified with minor changes for the sole purpose of allowing the software to operate on an operating system or hardware platform for which Percona has not yet released the software, provided that those third party changes do not affect the behavior, functionality, features, design or performance of the software. Users who acquire this Percona-branded software receive substantially exact implementations of the Percona software.

Percona reserves the right to revoke this authorization at any time in its sole discretion. For example, if Percona believes that your modification is beyond the scope of the limited license granted in this Policy or that your use of the Percona mark is detrimental to Percona, Percona will revoke this authorization. Upon revocation, you must immediately cease using the applicable Percona mark. If you do not immediately cease using the Percona mark upon revocation, Percona may take action to protect its rights and interests in the Percona mark. Percona does not grant any license to use any Percona mark for any other modified versions of Percona software; such use will require our prior written permission.

Neither trademark law nor any of the exceptions set forth in this Trademark Policy permit you to truncate, modify or otherwise use any Percona mark as part of your own brand. For example, if XYZ creates a modified version of the *Percona Server for MySQL*, XYZ may not brand that modification as "XYZ Percona Server" or "Percona XYZ Server", even if that modification otherwise complies with the third exception noted above.

In all cases, you must comply with applicable law, the underlying license, and this Trademark Policy, as amended from time to time. For instance, any mention of Percona trademarks should include the full trademarked name, with proper spelling and capitalization, along with attribution of ownership to Percona Inc. For example, the full proper name for XtraBackup is Percona XtraBackup. However, it is acceptable to omit the word "Percona" for brevity on the second and subsequent uses, where such omission does not cause confusion.

In the event of doubt as to any of the conditions or exceptions outlined in this Trademark Policy, please contact trademarks@percona.com for assistance and we will do our very best to be helpful.

CHAPTER

SEVENTYNINE

# INDEX OF INFORMATION_SCHEMA TABLES

This is a list of the INFORMATION_SCHEMA TABLES that exist in *Percona Server for MySQL* with *XtraDB*. The entry for each table points to the page in the documentation where it's described.

- *CLIENT_STATISTICS*
- *GLOBAL_TEMPORARY_TABLES*
- *INDEX_STATISTICS*
- *INNODB_CHANGED_PAGES*
- QUERY_RESPONSE_TIME
- *TABLE_STATISTICS*
- *TEMPORARY_TABLES*
- *THREAD_STATISTICS*
- *USER_STATISTICS*
- *XTRADB_INTERNAL_HASH_TABLES*
- *XTRADB_READ_VIEW*
- XTRADB_RSEG
- XTRADB_ZIP_DICT
- XTRADB_ZIP_DICT_COLS

CHAPTER

EIGHTY

# FREQUENTLY ASKED QUESTIONS

## Q: Will *Percona Server for MySQL* with *XtraDB* invalidate our *MySQL* support?

A: We don't know the details of your support contract. You should check with your *Oracle* representative. We have heard anecdotal stories from *MySQL* Support team members that they have customers who use *Percona Server for MySQL* with *XtraDB*, but you should not base your decision on that.

## Q: Will we have to *GPL* our whole application if we use *Percona Server for MySQL* with *XtraDB*?

A: This is a common misconception about the *GPL*. We suggest reading the *Free Software Foundation* 's excellent reference material on the GPL Version 2, which is the license that applies to *MySQL* and therefore to *Percona Server for MySQL* with *XtraDB*. That document contains links to many other documents which should answer your questions. *Percona* is unable to give legal advice about the *GPL*.

## Q: Do I need to install *Percona* client libraries?

A: No, you don't need to change anything on the clients. *Percona Server for MySQL* is 100% compatible with all existing client libraries and connectors.

## Q: When using the *Percona XtraBackup* to setup a replication slave on Debian based systems I'm getting: "ERROR 1045 (28000): Access denied for user 'debian-sys-maint'@'localhost' (using password: YES)"

A: In case you're using init script on Debian based system to start `mysqld`, be sure that the password for `debian-sys-maint` user has been updated and it's the same as that user's password from the server that the backup has been taken from. The password can be seen and updated in `/etc/mysql/debian.cnf`. For more information on how to set up a replication slave using *Percona XtraBackup* see this how-to.

CHAPTER

EIGHTYONE

# COPYRIGHT AND LICENSING INFORMATION

## Documentation Licensing



## Software License

*Percona Server for MySQL* is built upon MySQL from Oracle. Along with making our own modifications, we merge in changes from other sources such as community contributions and changes from MariaDB.

The original SHOW USER/TABLE/INDEX statistics code came from Google.

Percona does not require copyright assignment.

See the COPYING files accompanying the software distribution.

CHAPTER

EIGHTYTWO

# *PERCONA SERVER FOR MYSQL* 8.0 RELEASE NOTES

## *Percona Server for MySQL* 8.0.18-9

*Percona* announces the release of *Percona Server for MySQL* 8.0.18-9 on December 11, 2019 (downloads are available here and from the Percona Software Repositories).

This release includes fixes to bugs found in previous releases of *Percona Server for MySQL* 8.0.

*Percona Server for MySQL* 8.0.18-9 is now the current GA release in the 8.0 series. All of *Percona*'s software is open-source and free.

Percona Server for MySQL 8.0 includes all the features available in MySQL 8.0.18 Community Edition in addition to enterprise-grade features developed by Percona.

### Bugs Fixed

- Setting the `none` value for *`slow_query_log_use_global_control`* generates an error. Bugs fixed #5813.
- If pam_krb5 allows the user to change their password, and the password expired, a new password may cause a server exit. Bug fixed #6023.
- An incorrect assertion was triggered if any temporary tables should be logged to binlog. This event may cause a server exit. Bug fixed #5181.
- The Handler failed to trigger on Error 1049, SQLSTATE 42000, or plain sqlexception. Bug fixed #6094. (Upstream #97682)
- When executing `SHOW GLOBAL STATUS`, the variables may return incorrect values. Bug fixed #5966.
- The memory storage engine detected an incorrect `full` condition even though the space contained reusable memory chunks released by deleted records and the space could be reused. Bug fixed #1469.

Other bugs fixed:

#6051, #5876, #5996, #6021, #6052, #4775, #5836 (Upstream #96449), #6123, #5819, #5836, #6054, #6056, #6058, #6059, #6078, #6057, #6111, #6117, and #6073.

## *Percona Server for MySQL* 8.0.17-8

*Percona* announces the release of *Percona Server for MySQL* 8.0.17-8 on October 30, 2019 (downloads are available here and from the Percona Software Repositories).

This release includes fixes to bugs found in previous releases of *Percona Server for MySQL* 8.0.

*Percona Server for MySQL* 8.0.17-8 is now the current GA release in the 8.0 series. All of *Percona*'s software is open-source and free.

Percona Server for MySQL 8.0 includes all the features available in MySQL 8.0.17 Community Edition in addition to enterprise-grade features developed by Percona.

### New Features

*Percona Server for MySQL* has implemented the ability to have a *MySQL Utility user* who has system access to do administrative tasks but limited access to user schemas. The user is invisible to other users. This feature is especially useful to those who are operating *MySQL* as a Service. This feature has the same functionality as the utility user in earlier versions and has been delay-ported to version 8.0.

*Percona Server for MySQL* has implemented *Data Masking*.

### Bugs Fixed

- Changed the default of *innodb_empty_free_list_algorithm* to `backoff`. Bugs fixed #5881
- When the Adaptive Hash Index (AHI) was enabled or disabled, there was an AHI overhead during DDL operations. Bugs fixed #5747.
- An upgrade to `8.0.16-7` with encrypted tablespace fails on `innodb_dynamic_metadata`. Bugs fixed #5874.
- The `rocksdb.ttl_primary` test case sometimes fails. Bugs fixed #5722 (Louis Hust)
- The `rocksdb.ns_snapshot_read_committed` test case sometimes fails. Bugs fixed #5798 (Louis Hust).
- During a binlogging replication event, if the master crashes after the multi-threaded slave has begun copying to the slave's relay log and before the process has completed, a `STOP SLAVE` on the slave takes longer than expected. Bugs fixed #5824.
- The purpose of the sql_require_primary_key option is to avoid replication performance issues. Temporary tables are not replicated. The option cannot be used with temporary tables. Bugs fixed #5931.
- When using `skip-innodb_doublewrite` in my.cnf, a parallel doublewrite buffer is still created. Bugs fixed #3411.
- The metadata for every InnoDB table contains encryption information, either a 'Y' or an 'N' value based on the ENCRYPTION clause or the *default_table_encryption* value. You are unable to switch the storage engine from InnoDB to MyRocks because MyRocks does not support the ENCRYPTION clause. Bugs fixed #5865.
- MyRocks does not allow index condition pushdown optimization for specific data types, such as `varchar`. Bugs fixed #5024.

Other bugs fixed: #5880, #5427, #5838, #5682, #5979, #5793, #6020, #6025, #5327, #5839, #5933, #5939, #5659, #5924, #5687, #5926, #5925, #5875, #5533, #5867, #5864, #5760, #5909, #5985, #5941, #5954, #5790, and #5593.

## *Percona Server for MySQL* 8.0.16-7

*Percona* announces the release of *Percona Server for MySQL* 8.0.16-7 on August 15, 2019 (downloads are available here and from the Percona Software Repositories). This release includes fixes to bugs found in previous releases of *Percona Server for MySQL* 8.0. *Percona Server for MySQL* 8.0.16-7 is now the current GA release in the 8.0 series. All of *Percona*'s software is open-source and free.

Percona Server for MySQL 8.0 includes all the features and bug fixes available in MySQL 8.0.16 Community Edition in addition to enterprise-grade features developed by Percona.

## Encryption Features General Availability (GA)

- *Encrypting Temporary Files*
- *Encrypting the Undo Tablespace*
- *Encrypting the System Tablespace*
- *default_table_encryption* =OFF/ON
- table_encryption_privilege_check =OFF/ON
- *Encrypting the Redo Log* for master key encryption only
- *Merge-sort-encryption*
- *Encrypting Doublewrite Buffers*

## Bugs Fixed

- Parallel doublewrite buffer writes must crash the server on an I/O error occurs. Bug fixed #5678.
- After resetting the *innodb_temp_tablespace_encrypt* to OFF during runtime the subsequent file-per-table temporary tables continue to be encrypted. Bug fixed #5734.
- Setting the encryption to ON for the system tablespace generates an encryption key and encrypts system temporary tablespace pages. Resetting the encryption to OFF, all subsequent pages are written to the temporary tablespace without encryption. To allow any encrypted tables to be decrypted, the generated keys are not erased. Modifying the *innodb_temp_tablespace_encrypt* does not affect file-per-table temporary tables. This type of table is encrypted if ENCRYPTION ='Y' is set during table creation. Bug fixed #5736.
- An instance started with the default values but setting the redo-log without specifying the keyring plugin parameters does not fail or throw an error. Bug fixed #5476.
- The *rocksdb_large_prefix* allows index key prefixes up to 3072 bytes. The default value is changed to TRUE to match the behavior of the innodb_large_prefix. #5655.
- On a server with a large number of tables, a shutdown may take a measurable length of time. Bug fixed #5639.
- The changed page tracking uses the LOG flag during read operations. The redo log encryption may attempt to decrypt pages with a specific bit set and fail. This failure generates error messages. A NO_ENCRYPTION flag lets the read process safely disable decryption errors in this case. Bug fixed #5541.
- If large pages are enabled on MySQL side, the maximum size for innodb_buffer_pool_chunk_size is effectively limited to 4GB. Bug fixed #5517. (Upstream 94747)
- The TokuDB hot backup library continually dumps TRACE information to the server error log. The user cannot enable or disable the dump of this information. Bug fixed #4850.

Other bugs fixed: #5688, #5723, #5695, #5749, #5752, #5610, #5689, #5645, #5734, #5772, #5753, #5129, #5102, #5681, #5686, #5681, #5310, #5713, #5007, #5102, #5129, #5130, #5149, #5696, #3845, #5149, #5581, #5652, #5662, #5697, #5775, #5668, #5752, #5782, #5767, #5669, #5753, #5696, #5733, #5803, #5804, #5820, #5827, #5835, #5724, #5767, #5782, #5794, #5796, #5746 and, #5748.

### Known Issues

- #5865: *Percona Server for MySQL* 8.0.16-7 does not support encryption for the MyRocks storage engine. An attempt to move any table from InnoDB to MyRocks fails as MyRocks currently sees all InnoDB tables as being encrypted.

## *Percona Server for MySQL* 8.0.15-6

*Percona* announces the release of *Percona Server for MySQL* 8.0.15-6 on May 07, 2019 (downloads are available here and from the Percona Software Repositories).

This release includes fixes to bugs found in previous releases of *Percona Server for MySQL* 8.0.

*Percona Server for MySQL* 8.0.15-6 is now the current GA release in the 8.0 series. All of *Percona*'s software is open-source and free.

Percona Server for MySQL 8.0 includes all the features available in MySQL 8.0 Community Edition in addition to enterprise-grade features developed by Percona. For a list of highlighted features from both MySQL 8.0 and Percona Server for MySQL 8.0, please see the GA release announcement.

---

**Note:** If you are upgrading from 5.7 to 8.0, please ensure that you read the upgrade guide and the document Changed in Percona Server for MySQL 8.0.

---

### New Features

- The server part of MyRocks cross-engine consistent physical backups has been implemented by introducing `rocksdb_disable_file_deletions` and `rocksdb_create_temporary_checkpoint` session variables. These variables are intended to be used by backup tools. Prolonged use or other misuse can have serious side effects to the server instance.
- RocksDB WAL file information can now be seen in the `performance_schema.log_status` *table*.
- New `Audit_log_buffer_size_overflow` status variable has been implemented to track when an *Audit Log Plugin* entry was either dropped or written directly to the file due to its size being bigger than `audit_log_buffer_size` variable.

### Bugs Fixed

- TokuDB and MyRocks native partitioning handler objects were allocated from a wrong memory allocator. Memory was released only on shutdown and concurrent access to global memory allocator caused memory corruptions and therefore crashes. Bug fixed #5508.
- using TokuDB or MyRocks native partitioning and `index_merge` could lead to a server crash. Bugs fixed #5206, #5562.
- upgrade from *Percona Server for MySQL* 5.7.24 to `8.0.13-3` wasn't working with encrypted undo tablespaces. Bug fixed #5223.
- Keyring Vault plugin couldn't be initialized on *Ubuntu Cosmic 17.10*. Bug fixed #5453.
- rotated key encryption did not register `encryption_key_id` as a valid table option. Bug fixed #5482.
- `INFORMATION_SCHEMA.GLOBAL_TEMPORARY_TABLES` queries could crash if online `ALTER TABLE` was running in parallel. Bug fixed #5566.

- setting the *log_slow_verbosity* to include innodb value and enabling the slow_query_log could lead to a server crash. Bug fixed #4933.
- *Compression dictionary support* operations were not allowed under innodb-force-recovery. Now they work correctly when innodb_force_recovery is <= 2, and are forbidden when innodb_force_recovery is >= 3. Bug fixed #5148.
- BLOB entries in the binary log could become corrupted in case when a database with Blackhole tables served as an intermediate binary log server in a replication chain. Bug fixed #5353.
- FLUSH CHANGED_PAGE_BITMAPS would leave gaps between the last written bitmap LSN and the *InnoDB* checkpoint LSN. Bug fixed #5446.
- *XtraDB changed page tracking* was missing pages changed by the in-place DDL. Bug fixed #5447.
- innodb_system tablespace information was missing from the INFORMATION_SCHEMA.innodb_tablespaces view. Bug fixed #5473.
- undo log tablespace encryption status is now available through INFORMATION_SCHEMA.innodb_tablespaces view. Bug fixed #5485 (upstream #94665).
- enabling temporay tablespace encryption didn't mark the innodb_temporary tablespace with the encryption flag. Bug fixed #5490.
- server would crash during bootstrap if *innodb_encrypt_tables* was set to 1. Bug fixed #5492.
- fixed intermittent shutdown crashes that were happening if *Thread Pool* was enabled. Bug fixed #5510.
- compression dictionary INFORMATION_SCHEMA views were missing when datadir was upgraded from 8.0.13 to 8.0.15. Bug fixed #5529.
- *innodb_encrypt_tables* variable accepted FORCE option only as a string. Bug fixed #5538.
- ibd2sdi utility was missing in Debian/Ubuntu packages. Bug fixed #5549.
- Docker image is now ignoring password that is set in the configuration file when first initializing. Bug fixed #5573.
- long running ALTER TABLE ADD INDEX could cause a semaphore wait > 600 assertion. Bug fixed #3410 (upstream #82940).
- system keyring keys initialization wasn't thread safe. Bugs fixed #5554.
- *Backup Locks* was blocking DML for RocksDB. Bug fixed #5583.
- PerconaFT locktree library was re-licensed to Apache v2 license. Bug fixed #5501.

Other bugs fixed: #5537, #5243, #5371, #5475, #5484, #5512, #5514, #5523, #5528, #5536, #5550, #5570, #5578, #5441, #5442, #5456, #5462, #5487, #5489, #5520, and #5560.

## *Percona Server for MySQL* 8.0.15-5

*Percona* announces the release of *Percona Server for MySQL* 8.0.15-5 on March 15, 2019 (downloads are available here and from the Percona Software Repositories).

This release includes fixes to bugs found in previous releases of *Percona Server for MySQL* 8.0.

**Incompatible changes**

In previous releases, the audit log used to produce time stamps inconsistent with the ISO 8601 standard. Release 8.0.15-5 of *Percona Server for MySQL* solves this problem. This change, however, may break programs that rely on the old time stamp format.

---

Starting from release `8.0.15-5`, *Percona Server for MySQL* uses the upstream implementation of binary log encryption. The variable `encrypt_binlog` is removed and the related command line option `--encrypt_binlog` is not supported. It is important that you remove the `encrypt_binlog` variable from your configuration file before you attempt to upgrade either from another release in the *Percona Server for MySQL* 8.0 series or from *Percona Server for MySQL* 5.7. Otherwise, a server boot error will be produced reporting an unknown variable. The implemented binary log encryption is compatible with the old format: the binary log encrypted in a previous version of MySQL 8.0 series or Percona Server for MySQL are supported.

**See also:**

***MySQL* Documentation**

- Encrypting Binary Log Files and Relay Log Files
- binlog_encryption variable

This release is based on *MySQL* 8.0.14 and 8.0.15. It includes all bug fixes in these releases. *Percona Server for MySQL* `8.0.14` was skipped.

*Percona Server for MySQL* 8.0.15-5 is now the current GA release in the 8.0 series. All of *Percona*'s software is open-source and free.

Percona Server for MySQL 8.0 includes all the features available in MySQL 8.0 Community Edition in addition to enterprise-grade features developed by Percona. For a list of highlighted features from both MySQL 8.0 and Percona Server for MySQL 8.0, please see the GA release announcement.

---

**Note:** If you are upgrading from 5.7 to 8.0, please ensure that you read the upgrade guide and the document Changed in Percona Server for MySQL 8.0.

---

## Bugs Fixed

- The audit log produced time stamps inconsistent with the ISO8601 standard. Bug fixed #226.
- FLUSH commands written to the binary log could cause errors in case of replication. Bug fixed #1827 (upstream #88720).
- When *audit_plugin* was enabled, the server could use a lot of memory when handling large queries. Bug fixed #5395.
- The page cleaner could sleep for long time when the system clock was adjusted to an earlier point in time. Bug fixed #5221 (upstream #93708).
- In some cases, the MyRocks storage engine could crash without triggering the crash recovery. Bug fixed #5366.
- In some cases, when it failed to read from a file, InnoDB did not inform the name of the file in the related error message. Bug fixed #2455 (upstream #76020).
- The `ACCESS_DENIED` field of the `information_schema.user_statistics` table was not updated correctly. Bugs fixed #3956, #4996.
- `MyRocks` could crash while running `START TRANSACTION WITH CONSISTENT SNAPSHOT` if other transactions were in specific states. Bug fixed #4705.
- In some cases, the server using the the `MyRocks` storage engine could crash when TTL (Time to Live) was defined on a table. Bug fixed #4911.

- MyRocks incorrectly processed transactions in which multiple statements had to be rolled back. Bug fixed #5219.
- A stack buffer overrun could happen if the redo log encryption with key rotation was enabled. Bug fixed #5305.
- The TokuDB storage engine would assert on load when used with jemalloc 5.x. Bug fixed #5406.

Other bugs fixed: #4106, #4107, #4108, #4121, #4474, #4640, #5055, #5218, #5263, #5328, #5369.

## *Percona Server for MySQL* 8.0.14

Due to a critical fix, MySQL Community Server 8.0.15 was released shortly (11 days later) after MySQL Community Server 8.0.14. *Percona* has skipped the release of *Percona Server for MySQL* 8.0.14. The next release of *Percona Server for MySQL* is `8.0.15-5` which contains all bug fixes and contents of both MySQL Community Server 8.0.14 and MySQL Community Server 8.0.15.

*Percona Server for MySQL* 8.0 includes all the features available in MySQL 8.0 Community Edition in addition to enterprise-grade features developed by Percona. For a list of highlighted features from both MySQL 8.0 and Percona Server for MySQL 8.0, please see the GA release announcement.

---

**Note:** If you are upgrading from 5.7 to 8.0, please ensure that you read the upgrade guide and the document Changed in Percona Server for MySQL 8.0.

---

## *Percona Server for MySQL* 8.0.13-4

*Percona* announces the release of *Percona Server for MySQL* 8.0.13-4 on January 17, 2019 (downloads are available here and from the Percona Software Repositories). This release contains a fix for a critical bug that prevented *Percona Server for MySQL* 5.7.24-26 from being upgraded to version 8.0.13-3 if there were more than around 1000 tables, or if the maximum allocated InnoDB table ID was around 1000. *Percona Server for MySQL* 8.0.13-4 is now the current GA release in the 8.0 series. All of *Percona*'s software is open-source and free.

Percona Server for MySQL 8.0 includes all the features available in MySQL 8.0 Community Edition in addition to enterprise-grade features developed by Percona. For a list of highlighted features from both MySQL 8.0 and Percona Server for MySQL 8.0, please see the GA release announcement.

---

**Note:** If you are upgrading from 5.7 to 8.0, please ensure that you read the upgrade guide and the document Changed in Percona Server for MySQL 8.0.

---

### Bugs Fixed

- It was not possible to upgrade from MySQL 5.7.24-26 to 8.0.13-3 if there were more than around 1000 tables, or if the maximum allocated InnoDB table ID was around 1000. Bug fixed #5245.
- `SHOW BINLOG EVENTS FROM <bad offset>` is not diagnosed inside `Format_description_log_events`. Bug fixed #5126 (Upstream #93544).
- There was a typo in *mysqld_safe.sh*: **trottling** was replaced with **throttling**. Bug fixed #240. Thanks to Michael Coburn for the patch.

- *Percona Server for MySQL* 8.0 could crash with the "Assertion failure: dict0dict.cc:7451:space_id != SPACE_UNKNOWN" exception during an upgrade from *Percona Server for MySQL* 5.7.23 to *Percona Server for MySQL* 8.0.13-3 with `--innodb_file_per_table=OFF`. Bug fixed #5222.
- On Debian or Ubuntu, a conflict was reported on the `/usr/bin/innochecksum` file when attempting to install *Percona Server for MySQL* 8 over the *MySQL* 8. Bug fixed #5225.
- An out-of-bound read exception could occur on debug builds in the compressed columns with dictionaries feature. Bug fixed #5311:.
- The `innodb_data_pending_reads` server status variable contained an incorrect value. Bug fixed #5264:. Thanks to Fangxin Lou for the patch.
- **A memory leak and needless allocation in compression dictionaries** could happen in `mysqldump`. Bug fixed #5307.
- A compression-related memory leak could happen in `mysqlbinlog`. Bug fixed #5308:.

Other bugs fixed: #4797:, #5209, #5268, #5270:, #5306, #5309:

## *Percona Server for MySQL* 8.0.13-3

*Percona* announces the GA release of *Percona Server for MySQL* 8.0.13-3 on December 21, 2018 (downloads are available here and from the Percona SoftwareRepositories). This release merges changes of *MySQL* 8.0.13, including all the bug fixes in it. *Percona Server for MySQL* 8.0.13-3 is now the current GA release in the 8.0 series. All of *Percona*'s software is open-source and free.

Percona Server for MySQL 8.0 includes all the features available in MySQL 8.0 Community Edition in addition to enterprise-grade features developed by Percona. For a list of highlighted features from both MySQL 8.0 and Percona Server for MySQL 8.0, please see the GA release announcement.

---

**Note:** If you are upgrading from 5.7 to 8.0, please ensure that you read the upgrade guide and the document Changed in Percona Server for MySQL 8.0.

---

### Features Removed in Percona Server for MySQL 8.0

- Slow Query Log Rotation and Expiration: Not widely used, can be accomplished using `logrotate`
- CSV engine mode for standard-compliant quote and comma parsing
- Expanded program option modifiers
- The `ALL_O_DIRECT` InnoDB flush method: it is not compatible with the new redo logging implementation
- `XTRADB_RSEG` table from `INFORMATION_SCHEMA`
- InnoDB memory size information from `SHOW ENGINE INNODB STATUS;` the same information is available from Performance Schema memory summary tables
- Query cache enhancements: The query cache is no longer present in MySQL 8.0

### Features Being Deprecated in Percona Server for MySQL 8.0

- *TokuDB* Storage Engine: *TokuDB* will be supported throughout the *Percona Server for MySQL* 8.0 release series, but will not be available in the next major release. *Percona* encourages *TokuDB* users to explore the *MyRocks*

Storage Engine which provides similar benefits for the majority of workloads and has better optimized support for modern hardware.

## Issues Resolved in *Percona Server for MySQL* 8.0.13-3

### Improvements

- #5014: Update Percona Backup Locks feature to use the new `BACKUP_ADMIN` privilege in MySQL 8.0
- #4805: Re-Implemented Compressed Columns with Dictionaries feature in PS 8.0
- #4790: Improved accuracy of User Statistics feature

### Bugs Fixed Since 8.0.12-rc1

- Fixed a crash in `mysqldump` in the `--innodb-optimize-keys` functionality #4972
- Fixed a crash that can occur when system tables are locked by the user due to a `lock_wait_timeout` #5134
- Fixed a crash that can occur when system tables are locked by the user from a `SELECT FOR UPDATE` statement #5027
- Fixed a bug that caused `innodb_buffer_pool_size` to be uninitialized after a restart if it was set using `SET PERSIST` #5069
- Fixed a crash in TokuDB that can occur when a temporary table experiences an autoincrement rollover #5056
- Fixed a bug where marking an index as invisible would cause a table rebuild in TokuDB and also in MyRocks #5031
- Fixed a bug where audit logs could get corrupted if the `audit_log_rotations` was changed during runtime. #4950
- Fixed a bug where `LOCK INSTANCE FOR BACKUP` and `STOP SLAVE SQL_THREAD` would cause replication to be blocked and unable to be restarted. #4758 (Upstream #93649)

Other Bugs Fixed:

#5155, #5139, #5057, #5049, #4999, #4971, #4943, #4918, #4917, #4898, and #4744.

## Known Issues

We have a few features and issues outstanding that should be resolved in the next release.

### Pending Feature Re-Implementations and Improvements

- #4892: Re-Implement Expanded Fast Index Creation feature.
- #5216: Re-Implement Utility User feature.
- #5143: Identify Percona features which can make use of dynamic privileges instead of `SUPER`

### Notable Issues in Features

- #5148: Regression in Compressed Columns Feature when using `innodb-force-recovery`
- #4996: Regression in User Statistics feature where `TOTAL_CONNECTIONS` field report incorrect data
- #4933: Regression in Slow Query Logging Extensions feature where incorrect transaction idaccounting can cause an assert during certain DDLs.
- #5206: TokuDB: A crash can occur in TokuDB when using Native Partioning and the optimizer has `index_merge_union` enabled. Workaround by using `SET SESSION optimizer_switch="index_merge_union=off";`
- #5174: MyRocks: Attempting to use unsupported features against MyRocks can lead to a crash rather than an error.
- #5024: MyRocks: Queries can return the wrong results on tables with no primary key, non-unique `CHAR`/`VARCHAR` rows, and `UTF8MB4` charset.
- #5045: MyRocks: Altering a column or table comment cause the table to be rebuilt

Find the release notes for Percona Server for MySQL 8.0.13-3 in our online documentation. Report bugs in the Jira bug tracker.

## *Percona Server for MySQL* 8.0.12-2rc1

Following the alpha release announced earlier, Percona announces the release candidate of *Percona Server for MySQL* 8.0.12-2rc1 on October 31, 2018. Download the latest version from the Percona web site or the Percona Software Repositories.

This release is based on *MySQL* 8.0.12 and includes all bug fixes in it. It is a *Release Candidate* quality release and it is not intended for production. If you want a high quality, Generally Available release, use the current Stable version (the most recent stable release at the time of writing in the 5.7 series is 5.7.23-23).

Percona provides completely open-source and free software.

### Installation

As this is a release candidate, installation is performed by enabling the testing repository and installing the software via your package manager. For Debian based distributions, see apt installation instructions; for RPM based distributions. see yum installation instructions. Note that in both cases after installing the current percona-release package, you'll need to enable the testing repository in order to install *Percona Server for MySQL* for MySQL 8.0.12-2rc1. For manual installations, you can download from the testing repository directly through our website.

### New Features

- #4550: Native Partitioning support for MyRocks storage engine
- #3911: Native Partitioning support for TokuDB storage engine
- #4946: Add an option to prevent implicit creation of column family in MyRocks
- #4839: Better default configuration for MyRocks and TokuDB
- InnoDB changed page tracking has been rewritten to account for redo logging changes in MySQL 8.0.11. This fixes fast incremental backups for PS 8.0

- #4434: TokuDB ROW_FORMAT clause has been removed, compression may be set by using the session variable `tokudb_row_format` instead.

## Improvements

- Several packaging changes to bring Percona packages more in line with upstream, including split repositories. As you'll note from our instructions above we now ship a tool with our release packages to help manage this.

## Bugs Fixed

- #4785: Setting version_suffix to **NULL** could lead to *handle_fatal_signal* (sig=11) in *Sys_var_version::global_value_ptr*
- #4788: Setting *log_slow_verbosity* and enabling the *slow_query_log* could lead to a server crash
- #4937: Any index comment generated a new column family in MyRocks
- #1107: Binlog could be corrupted when *tmpdir* got full
- #1549: Server side prepared statements lead to a potential off-by-second timestamp on slaves
- #4937: `rocksdb_update_cf_options` was useless when specified in my.cnf or on command line.
- #4705: The server could crash on snapshot size check in RocksDB
- #4791: SQL injection on slave due to non-quoting in binlogged `ROLLBACK TO SAVEPOINT`
- #4953: *rocksdb.truncate_table3* was unstable

Other bugs fixed:

- #4811: 5.7 Merge and fixup for old DB-937 introduces possible regression
- #4885: `Using ALTER ... ROW_FORMAT=TOKUDB_QUICKLZ` lead to InnoDB: Assertion failure: `ha_innodb.cc:12198:m_form->s->row_type == m_create_info->row_type`
- Numerous testsuite failures/crashes

## Upcoming Features

- New encryption features in *Percona Server for MySQL* 5.7 will be ported forward to *Percona Server for MySQL* 8.0
- Adding back in column compression with custom data dictionaries and expanded fast index creation.

CHAPTER

# EIGHTYTHREE

# GLOSSARY

**ACID** Set of properties that guarantee database transactions are processed reliably. Stands for *Atomicity*, *Consistency*, *Isolation*, *Durability*.

**Atomicity** Atomicity means that database operations are applied following a "all or nothing" rule. A transaction is either fully applied or not at all.

**Consistency** Consistency means that each transaction that modifies the database takes it from one consistent state to another.

**Durability** Once a transaction is committed, it will remain so.

**Foreign Key** A referential constraint between two tables. Example: A purchase order in the purchase_orders table must have been made by a customer that exists in the customers table.

**Isolation** The Isolation requirement means that no transaction can interfere with another.

**InnoDB** A *Storage Engine* for MySQL and derivatives (*Percona Server*, *MariaDB*) originally written by Innobase Oy, since acquired by Oracle. It provides *ACID* compliant storage engine with *foreign key* support. As of *MySQL* version 5.5, InnoDB became the default storage engine on all platforms.

**Jenkins** Jenkins is a continuous integration system that we use to help ensure the continued quality of the software we produce. It helps us achieve the aims of:

- no failed tests in trunk on any platform,
- aid developers in ensuring merge requests build and test on all platforms,
- no known performance regressions (without a damn good explanation).

**LSN** Log Serial Number. A term used in relation to the *InnoDB* or *XtraDB* storage engines.

**MariaDB** A fork of *MySQL* that is maintained primarily by Monty Program AB. It aims to add features, fix bugs while maintaining 100% backwards compatibility with MySQL.

**my.cnf** The file name of the default MySQL configuration file.

**MyISAM** A *MySQL Storage Engine* that was the default until MySQL 5.5.

**MySQL** An open source database that has spawned several distributions and forks. MySQL AB was the primary maintainer and distributor until bought by Sun Microsystems, which was then acquired by Oracle. As Oracle owns the MySQL trademark, the term MySQL is often used for the Oracle distribution of MySQL as distinct from the drop-in replacements such as *MariaDB* and *Percona Server*.

**NUMA** Non-Uniform Memory Access (NUMA) is a computer memory design used in multiprocessing, where the memory access time depends on the memory location relative to a processor. Under NUMA, a processor can access its own local memory faster than non-local memory, that is, memory local to another processor or memory shared between processors. The whole system may still operate as one unit, and all memory is basically accessible from everywhere, but at a potentially higher latency and lower performance.

**Percona Server for MySQL** Percona's branch of *MySQL* with performance and management improvements.

**Percona Server** See *Percona Server for MySQL*

**Storage Engine** A *Storage Engine* is a piece of software that implements the details of data storage and retrieval for a database system. This term is primarily used within the *MySQL* ecosystem due to it being the first widely used relational database to have an abstraction layer around storage. It is analogous to a Virtual File System layer in an Operating System. A VFS layer allows an operating system to read and write multiple file systems (e.g. FAT, NTFS, XFS, ext3) and a Storage Engine layer allows a database server to access tables stored in different engines (e.g. *MyISAM*, InnoDB).

**XtraDB** Percona's improved version of *InnoDB* providing performance, features and reliability above what is shipped by Oracle in InnoDB.

- genindex
- modindex

# INDEX

## Symbols



## A



## B



## C



## D



## E



## F



## G



## H



## I

## J



## K



## L



## M



## N



## P



## R

## S



## T

## U



## X