

Dedicated to bazaar builders everywhere.

Licence

The Knowledge Base content from which this pdf is generated is either licensed under the terms of the [GPL, version 2](#), originally generated from the server fill_help_tables.sql file, or both of the following two licenses:

- The [Creative Commons Attribution/ShareAlike 3.0 Unported](#) license (CC-BY-SA).
- The [Gnu FDL](#) license (GFDL or FDL).

Please see the source page on the Knowledge Base for the definitive licence, and seek proper legal advice if you are in any doubt about what you are and are not allowed to do with material released under these licenses.

If you find any errors, please see [Reporting Documentation Bugs](#)

Preface

If you're contemplating whether to devote some time to this book, read this:

- MariaDB Server is a general-purpose, open source, relational database management system, optimised for performance and easy usability; it has its roots in MySQL Server, and is an alternative to Postgres, Oracle Database and other relational and NoSQL databases
- This book is the full documentation on MariaDB Server, a “Reference Manual Plus” which includes aspects of a User’s Guide; it is based on the contents of the MariaDB Knowledge base (<https://mariadb.com/kb/>), an open, community-edited site contributed to since the inception of MariaDB in 2009
- This edition is not specific to any version of MariaDB Server, but includes functionality up to the latest version of MariaDB at the time of generation

This preface describes the goals, structure and contents of the documentation. Reading it is intended as a helpful step in understanding how to best use the manual to improve productivity in using MariaDB Server.

This Book’s “Prehistory”

As noted, MariaDB Server has its roots in MySQL Server. It started as a fork of MySQL Server, using the same GPLv2 license. However, although the MySQL Server documentation was always publicly available, it was never released using a free documentation license. This means that the documentation of MariaDB Server was created from scratch. Or rather, from the online help texts, which had a compatible open licence that made them usable as a starting point.

The place to which documentation was written was labelled the “Knowledge Base”, by MySQL and MariaDB creator Michael “Monty” Widenius. The Knowledge Base was – and remains – a community effort. As with many community efforts, there are core contributors around whom the work is centered. This is where Daniel Bartholomew loaded the online help text, as a first seed. For roughly the last ten years, the core editor of the MariaDB Knowledge Base has been Ian Gilfillan, working for MariaDB Foundation and based in South Africa. Hence, his name is on the cover of the book. However, there are a large number of other contributors, many of whom come from MariaDB Corporation – both as developers of code and as documentation writers. They are listed on <https://mariadb.com/kb/stats/users/>.

With now some 3000 pages in this book, most of the initial holes in the documentation have been filled. There should now be no reason to do as in the very early days of MariaDB Server – namely look up MariaDB features in the MySQL documentation. On the contrary, the functionality of the two databases have diverged considerably, so you would be ill advised not to use this MariaDB Server specific documentation.

The First Edition

The first edition of the MariaDB Server Documentation as a PDF file was released in April 2022. Prior to this, the contents were accessible as individual Knowledge Base (KB) articles. But already in 2014 – over seven years before – the user base requested a PDF version, as seen by MariaDB’s Jira entry <https://jira.mariadb.org/browse/MDEV-6881> MariaDB Documentation improvements. There, user Stoykov pointed out that MariaDB documentation already had search capabilities and a way to mirror the KB in an offline version – but lacked downloadable PDF and EPUB versions,

Fast forward some seven years and a number of upvotes and watchers, we decided to devote resources to it. Creating a PDF from an HTML file is something Python is good at, and Dorje Gilfillan did all the tweaking necessary to merge the individual KB pages into one huge HTML file for PDF conversion.

This Book’s Structure

We had to impose a chapter structure on the book which is only indirectly visible from a collection of KB articles on the web. This means that the work in compiling the PDF isn’t just about merging many KB pages in an order that could be derived from the hierarchical pointers between the articles. It also involves cleaning up that structure.

As a result, you will see two tables of contents. One is a one-pager overview with just the two top levels of hierarchy. The other is over 30 pages long. True to the Open Source mantra of “release early, release often”, we believe that the structure can still be improved upon – but it is a good starting point. We have seven overall chapters, and the structures below them all make sense at some level.

To get the most out of the book, we recommend you to spend time making yourself familiar with the table of contents. It will give you an idea of existing functionality. Just browsing it through may give you ideas of commands you didn’t know existed.

This Book's Format

There is currently just one version of the book. It's delivered in the PDF format, and in the Golden Ratio aspect ratio – meaning, A4. As we envision it to be read mostly on-screen anyway, we wanted to avoid the additional complexity of also providing a US Letter format. If we meet demand for further versions, doing US Letter is of course an option; however, given there are many ways to improve the documentation, we would also like to understand how adding another aspect ratio of the PDF would benefit the users in practice.

We don't yet provide the ePub format. Again, if you desire ePub, please educate us as to what added benefits you expect of ePub on top of PDF.

Use Cases For This Book

We expect the main use case for the PDF version of the book to be offline access. Offline may be imposed by a flaky or non-existent internet, but also by self-imposed abstinence from the many distractions of being online.

We expect that browsing the PDF will enable concentrated time to be spent on learning about MariaDB Server. The search functionality of PDF browsers helps in finding out about commands and syntax you already know of; browsing through a PDF – in particular the clickable Table of Contents – will hopefully provide you with an educational overview better than the online KB does.

We expect downloading the manual into laptops, tablets and phones will make sense. If you have the MariaDB Server Documentation on your phone, you can turn waiting time into something productive, perhaps even fun.

What we should work on

We have lots of room for improvement. That said, our foremost goal now is to get the book out, to get it used. User feedback will help us determine the right priority for our already existing ideas for improvements. We will likely get other requests beyond what we currently have in mind.

In the area of basic usability, an index has been spoken about. Looking up commands through searches or through browsing the table of contents is ok, but an index also has use cases. Our plan here starts from automatic indexing based on keywords of the headers of individual articles.

In the area of layout, we are looking at finding icons that make the PDF look more like a book, and less like a web page. We already solved the first issue, which was to find a clearer visual distinction between links within the PDF and links to the web.

In the area of structure, the length of individual chapters varies a lot. It may make sense to move around chapters in the TOC tree, to be more balanced. It may be that the reader expects another ordering based on experiences from other databases. It may even be that we lack entire topics. For instance, we eliminated the Release Notes for unsupported versions of MariaDB, even though these are still accessible on the KB.

In the area of accessibility, there may be places we should publish the PDF to make it easier to find, download, and use.

The common denominator for all of the above is that we need your feedback on what makes sense for you as a user of MariaDB Server.

Give Us Feedback

We would like to pick the brains of individual users. At conferences, asking open-ended questions is easy and feels productive for both parties, when meeting in the corridors between talks. Replicating the same productive discussion on-line is much harder. It takes effort from both parties. It feels like work.

We are still looking for the best way for you to give us meaningful feedback. Feel free to approach us over Zulip (<https://mariadb.zulipchat.com/> – the Documentation topic). Also email to foundation@mariadb.org will find its way to us.

When you find individual bugs, please enter them into Jira using the guidelines mentioned in the KB article <https://mariadb.com/kb/en/reporting-documentation-bugs/>.

Acknowledgements

Compiling any book requires more effort than expected by the authors, and more than visible to the readers. This book is no exception. It has been over ten years in the making.

The primary thanks go to Ian Gilfillan, as the overall editor of the book and as the individually most productive author.

Close to Ian, we have Daniel Bartholomew. Daniel even beats Ian when it comes to articles created, and comes second on articles edited.

Among the community contributors, we want to highlight Federico Razzoli. He has two accounts, totalling 4488, at the time of writing – making him rank third amongst personal contributors.

When it comes to organisational contributors, the largest one is MariaDB Corporation. With them coding most of the features, they also stand for the lion's share of their documentation. As writers, besides Daniel Bartholomew whom we already mentioned several times, we want to highlight Russell Dyer, Kenneth Dyer, Geoff Montee, and Jacob Moorman.

As the developer of the KB software itself, Bryan Alsdorf deserves special acknowledgement.

A special thanks goes to Michael “Monty” Widenius, the creator of MariaDB. Monty has always understood the importance of documentation. He is leading by example, with a large number of personal edits. In fact, Monty has the second highest number of edits amongst developers, after Sergei Golubchik and followed by Sergey Petrunia – all of which have over a thousand edits.

Amongst the prolific contributors within the MariaDB Corporation Engineering team, the Connectors team stands out, with Diego Dupin, Georg Richter, and Lawrin Novitzky ranking near the top. However, we have decided not to include Connectors documentation in this first edition; we are contemplating whether it should be a separate PDF manual.

Other past and present Engineering team members, in decreasing order of number of edits, are David Hill, Dipti Joshi, David Thompson, Massimiliano Pinto, Kolbe Kegel, Vladislav Vaintrob, Ralf Gebhardt, Markus Mäkelä, Sunanda Menon, the late Rasmus Johansson, Todd Stoffel, Elena Stepanova, Julien Fritsch, and Alexander Barkov. They all have more than one hundred edits, which is a lot.

As a true Open Source project, MariaDB Server documentation attracts attention and plentiful contributions also from outside the MariaDB Corporation Documentation and Engineering teams. We want to highlight those with over a hundred edits: Colin Charles and Stephane Varoqui, both of MariaDB Corporation, and Daniel Black, of MariaDB Foundation.

Amongst community contributors in the over-a-hundred-edits category, we want to mention especially Alena Subotina, with edits related to the dbforge documentation tool, and Juan Telleria, with edits often related to R Statistical Programming. Prolific contributors whose contributions are not visible in this English manual are Esper Ecyan (Japanese) and Hector Stredel (French); Federico Razzoli (Italian) has many edits also in English.

We also want to extend a thank you to the code developers who make work easy for the documentation team through thoroughly prepared, reusable texts in Jira; in this category, Marko Mäkelä and Oleksandr Byelkin come to mind.

As for the PDF manual, it has been teamwork between Ian and his son Dorje Gilfillan. Ian has done what editors do, Dorje has coded the Python code that compiles the KB pages into one.

All in all, thank you to everyone who has contributed to this book! We hope compiling it into one volume is of use for you, and we would love to hear what you think about the end result.

Munich, Germany, October 2022

Kaj Arnö, CEO, MariaDB Foundation

Chapter Contents

Chapter 1 Using MariaDB Server.....	x
1.1 SQL Statements & Structure.....	x

Table of Contents

Chapter 1 Using MariaDB Server.....	x
1.1 SQL Statements & Structure.....	x
1.1.1 SQL Statements.....	x
1.1.1.1 Account Management SQL Commands.....	x
1.1.1.1.1 CREATE USER.....	x
1.1.1.1.2 ALTER USER.....	x
1.1.1.1.3 DROP USER.....	x
1.1.1.1.4 GRANT.....	x
1.1.1.1.5 RENAME USER.....	x
1.1.1.1.6 REVOKE.....	x
1.1.1.1.7 SET PASSWORD.....	x
1.1.1.1.8 CREATE ROLE.....	x
1.1.1.1.9 DROP ROLE.....	x
1.1.1.1.10 SET ROLE.....	x
1.1.1.1.11 SET DEFAULT ROLE.....	x
1.1.1.1.12 SHOW GRANTS.....	x
1.1.1.1.13 SHOW CREATE USER.....	x
1.1.1.2 Administrative SQL Statements.....	x
1.1.1.2.1 Table Statements.....	x
1.1.1.2.1.1 ALTER.....	x
1.1.1.2.1.1.1 ALTER TABLE.....	x
1.1.1.2.1.1.2 ALTER DATABASE.....	x
1.1.1.2.1.1.3 ALTER EVENT.....	x
1.1.1.2.1.1.4 ALTER FUNCTION.....	x
1.1.1.2.1.1.5 ALTER LOGFILE GROUP.....	x
1.1.1.2.1.1.6 ALTER PROCEDURE.....	x
1.1.1.2.1.1.7 ALTER SEQUENCE.....	x
1.1.1.2.1.1.8 ALTER SERVER.....	x
1.1.1.2.1.1.9 ALTER TABLESPACE.....	x
1.1.1.2.1.1.10 ALTER USER.....	x
1.1.1.2.1.1.11 ALTER VIEW.....	x
1.1.1.2.1.2 ANALYZE TABLE.....	x
1.1.1.2.1.3 CHECK TABLE.....	x
1.1.1.2.1.4 CHECK VIEW.....	x
1.1.1.2.1.5 CHECKSUM TABLE.....	x
1.1.1.2.1.6 CREATE TABLE.....	x
1.1.1.2.1.7 DELETE.....	x
1.1.1.2.1.8 DROP TABLE.....	x
1.1.1.2.1.9 Installing System Tables (mysql_install_db).....	x
1.1.1.2.1.10 mysqlcheck.....	x
1.1.1.2.1.11 OPTIMIZE TABLE.....	x
1.1.1.2.1.12 RENAME TABLE.....	x
1.1.1.2.1.13 REPAIR TABLE.....	x
1.1.1.2.1.14 REPAIR VIEW.....	x
1.1.1.2.1.15 REPLACE.....	x
1.1.1.2.1.16 SHOW COLUMNS.....	x
1.1.1.2.1.17 SHOW CREATE TABLE.....	x
1.1.1.2.1.18 SHOW INDEX.....	x
1.1.1.2.1.19 TRUNCATE TABLE.....	x
1.1.1.2.1.20 UPDATE.....	x
1.1.1.2.1.21 IGNORE.....	x
1.1.1.2.1.22 System-Versioned Tables.....	x
1.1.1.2.2 ANALYZE and EXPLAIN Statements.....	x
1.1.1.2.2.1 ANALYZE FORMAT=JSON.....	x
1.1.1.2.2.2 ANALYZE FORMAT=JSON Examples.....	x
1.1.1.2.2.3 ANALYZE Statement.....	x
1.1.1.2.2.4 EXPLAIN.....	x

1.1.1.2.2.5 EXPLAIN ANALYZE.....	X
1.1.1.2.2.6 EXPLAIN FORMAT=JSON.....	X
1.1.1.2.2.7 SHOW EXPLAIN.....	X
1.1.1.2.2.8 Using Buffer UPDATE Algorithm.....	X
1.1.1.2.3 BACKUP Commands.....	X
1.1.1.2.3.1 BACKUP STAGE.....	X
1.1.1.2.3.2 BACKUP LOCK.....	X
1.1.1.2.3.3 Mariabackup and BACKUP STAGE Commands.....	X
1.1.1.2.3.4 Storage Snapshots and BACKUP STAGE Commands.....	X
1.1.1.2.4 FLUSH Commands.....	X
1.1.1.2.4.1 FLUSH.....	X
1.1.1.2.4.2 FLUSH QUERY CACHE.....	X
1.1.1.2.4.3 FLUSH TABLES FOR EXPORT.....	X
1.1.1.2.5 Replication Commands.....	X
1.1.1.2.5.1 CHANGE MASTER TO.....	X
1.1.1.2.5.2 START SLAVE.....	X
1.1.1.2.5.3 STOP SLAVE.....	X
1.1.1.2.5.4 RESET REPLICA/SLAVE.....	X
1.1.1.2.5.5 SET GLOBAL SQL_SLAVE_SKIP_COUNTER.....	X
1.1.1.2.5.6 SHOW RELAYLOG EVENTS.....	X
1.1.1.2.5.7 SHOW SLAVE STATUS.....	X
1.1.1.2.5.8 SHOW MASTER STATUS.....	X
1.1.1.2.5.9 SHOW SLAVE HOSTS.....	X
1.1.1.2.5.10 RESET MASTER.....	X
1.1.1.2.6 Plugin SQL Statements.....	X
1.1.1.2.6.1 SHOW PLUGINS.....	X
1.1.1.2.6.2 SHOW PLUGINS SONAME.....	X
1.1.1.2.6.3 INSTALL PLUGIN.....	X
1.1.1.2.6.4 UNINSTALL PLUGIN.....	X
1.1.1.2.6.5 INSTALL SONAME.....	X
1.1.1.2.6.6 UNINSTALL SONAME.....	X
1.1.1.2.6.7 mysql_plugin.....	X
1.1.1.2.7 SET Commands.....	X
1.1.1.2.7.1 SET.....	X
1.1.1.2.7.2 SET CHARACTER SET.....	X
1.1.1.2.7.3 SET GLOBAL SQL_SLAVE_SKIP_COUNTER.....	X
1.1.1.2.7.4 SET NAMES.....	X
1.1.1.2.7.5 SET PASSWORD.....	X
1.1.1.2.7.6 SET ROLE.....	X
1.1.1.2.7.7 SET SQL_LOG_BIN.....	X
1.1.1.2.7.8 SET STATEMENT.....	X
1.1.1.2.7.9 SET TRANSACTION.....	X
1.1.1.2.7.10 SET Variable.....	X
1.1.1.2.8 SHOW.....	X
1.1.1.2.8.1 About SHOW.....	X
1.1.1.2.8.2 Extended Show.....	X
1.1.1.2.8.3 SHOW AUTHORS.....	X
1.1.1.2.8.4 SHOW BINARY LOGS.....	X
1.1.1.2.8.5 SHOW BINLOG EVENTS.....	X
1.1.1.2.8.6 SHOW CHARACTER SET.....	X
1.1.1.2.8.7 SHOW CLIENT_STATISTICS.....	X
1.1.1.2.8.8 SHOW COLLATION.....	X
1.1.1.2.8.9 SHOW COLUMNS.....	X
1.1.1.2.8.10 SHOW CONTRIBUTORS.....	X
1.1.1.2.8.11 SHOW CREATE DATABASE.....	X
1.1.1.2.8.12 SHOW CREATE EVENT.....	X
1.1.1.2.8.13 SHOW CREATE FUNCTION.....	X
1.1.1.2.8.14 SHOW CREATE PACKAGE.....	X
1.1.1.2.8.15 SHOW CREATE PACKAGE BODY.....	X
1.1.1.2.8.16 SHOW CREATE PROCEDURE.....	X
1.1.1.2.8.17 SHOW CREATE SEQUENCE.....	X
1.1.1.2.8.18 SHOW CREATE TABLE.....	X
1.1.1.2.8.19 SHOW CREATE TRIGGER.....	X

1.1.1.2.8.20 SHOW CREATE USER.....	X
1.1.1.2.8.21 SHOW CREATE VIEW.....	X
1.1.1.2.8.22 SHOW DATABASES.....	X
1.1.1.2.8.23 SHOW ENGINE.....	X
1.1.1.2.8.24 SHOW ENGINE INNODB STATUS.....	X
1.1.1.2.8.25 SHOW ENGINES.....	X
1.1.1.2.8.26 SHOW ERRORS.....	X
1.1.1.2.8.27 SHOW EVENTS.....	X
1.1.1.2.8.28 SHOW FUNCTION CODE.....	X
1.1.1.2.8.29 SHOW FUNCTION STATUS.....	X
1.1.1.2.8.30 SHOW GRANTS.....	X
1.1.1.2.8.31 SHOW INDEX.....	X
1.1.1.2.8.32 SHOW INDEX_STATISTICS.....	X
1.1.1.2.8.33 SHOW LOCALES.....	X
1.1.1.2.8.34 SHOW BINLOG STATUS.....	X
1.1.1.2.8.35 SHOW OPEN TABLES.....	X
1.1.1.2.8.36 SHOW PACKAGE BODY STATUS.....	X
1.1.1.2.8.37 SHOW PACKAGE STATUS.....	X
1.1.1.2.8.38 SHOW PLUGINS.....	X
1.1.1.2.8.39 SHOW PLUGINS SONAME.....	X
1.1.1.2.8.40 SHOW PRIVILEGES.....	X
1.1.1.2.8.41 SHOW PROCEDURE CODE.....	X
1.1.1.2.8.42 SHOW PROCEDURE STATUS.....	X
1.1.1.2.8.43 SHOW PROCESSLIST.....	X
1.1.1.2.8.44 SHOW PROFILE.....	X
1.1.1.2.8.45 SHOW PROFILES.....	X
1.1.1.2.8.46 SHOW QUERY_RESPONSE_TIME.....	X
1.1.1.2.8.47 SHOW RELAYLOG EVENTS.....	X
1.1.1.2.8.48 SHOW REPLICA HOSTS.....	X
1.1.1.2.8.49 SHOW REPLICA STATUS.....	X
1.1.1.2.8.50 SHOW STATUS.....	X
1.1.1.2.8.51 SHOW TABLE STATUS.....	X
1.1.1.2.8.52 SHOW TABLES.....	X
1.1.1.2.8.53 SHOW TABLE_STATISTICS.....	X
1.1.1.2.8.54 SHOW TRIGGERS.....	X
1.1.1.2.8.55 SHOW USER_STATISTICS.....	X
1.1.1.2.8.56 SHOW VARIABLES.....	X
1.1.1.2.8.57 SHOW WARNINGS.....	X
1.1.1.2.8.58 SHOW WSREP_MEMBERSHIP.....	X
1.1.1.2.8.59 SHOW WSREP_STATUS.....	X
1.1.1.2.9 System Tables.....	X
1.1.1.2.9.1 Information Schema.....	X
1.1.1.2.9.1.1 Information Schema Tables.....	X
1.1.1.2.9.1.1.1 Information Schema InnoDB Tables.....	X
1.1.1.2.9.1.1.1.1 Information Schema INNODB_BUFFER_PAGE Table.....	X
1.1.1.2.9.1.1.1.2 Information Schema INNODB_BUFFER_PAGE_LRU Table.....	X
1.1.1.2.9.1.1.1.3 Information Schema INNODB_BUFFER_POOL_PAGES Table.....	X
1.1.1.2.9.1.1.1.4 Information Schema INNODB_BUFFER_POOL_PAGES_BLOB Table.....	X
1.1.1.2.9.1.1.1.5 Information Schema INNODB_BUFFER_POOL_PAGES_INDEX Table.....	X
1.1.1.2.9.1.1.1.6 Information Schema INNODB_BUFFER_POOL_STATS Table.....	X
1.1.1.2.9.1.1.1.7 Information Schema INNODB_CHANGED_PAGES Table.....	X
1.1.1.2.9.1.1.1.8 Information Schema INNODB_CMP and INNODB_CMP_RESET Tables.....	X
1.1.1.2.9.1.1.1.9 Information Schema INNODB_CPMEM and INNODB_CPMEM_RESET Tables.....	X
1.1.1.2.9.1.1.1.10 Information Schema INNODB_CMP_PER_INDEX and INNODB_CMP_PER_INDEX_RESET Tables.....	X
1.1.1.2.9.1.1.1.11 Information Schema INNODB_FT_BEING_DELETED Table.....	X
1.1.1.2.9.1.1.1.12 Information Schema INNODB_FT_CONFIG Table.....	X
1.1.1.2.9.1.1.1.13 Information Schema INNODB_FT_DEFAULT_STOPWORD Table.....	X
1.1.1.2.9.1.1.1.14 Information Schema INNODB_FT_DELETED Table.....	X
1.1.1.2.9.1.1.1.15 Information Schema INNODB_FT_INDEX_CACHE Table.....	X
1.1.1.2.9.1.1.1.16 Information Schema INNODB_FT_INDEX_TABLE Table.....	X
1.1.1.2.9.1.1.1.17 Information Schema INNODB_LOCK_WAITS Table.....	X
1.1.1.2.9.1.1.1.18 Information Schema INNODB_LOCKS Table.....	X
1.1.1.2.9.1.1.1.19 Information Schema INNODB_METRICS Table.....	X

1.1.1.2.9.1.1.1.20 Information Schema INNODB_MUTEXES Table.....	X
1.1.1.2.9.1.1.1.21 Information Schema INNODB_SYS_COLUMNS Table.....	X
1.1.1.2.9.1.1.1.22 Information Schema INNODB_SYS_DATAFILES Table.....	X
1.1.1.2.9.1.1.1.23 Information Schema INNODB_SYS_FIELDS Table.....	X
1.1.1.2.9.1.1.1.24 Information Schema INNODB_SYS_FOREIGN Table.....	X
1.1.1.2.9.1.1.1.25 Information Schema INNODB_SYS_FOREIGN_COLS Table.....	X
1.1.1.2.9.1.1.1.26 Information Schema INNODB_SYS_INDEXES Table.....	X
1.1.1.2.9.1.1.1.27 Information Schema INNODB_SYS_SEMAPHORE_WAITS Table.....	X
1.1.1.2.9.1.1.1.28 Information Schema INNODB_SYS_TABLES Table.....	X
1.1.1.2.9.1.1.1.29 Information Schema INNODB_SYS_TABLESPACES Table.....	X
1.1.1.2.9.1.1.1.30 Information Schema INNODB_SYS_TABLESTATS Table.....	X
1.1.1.2.9.1.1.1.31 Information Schema INNODB_SYS_VIRTUAL Table.....	X
1.1.1.2.9.1.1.1.32 Information Schema INNODB_TABLESPACES_ENCRYPTION Table.....	X
1.1.1.2.9.1.1.1.33 Information Schema INNODB_TABLESPACES_SCRUBBING Table.....	X
1.1.1.2.9.1.1.1.34 Information Schema INNODB_TRX Table.....	X
1.1.1.2.9.1.1.1.35 Information Schema TEMP_TABLES_INFO Table.....	X
1.1.1.2.9.1.1.2 Information Schema MyRocks Tables.....	X

H3Dr Using MariaDB Server

Documentation on using MariaDB Server.



SQL Statements & Structure

SQL statements, structure, and rules.



Built-in Functions

Functions and procedures in MariaDB. ↗



Clients & Utilities

Client and utility programs for MariaDB. ↗

H3Dr SQL Statements & Structure

The letters *SQL* stand for Structured Query Language. As with all languages—even computer languages—there are grammar rules. This includes a certain structure to statements, acceptable punctuation (i.e., operators and delimiters), and a vocabulary (i.e., reserve words).



SQL Statements

Explanations of all of the MariaDB SQL statements.



SQL Language Structure

Explanation of SQL grammar rules, including reserved words and literals. ↗



Geographic & Geometric Features

Spatial extensions for geographic and geometric features. ↗



NoSQL

NoSQL-related commands and interfaces. ↗



Operators

Operators for comparing and assigning values. ↗



Sequences

Sequence objects, an alternative to AUTO_INCREMENT. ↗



Temporal Tables

MariaDB supports system-versioning, application-time periods and bitemporal tables. ↗

There are [9 related questions](#) ↗.

H3Dr1 SQL Statements

Complete list of SQL statements for data definition, data manipulation, etc.



Account Management SQL Commands

CREATE/DROP USER, GRANT, REVOKE, SET PASSWORD etc.



Administrative SQL Statements

SQL statements for setting, flushing and displaying server variables and resources.



Data Definition

SQL commands for defining data, such as ALTER, CREATE, DROP, RENAME etc. ↗



Data Manipulation

SQL commands for querying and manipulating data, such as SELECT, UPDATE, DELETE etc. ↗



Prepared Statements

Prepared statements from any client using the text based prepared statement interface. ↗



Programmatic & Compound Statements

Compound SQL statements for stored routines and in general. ↗



Stored Routine Statements

SQL statements related to creating and using stored routines. ↗



Table Statements

Documentation on creating, altering, analyzing and maintaining tables.



Transactions

Sequence of statements that are either completely successful, or have no effect on any schemas ↗



HELP Command

The HELP command will retrieve syntax and help within the mysql client. ↗



Comment Syntax

Comment syntax and style. ↗



Built-in Functions

Functions and procedures in MariaDB. ↗

There are 17 related questions ↗.

H3Dr1.1 Account Management SQL Commands

CREATE/DROP USER, GRANT, REVOKE, SET PASSWORD etc.



CREATE USER

Create new MariaDB accounts.



ALTER USER

Modify an existing MariaDB account.



DROP USER

Remove one or more MariaDB accounts.



GRANT

Create accounts and set privileges or roles.



RENAME USER

Rename user account.



REVOKE

Remove privileges or roles.



SET PASSWORD

Assign password to an existing MariaDB user.



CREATE ROLE

Add new roles.



DROP ROLE

Drop a role.



SET ROLE

Enable a role.



SET DEFAULT ROLE

Sets a default role for a specified (or current) user.



SHOW GRANTS

View GRANT statements.



SHOW CREATE USER

Show the CREATE USER statement for a specified user.

There are 2 related questions

H3Dr1.1.1 CREATE USER

Syntax

```
CREATE [OR REPLACE] USER [IF NOT EXISTS]
  user_specification [,user_specification ...]
  [REQUIRE {NONE | tls_option [[AND] tls_option ...] }]
  [WITH resource_option [resource_option ...] ]
  [lock_option] [password_option]

  user_specification:
    username [authentication_option]

  authentication_option:
    IDENTIFIED BY 'password'
    | IDENTIFIED BY PASSWORD 'password_hash'
    | IDENTIFIED {VIA|WITH} authentication_rule [OR authentication_rule ...]

  authentication_rule:
    authentication_plugin
    | authentication_plugin {USING|AS} 'authentication_string'
    | authentication_plugin {USING|AS} PASSWORD('password')

  tls_option:
    SSL
    | X509
    | CIPHER 'cipher'
    | ISSUER 'issuer'
    | SUBJECT 'subject'

  resource_option:
    MAX_QUERIES_PER_HOUR count
    | MAX_UPDATES_PER_HOUR count
    | MAX_CONNECTIONS_PER_HOUR count
    | MAX_USER_CONNECTIONS count
    | MAX_STATEMENT_TIME time

  password_option:
    PASSWORD EXPIRE
    | PASSWORD EXPIRE DEFAULT
    | PASSWORD EXPIRE NEVER
    | PASSWORD EXPIRE INTERVAL N DAY

  lock_option:
    ACCOUNT LOCK
    | ACCOUNT UNLOCK
}
```

Contents

1. [Syntax](#)
2. [Description](#)
3. [OR REPLACE](#)
4. [IF NOT EXISTS](#)
5. [Authentication Options](#)
 1. [IDENTIFIED BY 'password'](#)
 2. [IDENTIFIED BY PASSWORD
'password_hash'](#)
 3. [IDENTIFIED {VIA|WITH}
authentication_plugin](#)
6. [TLS Options](#)
7. [Resource Limit Options](#)
8. [Account Names](#)
 1. [Host Name Component](#)
 2. [User Name Component](#)
 3. [Anonymous Accounts](#)
 1. [Fixing a Legacy Default Anonymous Account](#)
9. [Password Expiry](#)
10. [Account Locking](#)
11. [See Also](#)

Description

The `CREATE USER` statement creates new MariaDB accounts. To use it, you must have the global `CREATE USER` privilege or the `INSERT` privilege for the `mysql` database. For each account, `CREATE USER` creates a new row in `mysql.user` (until MariaDB 10.3 this is a table, from MariaDB 10.4 it's a view) or `mysql.global_priv_table` (from MariaDB 10.4) that has no privileges.

If any of the specified accounts, or any permissions for the specified accounts, already exist, then the server returns `ERROR 1396 (HY000)`. If an error occurs, `CREATE USER` will still create the accounts that do not result in an error. Only one error is produced for all users which have not been created:

```
ERROR 1396 (HY000):  
Operation CREATE USER failed for 'u1'@'%', 'u2'@'%'
```

`CREATE USER`, `DROP USER`, `CREATE ROLE`, and `DROP ROLE` all produce the same error code when they fail.

See [Account Names](#) below for details on how account names are specified.

OR REPLACE

If the optional `OR REPLACE` clause is used, it is basically a shortcut for:

```
DROP USER IF EXISTS name;  
CREATE USER name ...;
```

For example:

```
CREATE USER foo2@test IDENTIFIED BY 'password';  
ERROR 1396 (HY000): Operation CREATE USER failed for 'foo2'@'test'  
  
CREATE OR REPLACE USER foo2@test IDENTIFIED BY 'password';  
Query OK, 0 rows affected (0.00 sec)
```

IF NOT EXISTS

When the `IF NOT EXISTS` clause is used, MariaDB will return a warning instead of an error if the specified user already exists.

For example:

```
CREATE USER foo2@test IDENTIFIED BY 'password';
ERROR 1396 (HY000): Operation CREATE USER failed for 'foo2'@'test'

CREATE USER IF NOT EXISTS foo2@test IDENTIFIED BY 'password';
Query OK, 0 rows affected, 1 warning (0.00 sec)

SHOW WARNINGS;
+-----+-----+
| Level | Code | Message           |
+-----+-----+
| Note  | 1973 | Can't create user 'foo2'@'test'; it already exists |
+-----+-----+
```

Authentication Options

IDENTIFIED BY 'password'

The optional `IDENTIFIED BY` clause can be used to provide an account with a password. The password should be specified in plain text. It will be hashed by the [PASSWORD](#) function prior to being stored in the [mysql.user](#)/[mysql.global_priv_table](#) table.

For example, if our password is `mariadb`, then we can create the user with:

```
CREATE USER foo2@test IDENTIFIED BY 'mariadb';
```

If you do not specify a password with the `IDENTIFIED BY` clause, the user will be able to connect without a password. A blank password is not a wildcard to match any password. The user must connect without providing a password if no password is set.

The only [authentication plugins](#) that this clause supports are [mysql_native_password](#) and [mysql_old_password](#).

IDENTIFIED BY PASSWORD 'password_hash'

The optional `IDENTIFIED BY PASSWORD` clause can be used to provide an account with a password that has already been hashed. The password should be specified as a hash that was provided by the [PASSWORD](#) function. It will be stored in the [mysql.user](#)/[mysql.global_priv_table](#) table as-is.

For example, if our password is `mariadb`, then we can find the hash with:

```
SELECT PASSWORD('mariadb');
+-----+
| PASSWORD('mariadb')          |
+-----+
| *54958E764CE10E50764C2EECB71D01F08549980 |
+-----+
1 row in set (0.00 sec)
```

And then we can create a user with the hash:

```
CREATE USER foo2@test IDENTIFIED BY PASSWORD '*54958E764CE10E50764C2EECB71D01F08549980';
```

If you do not specify a password with the `IDENTIFIED BY` clause, the user will be able to connect without a password. A blank password is not a wildcard to match any password. The user must connect without providing a password if no password is set.

The only [authentication plugins](#) that this clause supports are [mysql_native_password](#) and [mysql_old_password](#).

IDENTIFIED {VIA|WITH} authentication_plugin

The optional `IDENTIFIED VIA` `authentication_plugin` allows you to specify that the account should be

authenticated by a specific [authentication plugin](#). The plugin name must be an active authentication plugin as per [SHOW PLUGINS](#). If it doesn't show up in that output, then you will need to install it with [INSTALL PLUGIN](#) or [INSTALL SONAME](#).

For example, this could be used with the [PAM authentication plugin](#):

```
CREATE USER foo2@test IDENTIFIED VIA pam;
```

Some authentication plugins allow additional arguments to be specified after a `USING` or `AS` keyword. For example, the [PAM authentication plugin](#) accepts a service name:

```
CREATE USER foo2@test IDENTIFIED VIA pam USING 'mariadb';
```

The exact meaning of the additional argument would depend on the specific authentication plugin.

MariaDB starting with [10.4.0](#)

The `USING` or `AS` keyword can also be used to provide a plain-text password to a plugin if it's provided as an argument to the [PASSWORD\(\)](#) function. This is only valid for [authentication plugins](#) that have implemented a hook for the [PASSWORD\(\)](#) function. For example, the [ed25519](#) authentication plugin supports this:

```
CREATE USER safe@'%' IDENTIFIED VIA ed25519 USING PASSWORD('secret');
```

MariaDB starting with [10.4.3](#)

One can specify many authentication plugins, they all work as alternatives ways of authenticating a user:

```
CREATE USER safe@'%' IDENTIFIED VIA ed25519 USING PASSWORD('secret') OR unix_socket;
```

By default, when you create a user without specifying an authentication plugin, MariaDB uses the [mysql_native_password](#) plugin.

TLS Options

By default, MariaDB transmits data between the server and clients without encrypting it. This is generally acceptable when the server and client run on the same host or in networks where security is guaranteed through other means. However, in cases where the server and client exist on separate networks or they are in a high-risk network, the lack of encryption does introduce security concerns as a malicious actor could potentially eavesdrop on the traffic as it is sent over the network between them.

To mitigate this concern, MariaDB allows you to encrypt data in transit between the server and clients using the Transport Layer Security (TLS) protocol. TLS was formerly known as Secure Socket Layer (SSL), but strictly speaking the SSL protocol is a predecessor to TLS and, that version of the protocol is now considered insecure. The documentation still uses the term SSL often and for compatibility reasons TLS-related server system and status variables still use the prefix `ssl_`, but internally, MariaDB only supports its secure successors.

See [Secure Connections Overview](#) for more information about how to determine whether your MariaDB server has TLS support.

You can set certain TLS-related restrictions for specific user accounts. For instance, you might use this with user accounts that require access to sensitive data while sending it across networks that you do not control. These restrictions can be enabled for a user account with the [CREATE USER](#), [ALTER USER](#), or [GRANT](#) statements. The following options are available:

Option	Description
REQUIRE NONE	TLS is not required for this account, but can still be used.

REQUIRE SSL	The account must use TLS, but no valid X509 certificate is required. This option cannot be combined with other TLS options.
REQUIRE X509	The account must use TLS and must have a valid X509 certificate. This option implies REQUIRE SSL . This option cannot be combined with other TLS options.
REQUIRE ISSUER 'issuer'	The account must use TLS and must have a valid X509 certificate. Also, the Certificate Authority must be the one specified via the string <code>issuer</code> . This option implies REQUIRE X509 . This option can be combined with the SUBJECT , and CIPHER options in any order.
REQUIRE SUBJECT 'subject'	The account must use TLS and must have a valid X509 certificate. Also, the certificate's Subject must be the one specified via the string <code>subject</code> . This option implies REQUIRE X509 . This option can be combined with the ISSUER , and CIPHER options in any order.
REQUIRE CIPHER 'cipher'	The account must use TLS, but no valid X509 certificate is required. Also, the encryption used for the connection must use a specific cipher method specified in the string <code>cipher</code> . This option implies REQUIRE SSL . This option can be combined with the ISSUER , and SUBJECT options in any order.

The REQUIRE keyword must be used only once for all specified options, and the AND keyword can be used to separate individual options, but it is not required.

For example, you can create a user account that requires these TLS options with the following:

```
CREATE USER 'alice'@'%'
REQUIRE SUBJECT '/CN=alice/O=My Dom, Inc./C=US/ST=Oregon/L=Portland'
AND ISSUER '/C=FI/ST=Somewhere/L=City/ O=Some Company/CN=Peter
Parker/emailAddress=p.parker@marvel.com'
AND CIPHER 'SHA-DES-CBC3-EDH-RSA';
```

If any of these options are set for a specific user account, then any client who tries to connect with that user account will have to be configured to connect with TLS.

See [Securing Connections for Client and Server](#) for information on how to enable TLS on the client and server.

Resource Limit Options

MariaDB starting with [10.2.0](#)

[MariaDB 10.2.0](#) introduced a number of resource limit options.

It is possible to set per-account limits for certain server resources. The following table shows the values that can be set per account:

Limit Type	Description
MAX_QUERIES_PER_HOUR	Number of statements that the account can issue per hour (including updates)
MAX_UPDATES_PER_HOUR	Number of updates (not queries) that the account can issue per hour
MAX_CONNECTIONS_PER_HOUR	Number of connections that the account can start per hour
MAX_USER_CONNECTIONS	Number of simultaneous connections that can be accepted from the same account; if it is 0, <code>max_connections</code> will be used instead; if <code>max_connections</code> is 0, there is no limit for this account's simultaneous connections.
MAX_STATEMENT_TIME	Timeout, in seconds, for statements executed by the user. See also Aborting Statements that Exceed a Certain Time to Execute .

If any of these limits are set to 0 , then there is no limit for that resource for that user.

Here is an example showing how to create a user with resource limits:

```
CREATE USER 'someone'@'localhost' WITH  
MAX_USER_CONNECTIONS 10  
MAX_QUERIES_PER_HOUR 200;
```

The resources are tracked per account, which means 'user'@'server'; not per user name or per connection.

The count can be reset for all users using [FLUSH USER_RESOURCES](#), [FLUSH PRIVILEGES](#) or [mysqladmin reload](#).

Per account resource limits are stored in the [user](#) table, in the [mysql](#) database. Columns used for resources limits are named `max_questions`, `max_updates`, `max_connections` (for `MAX_CONNECTIONS_PER_HOUR`), and `max_user_connections` (for `MAX_USER_CONNECTIONS`).

Account Names

Account names have both a user name component and a host name component, and are specified as '`user_name`'@'`host_name`'.

The user name and host name may be unquoted, quoted as strings using double quotes (") or single quotes ('), or quoted as identifiers using backticks (`). You must use quotes when using special characters (such as a hyphen) or wildcard characters. If you quote, you must quote the user name and host name separately (for example '`user_name`'@'`host_name`').

Host Name Component

If the host name is not provided, it is assumed to be '%' .

Host names may contain the wildcard characters % and _ . They are matched as if by the [LIKE](#) clause. If you need to use a wildcard character literally (for example, to match a domain name with an underscore), prefix the character with a backslash. See [LIKE](#) for more information on escaping wildcard characters.

Host name matches are case-insensitive. Host names can match either domain names or IP addresses. Use '`localhost`' as the host name to allow only local client connections.

You can use a netmask to match a range of IP addresses using '`base_ip/netmask`' as the host name. A user with an IP address `ip_addr` will be allowed to connect if the following condition is true:

```
ip_addr & netmask = base_ip
```

For example, given a user:

```
CREATE USER 'maria'@'247.150.130.0/255.255.255.0';
```

the IP addresses satisfying this condition range from 247.150.130.0 to 247.150.130.255.

Using 255.255.255.255 is equivalent to not using a netmask at all. Netmasks cannot be used for IPv6 addresses.

Note that the credentials added when creating a user with the '%' wildcard host will not grant access in all cases. For example, some systems come with an anonymous localhost user, and when connecting from localhost this will take precedence.

Before [MariaDB 10.6](#), the host name component could be up to 60 characters in length. Starting from [MariaDB 10.6](#), it can be up to 255 characters.

User Name Component

User names must match exactly, including case. A user name that is empty is known as an anonymous account and is allowed to match a login attempt with any user name component. These are described more in the next section.

For valid identifiers to use as user names, see [Identifier Names](#).

It is possible for more than one account to match when a user connects. MariaDB selects the first

matching account after sorting according to the following criteria:

- Accounts with an exact host name are sorted before accounts using a wildcard in the host name. Host names using a netmask are considered to be exact for sorting.
- Accounts with a wildcard in the host name are sorted according to the position of the first wildcard character. Those with a wildcard character later in the host name sort before those with a wildcard character earlier in the host name.
- Accounts with a non-empty user name sort before accounts with an empty user name.
- Accounts with an empty user name are sorted last. As mentioned previously, these are known as anonymous accounts. These are described more in the next section.

The following table shows a list of example account as sorted by these criteria:

User	Host
joffrey	192.168.0.3
	192.168.0.%
joffrey	192.168.%
	192.168.%

Once connected, you only have the privileges granted to the account that matched, not all accounts that could have matched. For example, consider the following commands:

```
CREATE USER 'joffrey'@'192.168.0.3';
CREATE USER 'joffrey'@'%';
GRANT SELECT ON test.t1 TO 'joffrey'@'192.168.0.3';
GRANT SELECT ON test.t2 TO 'joffrey'@'%';
```

If you connect as joffrey from 192.168.0.3 , you will have the `SELECT` privilege on the table `test.t1` , but not on the table `test.t2` . If you connect as joffrey from any other IP address, you will have the `SELECT` privilege on the table `test.t2` , but not on the table `test.t1` .

Usernames can be up to 80 characters long before 10.6 and starting from 10.6 it can be 128 characters long.

Anonymous Accounts

Anonymous accounts are accounts where the user name portion of the account name is empty. These accounts act as special catch-all accounts. If a user attempts to log into the system from a host, and an anonymous account exists with a host name portion that matches the user's host, then the user will log in as the anonymous account if there is no more specific account match for the user name that the user entered.

For example, here are some anonymous accounts:

```
CREATE USER ''@'localhost';
CREATE USER ''@'192.168.0.3';
```

Fixing a Legacy Default Anonymous Account

On some systems, the `mysql.db` table has some entries for the ''@'' anonymous account by default. Unfortunately, there is no matching entry in the `mysql.user`/`mysql.global_priv_table` table, which means that this anonymous account doesn't exactly exist, but it does have privileges--usually on the default `test` database created by `mysql_install_db`. These account-less privileges are a legacy that is leftover from a time when MySQL's privilege system was less advanced.

This situation means that you will run into errors if you try to create a ''@'' account. For example:

```
CREATE USER ''@'%';
ERROR 1396 (HY000): Operation CREATE USER failed for ''@'%'
```

The fix is to `DELETE` the row in the `mysql.db` table and then execute `FLUSH PRIVILEGES`:

```
DELETE FROM mysql.db WHERE User=' ' AND Host='%';
FLUSH PRIVILEGES;
```

And then the account can be created:

```
CREATE USER ''@'%';
Query OK, 0 rows affected (0.01 sec)
```

See [MDEV-13486](#) for more information.

Password Expiry

MariaDB starting with [10.4.3](#)

Besides automatic password expiry, as determined by [default_password_lifetime](#), password expiry times can be set on an individual user basis, overriding the global setting, for example:

```
CREATE USER 'monty'@'localhost' PASSWORD EXPIRE INTERVAL 120 DAY;
```

See [User Password Expiry](#) for more details.

Account Locking

MariaDB starting with [10.4.2](#)

Account locking permits privileged administrators to lock/unlock user accounts. No new client connections will be permitted if an account is locked (existing connections are not affected). For example:

```
CREATE USER 'marijn'@'localhost' ACCOUNT LOCK;
```

See [Account Locking](#) for more details.

From [MariaDB 10.4.7](#) and [MariaDB 10.5.8](#), the *lock_option* and *password_option* clauses can occur in either order.

See Also

- [Troubleshooting Connection Issues](#)
- [Authentication from MariaDB 10.4](#)
- [Identifier Names](#)
- [GRANT](#)
- [ALTER USER](#)
- [DROP USER](#)
- [SET PASSWORD](#)
- [SHOW CREATE USER](#)
- [mysql.user table](#)
- [mysql.global_priv_table](#)
- [Password Validation Plugins](#) - permits the setting of basic criteria for passwords
- [Authentication Plugins](#) - allow various authentication methods to be used, and new ones to be developed.

H3Dr1.1.2 ALTER USER

MariaDB starting with [10.2.0](#)

The ALTER USER statement was introduced in [MariaDB 10.2.0](#).

Syntax

```
ALTER USER [IF EXISTS]
  user_specification [,user_specification] ...
  [REQUIRE {NONE | tls_option [[AND] tls_option] ...}]
  [WITH resource_option [resource_option] ...]
  [Lock_option] [password_option]

  user_specification:
    username [authentication_option]

  authentication_option:
    IDENTIFIED BY 'password'
    | IDENTIFIED BY PASSWORD 'password_hash'
    | IDENTIFIED {VIA|WITH} authentication_rule [OR authentication_rule] ...

  authentication_rule:
    authentication_plugin
    | authentication_plugin {USING|AS} 'authentication_string'
    | authentication_plugin {USING|AS} PASSWORD('password')

  tls_option
    SSL
    | X509
    | CIPHER 'cipher'
    | ISSUER 'issuer'
    | SUBJECT 'subject'

  resource_option
    MAX_QUERIES_PER_HOUR count
    | MAX_UPDATES_PER_HOUR count
    | MAX_CONNECTIONS_PER_HOUR count
    | MAX_USER_CONNECTIONS count
    | MAX_STATEMENT_TIME time

  password_option:
    PASSWORD EXPIRE
    | PASSWORD EXPIRE DEFAULT
    | PASSWORD EXPIRE NEVER
    | PASSWORD EXPIRE INTERVAL N DAY

  lock_option:
    ACCOUNT LOCK
    | ACCOUNT UNLOCK
}
```

Contents

1. [Syntax](#)
2. [Description](#)
3. [IF EXISTS](#)
4. [Account Names](#)
5. [Authentication Options](#)
 1. [IDENTIFIED BY 'password'](#)
 2. [IDENTIFIED BY PASSWORD
'password_hash'](#)
 3. [IDENTIFIED {VIA|WITH}
authentication_plugin](#)
6. [TLS Options](#)
7. [Resource Limit Options](#)
8. [Password Expiry](#)
9. [Account Locking](#)
10. [See Also](#)

Description

The `ALTER USER` statement modifies existing MariaDB accounts. To use it, you must have the global `CREATE USER` privilege or the `UPDATE` privilege for the `mysql` database. The global `SUPER` privilege is also required if the `read_only` system variable is enabled.

If any of the specified user accounts do not yet exist, an error results. If an error occurs, `ALTER USER` will still modify the accounts that do not result in an error. Only one error is produced for all users which have not been modified.

IF EXISTS

When the `IF EXISTS` clause is used, MariaDB will return a warning instead of an error for each specified user that does not exist.

Account Names

For `ALTER USER` statements, account names are specified as the `username` argument in the same way as they are for `CREATE USER` statements. See [account names](#) from the `CREATE USER` page for details on how account names are specified.

`CURRENT_USER` or `CURRENT_USER()` can also be used to alter the account logged into the current session. For example, to change the current user's password to `mariadb`:

```
ALTER USER CURRENT_USER() IDENTIFIED BY 'mariadb';
```

Authentication Options

MariaDB starting with [10.4](#)

From [MariaDB 10.4](#), it is possible to use more than one authentication plugin for each user account. For example, this can be useful to slowly migrate users to the more secure ed25519 authentication plugin over time, while allowing the old `mysql_native_password` authentication plugin as an alternative for the transitional period. See [Authentication from MariaDB 10.4](#) for more.

When running `ALTER USER`, not specifying an authentication option in the `IDENTIFIED VIA` clause will remove that authentication method. (However this was not the case before [MariaDB 10.4.13](#), see [MDEV-21928](#))

For example, a user is created with the ability to authenticate via both a password and `unix_socket`:

```

CREATE USER 'bob'@'localhost'
  IDENTIFIED VIA mysql_native_password USING PASSWORD('pwd')
  OR unix_socket;

SHOW CREATE USER 'bob'@'localhost'\G
***** 1. row *****
CREATE USER for bob@localhost: CREATE USER `bob`@`localhost`
  IDENTIFIED VIA mysql_native_password
  USING '*975B2CD4FF9AE554FE8AD33168FBFC326D2021DD'
  OR unix_socket

```

If the user's password is updated, but unix_socket authentication is not specified in the IDENTIFIED VIA clause, unix_socket authentication will no longer be permitted.

```

ALTER USER 'bob'@'localhost' IDENTIFIED VIA mysql_native_password
  USING PASSWORD('pwd2');

SHOW CREATE USER 'bob'@'localhost'\G
***** 1. row *****
CREATE USER for bob@localhost: CREATE USER `bob`@`localhost`
  IDENTIFIED BY PASSWORD '*38366FDA01695B6A5A9DD4E428D9FB8F7EB75512'

```

IDENTIFIED BY 'password'

The optional IDENTIFIED BY clause can be used to provide an account with a password. The password should be specified in plain text. It will be hashed by the [PASSWORD](#) function prior to being stored to the [mysql.user](#) table.

For example, if our password is `mariadb`, then we can set the account's password with:

```
ALTER USER foo2@test IDENTIFIED BY 'mariadb';
```

If you do not specify a password with the IDENTIFIED BY clause, the user will be able to connect without a password. A blank password is not a wildcard to match any password. The user must connect without providing a password if no password is set.

The only [authentication plugins](#) that this clause supports are [mysql_native_password](#) and [mysql_old_password](#).

IDENTIFIED BY PASSWORD 'password_hash'

The optional IDENTIFIED BY PASSWORD clause can be used to provide an account with a password that has already been hashed. The password should be specified as a hash that was provided by the [PASSWORD](#) function. It will be stored to the [mysql.user](#) table as-is.

For example, if our password is `mariadb`, then we can find the hash with:

```
SELECT PASSWORD('mariadb');
+-----+
| PASSWORD('mariadb') |
+-----+
| *54958E764CE10E50764C2EECBB71D01F08549980 |
+-----+
```

And then we can set an account's password with the hash:

```
ALTER USER foo2@test
  IDENTIFIED BY PASSWORD '*54958E764CE10E50764C2EECBB71D01F08549980';
```

If you do not specify a password with the IDENTIFIED BY clause, the user will be able to connect without a password. A blank password is not a wildcard to match any password. The user must connect without providing a password if no password is set.

The only [authentication plugins](#) that this clause supports are [mysql_native_password](#) and

IDENTIFIED {VIA|WITH} authentication_plugin

The optional `IDENTIFIED VIA` `authentication_plugin` allows you to specify that the account should be authenticated by a specific [authentication plugin](#). The plugin name must be an active authentication plugin as per `SHOW PLUGINS`. If it doesn't show up in that output, then you will need to install it with `INSTALL PLUGIN` or `INSTALL SONAME`.

For example, this could be used with the [PAM authentication plugin](#):

```
ALTER USER foo2@test IDENTIFIED VIA pam;
```

Some authentication plugins allow additional arguments to be specified after a `USING` or `AS` keyword. For example, the [PAM authentication plugin](#) accepts a service name:

```
ALTER USER foo2@test IDENTIFIED VIA pam USING 'mariadb';
```

The exact meaning of the additional argument would depend on the specific authentication plugin.

In [MariaDB 10.4](#) and later, the `USING` or `AS` keyword can also be used to provide a plain-text password to a plugin if it's provided as an argument to the [PASSWORD\(\)](#) function. This is only valid for [authentication plugins](#) that have implemented a hook for the [PASSWORD\(\)](#) function. For example, the [ed25519](#) authentication plugin supports this:

```
ALTER USER safe@'%' IDENTIFIED VIA ed25519 USING PASSWORD('secret');
```

TLS Options

By default, MariaDB transmits data between the server and clients without encrypting it. This is generally acceptable when the server and client run on the same host or in networks where security is guaranteed through other means. However, in cases where the server and client exist on separate networks or they are in a high-risk network, the lack of encryption does introduce security concerns as a malicious actor could potentially eavesdrop on the traffic as it is sent over the network between them.

To mitigate this concern, MariaDB allows you to encrypt data in transit between the server and clients using the Transport Layer Security (TLS) protocol. TLS was formerly known as Secure Socket Layer (SSL), but strictly speaking the SSL protocol is a predecessor to TLS and, that version of the protocol is now considered insecure. The documentation still uses the term SSL often and for compatibility reasons TLS-related server system and status variables still use the prefix `ssl_`, but internally, MariaDB only supports its secure successors.

See [Secure Connections Overview](#) for more information about how to determine whether your MariaDB server has TLS support.

You can set certain TLS-related restrictions for specific user accounts. For instance, you might use this with user accounts that require access to sensitive data while sending it across networks that you do not control. These restrictions can be enabled for a user account with the [CREATE USER](#), [ALTER USER](#), or [GRANT](#) statements. The following options are available:

Option	Description
REQUIRE NONE	TLS is not required for this account, but can still be used.
REQUIRE SSL	The account must use TLS, but no valid X509 certificate is required. This option cannot be combined with other TLS options.
REQUIRE X509	The account must use TLS and must have a valid X509 certificate. This option implies <code>REQUIRE SSL</code> . This option cannot be combined with other TLS options.
REQUIRE ISSUER 'issuer'	The account must use TLS and must have a valid X509 certificate. Also, the Certificate Authority must be the one specified via the string <code>issuer</code> . This option implies <code>REQUIRE X509</code> . This option can be combined with the <code>SUBJECT</code> , and <code>CIPHER</code> options in any order.

REQUIRE SUBJECT 'subject'	The account must use TLS and must have a valid X509 certificate. Also, the certificate's Subject must be the one specified via the string <code>subject</code> . This option implies REQUIRE x509 . This option can be combined with the ISSUER , and CIPHER options in any order.
REQUIRE CIPHER 'cipher'	The account must use TLS, but no valid X509 certificate is required. Also, the encryption used for the connection must use a specific cipher method specified in the string <code>cipher</code> . This option implies REQUIRE SSL . This option can be combined with the ISSUER , and SUBJECT options in any order.

The REQUIRE keyword must be used only once for all specified options, and the AND keyword can be used to separate individual options, but it is not required.

For example, you can alter a user account to require these TLS options with the following:

```
ALTER USER 'alice'@'%'
REQUIRE SUBJECT '/CN=alice/0=My Dom, Inc./C=US/ST=Oregon/L=Portland' AND
ISSUER '/C=FI/ST=Somewhere/L=City/ O=Some Company/CN=Peter Parker/emailAddress=p.parker@mar
AND CIPHER 'SHA-DES-CBC3-EDH-RSA';
```

If any of these options are set for a specific user account, then any client who tries to connect with that user account will have to be configured to connect with TLS.

See [Securing Connections for Client and Server](#) for information on how to enable TLS on the client and server.

Resource Limit Options

MariaDB starting with [10.2.0](#)

MariaDB [10.2.0](#) introduced a number of resource limit options.

It is possible to set per-account limits for certain server resources. The following table shows the values that can be set per account:

Limit Type	Description
MAX_QUERIES_PER_HOUR	Number of statements that the account can issue per hour (including updates)
MAX_UPDATES_PER_HOUR	Number of updates (not queries) that the account can issue per hour
MAX_CONNECTIONS_PER_HOUR	Number of connections that the account can start per hour
MAX_USER_CONNECTIONS	Number of simultaneous connections that can be accepted from the same account; if it is 0, <code>max_connections</code> will be used instead; if <code>max_connections</code> is 0, there is no limit for this account's simultaneous connections.
MAX_STATEMENT_TIME	Timeout, in seconds, for statements executed by the user. See also Aborting Statements that Exceed a Certain Time to Execute .

If any of these limits are set to 0 , then there is no limit for that resource for that user.

Here is an example showing how to set an account's resource limits:

```
ALTER USER 'someone'@'localhost' WITH
  MAX_USER_CONNECTIONS 10
  MAX_QUERIES_PER_HOUR 200;
```

The resources are tracked per account, which means '`user'@'server'`' ; not per user name or per connection.

The count can be reset for all users using [FLUSH USER_RESOURCES](#), [FLUSH PRIVILEGES](#) or [mysqladmin reload](#).

Per account resource limits are stored in the `user` table, in the `mysql` database. Columns used for resources limits are named `max_questions` , `max_updates` , `max_connections` (for

Password Expiry

MariaDB starting with [10.4.3](#)

Besides automatic password expiry, as determined by [default_password_lifetime](#), password expiry times can be set on an individual user basis, overriding the global setting, for example:

```
ALTER USER 'monty'@'localhost' PASSWORD EXPIRE INTERVAL 120 DAY;
ALTER USER 'monty'@'localhost' PASSWORD EXPIRE NEVER;
ALTER USER 'monty'@'localhost' PASSWORD EXPIRE DEFAULT;
```

See [User Password Expiry](#) for more details.

Account Locking

MariaDB starting with [10.4.2](#)

Account locking permits privileged administrators to lock/unlock user accounts. No new client connections will be permitted if an account is locked (existing connections are not affected). For example:

```
ALTER USER 'marijn'@'localhost' ACCOUNT LOCK;
```

See [Account Locking](#) for more details.

From [MariaDB 10.4.7](#) and [MariaDB 10.5.8](#), the *lock_option* and *password_option* clauses can occur in either order.

See Also

- [Authentication from MariaDB 10.4](#)
- [GRANT](#)
- [CREATE USER](#)
- [DROP USER](#)
- [SET PASSWORD](#)
- [SHOW CREATE USER](#)
- [mysql.user table](#)
- [Password Validation Plugins](#) - permits the setting of basic criteria for passwords
- [Authentication Plugins](#) - allow various authentication methods to be used, and new ones to be developed.

H3Dr1.1.3 DROP USER

Syntax

```
DROP USER [IF EXISTS] user_name [, user_name] ...
```

Contents

1. [Syntax](#)
2. [Description](#)
 1. [IF EXISTS](#)
3. [Examples](#)
4. [See Also](#)

Description

The `DROP USER` statement removes one or more MariaDB accounts. It removes privilege rows for the account from all grant tables. To use this statement, you must have the global [CREATE USER](#) privilege or the [DELETE](#) privilege for the `mysql` database. Each account is named using the same format as for the `CREATE USER` statement; for example, `'jeffrey'@'localhost'`. If you specify only the user name part of the account name, a host name part of `'%'` is used. For additional information about specifying account names, see [CREATE USER](#).

Note that, if you specify an account that is currently connected, it will not be deleted until the connection is closed. The connection will not be automatically closed.

If any of the specified user accounts do not exist, `ERROR 1396 (HY000)` results. If an error occurs, `DROP USER` will still drop the accounts that do not result in an error. Only one error is produced for all users which have not been dropped:

```
ERROR 1396 (HY000): Operation DROP USER failed for 'u1'@'%','u2'@'%'
```

Failed `CREATE` or `DROP` operations, for both users and roles, produce the same error code.

IF EXISTS

If the `IF EXISTS` clause is used, MariaDB will return a note instead of an error if the user does not exist.

Examples

```
DROP USER bob;
```

IF EXISTS :

```
DROP USER bob;
ERROR 1396 (HY000): Operation DROP USER failed for 'bob'@'%'
```

```
DROP USER IF EXISTS bob;
Query OK, 0 rows affected, 1 warning (0.00 sec)
```

```
SHOW WARNINGS;
+-----+
| Level | Code | Message |
+-----+
| Note  | 1974 | Can't drop user 'bob'@'%'; it doesn't exist |
+-----+
```

See Also

- [CREATE USER](#)
- [ALTER USER](#)
- [GRANT](#)
- [SHOW CREATE USER](#)
- [mysql.user table](#)

H3Dr1.1.4 GRANT

Contents

- 1. [Syntax](#)
- 2. [Description](#)
- 3. [Account Names](#)
- 4. [Implicit Account Creation](#)
- 5. [Privilege Levels](#)
 - 1. [The USAGE Privilege](#)
 - 2. [The ALL PRIVILEGES Privilege](#)
 - 3. [The GRANT OPTION Privilege](#)
 - 4. [Global Privileges](#)
 - 1. [BINLOG ADMIN](#)
 - 2. [BINLOG MONITOR](#)
 - 3. [BINLOG REPLAY](#)
 - 4. [CONNECTION ADMIN](#)
 - 5. [CREATE USER](#)
 - 6. [FEDERATED ADMIN](#)
 - 7. [FILE](#)
 - 8. [GRANT OPTION](#)
 - 9. [PROCESS](#)
 - 10. [READ_ONLY ADMIN](#)
 - 11. [RELOAD](#)
 - 12. [REPLICATION CLIENT](#)
 - 13. [REPLICATION MASTER ADMIN](#)
 - 14. [REPLICA MONITOR](#)
 - 15. [REPLICATION REPLICA](#)
 - 16. [REPLICATION SLAVE](#)
 - 17. [REPLICATION SLAVE ADMIN](#)
 - 18. [SET USER](#)
 - 19. [SHOW DATABASES](#)
 - 20. [SHUTDOWN](#)
 - 21. [SUPER](#)
 - 5. [Database Privileges](#)
 - 6. [Table Privileges](#)
 - 7. [Column Privileges](#)
 - 8. [Function Privileges](#)
 - 9. [Procedure Privileges](#)
 - 10. [Proxy Privileges](#)
- 3. [Authentication Options](#)
 - 1. [IDENTIFIED BY 'password'](#)
 - 2. [IDENTIFIED BY PASSWORD 'password_hash'](#)
 - 3. [IDENTIFIED {VIA|WITH} authentication_plugin](#)
- 7. [Resource Limit Options](#)
- 3. [TLS Options](#)
- 3. [Roles](#)
 - 1. [Syntax](#)
- 3. [Grant Examples](#)
 - 1. [Granting Root-like Privileges](#)
- 1. [See Also](#)

Syntax

```

GRANT
    priv_type [(column_list)]
        [, priv_type (column_list)] ...
    ON [object_type] priv_level
    TO user_specification [user_options ...]

user_specification:
    username [authentication_option]

authentication_option:
    IDENTIFIED BY 'password'
    | IDENTIFIED BY PASSWORD 'password_hash'
    | IDENTIFIED {VIA|WITH} authentication_rule [OR authentication_rule ...]

authentication_rule:
    authentication_plugin
    | authentication_plugin {USING|AS} 'authentication_string'
    | authentication_plugin {USING|AS} PASSWORD('password')

GRANT PROXY ON username
    TO user_specification [, user_specification ...]
    [WITH GRANT OPTION]

GRANT rolename TO grantee [, grantee ...]
    [WITH ADMIN OPTION]

grantee:
    rolename
    username [authentication_option]

user_options:
    [REQUIRE {NONE | tls_option [[AND] tls_option] ...}]
    [WITH with_option [with_option] ...]

object_type:
    TABLE
    | FUNCTION
    | PROCEDURE
    | PACKAGE

priv_level:
    *
    | *.*
    | db_name.*
    | db_name.tbl_name
    | tbl_name
    | db_name.routine_name

with_option:
    GRANT OPTION
    | resource_option

resource_option:
    MAX_QUERIES_PER_HOUR count
    | MAX_UPDATES_PER_HOUR count
    | MAX_CONNECTIONS_PER_HOUR count
    | MAX_USER_CONNECTIONS count
    | MAX_STATEMENT_TIME time

tls_option:
    SSL
    | X509
    | CIPHER 'cipher'
    | ISSUER 'issuer'
    | SUBJECT 'subject'

```

Description

The `GRANT` statement allows you to grant privileges or [roles](#) to accounts. To use `GRANT`, you must have

the `GRANT OPTION` privilege, and you must have the privileges that you are granting.

Use the `REVOKE` statement to revoke privileges granted with the `GRANT` statement.

Use the `SHOW GRANTS` statement to determine what privileges an account has.

Account Names

For `GRANT` statements, account names are specified as the `username` argument in the same way as they are for `CREATE USER` statements. See [account names](#) from the `CREATE USER` page for details on how account names are specified.

Implicit Account Creation

The `GRANT` statement also allows you to implicitly create accounts in some cases.

If the account does not yet exist, then `GRANT` can implicitly create it. To implicitly create an account with `GRANT`, a user is required to have the same privileges that would be required to explicitly create the account with the `CREATE USER` statement.

If the `NO_AUTO_CREATE_USER` [SQL_MODE](#) is set, then accounts can only be created if authentication information is specified, or with a `CREATE USER` statement. If no authentication information is provided, `GRANT` will produce an error when the specified account does not exist, for example:

```
show variables like '%sql_mode%' ;
+-----+-----+
| Variable_name | Value           |
+-----+-----+
| sql_mode      | NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION |
+-----+-----+

GRANT USAGE ON *.* TO 'user123'@'%' IDENTIFIED BY '';
ERROR 1133 (28000): Can't find any matching row in the user table

GRANT USAGE ON *.* TO 'user123'@'%'
  IDENTIFIED VIA PAM using 'mariadb' require ssl ;
Query OK, 0 rows affected (0.00 sec)

select host, user from mysql.user where user='user123' ;

+-----+-----+
| host | user   |
+-----+-----+
| %    | user123 |
+-----+-----+
```

Privilege Levels

Privileges can be set globally, for an entire database, for a table or routine, or for individual columns in a table. Certain privileges can only be set at certain levels.

- [Global privileges](#) `priv_type` are granted using `*.*` for `priv_level`. Global privileges include privileges to administer the database and manage user accounts, as well as privileges for all tables, functions, and procedures. Global privileges are stored in the [mysql.user table](#) prior to MariaDB 10.4, and in [mysql.global_priv table](#) afterwards.
- [Database privileges](#) `priv_type` are granted using `db_name.*` for `priv_level`, or using just `*` to use default database. Database privileges include privileges to create tables and functions, as well as privileges for all tables, functions, and procedures in the database. Database privileges are stored in the [mysql.db table](#).
- [Table privileges](#) `priv_type` are granted using `db_name.tbl_name` for `priv_level`, or using just `tbl_name` to specify a table in the default database. The `TABLE` keyword is optional. Table privileges include the ability to select and change data in the table. Certain table privileges can be granted for individual columns.
- [Column privileges](#) `priv_type` are granted by specifying a table for `priv_level` and providing a column list after the privilege type. They allow you to control exactly which columns in a table users can select and change.

- Function privileges *priv_type* are granted using `FUNCTION db_name.routine_name` for *priv_level*, or using just `FUNCTION routine_name` to specify a function in the default database.
- Procedure privileges *priv_type* are granted using `PROCEDURE db_name.routine_name` for *priv_level*, or using just `PROCEDURE routine_name` to specify a procedure in the default database.

The USAGE Privilege

The `USAGE` privilege grants no real privileges. The `SHOW GRANTS` statement will show a global `USAGE` privilege for a newly-created user. You can use `USAGE` with the `GRANT` statement to change options like `GRANT OPTION` and `MAX_USER_CONNECTIONS` without changing any account privileges.

The ALL PRIVILEGES Privilege

The `ALL PRIVILEGES` privilege grants all available privileges. Granting all privileges only affects the given privilege level. For example, granting all privileges on a table does not grant any privileges on the database or globally.

Using `ALL PRIVILEGES` does not grant the special `GRANT OPTION` privilege.

You can use `ALL` instead of `ALL PRIVILEGES`.

The GRANT OPTION Privilege

Use the `WITH GRANT OPTION` clause to give users the ability to grant privileges to other users at the given privilege level. Users with the `GRANT OPTION` privilege can only grant privileges they have. They cannot grant privileges at a higher privilege level than they have the `GRANT OPTION` privilege.

The `GRANT OPTION` privilege cannot be set for individual columns. If you use `WITH GRANT OPTION` when specifying `column privileges`, the `GRANT OPTION` privilege will be granted for the entire table.

Using the `WITH GRANT OPTION` clause is equivalent to listing `GRANT OPTION` as a privilege.

Global Privileges

The following table lists the privileges that can be granted globally. You can also grant all database, table, and function privileges globally. When granted globally, these privileges apply to all databases, tables, or functions, including those created later.

To set a global privilege, use `*.*` for *priv_level*.

BINLOG ADMIN

Enables administration of the [binary log](#), including the [PURGE BINARY LOGS](#) statement and setting the system variables:

- [binlog_annotation_row_events](#)
- [binlog_cache_size](#)
- [binlog_commit_wait_count](#)
- [binlog_commit_wait_usecs](#)
- [binlog_direct_non_transactional_updates](#)
- [binlog_expire_logs_seconds](#)
- [binlog_file_cache_size](#)
- [binlog_format](#)
- [binlog_row_image](#)
- [binlog_row_metadata](#)
- [binlog_stmt_cache_size](#)
- [expire_logs_days](#)
- [log_bin_compress](#)
- [log_bin_compress_min_len](#)
- [log_bin_trust_function_creators](#)
- [max_binlog_cache_size](#)
- [max_binlog_size](#)
- [max_binlog_stmt_cache_size](#)
- [sql_log_bin](#) and

- [sync_binlog](#).

Added in [MariaDB 10.5.2](#).

BINLOG MONITOR

New name for [REPLICATION CLIENT](#) from [MariaDB 10.5.2](#), ([REPLICATION CLIENT](#) still supported as an alias for compatibility purposes). Permits running SHOW commands related to the [binary log](#), in particular the [SHOW BINLOG STATUS](#) and [SHOW BINARY LOGS](#) statements. Unlike [REPLICATION CLIENT](#) prior to [MariaDB 10.5](#), [SHOW REPLICA STATUS](#) isn't included in this privilege, and [REPLICA MONITOR](#) is required.

BINLOG REPLAY

Enables replaying the binary log with the [BINLOG](#) statement (generated by [mariadb-binlog](#)), executing [SET timestamp](#) when [secure_timestamp](#) is set to `replication`, and setting the session values of system variables usually included in BINLOG output, in particular:

- [gtid_domain_id](#)
- [gtid_seq_no](#)
- [pseudo_thread_id](#)
- [server_id](#).

Added in [MariaDB 10.5.2](#).

CONNECTION ADMIN

Enables administering connection resource limit options. This includes ignoring the limits specified by:

- [max_connections](#)
- [max_user_connections](#) and
- [max_password_errors](#).

The statements specified in [init_connect](#) are not executed, [killing connections and queries](#) owned by other users is permitted. The following connection-related system variables can be changed:

- [connect_timeout](#)
- [disconnect_on_expired_password](#)
- [extra_max_connections](#)
- [init_connect](#)
- [max_connections](#)
- [max_connect_errors](#)
- [max_password_errors](#)
- [proxy_protocol_networks](#)
- [secure_auth](#)
- [slow_launch_time](#)
- [thread_pool_exact_stats](#)
- [thread_pool_dedicated_listener](#)
- [thread_pool_idle_timeout](#)
- [thread_pool_max_threads](#)
- [thread_pool_min_threads](#)
- [thread_pool_oversubscribe](#)
- [thread_pool_prio_kickup_timer](#)
- [thread_pool_priority](#)
- [thread_pool_size](#), and
- [thread_pool_stall_limit](#).

Added in [MariaDB 10.5.2](#).

CREATE USER

Create a user using the [CREATE USER](#) statement, or implicitly create a user with the [GRANT](#) statement.

FEDERATED ADMIN

Execute [CREATE SERVER](#), [ALTER SERVER](#), and [DROP SERVER](#) statements. Added in [MariaDB 10.5.2](#).

FILE

Read and write files on the server, using statements like [LOAD DATA INFILE](#) or functions like [LOAD_FILE\(\)](#). Also needed to create [CONNECT](#) outward tables. MariaDB server must have the permissions to access those files.

GRANT OPTION

Grant global privileges. You can only grant privileges that you have.

PROCESS

Show information about the active processes, for example via [SHOW PROCESSLIST](#) or [mysqladmin processlist](#). If you have the PROCESS privilege, you can see all threads. Otherwise, you can see only your own threads (that is, threads associated with the MariaDB account that you are using).

READ_ONLY ADMIN

User can set the [read_only](#) system variable and allows the user to perform write operations, even when the `read_only` option is active. Added in [MariaDB 10.5.2](#).

RELOAD

Execute [FLUSH](#) statements or equivalent [mariadb-admin/mysqladmin](#) commands.

REPLICATION CLIENT

Execute [SHOW MASTER STATUS](#) and [SHOW BINARY LOGS](#) informative statements. Renamed to [BINLOG MONITOR](#) in [MariaDB 10.5.2](#) (but still supported as an alias for compatibility reasons). [SHOW SLAVE STATUS](#) was part of [REPLICATION CLIENT](#) prior to [MariaDB 10.5](#).

REPLICATION MASTER ADMIN

Permits administration of primary servers, including the [SHOW REPLICA HOSTS](#) statement, and setting the [gtid_binlog_state](#), [gtid_domain_id](#), [master_verify_checksum](#) and [server_id](#) system variables. Added in [MariaDB 10.5.2](#).

REPLICA MONITOR

Permit [SHOW REPLICA STATUS](#) and [SHOW RELAYLOG EVENTS](#). From [MariaDB 10.5.9](#).

When a user would upgrade from an older major release to a [MariaDB 10.5](#) minor release prior to [MariaDB 10.5.9](#), certain user accounts would lose capabilities. For example, a user account that had the REPLICATION CLIENT privilege in older major releases could run [SHOW REPLICA STATUS](#), but after upgrading to a [MariaDB 10.5](#) minor release prior to [MariaDB 10.5.9](#), they could no longer run [SHOW REPLICA STATUS](#), because that statement was changed to require the REPLICATION REPLICA ADMIN privilege.

This issue is fixed in [MariaDB 10.5.9](#) with this new privilege, which now grants the user the ability to execute `SHOW [ALL] (SLAVE | REPLICA) STATUS`.

When a database is upgraded from an older major release to MariaDB Server 10.5.9 or later, any user accounts with the REPLICATION CLIENT or REPLICATION SLAVE privileges will automatically be granted the new REPLICATION MONITOR privilege. The privilege fix occurs when the server is started up, not when mariadb-upgrade is performed.

However, when a database is upgraded from an early 10.5 minor release to 10.5.9 and later, the user will have to fix any user account privileges manually.

REPLICATION REPLICA

Synonym for [REPLICATION SLAVE](#). From [MariaDB 10.5.1](#).

REPLICATION SLAVE

Accounts used by replica servers on the primary need this privilege. This is needed to get the updates made on the master. From MariaDB 10.5.1, REPLICATION REPLICA is an alias for REPLICATION SLAVE .

REPLICATION SLAVE ADMIN

Permits administering replica servers, including START REPLICA/SLAVE, STOP REPLICA/SLAVE, CHANGE MASTER, SHOW REPLICA/SLAVE STATUS, SHOW RELAYLOG EVENTS statements, replaying the binary log with the BINLOG statement (generated by mariadb-binlog), and setting the system variables:

- gtid_cleanup_batch_size
- gtid_ignore_duplicates
- gtid_pos_auto_engines
- gtid_slave_pos
- gtid_strict_mode
- init_slave
- read_binlog_speed_limit
- relay_log_purge
- relay_log_recovery
- replicate_do_db
- replicate_do_table
- replicate_events_marked_for_skip
- replicate_ignore_db
- replicate_ignore_table
- replicate_wild_do_table
- replicate_wild_ignore_table
- slave_compressed_protocol
- slave_ddl_exec_mode
- slave_domain_parallel_threads
- slave_exec_mode
- slave_max_allowed_packet
- slave_net_timeout
- slave_parallel_max_queued
- slave_parallel_mode
- slave_parallel_threads
- slave_parallel_workers
- slave_run_triggers_for_rbr
- slave_sql_verify_checksum
- slave_transaction_retry_interval
- slave_type_conversions
- sync_master_info
- sync_relay_log, and
- sync_relay_log_info.

Added in MariaDB 10.5.2.

SET USER

Enables setting the DEFINER when creating triggers, views, stored functions and stored procedures. Added in MariaDB 10.5.2.

SHOW DATABASES

List all databases using the SHOW DATABASES statement. Without the SHOW DATABASES privilege, you can still issue the SHOW DATABASES statement, but it will only list databases containing tables on which you have privileges.

SHUTDOWN

Shut down the server using SHUTDOWN or the mysqladmin shutdown command.

SUPER

Execute superuser statements: [CHANGE MASTER TO](#), [KILL](#) (users who do not have this privilege can only [KILL](#) their own threads), [PURGE LOGS](#), [SET global system variables](#), or the [mysqladmin debug](#) command. Also, this permission allows the user to write data even if the [read_only](#) startup option is set, enable or disable logging, enable or disable replication on replica, specify a [DEFINER](#) for statements that support that clause, connect once reaching the [MAX_CONNECTIONS](#). If a statement has been specified for the [init-connect](#) mysqld option, that command will not be executed when a user with [SUPER](#) privileges connects to the server.

The [SUPER](#) privilege has been split into multiple smaller privileges from MariaDB 10.5.2 to allow for more fine-grained privileges, although it remains an alias for these smaller privileges.

Database Privileges

The following table lists the privileges that can be granted at the database level. You can also grant all table and function privileges at the database level. Table and function privileges on a database apply to all tables or functions in that database, including those created later.

To set a privilege for a database, specify the database using `db_name.*` for *priv_level*, or just use `*` to specify the default database.

Privilege	Description
CREATE	Create a database using the CREATE DATABASE statement, when the privilege is granted for a database. You can grant the <code>CREATE</code> privilege on databases that do not yet exist. This also grants the <code>CREATE</code> privilege on all tables in the database.
CREATE ROUTINE	Create Stored Programs using the CREATE PROCEDURE and CREATE FUNCTION statements.
CREATE TEMPORARY TABLES	Create temporary tables with the CREATE TEMPORARY TABLE statement. This privilege enable writing and dropping those temporary tables
DROP	Drop a database using the DROP DATABASE statement, when the privilege is granted for a database. This also grants the <code>DROP</code> privilege on all tables in the database.
EVENT	Create, drop and alter <code>EVENT</code> s.
GRANT OPTION	Grant database privileges. You can only grant privileges that you have.
LOCK TABLES	Acquire explicit locks using the LOCK TABLES statement; you also need to have the <code>SELECT</code> privilege on a table, in order to lock it.

Table Privileges

Privilege	Description
ALTER	Change the structure of an existing table using the ALTER TABLE statement.
CREATE	Create a table using the CREATE TABLE statement. You can grant the <code>CREATE</code> privilege on tables that do not yet exist.
CREATE VIEW	Create a view using the CREATE VIEW statement.
DELETE	Remove rows from a table using the DELETE statement.
DELETE HISTORY	Remove historical rows from a table using the DELETE HISTORY statement. Displays as <code>DELETE VERSIONING ROWS</code> when running SHOW GRANTS until MariaDB 10.3.15 and until MariaDB 10.4.5 (MDEV-17655), or when running SHOW PRIVILEGES until MariaDB 10.5.2, MariaDB 10.4.13 and MariaDB 10.3.23 (MDEV-20382). From MariaDB 10.3.4. From MariaDB 10.3.5, if a user has the <code>SUPER</code> privilege but not this privilege, running mysql_upgrade will grant this privilege as well.
DROP	Drop a table using the DROP TABLE statement or a view using the DROP VIEW statement. Also required to execute the TRUNCATE TABLE statement.

GRANT OPTION	Grant table privileges. You can only grant privileges that you have.
INDEX	Create an index on a table using the CREATE INDEX statement. Without the <code>INDEX</code> privilege, you can still create indexes when creating a table using the CREATE TABLE statement if you have the <code>CREATE</code> privilege, and you can create indexes using the ALTER TABLE statement if you have the <code>ALTER</code> privilege.
INSERT	Add rows to a table using the INSERT statement. The <code>INSERT</code> privilege can also be set on individual columns; see Column Privileges below for details.
REFERENCES	Unused.
SELECT	Read data from a table using the SELECT statement. The <code>SELECT</code> privilege can also be set on individual columns; see Column Privileges below for details.
SHOW VIEW	Show the CREATE VIEW statement to create a view using the SHOW CREATE VIEW statement.
TRIGGER	Execute triggers associated to tables you update, execute the CREATE TRIGGER and DROP TRIGGER statements. You will still be able to see triggers.
UPDATE	Update existing rows in a table using the UPDATE statement. <code>UPDATE</code> statements usually include a <code>WHERE</code> clause to update only certain rows. You must have <code>SELECT</code> privileges on the table or the appropriate columns for the <code>WHERE</code> clause. The <code>UPDATE</code> privilege can also be set on individual columns; see Column Privileges below for details.

Column Privileges

Some table privileges can be set for individual columns of a table. To use column privileges, specify the table explicitly and provide a list of column names after the privilege type. For example, the following statement would allow the user to read the names and positions of employees, but not other information from the same table, such as salaries.

```
GRANT SELECT (name, position) ON Employee TO 'jeffrey'@'localhost';
```

Privilege	Description
INSERT (column_list)	Add rows specifying values in columns using the INSERT statement. If you only have column-level <code>INSERT</code> privileges, you must specify the columns you are setting in the <code>INSERT</code> statement. All other columns will be set to their default values, or <code>NULL</code> .
REFERENCES (column_list)	Unused.
SELECT (column_list)	Read values in columns using the SELECT statement. You cannot access or query any columns for which you do not have <code>SELECT</code> privileges, including in <code>WHERE</code> , <code>ON</code> , <code>GROUP BY</code> , and <code>ORDER BY</code> clauses.
UPDATE (column_list)	Update values in columns of existing rows using the UPDATE statement. <code>UPDATE</code> statements usually include a <code>WHERE</code> clause to update only certain rows. You must have <code>SELECT</code> privileges on the table or the appropriate columns for the <code>WHERE</code> clause.

Function Privileges

Privilege	Description
ALTER ROUTINE	Change the characteristics of a stored function using the ALTER FUNCTION statement.
EXECUTE	Use a stored function. You need <code>SELECT</code> privileges for any tables or columns accessed by the function.
GRANT OPTION	Grant function privileges. You can only grant privileges that you have.

Procedure Privileges

Privilege	Description
ALTER ROUTINE	Change the characteristics of a stored procedure using the ALTER PROCEDURE statement.
EXECUTE	Execute a stored procedure using the CALL statement. The privilege to call a procedure may allow you to perform actions you wouldn't otherwise be able to do, such as insert rows into a table.
GRANT OPTION	Grant procedure privileges. You can only grant privileges that you have.

```
GRANT EXECUTE ON PROCEDURE mysql.create_db TO maintainer;
```

Proxy Privileges

Privilege	Description
PROXY	Permits one user to be a proxy for another.

The `PROXY` privilege allows one user to proxy as another user, which means their privileges change to that of the proxy user, and the [CURRENT_USER\(\)](#) function returns the user name of the proxy user.

The `PROXY` privilege only works with authentication plugins that support it. The default [mysql_native_password](#) authentication plugin does not support proxy users.

The [pam](#) authentication plugin is the only plugin included with MariaDB that currently supports proxy users. The `PROXY` privilege is commonly used with the [pam](#) authentication plugin to enable [user and group mapping with PAM](#).

For example, to grant the `PROXY` privilege to an [anonymous account](#) that authenticates with the [pam](#) authentication plugin, you could execute the following:

```
CREATE USER 'dba'@'%' IDENTIFIED BY 'strongpassword';
GRANT ALL PRIVILEGES ON *.* TO 'dba'@'%';

CREATE USER ''@'%' IDENTIFIED VIA pam USING 'mariadb';
GRANT PROXY ON 'dba'@'%' TO ''@'%';
```

A user account can only grant the `PROXY` privilege for a specific user account if the grantor also has the `PROXY` privilege for that specific user account, and if that privilege is defined `WITH GRANT OPTION`. For example, the following example fails because the grantor does not have the `PROXY` privilege for that specific user account at all:

```
SELECT USER(), CURRENT_USER();
+-----+-----+
| USER()      | CURRENT_USER() |
+-----+-----+
| alice@localhost | alice@localhost |

SHOW GRANTS;
+-----+
| Grants for alice@localhost
|
+-----+
| GRANT ALL PRIVILEGES ON *.* TO 'alice'@'localhost' IDENTIFIED BY PASSWORD
'*2470C0C06DEE42FD1618BB99005ADCA2EC9D1E19' |
+-----+
| GRANT PROXY ON 'dba'@'localhost' TO 'bob'@'localhost';
ERROR 1698 (28000): Access denied for user 'alice'@'localhost'
```

And the following example fails because the grantor does have the `PROXY` privilege for that specific user account, but it is not defined `WITH GRANT OPTION`:

```
SELECT USER(), CURRENT_USER();
+-----+
| USER()      | CURRENT_USER() |
+-----+
| alice@localhost | alice@localhost |
+-----+


SHOW GRANTS;
+-----+
| Grants for alice@localhost
|
+-----+
| GRANT ALL PRIVILEGES ON *.* TO 'alice'@'localhost' IDENTIFIED BY PASSWORD
'*2470C0C06DEE42FD1618BB99005ADCA2EC9D1E19' |
| GRANT PROXY ON 'dba'@'localhost' TO 'alice'@'localhost'
|
+-----+
| GRANT PROXY ON 'dba'@'localhost' TO 'bob'@'localhost';
ERROR 1698 (28000): Access denied for user 'alice'@'localhost'
```

But the following example succeeds because the grantor does have the `PROXY` privilege for that specific user account, and it is defined `WITH GRANT OPTION`:

```
SELECT USER(), CURRENT_USER();
+-----+
| USER()      | CURRENT_USER() |
+-----+
| alice@localhost | alice@localhost |
+-----+


SHOW GRANTS;
+-----+
| Grants for alice@localhost
|
+-----+
| GRANT ALL PRIVILEGES ON *.* TO 'alice'@'localhost' IDENTIFIED BY PASSWORD
'*2470C0C06DEE42FD1618BB99005ADCA2EC9D1E19' WITH GRANT OPTION |
| GRANT PROXY ON 'dba'@'localhost' TO 'alice'@'localhost' WITH GRANT OPTION
|
+-----+
| GRANT PROXY ON 'dba'@'localhost' TO 'bob'@'localhost';
```

A user account can grant the `PROXY` privilege for any other user account if the grantor has the `PROXY` privilege for the '`'@'%'`' anonymous user account, like this:

```
GRANT PROXY ON ''@''%'' TO 'dba'@'localhost' WITH GRANT OPTION;
```

For example, the following example succeeds because the user can grant the `PROXY` privilege for any other user account:

```

SELECT USER(), CURRENT_USER();
+-----+
| USER()      | CURRENT_USER() |
+-----+
| alice@localhost | alice@localhost |
+-----+

SHOW GRANTS;
+-----+
| Grants for alice@localhost
|
+-----+
| GRANT ALL PRIVILEGES ON *.* TO 'alice'@'localhost' IDENTIFIED BY PASSWORD
'*2470C0C06DEE42FD1618BB99005ADCA2EC9D1E19' WITH GRANT OPTION |
| GRANT PROXY ON ''@ '%' TO 'alice'@'localhost' WITH GRANT OPTION
|
+-----+
| GRANT PROXY ON 'app1_db'@'localhost' TO 'bob'@'localhost';
Query OK, 0 rows affected (0.004 sec)

GRANT PROXY ON 'app2_db'@'localhost' TO 'carol'@'localhost';
Query OK, 0 rows affected (0.004 sec)

```

The default `root` user accounts created by [mysql_install_db](#) have this privilege. For example:

```

GRANT ALL PRIVILEGES ON *.* TO 'root'@'localhost' WITH GRANT OPTION;
GRANT PROXY ON ''@ '%' TO 'root'@'localhost' WITH GRANT OPTION;

```

This allows the default `root` user accounts to grant the `PROXY` privilege for any other user account, and it also allows the default `root` user accounts to grant others the privilege to do the same.

Authentication Options

The authentication options for the `GRANT` statement are the same as those for the [CREATE USER](#) statement.

IDENTIFIED BY 'password'

The optional `IDENTIFIED BY` clause can be used to provide an account with a password. The password should be specified in plain text. It will be hashed by the [PASSWORD](#) function prior to being stored.

For example, if our password is `mariadb` , then we can create the user with:

```
GRANT USAGE ON *.* TO foo2@test IDENTIFIED BY 'mariadb';
```

If you do not specify a password with the `IDENTIFIED BY` clause, the user will be able to connect without a password. A blank password is not a wildcard to match any password. The user must connect without providing a password if no password is set.

If the user account already exists and if you provide the `IDENTIFIED BY` clause, then the user's password will be changed. You must have the privileges needed for the [SET PASSWORD](#) statement to change a user's password with `GRANT` .

The only [authentication plugins](#) that this clause supports are [mysql_native_password](#) and [mysql_old_password](#).

IDENTIFIED BY PASSWORD 'password_hash'

The optional `IDENTIFIED BY PASSWORD` clause can be used to provide an account with a password that has already been hashed. The password should be specified as a hash that was provided by the [PASSWORD](#) function. It will be stored as-is.

For example, if our password is `mariadb`, then we can find the hash with:

```
SELECT PASSWORD('mariadb');
+-----+
| PASSWORD('mariadb') |
+-----+
| *54958E764CE10E50764C2EECBB71D01F08549980 |
+-----+
1 row in set (0.00 sec)
```

And then we can create a user with the hash:

```
GRANT USAGE ON *.* TO foo2@test IDENTIFIED BY
PASSWORD '*54958E764CE10E50764C2EECBB71D01F08549980';
```

If you do not specify a password with the `IDENTIFIED BY` clause, the user will be able to connect without a password. A blank password is not a wildcard to match any password. The user must connect without providing a password if no password is set.

If the user account already exists and if you provide the `IDENTIFIED BY` clause, then the user's password will be changed. You must have the privileges needed for the `SET PASSWORD` statement to change a user's password with `GRANT`.

The only [authentication plugins](#) that this clause supports are `mysql_native_password` and `mysql_old_password`.

IDENTIFIED {VIA|WITH} authentication_plugin

The optional `IDENTIFIED VIA authentication_plugin` allows you to specify that the account should be authenticated by a specific [authentication plugin](#). The plugin name must be an active authentication plugin as per `SHOW PLUGINS`. If it doesn't show up in that output, then you will need to install it with `INSTALL PLUGIN` or `INSTALL SONAME`.

For example, this could be used with the [PAM authentication plugin](#):

```
GRANT USAGE ON *.* TO foo2@test IDENTIFIED VIA pam;
```

Some authentication plugins allow additional arguments to be specified after a `USING` or `AS` keyword. For example, the [PAM authentication plugin](#) accepts a [service name](#):

```
GRANT USAGE ON *.* TO foo2@test IDENTIFIED VIA pam USING 'mariadb';
```

The exact meaning of the additional argument would depend on the specific authentication plugin.

MariaDB starting with 10.4.0

The `USING` or `AS` keyword can also be used to provide a plain-text password to a plugin if it's provided as an argument to the `PASSWORD()` function. This is only valid for [authentication plugins](#) that have implemented a hook for the `PASSWORD()` function. For example, the `ed25519` authentication plugin supports this:

```
CREATE USER safe@'%' IDENTIFIED VIA ed25519
    USING PASSWORD('secret');
```

MariaDB starting with 10.4.3

One can specify many authentication plugins, they all work as alternatives ways of authenticating a user:

```
CREATE USER safe@'%' IDENTIFIED VIA ed25519
    USING PASSWORD('secret') OR unix_socket;
```

By default, when you create a user without specifying an authentication plugin, MariaDB uses the [mysql_native_password](#) plugin.

Resource Limit Options

MariaDB starting with [10.2.0](#)

[MariaDB 10.2.0](#) introduced a number of resource limit options.

It is possible to set per-account limits for certain server resources. The following table shows the values that can be set per account:

Limit Type	Description
MAX_QUERIES_PER_HOUR	Number of statements that the account can issue per hour (including updates)
MAX_UPDATES_PER_HOUR	Number of updates (not queries) that the account can issue per hour
MAX_CONNECTIONS_PER_HOUR	Number of connections that the account can start per hour
MAX_USER_CONNECTIONS	Number of simultaneous connections that can be accepted from the same account; if it is 0, <code>max_connections</code> will be used instead; if <code>max_connections</code> is 0, there is no limit for this account's simultaneous connections.
MAX_STATEMENT_TIME	Timeout, in seconds, for statements executed by the user. See also Aborting Statements that Exceed a Certain Time to Execute .

If any of these limits are set to 0, then there is no limit for that resource for that user.

To set resource limits for an account, if you do not want to change that account's privileges, you can issue a `GRANT` statement with the `USAGE` privilege, which has no meaning. The statement can name some or all limit types, in any order.

Here is an example showing how to set resource limits:

```
GRANT USAGE ON *.* TO 'someone'@'localhost' WITH
    MAX_USER_CONNECTIONS 0
    MAX_QUERIES_PER_HOUR 200;
```

The resources are tracked per account, which means '`user'@'server'`' ; not per user name or per connection.

The count can be reset for all users using [FLUSH USER_RESOURCES](#), [FLUSH PRIVILEGES](#) or [mysqladmin reload](#).

Users with the `CONNECTION ADMIN` privilege (in [MariaDB 10.5.2](#) and later) or the `SUPER` privilege are not restricted by `max_user_connections`, `max_connections`, or `max_password_errors`.

Per account resource limits are stored in the `user` table, in the `mysql` database. Columns used for resources limits are named `max_questions`, `max_updates`, `max_connections` (for `MAX_CONNECTIONS_PER_HOUR`), and `max_user_connections` (for `MAX_USER_CONNECTIONS`).

TLS Options

By default, MariaDB transmits data between the server and clients without encrypting it. This is generally acceptable when the server and client run on the same host or in networks where security is guaranteed through other means. However, in cases where the server and client exist on separate networks or they are in a high-risk network, the lack of encryption does introduce security concerns as a malicious actor could potentially eavesdrop on the traffic as it is sent over the network between them.

To mitigate this concern, MariaDB allows you to encrypt data in transit between the server and clients using the Transport Layer Security (TLS) protocol. TLS was formerly known as Secure Socket Layer (SSL), but strictly speaking the SSL protocol is a predecessor to TLS and, that version of the protocol is now considered insecure. The documentation still uses the term SSL often and for compatibility reasons TLS-related server system and status variables still use the prefix `ssl_`, but internally, MariaDB only

supports its secure successors.

See [Secure Connections Overview](#) for more information about how to determine whether your MariaDB server has TLS support.

You can set certain TLS-related restrictions for specific user accounts. For instance, you might use this with user accounts that require access to sensitive data while sending it across networks that you do not control. These restrictions can be enabled for a user account with the [CREATE USER](#), [ALTER USER](#), or [GRANT](#) statements. The following options are available:

Option	Description
REQUIRE NONE	TLS is not required for this account, but can still be used.
REQUIRE SSL	The account must use TLS, but no valid X509 certificate is required. This option cannot be combined with other TLS options.
REQUIRE X509	The account must use TLS and must have a valid X509 certificate. This option implies REQUIRE SSL . This option cannot be combined with other TLS options.
REQUIRE ISSUER 'issuer'	The account must use TLS and must have a valid X509 certificate. Also, the Certificate Authority must be the one specified via the string <code>issuer</code> . This option implies REQUIRE x509 . This option can be combined with the SUBJECT , and CIPHER options in any order.
REQUIRE SUBJECT 'subject'	The account must use TLS and must have a valid X509 certificate. Also, the certificate's Subject must be the one specified via the string <code>subject</code> . This option implies REQUIRE x509 . This option can be combined with the ISSUER , and CIPHER options in any order.
REQUIRE CIPHER 'cipher'	The account must use TLS, but no valid X509 certificate is required. Also, the encryption used for the connection must use a specific cipher method specified in the string <code>cipher</code> . This option implies REQUIRE SSL . This option can be combined with the ISSUER , and SUBJECT options in any order.

The REQUIRE keyword must be used only once for all specified options, and the AND keyword can be used to separate individual options, but it is not required.

For example, you can create a user account that requires these TLS options with the following:

```
GRANT USAGE ON *.* TO 'alice'@'%'
  REQUIRE SUBJECT '/CN=alice/O=My Dom, Inc./C=US/ST=Oregon/L=Portland'
    AND ISSUER '/C=FI/ST=Somewhere/L=City/ O=Some Company/CN=Peter
Parker/emailAddress=p.parker@marvel.com'
  AND CIPHER 'SHA-DES-CBC3-EDH-RSA';
```

If any of these options are set for a specific user account, then any client who tries to connect with that user account will have to be configured to connect with TLS.

See [Securing Connections for Client and Server](#) for information on how to enable TLS on the client and server.

Roles

Syntax

```
GRANT role TO grantee [, grantee ... ]
  [ WITH ADMIN OPTION ]

grantee:
  rolename
  username [authentication_option]
```

The GRANT statement is also used to grant the use a [role](#) to one or more users or other roles. In order to be able to grant a role, the grantor doing so must have permission to do so (see WITH ADMIN in the [CREATE ROLE](#) article).

Specifying the WITH ADMIN OPTION permits the grantee to in turn grant the role to another.

For example, the following commands show how to grant the same role to a couple different users.

```
GRANT journalist TO hulda;  
GRANT journalist TO berengar WITH ADMIN OPTION;
```

If a user has been granted a role, they do not automatically obtain all permissions associated with that role. These permissions are only in use when the user activates the role with the [SET ROLE](#) statement.

Grant Examples

Granting Root-like Privileges

You can create a user that has privileges similar to the default `root` accounts by executing the following:

```
CREATE USER 'alexander'@'localhost';  
GRANT ALL PRIVILEGES ON *.* TO 'alexander'@'localhost' WITH GRANT OPTION;
```

See Also

- [Troubleshooting Connection Issues](#)
- [--skip-grant-tables](#) allows you to start MariaDB without `GRANT`. This is useful if you lost your root password.
- [CREATE USER](#)
- [ALTER USER](#)
- [DROP USER](#)
- [SET PASSWORD](#)
- [SHOW CREATE USER](#)
- [mysql.global_priv table](#)
- [mysql.user table](#)
- [Password Validation Plugins](#) - permits the setting of basic criteria for passwords
- [Authentication Plugins](#) - allow various authentication methods to be used, and new ones to be developed.

H3Dr1.1.5 RENAME USER

Syntax

```
RENAME USER old_user TO new_user  
[, old_user TO new_user] ...
```

Description

The `RENAME USER` statement renames existing MariaDB accounts. To use it, you must have the global [CREATE USER](#) privilege or the [UPDATE](#) privilege for the `mysql` database. Each account is named using the same format as for the [CREATE USER](#) statement; for example, `'jeffrey'@'localhost'`. If you specify only the user name part of the account name, a host name part of `'%'` is used.

If any of the old user accounts do not exist or any of the new user accounts already exist, `ERROR 1396 (HY000)` results. If an error occurs, `RENAME USER` will still rename the accounts that do not result in an error.

Examples

```
CREATE USER 'donald', 'mickey';  
RENAME USER 'donald' TO 'duck'@'localhost', 'mickey' TO 'mouse'@'localhost';
```

H3Dr1.1.6 REVOKE

Contents

- 1. [Privileges](#)
 - 1. [Syntax](#)
 - 2. [Description](#)
 - 3. [Examples](#)
- 2. [Roles](#)
 - 1. [Syntax](#)
 - 2. [Description](#)
 - 3. [Example](#)

Privileges

Syntax

```
REVOKE  
    priv_type [(column_list)]  
    [, priv_type [(column_list)]] ...  
    ON [object_type] priv_level  
    FROM user [, user] ...  
  
REVOKE ALL PRIVILEGES, GRANT OPTION  
    FROM user [, user] ...
```

Description

The `REVOKE` statement enables system administrators to revoke privileges (or roles - see [section below](#)) from MariaDB accounts. Each account is named using the same format as for the `GRANT` statement; for example, '`'jeffrey'@'localhost'`'. If you specify only the user name part of the account name, a host name part of '`%`' is used. For details on the levels at which privileges exist, the allowable `priv_type` and `priv_level` values, and the syntax for specifying users and passwords, see [GRANT](#).

To use the first `REVOKE` syntax, you must have the `GRANT OPTION` privilege, and you must have the privileges that you are revoking.

To revoke all privileges, use the second syntax, which drops all global, database, table, column, and routine privileges for the named user or users:

```
REVOKE ALL PRIVILEGES, GRANT OPTION FROM user [, user] ...
```

To use this `REVOKE` syntax, you must have the global `CREATE USER` privilege or the `UPDATE` privilege for the `mysql` database. See [GRANT](#).

Examples

```
REVOKE SUPER ON *.* FROM 'alexander'@'localhost';
```

Roles

Syntax

```
REVOKE role [, role ...]  
    FROM grantee [, grantee2 ... ]  
  
REVOKE ADMIN OPTION FOR role FROM grantee [, grantee2]
```

Description

`REVOKE` is also used to remove a [role](#) from a user or another role that it's previously been assigned to. If a role has previously been set as a [default role](#), `REVOKE` does not remove the record of the default role from the [mysql.user](#) table. If the role is subsequently granted again, it will again be the user's default. Use `SET DEFAULT ROLE NONE` to explicitly remove this.

Before MariaDB 10.1.13, the `REVOKE` role statement was not permitted in [prepared statements](#).

Example

```
REVOKE journalist FROM hulda
```

H3Dr1.1.7 SET PASSWORD

Syntax

```
SET PASSWORD [FOR user] =
{
    PASSWORD('some password')
    | OLD_PASSWORD('some password')
    | 'encrypted password'
}
```

Contents

1. [Syntax](#)
2. [Description](#)
3. [Authentication Plugin Support](#)
4. [Passwordless User Accounts](#)
5. [Example](#)
6. [See Also](#)

Description

The `SET PASSWORD` statement assigns a password to an existing MariaDB user account.

If the password is specified using the `PASSWORD()` or `OLD_PASSWORD()` function, the literal text of the password should be given. If the password is specified without using either function, the password should be the already-encrypted password value as returned by `PASSWORD()`.

`OLD_PASSWORD()` should only be used if your MariaDB/MySQL clients are very old (< 4.0.0).

With no `FOR` clause, this statement sets the password for the current user. Any client that has connected to the server using a non-anonymous account can change the password for that account.

With a `FOR` clause, this statement sets the password for a specific account on the current server host. Only clients that have the `UPDATE` privilege for the `mysql` database can do this. The `user` value should be given in `user_name@host_name` format, where `user_name` and `host_name` are exactly as they are listed in the `User` and `Host` columns of the [mysql.user](#) table (or view in MariaDB-10.4 onwards) entry.

The argument to `PASSWORD()` and the password given to MariaDB clients can be of arbitrary length.

Authentication Plugin Support

MariaDB starting with 10.4

In MariaDB 10.4 and later, `SET PASSWORD` (with or without `PASSWORD()`) works for accounts authenticated via any [authentication plugin](#) that supports passwords stored in the [mysql.global_priv](#) table.

The `ed25519`, `mysql_native_password`, and `mysql_old_password` authentication plugins store passwords in the [mysql.global_priv](#) table.

If you run `SET PASSWORD` on an account that authenticates with one of these authentication plugins that stores passwords in the `mysql.global_priv` table, then the `PASSWORD()` function is evaluated by the specific authentication plugin used by the account. The authentication plugin hashes the password with a method that is compatible with that specific authentication plugin.

The `unix_socket`, `named_pipe`, `gssapi`, and `pam` authentication plugins do **not** store passwords in the `mysql.global_priv` table. These authentication plugins rely on other methods to authenticate the user.

If you attempt to run `SET PASSWORD` on an account that authenticates with one of these authentication plugins that doesn't store a password in the `mysql.global_priv` table, then MariaDB Server will raise a warning like the following:

```
SET PASSWORD is ignored for users authenticating via unix_socket plugin
```

See [Authentication from MariaDB 10.4](#) for an overview of authentication changes in [MariaDB 10.4](#).

MariaDB until 10.3

In [MariaDB 10.3](#) and before, `SET PASSWORD` (with or without `PASSWORD()`) only works for accounts authenticated via `mysql_native_password` or `mysql_old_password` authentication plugins

Passwordless User Accounts

User accounts do not always require passwords to login.

The `unix_socket`, `named_pipe` and `gssapi` authentication plugins do not require a password to authenticate the user.

The `pam` authentication plugin may or may not require a password to authenticate the user, depending on the specific configuration.

The `mysql_native_password` and `mysql_old_password` authentication plugins require passwords for authentication, but the password can be blank. In that case, no password is required.

If you provide a password while attempting to log into the server as an account that doesn't require a password, then MariaDB server will simply ignore the password.

MariaDB starting with 10.4

In [MariaDB 10.4](#) and later, a user account can be defined to use multiple authentication plugins in a specific order of preference. This specific scenario may be more noticeable in these versions, since an account could be associated with some authentication plugins that require a password, and some that do not.

Example

For example, if you had an entry with User and Host column values of 'bob' and '%.loc.gov', you would write the statement like this:

```
SET PASSWORD FOR 'bob'@'%.loc.gov' = PASSWORD('newpass');
```

If you want to delete a password for a user, you would do:

```
SET PASSWORD FOR 'bob'@localhost = PASSWORD('');
```

See Also

- [Password Validation Plugins](#) - permits the setting of basic criteria for passwords
- [ALTER USER](#)

H3Dr1.1.8 CREATE ROLE

Syntax

```
CREATE [OR REPLACE] ROLE [IF NOT EXISTS] role  
[WITH ADMIN  
{CURRENT_USER | CURRENT_ROLE | user | role}]
```

Contents

1. [Syntax](#)
2. [Description](#)
 1. [WITH ADMIN](#)
 2. [OR REPLACE](#)
 3. [IF NOT EXISTS](#)
3. [Examples](#)
4. [See Also](#)

Description

The `CREATE ROLE` statement creates one or more MariaDB [roles](#). To use it, you must have the global `CREATE USER` privilege or the `INSERT` privilege for the `mysql` database. For each account, `CREATE ROLE` creates a new row in the [mysql.user](#) table that has no privileges, and with the corresponding `is_role` field set to `y`. It also creates a record in the [mysql.roles_mapping](#) table.

If any of the specified roles already exist, `ERROR 1396 (HY000)` results. If an error occurs, `CREATE ROLE` will still create the roles that do not result in an error. The maximum length for a role is 128 characters. Role names can be quoted, as explained in the [Identifier names](#) page. Only one error is produced for all roles which have not been created:

```
ERROR 1396 (HY000): Operation CREATE ROLE failed for 'a','b','c'
```

Failed `CREATE` or `DROP` operations, for both users and roles, produce the same error code.

`PUBLIC` and `NONE` are reserved, and cannot be used as role names. `NONE` is used to [unset a role](#) and `PUBLIC` has a special use in other systems, such as Oracle, so is reserved for compatibility purposes.

For valid identifiers to use as role names, see [Identifier Names](#).

WITH ADMIN

The optional `WITH ADMIN` clause determines whether the current user, the current role or another user or role has use of the newly created role. If the clause is omitted, `WITH ADMIN CURRENT_USER` is treated as the default, which means that the current user will be able to `GRANT` this role to users.

OR REPLACE

If the optional `OR REPLACE` clause is used, it acts as a shortcut for:

```
DROP ROLE IF EXISTS name;  
CREATE ROLE name ...;
```

IF NOT EXISTS

When the `IF NOT EXISTS` clause is used, MariaDB will return a warning instead of an error if the specified role already exists. Cannot be used together with the `OR REPLACE` clause.

Examples

```
CREATE ROLE journalist;  
  
CREATE ROLE developer WITH ADMIN lorinda@localhost;
```

Granting the role to another user. Only user `lorinda@localhost` has permission to grant the developer role:

```
SELECT USER();  
+-----+  
| USER() |  
+-----+  
| henning@localhost |  
+-----+  
...  
GRANT developer TO ian@localhost;  
Access denied for user 'henning'@'localhost'  
  
SELECT USER();  
+-----+  
| USER() |  
+-----+  
| lorinda@localhost |  
+-----+  
  
GRANT m_role TO ian@localhost;
```

The `OR REPLACE` and `IF NOT EXISTS` clauses. The `journalist` role already exists:

```
CREATE ROLE journalist;  
ERROR 1396 (HY000): Operation CREATE ROLE failed for 'journalist'  
  
CREATE OR REPLACE ROLE journalist;  
Query OK, 0 rows affected (0.00 sec)  
  
CREATE ROLE IF NOT EXISTS journalist;  
Query OK, 0 rows affected, 1 warning (0.00 sec)
```

```
SHOW WARNINGS;  
+-----+-----+  
| Level | Code | Message |  
+-----+-----+  
| Note | 1975 | Can't create role 'journalist'; it already exists |  
+-----+-----+
```

See Also

- [Identifier Names](#)
- [Roles Overview](#)
- [DROP ROLE](#)

H3Dr1.1.9 DROP ROLE

Syntax

```
DROP ROLE [IF EXISTS] role_name [,role_name ...]
```

Contents

1. [Syntax](#)
2. [Description](#)
 1. [IF EXISTS](#)
3. [Examples](#)
4. [See Also](#)

Description

The `DROP ROLE` statement removes one or more MariaDB [roles](#). To use this statement, you must have the global `CREATE USER` privilege or the `DELETE` privilege for the `mysql` database.

`DROP ROLE` does not disable roles for connections which selected them with `SET ROLE`. If a role has previously been set as a `default role`, `DROP ROLE` does not remove the record of the default role from the `mysql.user` table. If the role is subsequently recreated and granted, it will again be the user's default. Use `SET DEFAULT ROLE NONE` to explicitly remove this.

If any of the specified user accounts do not exist, `ERROR 1396 (HY000)` results. If an error occurs, `DROP ROLE` will still drop the roles that do not result in an error. Only one error is produced for all roles which have not been dropped:

```
ERROR 1396 (HY000): Operation DROP ROLE failed for 'a', 'b', 'c'
```

Failed `CREATE` or `DROP` operations, for both users and roles, produce the same error code.

IF EXISTS

If the `IF EXISTS` clause is used, MariaDB will return a warning instead of an error if the role does not exist.

Examples

```
DROP ROLE journalist;
```

The same thing using the optional `IF EXISTS` clause:

```
DROP ROLE journalist;
ERROR 1396 (HY000): Operation DROP ROLE failed for 'journalist'
```

```
DROP ROLE IF EXISTS journalist;
Query OK, 0 rows affected, 1 warning (0.00 sec)
```

```
Note (Code 1975): Can't drop role 'journalist'; it doesn't exist
```

See Also

- [Roles Overview](#)
- [CREATE ROLE](#)

H3Dr1.1.10 SET ROLE

Syntax

```
SET ROLE { role | NONE }
```

Contents

1. [Syntax](#)
2. [Description](#)
3. [Example](#)

Description

The `SET ROLE` statement enables a [role](#), along with all of its associated permissions, for the current session. To unset a role, use `NONE`.

If a role that doesn't exist, or to which the user has not been assigned, is specified, an `ERROR 1959 (OP000)`: Invalid role specification error occurs.

An automatic `SET ROLE` is implicitly performed when a user connects if that user has been assigned a default role. See [SET DEFAULT ROLE](#).

Example

```
SELECT CURRENT_ROLE;
+-----+
| CURRENT_ROLE |
+-----+
| NULL         |
+-----+

SET ROLE staff;

SELECT CURRENT_ROLE;
+-----+
| CURRENT_ROLE |
+-----+
| staff        |
+-----+

SET ROLE NONE;

SELECT CURRENT_ROLE();
+-----+
| CURRENT_ROLE() |
+-----+
| NULL          |
+-----+
```

H3Dr1.1.11 SET DEFAULT ROLE

Contents

1. [Syntax](#)
2. [Description](#)
3. [Examples](#)

Syntax

```
SET DEFAULT ROLE { role | NONE } [ FOR user@host ]
```

Description

The `SET DEFAULT ROLE` statement sets a [default role](#) for a specified (or current) user. A default role is automatically enabled when a user connects (an implicit `SET ROLE` statement is executed immediately after a connection is established).

To be able to set a role as a default, the role must already have been granted to that user, and one needs

the privileges to enable this role (if you cannot do `SET ROLE X`, you won't be able to do `SET DEFAULT ROLE X`). To set a default role for another user one needs to have write access to the `mysql` database.

To remove a user's default role, use `SET DEFAULT ROLE NONE [FOR user@host]`. The record of the default role is not removed if the role is [dropped](#) or [revoked](#), so if the role is subsequently re-created or granted, it will again be the user's default role.

The default role is stored in the `default_role` column in the [mysql.user](#) table/view, as well as in the [Information Schema APPLICABLE_ROLES table](#), so these can be viewed to see which role has been assigned to a user as the default.

Examples

Setting a default role for the current user:

```
SET DEFAULT ROLE journalist;
```

Removing a default role from the current user:

```
SET DEFAULT ROLE NONE;
```

Setting a default role for another user. The role has to have been granted to the user before it can be set as default:

```
CREATE ROLE journalist;
CREATE USER taniel;

SET DEFAULT ROLE journalist FOR taniel;
ERROR 1959 (OP000): Invalid role specification `journalist'

GRANT journalist TO taniel;
SET DEFAULT ROLE journalist FOR taniel;
```

Viewing `mysql.user`:

```
select * from mysql.user where user='taniel'\G
*****
      1. row *****

      Host: %
      User: taniel
      ...
      is_role: N
      default_role: journalist
      ...
```

Removing a default role for another user

```
SET DEFAULT ROLE NONE FOR taniel;
```

H3Dr1.1.12 SHOW GRANTS

Contents

1. [Syntax](#)
2. [Description](#)
 1. [Users](#)
 2. [Roles](#)
 1. [Example](#)
3. [See Also](#)

Syntax

```
SHOW GRANTS [FOR user|role]
```

Description

The `SHOW GRANTS` statement lists privileges granted to a particular user or role.

Users

The statement lists the `GRANT` statement or statements that must be issued to duplicate the privileges that are granted to a MariaDB user account. The account is named using the same format as for the `GRANT` statement; for example, '`'jeffrey'@'localhost'`'. If you specify only the user name part of the account name, a host name part of '`%`' is used. For additional information about specifying account names, see [GRANT](#).

```
SHOW GRANTS FOR 'root'@'localhost';
+-----+
| Grants for root@localhost |
+-----+
| GRANT ALL PRIVILEGES ON *.* TO 'root'@'localhost' WITH GRANT OPTION |
+-----+
```

To list the privileges granted to the account that you are using to connect to the server, you can use any of the following statements:

```
SHOW GRANTS;
SHOW GRANTS FOR CURRENT_USER;
SHOW GRANTS FOR CURRENT_USER();
```

If `SHOW GRANTS FOR CURRENT_USER` (or any of the equivalent syntaxes) is used in `DEFINER` context (such as within a stored procedure that is defined with `SQL SECURITY DEFINER`), the grants displayed are those of the definer and not the invoker.

Note that the `DELETE HISTORY` privilege, introduced in [MariaDB 10.3.4](#), was displayed as `DELETE VERSIONING ROWS` when running `SHOW GRANTS` until [MariaDB 10.3.15](#) ([MDEV-17655](#)).

Roles

`SHOW GRANTS` can also be used to view the privileges granted to a [role](#).

Example

```
SHOW GRANTS FOR journalist;
+-----+
| Grants for journalist |
+-----+
| GRANT USAGE ON *.* TO 'journalist' |
| GRANT DELETE ON `test`.* TO 'journalist' |
+-----+
```

See Also

- [Authentication from MariaDB 10.4](#)
- [SHOW CREATE USER](#) shows how the user was created.
- [SHOW PRIVILEGES](#) shows the privileges supported by MariaDB.
- [Roles](#)

H3Dr1.1.13 SHOW CREATE USER

Syntax

```
SHOW CREATE USER user_name
```

Description

Shows the [CREATE USER](#) statement that created the given user. The statement requires the [SELECT](#) privilege for the [mysql](#) database, except for the current user.

Examples

```
CREATE USER foo4@test require cipher 'text'
  issuer 'foo_issuer' subject 'foo_subject';

SHOW CREATE USER foo4@test\G
***** 1. row *****
CREATE USER 'foo4'@'test'
  REQUIRE ISSUER 'foo_issuer'
  SUBJECT 'foo_subject'
  CIPHER 'text'
```

User Password Expiry [↗](#):

```
CREATE USER 'monty'@'localhost' PASSWORD EXPIRE INTERVAL 120 DAY;

SHOW CREATE USER 'monty'@'localhost';
+-----+
| CREATE USER for monty@localhost          |
+-----+
| CREATE USER 'monty'@'localhost' PASSWORD EXPIRE INTERVAL 120 DAY |
+-----+
```

See Also

- [CREATE USER](#)
- [ALTER USER](#)
- [SHOW GRANTS](#) shows the [GRANTS/PRIVILEGES](#) for a user.
- [SHOW PRIVILEGES](#) shows the privileges supported by MariaDB.

H3Dr1.2 Administrative SQL Statements

SQL statements for administering MariaDB.



Table Statements

[Documentation on creating, altering, analyzing and maintaining tables.](#)



ANALYZE and EXPLAIN Statements

[Articles on the ANALYZE and EXPLAIN statements](#)



BACKUP Commands

[Commands used by backup tools.](#)



FLUSH Commands

[Commands to flush or reset various caches in MariaDB.](#)



Replication Commands

List of replication-related commands.



Plugin SQL Statements

List of SQL statements related to plugins.



SET Commands

The SET commands



SHOW

Articles on the various SHOW commands.



System Tables



BINLOG

Generated by mysqlbinlog [🔗](#)



PURGE BINARY LOGS

PURGE BINARY LOGS removes all binary logs from the server, prior to the provided date or log file. [🔗](#)



CACHE INDEX

Caches MyISAM or Aria indexes [🔗](#)



DESCRIBE

Information about columns in a table. [🔗](#)



EXECUTE Statement

Executes a previously PREPARED statement [🔗](#)



HELP Command

The HELP command will retrieve syntax and help within the mysql client. [🔗](#)



KILL [CONNECTION | QUERY]

Kill connection by query or thread id. [🔗](#)



LOAD INDEX

Loads one or more indexes from one or more MyISAM/Aria tables into a key buffer. [🔗](#)



RESET

Overall description of the different RESET commands [🔗](#)



SHUTDOWN

Shuts down the server. [🔗](#)



USE

Set the current default database. [🔗](#)

There are 1 related questions [🔗](#).

H3Dr1.2.1 Table Statements

Articles about creating, modifying, and maintaining tables in MariaDB.



ALTER

The various ALTER statements in MariaDB.



ANALYZE TABLE

Store key distributions for a table.



CHECK TABLE

Check table for errors.



CHECK VIEW

Check whether the view algorithm is correct.



CHECKSUM TABLE

Report a table checksum.



CREATE TABLE

Creates a new table.



DELETE

Delete rows from one or more tables.



DROP TABLE

Removes definition and data from one or more tables.



Installing System Tables (`mysql_install_db`)

Using `mysql_install_db` to create the system tables in the 'mysql' database directory.



mysqlcheck

Tool for checking, repairing, analyzing and optimizing tables.



mysql_upgrade

Update to the latest version. ↗



OPTIMIZE TABLE

Reclaim unused space and defragment data.



RENAME TABLE

Change a table's name.



REPAIR TABLE

Rapairs a table, if the storage engine supports this statement.



REPAIR VIEW

Fix view if the algorithms are swapped.



REPLACE

Equivalent to `DELETE + INSERT`, or just an `INSERT` if no rows are returned.



SHOW COLUMNS

Column information.



SHOW CREATE TABLE

Shows the `CREATE TABLE` statement that created the table.



SHOW INDEX

Information about table indexes.



TRUNCATE TABLE

DROP and re-CREATE a table.



UPDATE

Modify rows in one or more tables.



Obsolete Table Commands

Table commands that have been removed from MariaDB.



IGNORE

SUPPRESS errors while trying to violate a UNIQUE constraint.



System-Versioned Tables

System-versioned tables record the history of all changes to table data.

There are 1 related questions [↗](#).

H3Dr1.2.1.1 ALTER

This category is for documentation on the various ALTER statements.



ALTER TABLE

Modify a table's definition.



ALTER DATABASE

Change the overall characteristics of a database.



ALTER EVENT

Change an existing event.



ALTER FUNCTION

Change the characteristics of a stored function.



ALTER LOGFILE GROUP

Only useful with MySQL Cluster, and has no effect in MariaDB.



ALTER PROCEDURE

Change stored procedure characteristics.



ALTER SEQUENCE

Change options for a SEQUENCE.



ALTER SERVER

Updates mysql.servers table.



ALTER TABLESPACE

ALTER TABLESPACE is not available in MariaDB.



ALTER USER

Modify an existing MariaDB account.



ALTER VIEW

Change a view definition.

There are 1 related questions [↗](#).

H3Dr1.2.1.1 ALTER TABLE

Syntax

```
ALTER [ONLINE] [IGNORE] TABLE [IF EXISTS] tbl_name
    [WAIT n | NOWAIT]
    alter_specification [, alter_specification] ...

alter_specification:
    table_option ...
    | ADD [COLUMN] [IF NOT EXISTS] col_name column_definition
        [FIRST | AFTER col_name ]
    | ADD [COLUMN] [IF NOT EXISTS] (col_name column_definition,...)
    | ADD {INDEX|KEY} [IF NOT EXISTS] [index_name]
        [index_type] (index_col_name,...) [index_option] ...
    | ADD [CONSTRAINT [symbol]] PRIMARY KEY
        [index_type] (index_col_name,...) [index_option] ...
    | ADD [CONSTRAINT [symbol]]
        UNIQUE [INDEX|KEY] [index_name]
        [index_type] (index_col_name,...) [index_option] ...
    | ADD FULLTEXT [INDEX|KEY] [index_name]
        (index_col_name,...) [index_option] ...
    | ADD SPATIAL [INDEX|KEY] [index_name]
        (index_col_name,...) [index_option] ...
    | ADD [CONSTRAINT [symbol]]
        FOREIGN KEY [IF NOT EXISTS] [index_name] (index_col_name,...)
        reference_definition
    | ADD PERIOD FOR SYSTEM_TIME (start_column_name, end_column_name)
    | ALTER [COLUMN] col_name SET DEFAULT literal | (expression)
    | ALTER [COLUMN] col_name DROP DEFAULT
    | ALTER {INDEX|KEY} index_name [NOT] INVISIBLE
    | CHANGE [COLUMN] [IF EXISTS] old_col_name new_col_name column_definition
        [FIRST|AFTER col_name]
    | MODIFY [COLUMN] [IF EXISTS] col_name column_definition
        [FIRST | AFTER col_name]
    | DROP [COLUMN] [IF EXISTS] col_name [RESTRICT|CASCADE]
    | DROP PRIMARY KEY
    | DROP {INDEX|KEY} [IF EXISTS] index_name
    | DROP FOREIGN KEY [IF EXISTS] fk_symbol
    | DROP CONSTRAINT [IF EXISTS] constraint_name
    | DISABLE KEYS
    | ENABLE KEYS
    | RENAME [TO] new_tbl_name
    | ORDER BY col_name [, col_name] ...
    | RENAME COLUMN old_col_name TO new_col_name
    | RENAME {INDEX|KEY} old_index_name TO new_index_name
    | CONVERT TO CHARACTER SET charset_name [COLLATE collation_name]
    | [DEFAULT] CHARACTER SET [=] charset_name
    | [DEFAULT] COLLATE [=] collation_name
    | DISCARD TABLESPACE
    | IMPORT TABLESPACE
    | ALGORITHM [=] {DEFAULT|INPLACE|COPY|NOCOPY|INSTANT}
    | LOCK [=] {DEFAULT|NONE|SHARED|EXCLUSIVE}
    | FORCE
    | partition_options
    | ADD PARTITION [IF NOT EXISTS] (partition_definition)
    | DROP PARTITION [IF EXISTS] partition_names
    | COALESCE PARTITION number
    | REORGANIZE PARTITION [partition_names INTO (partition_definitions)]
    | ANALYZE PARTITION partition_names
    | CHECK PARTITION partition_names
    | OPTIMIZE PARTITION partition_names
```

```
| REBUILD PARTITION partition_names
| REPAIR PARTITION partition_names
| EXCHANGE PARTITION partition_name WITH TABLE tbl_name
| REMOVE PARTITIONING
| ADD SYSTEM VERSIONING
| DROP SYSTEM VERSIONING

index_col_name:
    col_name [(length)] [ASC | DESC]

index_type:
    USING {BTREE | HASH | RTREE}

index_option:
    [ KEY_BLOCK_SIZE [=] value
    | index_type
    | WITH PARSER parser_name
    | COMMENT 'string'
    | CLUSTERING={YES| NO} ]
    [ IGNORED | NOT IGNORED ]

table_options:
    table_option [[,] table_option] ...
```

Contents

1. [Syntax](#)
2. [Description](#)
3. [Privileges](#)
4. [Online DDL](#)
 1. [ALTER ONLINE TABLE](#)
5. [WAIT/NOWAIT](#)
6. [IF EXISTS](#)
7. [Column Definitions](#)
8. [Index Definitions](#)
9. [Character Sets and Collations](#)
10. [Alter Specifications](#)
 1. [Table Options](#)
 2. [ADD COLUMN](#)
 3. [DROP COLUMN](#)
 4. [MODIFY COLUMN](#)
 5. [CHANGE COLUMN](#)
 6. [ALTER COLUMN](#)
 7. [RENAME INDEX/KEY](#)
 8. [RENAME COLUMN](#)
 9. [ADD PRIMARY KEY](#)
 10. [DROP PRIMARY KEY](#)
 11. [ADD FOREIGN KEY](#)
 12. [DROP FOREIGN KEY](#)
 13. [ADD INDEX](#)
 14. [DROP INDEX](#)
 15. [ADD UNIQUE INDEX](#)
 16. [DROP UNIQUE INDEX](#)
 17. [ADD FULLTEXT INDEX](#)
 18. [DROP FULLTEXT INDEX](#)
 19. [ADD SPATIAL INDEX](#)
 20. [DROP SPATIAL INDEX](#)
 21. [ENABLE/ DISABLE KEYS](#)
 22. [RENAME TO](#)
 23. [ADD CONSTRAINT](#)
 24. [DROP CONSTRAINT](#)
 25. [ADD SYSTEM VERSIONING](#)
 26. [DROP SYSTEM VERSIONING](#)
 27. [ADD PERIOD FOR SYSTEM_TIME](#)
 28. [FORCE](#)
 29. [EXCHANGE PARTITION](#)
 30. [DISCARD TABLESPACE](#)
 31. [IMPORT TABLESPACE](#)
 32. [ALGORITHM](#)
 1. [ALGORITHM=DEFAULT](#)
 2. [ALGORITHM=COPY](#)
 3. [ALGORITHM=INPLACE](#)
 4. [ALGORITHM=NOCOPY](#)
 5. [ALGORITHM=INSTANT](#)
 33. [LOCK](#)
11. [Progress Reporting](#)
12. [Aborting ALTER TABLE Operations](#)
13. [Atomic ALTER TABLE](#)
14. [Replication](#)
15. [Examples](#)
16. [See Also](#)

Description

`ALTER TABLE` enables you to change the structure of an existing table. For example, you can add or delete columns, create or destroy indexes, change the type of existing columns, or rename columns or the table itself. You can also change the comment for the table and the storage engine of the table.

If another connection is using the table, a [metadata lock](#) is active, and this statement will wait until the lock is released. This is also true for non-transactional tables.

When adding a `UNIQUE` index on a column (or a set of columns) which have duplicated values, an error will be produced and the statement will be stopped. To suppress the error and force the creation of `UNIQUE` indexes, discarding duplicates, the `IGNORE` option can be specified. This can be useful if a column (or a set of columns) should be `UNIQUE` but it contains duplicate values; however, this technique provides no control on which rows are preserved and which are deleted. Also, note that `IGNORE` is accepted but ignored in `ALTER TABLE ... EXCHANGE PARTITION` statements.

This statement can also be used to rename a table. For details see [RENAME TABLE](#).

When an index is created, the storage engine may use a configurable buffer in the process. Incrementing the buffer speeds up the index creation. [Aria](#) and [MyISAM](#) allocate a buffer whose size is defined by `aria_sort_buffer_size` or `myisam_sort_buffer_size`, also used for [REPAIR TABLE](#). [InnoDB](#) allocates three buffers whose size is defined by `innodb_sort_buffer_size`.

Privileges

Executing the `ALTER TABLE` statement generally requires at least the `ALTER` privilege for the table or the database..

If you are renaming a table, then it also requires the `DROP`, `CREATE` and `INSERT` privileges for the table or the database as well.

Online DDL

Online DDL is supported with the `ALGORITHM` and `LOCK` clauses.

See [InnoDB Online DDL Overview](#) for more information on online DDL with [InnoDB](#).

ALTER ONLINE TABLE

`ALTER ONLINE TABLE` also works for partitioned tables.

Online `ALTER TABLE` is available by executing the following:

```
ALTER ONLINE TABLE ...;
```

This statement has the following semantics:

This statement is equivalent to the following:

```
ALTER TABLE ... LOCK=NONE;
```

See the `LOCK` alter specification for more information.

This statement is equivalent to the following:

```
ALTER TABLE ... ALGORITHM=INPLACE;
```

See the `ALGORITHM` alter specification for more information.

WAIT/NOWAIT

MariaDB starting with [10.3.0](#)

Set the lock wait timeout. See [WAIT](#) and [NOWAIT](#).

IF EXISTS

The `IF EXISTS` and `IF NOT EXISTS` clauses are available for the following:

```
ADD COLUMN      [IF NOT EXISTS]
ADD INDEX       [IF NOT EXISTS]
ADD FOREIGN KEY [IF NOT EXISTS]
ADD PARTITION   [IF NOT EXISTS]
CREATE INDEX    [IF NOT EXISTS]

DROP COLUMN     [IF EXISTS]
DROP INDEX      [IF EXISTS]
DROP FOREIGN KEY [IF EXISTS]
DROP PARTITION  [IF EXISTS]
CHANGE COLUMN   [IF EXISTS]
MODIFY COLUMN   [IF EXISTS]
DROP INDEX      [IF EXISTS]
```

When `IF EXISTS` and `IF NOT EXISTS` are used in clauses, queries will not report errors when the condition is triggered for that clause. A warning with the same message text will be issued and the `ALTER` will move on to the next clause in the statement (or end if finished).

MariaDB starting with [10.5.2](#)

If this directive is used after `ALTER ... TABLE`, one will not get an error if the table doesn't exist.

Column Definitions

See [CREATE TABLE: Column Definitions](#) for information about column definitions.

Index Definitions

See [CREATE TABLE: Index Definitions](#) for information about index definitions.

The [CREATE INDEX](#) and [DROP INDEX](#) statements can also be used to add or remove an index.

Character Sets and Collations

```
CONVERT TO CHARACTER SET charset_name [COLLATE collation_name]
[DEFAULT] CHARACTER SET [=] charset_name
[DEFAULT] COLLATE [=] collation_name
```

See [Setting Character Sets and Collations](#) for details on setting the [character sets and collations](#).

Alter Specifications

Table Options

See [CREATE TABLE: Table Options](#) for information about table options.

ADD COLUMN

```
... ADD COLUMN [IF NOT EXISTS] (col_name column_definition,...)
```

Adds a column to the table. The syntax is the same as in [CREATE TABLE](#). If you are using `IF NOT EXISTS` the column will not be added if it was not there already. This is very useful when doing scripts to modify tables.

The `FIRST` and `AFTER` clauses affect the physical order of columns in the datafile. Use `FIRST` to add a column in the first (leftmost) position, or `AFTER` followed by a column name to add the new column in any other position. Note that, nowadays, the physical position of a column is usually irrelevant.

DROP COLUMN

```
... DROP COLUMN [IF EXISTS] col_name [CASCADE|RESTRICT]
```

Drops the column from the table. If you are using `IF EXISTS` you will not get an error if the column didn't exist. If the column is part of any index, the column will be dropped from them, except if you add a new column with identical name at the same time. The index will be dropped if all columns from the index were dropped. If the column was used in a view or trigger, you will get an error next time the view or trigger is accessed.

MariaDB starting with [10.2.8](#)

Dropping a column that is part of a multi-column `UNIQUE` constraint is not permitted. For example:

```
CREATE TABLE a (
    a int,
    b int,
    primary key (a,b)
);

ALTER TABLE x DROP COLUMN a;
[42000][1072] Key column 'A' doesn't exist in table
```

The reason is that dropping column `a` would result in the new constraint that all values in column `b` be unique. In order to drop the column, an explicit `DROP PRIMARY KEY` and `ADD PRIMARY KEY` would be required. Up until [MariaDB 10.2.7](#), the column was dropped and the additional constraint applied, resulting in the following structure:

```
ALTER TABLE x DROP COLUMN a;
Query OK, 0 rows affected (0.46 sec)

DESC x;
+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+
| b     | int(11) | NO   | PRI | NULL    |       |
+-----+-----+-----+-----+
```

MariaDB starting with [10.4.0](#)

[MariaDB 10.4.0](#) supports instant `DROP COLUMN`. `DROP COLUMN` of an indexed column would imply `DROP INDEX` (and in the case of a non-`UNIQUE` multi-column index, possibly `ADD INDEX`). These will not be allowed with `ALGORITHM=INSTANT`, but unlike before, they can be allowed with `ALGORITHM=NOCOPY`

`RESTRICT` and `CASCADE` are allowed to make porting from other database systems easier. In MariaDB, they do nothing.

MODIFY COLUMN

Allows you to modify the type of a column. The column will be at the same place as the original column and all indexes on the column will be kept. Note that when modifying column, you should specify all attributes for the new column.

```
CREATE TABLE t1 (a INT UNSIGNED AUTO_INCREMENT, PRIMARY KEY((a)));
ALTER TABLE t1 MODIFY a BIGINT UNSIGNED AUTO_INCREMENT;
```

CHANGE COLUMN

Works like `MODIFY COLUMN` except that you can also change the name of the column. The column will be at the same place as the original column and all index on the column will be kept.

```
CREATE TABLE t1 (a INT UNSIGNED AUTO_INCREMENT, PRIMARY KEY(a));
ALTER TABLE t1 CHANGE a b BIGINT UNSIGNED AUTO_INCREMENT;
```

ALTER COLUMN

This lets you change column options.

```
CREATE TABLE t1 (a INT UNSIGNED AUTO_INCREMENT, b varchar(50), PRIMARY KEY(a));
ALTER TABLE t1 ALTER b SET DEFAULT 'hello';
```

RENAME INDEX/KEY

MariaDB starting with [10.5.2](#)

From [MariaDB 10.5.2](#), it is possible to rename an index using the `RENAME INDEX` (or `RENAME KEY`) syntax, for example:

```
ALTER TABLE t1 RENAME INDEX i_old TO i_new;
```

RENAME COLUMN

MariaDB starting with [10.5.2](#)

From [MariaDB 10.5.2](#), it is possible to rename a column using the `RENAME COLUMN` syntax, for example:

```
ALTER TABLE t1 RENAME COLUMN c_old TO c_new;
```

ADD PRIMARY KEY

Add a primary key.

For `PRIMARY KEY` indexes, you can specify a name for the index, but it is silently ignored, and the name of the index is always `PRIMARY`.

See [Getting Started with Indexes: Primary Key](#) for more information.

DROP PRIMARY KEY

Drop a primary key.

For `PRIMARY KEY` indexes, you can specify a name for the index, but it is silently ignored, and the name of the index is always `PRIMARY`.

See [Getting Started with Indexes: Primary Key](#) for more information.

ADD FOREIGN KEY

Add a foreign key.

For `FOREIGN KEY` indexes, a reference definition must be provided.

For `FOREIGN KEY` indexes, you can specify a name for the constraint, using the `CONSTRAINT` keyword. That name will be used in error messages.

First, you have to specify the name of the target (parent) table and a column or a column list which must be indexed and whose values must match to the foreign key's values. The `MATCH` clause is accepted to improve the compatibility with other DBMS's, but has no meaning in MariaDB. The `ON DELETE` and `ON`

`UPDATE` clauses specify what must be done when a `DELETE` (or a `REPLACE`) statement attempts to delete a referenced row from the parent table, and when an `UPDATE` statement attempts to modify the referenced foreign key columns in a parent table row, respectively. The following options are allowed:

- `RESTRICT` : The delete/update operation is not performed. The statement terminates with a 1451 error (SQLSTATE '2300').
- `NO ACTION` : Synonym for `RESTRICT`.
- `CASCADE` : The delete/update operation is performed in both tables.
- `SET NULL` : The update or delete goes ahead in the parent table, and the corresponding foreign key fields in the child table are set to `NULL`. (They must not be defined as `NOT NULL` for this to succeed).
- `SET DEFAULT` : This option is implemented only for the legacy PBXT storage engine, which is disabled by default and no longer maintained. It sets the child table's foreign key fields to their `DEFAULT` values when the referenced parent table key entries are updated or deleted.

If either clause is omitted, the default behavior for the omitted clause is `RESTRICT`.

See [Foreign Keys](#) for more information.

DROP FOREIGN KEY

Drop a foreign key.

See [Foreign Keys](#) for more information.

ADD INDEX

Add a plain index.

Plain indexes are regular indexes that are not unique, and are not acting as a primary key or a foreign key. They are also not the "specialized" `FULLTEXT` or `SPATIAL` indexes.

See [Getting Started with Indexes: Plain Indexes](#) for more information.

DROP INDEX

Drop a plain index.

Plain indexes are regular indexes that are not unique, and are not acting as a primary key or a foreign key. They are also not the "specialized" `FULLTEXT` or `SPATIAL` indexes.

See [Getting Started with Indexes: Plain Indexes](#) for more information.

ADD UNIQUE INDEX

Add a unique index.

The `UNIQUE` keyword means that the index will not accept duplicated values, except for `NULLs`. An error will raise if you try to insert duplicate values in a `UNIQUE` index.

For `UNIQUE` indexes, you can specify a name for the constraint, using the `CONSTRAINT` keyword. That name will be used in error messages.

See [Getting Started with Indexes: Unique Index](#) for more information.

DROP UNIQUE INDEX

Drop a unique index.

The `UNIQUE` keyword means that the index will not accept duplicated values, except for `NULLs`. An error will raise if you try to insert duplicate values in a `UNIQUE` index.

For `UNIQUE` indexes, you can specify a name for the constraint, using the `CONSTRAINT` keyword. That name will be used in error messages.

See [Getting Started with Indexes: Unique Index](#) for more information.

ADD FULLTEXT INDEX

Add a FULLTEXT index.

See [Full-Text Indexes](#) for more information.

DROP FULLTEXT INDEX

Drop a FULLTEXT index.

See [Full-Text Indexes](#) for more information.

ADD SPATIAL INDEX

Add a SPATIAL index.

See [SPATIAL INDEX](#) for more information.

DROP SPATIAL INDEX

Drop a SPATIAL index.

See [SPATIAL INDEX](#) for more information.

ENABLE/ DISABLE KEYS

DISABLE KEYS will disable all non unique keys for the table for storage engines that support this (at least MyISAM and Aria). This can be used to [speed up inserts](#) into empty tables.

ENABLE KEYS will enable all disabled keys.

RENAME TO

Renames the table. See also [RENAME TABLE](#).

ADD CONSTRAINT

Modifies the table adding a [constraint](#) on a particular column or columns.

MariaDB starting with [10.2.1](#)

[MariaDB 10.2.1](#) introduced new ways to define a constraint.

Note: Before [MariaDB 10.2.1](#), constraint expressions were accepted in syntax, but ignored.

```
ALTER TABLE table_name  
ADD CONSTRAINT [constraint_name] CHECK(expression);
```

Before a row is inserted or updated, all constraints are evaluated in the order they are defined. If any constraint fails, then the row will not be updated. One can use most deterministic functions in a constraint, including [UDF's](#).

```
CREATE TABLE account_ledger (  
    id INT PRIMARY KEY AUTO_INCREMENT,  
    transaction_name VARCHAR(100),  
    credit_account VARCHAR(100),  
    credit_amount INT,  
    debit_account VARCHAR(100),  
    debit_amount INT);  
  
ALTER TABLE account_ledger  
ADD CONSTRAINT is_balanced  
    CHECK((debit_amount + credit_amount) = 0);
```

The `constraint_name` is optional. If you don't provide one in the `ALTER TABLE` statement, MariaDB auto-generates a name for you. This is done so that you can remove it later using [DROP CONSTRAINT](#)

clause.

You can disable all constraint expression checks by setting the variable `check_constraint_checks` to `OFF`. You may find this useful when loading a table that violates some constraints that you want to later find and fix in SQL.

To view constraints on a table, query `information_schema.TABLE_CONSTRAINTS`:

```
SELECT CONSTRAINT_NAME, TABLE_NAME, CONSTRAINT_TYPE
FROM information_schema.TABLE_CONSTRAINTS
WHERE TABLE_NAME = 'account_ledger';

+-----+-----+-----+
| CONSTRAINT_NAME | TABLE_NAME      | CONSTRAINT_TYPE |
+-----+-----+-----+
| is_balanced    | account_ledger | CHECK          |
+-----+-----+-----+
```

DROP CONSTRAINT

MariaDB starting with 10.2.22

`DROP CONSTRAINT` for `UNIQUE` and `FOREIGN KEY` constraints was introduced in MariaDB 10.2.22 and MariaDB 10.3.13.

MariaDB starting with 10.2.1

`DROP CONSTRAINT` for `CHECK` constraints was introduced in MariaDB 10.2.1.

Modifies the table, removing the given constraint.

```
ALTER TABLE table_name
DROP CONSTRAINT constraint_name;
```

When you add a constraint to a table, whether through a `CREATE TABLE` or `ALTER TABLE...ADD CONSTRAINT` statement, you can either set a `constraint_name` yourself, or allow MariaDB to auto-generate one for you. To view constraints on a table, query `information_schema.TABLE_CONSTRAINTS`. For instance,

```
CREATE TABLE t (
  a INT,
  b INT,
  c INT,
  CONSTRAINT CHECK(a > b),
  CONSTRAINT check_equals CHECK(a = c));

SELECT CONSTRAINT_NAME, TABLE_NAME, CONSTRAINT_TYPE
FROM information_schema.TABLE_CONSTRAINTS
WHERE TABLE_NAME = 't';

+-----+-----+-----+
| CONSTRAINT_NAME | TABLE_NAME      | CONSTRAINT_TYPE |
+-----+-----+-----+
| check_equals   | t              | CHECK          |
| CONSTRAINT_1   | t              | CHECK          |
+-----+-----+-----+
```

To remove a constraint from the table, issue an `ALTER TABLE...DROP CONSTRAINT` statement. For example,

```
ALTER TABLE t DROP CONSTRAINT is_unique;
```

ADD SYSTEM VERSIONING

MariaDB starting with [10.3.4](#)

[System-versioned tables](#) was added in [MariaDB 10.3.4](#).

Add system versioning.

DROP SYSTEM VERSIONING

MariaDB starting with [10.3.4](#)

[System-versioned tables](#) was added in [MariaDB 10.3.4](#).

Drop system versioning.

ADD PERIOD FOR SYSTEM_TIME

MariaDB starting with [10.3.4](#)

[System-versioned tables](#) was added in [MariaDB 10.3.4](#).

FORCE

`ALTER TABLE ... FORCE` can force MariaDB to re-build the table.

In [MariaDB 5.5](#) and before, this could only be done by setting the `ENGINE` table option to its old value. For example, for an InnoDB table, one could execute the following:

```
ALTER TABLE tab_name ENGINE = InnoDB;
```

The `FORCE` option can be used instead. For example, :

```
ALTER TABLE tab_name FORCE;
```

With InnoDB, the table rebuild will only reclaim unused space (i.e. the space previously used for deleted rows) if the `innodb_file_per_table` system variable is set to `ON`. If the system variable is `OFF`, then the space will not be reclaimed, but it will be-re-used for new data that's later added.

EXCHANGE PARTITION

This is used to exchange the tablespace files between a partition and another table.

See [copying InnoDB's transportable tablespaces](#) for more information.

DISCARD TABLESPACE

This is used to discard an InnoDB table's tablespace.

See [copying InnoDB's transportable tablespaces](#) for more information.

IMPORT TABLESPACE

This is used to import an InnoDB table's tablespace. The tablespace should have been copied from its original server after executing [FLUSH TABLES FOR EXPORT](#).

See [copying InnoDB's transportable tablespaces](#) for more information.

`ALTER TABLE ... IMPORT` only applies to InnoDB tables. Most other popular storage engines, such as Aria and MyISAM, will recognize their data files as soon as they've been placed in the proper directory under the datadir, and no special DDL is required to import them.

ALGORITHM

The `ALTER TABLE` statement supports the `ALGORITHM` clause. This clause is one of the clauses that is used to implement online DDL. `ALTER TABLE` supports several different algorithms. An algorithm can be explicitly chosen for an `ALTER TABLE` operation by setting the `ALGORITHM` clause. The supported values are:

- `ALGORITHM=DEFAULT` - This implies the default behavior for the specific statement, such as if no `ALGORITHM` clause is specified.
- `ALGORITHM=COPY`
- `ALGORITHM=INPLACE`
- `ALGORITHM=NOCOPY` - This was added in [MariaDB 10.3.7](#).
- `ALGORITHM=INSTANT` - This was added in [MariaDB 10.3.7](#).

See [InnoDB Online DDL Overview: ALGORITHM](#) for information on how the `ALGORITHM` clause affects InnoDB.

ALGORITHM=DEFAULT

The default behavior, which occurs if `ALGORITHM=DEFAULT` is specified, or if `ALGORITHM` is not specified at all, usually only makes a copy if the operation doesn't support being done in-place at all. In this case, the most efficient available algorithm will usually be used.

However, in [MariaDB 10.3.6](#) and before, if the value of the `old_alter_table` system variable is set to `ON`, then the default behavior is to perform `ALTER TABLE` operations by making a copy of the table using the old algorithm.

In [MariaDB 10.3.7](#) and later, the `old_alter_table` system variable is deprecated. Instead, the `alter_algorithm` system variable defines the default algorithm for `ALTER TABLE` operations.

ALGORITHM=COPY

`ALGORITHM=COPY` is the name for the original `ALTER TABLE` algorithm from early MariaDB versions.

When `ALGORITHM=COPY` is set, MariaDB essentially does the following operations:

```
-- Create a temporary table with the new definition
CREATE TEMPORARY TABLE tmp_tab (
...
);

-- Copy the data from the original table
INSERT INTO tmp_tab
    SELECT * FROM original_tab;

-- Drop the original table
DROP TABLE original_tab;

-- Rename the temporary table, so that it replaces the original one
RENAME TABLE tmp_tab TO original_tab;
```

This algorithm is very inefficient, but it is generic, so it works for all storage engines.

If `ALGORITHM=COPY` is specified, then the copy algorithm will be used even if it is not necessary. This can result in a lengthy table copy. If multiple `ALTER TABLE` operations are required that each require the table to be rebuilt, then it is best to specify all operations in a single `ALTER TABLE` statement, so that the table is only rebuilt once.

ALGORITHM=INPLACE

`ALGORITHM=COPY` can be incredibly slow, because the whole table has to be copied and rebuilt. `ALGORITHM=INPLACE` was introduced as a way to avoid this by performing operations in-place and avoiding the table copy and rebuild, when possible.

When `ALGORITHM=INPLACE` is set, the underlying storage engine uses optimizations to perform the operation while avoiding the table copy and rebuild. However, `INPLACE` is a bit of a misnomer, since some operations may still require the table to be rebuilt for some storage engines. Regardless, several operations can be performed without a full copy of the table for some storage engines.

A more accurate name would have been `ALGORITHM=ENGINE`, where `ENGINE` refers to an "engine-specific"

algorithm.

If an `ALTER TABLE` operation supports `ALGORITHM=INPLACE`, then it can be performed using optimizations by the underlying storage engine, but it may rebuilt.

See [InnoDB Online DDL Operations with ALGORITHM=INPLACE](#) for more.

ALGORITHM=NOCOPY

`ALGORITHM=NOCOPY` was introduced in [MariaDB 10.3.7](#).

`ALGORITHM=INPLACE` can sometimes be surprisingly slow in instances where it has to rebuild the clustered index, because when the clustered index has to be rebuilt, the whole table has to be rebuilt.

`ALGORITHM=NOCOPY` was introduced as a way to avoid this.

If an `ALTER TABLE` operation supports `ALGORITHM=NOCOPY`, then it can be performed without rebuilding the clustered index.

If `ALGORITHM=NOCOPY` is specified for an `ALTER TABLE` operation that does not support `ALGORITHM=NOCOPY`, then an error will be raised. In this case, raising an error is preferable, if the alternative is for the operation to rebuild the clustered index, and perform unexpectedly slowly.

See [InnoDB Online DDL Operations with ALGORITHM=NOCOPY](#) for more.

ALGORITHM=INSTANT

`ALGORITHM=INSTANT` was introduced in [MariaDB 10.3.7](#).

`ALGORITHM=INPLACE` can sometimes be surprisingly slow in instances where it has to modify data files.

`ALGORITHM=INSTANT` was introduced as a way to avoid this.

If an `ALTER TABLE` operation supports `ALGORITHM=INSTANT`, then it can be performed without modifying any data files.

If `ALGORITHM=INSTANT` is specified for an `ALTER TABLE` operation that does not support `ALGORITHM=INSTANT`, then an error will be raised. In this case, raising an error is preferable, if the alternative is for the operation to modify data files, and perform unexpectedly slowly.

See [InnoDB Online DDL Operations with ALGORITHM=INSTANT](#) for more.

LOCK

The `ALTER TABLE` statement supports the `LOCK` clause. This clause is one of the clauses that is used to implement online DDL. `ALTER TABLE` supports several different locking strategies. A locking strategy can be explicitly chosen for an `ALTER TABLE` operation by setting the `LOCK` clause. The supported values are:

- `DEFAULT` : Acquire the least restrictive lock on the table that is supported for the specific operation. Permit the maximum amount of concurrency that is supported for the specific operation.
- `NONE` : Acquire no lock on the table. Permit **all** concurrent DML. If this locking strategy is not permitted for an operation, then an error is raised.
- `SHARED` : Acquire a read lock on the table. Permit **read-only** concurrent DML. If this locking strategy is not permitted for an operation, then an error is raised.
- `EXCLUSIVE` : Acquire a write lock on the table. Do **not** permit concurrent DML.

Different storage engines support different locking strategies for different operations. If a specific locking strategy is chosen for an `ALTER TABLE` operation, and that table's storage engine does not support that locking strategy for that specific operation, then an error will be raised.

If the `LOCK` clause is not explicitly set, then the operation uses `LOCK=DEFAULT`.

`ALTER ONLINE TABLE` is equivalent to `LOCK=NONE`. Therefore, the `ALTER ONLINE TABLE` statement can be used to ensure that your `ALTER TABLE` operation allows all concurrent DML.

See [InnoDB Online DDL Overview: LOCK](#) for information on how the `LOCK` clause affects InnoDB.

Progress Reporting

MariaDB provides progress reporting for `ALTER TABLE` statement for clients that support the new progress reporting protocol. For example, if you were using the `mysql` client, then the progress report might look

like this::

```
ALTER TABLE test ENGINE=Aria;
Stage: 1 of 2 'copy to tmp table'    46% of stage
```

The progress report is also shown in the output of the `SHOW PROCESSLIST` statement and in the contents of the `information_schema.PROCESSLIST` table.

See [Progress Reporting](#) for more information.

Aborting ALTER TABLE Operations

If an `ALTER TABLE` operation is being performed and the connection is killed, the changes will be rolled back in a controlled manner. The rollback can be a slow operation as the time it takes is relative to how far the operation has progressed.

MariaDB starting with [10.2.13](#)

Aborting `ALTER TABLE ... ALGORITHM=COPY` was made faster by removing excessive undo logging ([MDEV-11415](#)). This significantly shortens the time it takes to abort a running `ALTER TABLE` operation.

Atomic ALTER TABLE

MariaDB starting with [10.6.1](#)

From [MariaDB 10.6](#), `ALTER TABLE` is atomic for most engines, including InnoDB, MyRocks, MyISAM and Aria ([MDEV-25180](#)). This means that if there is a crash (server down or power outage) during an `ALTER TABLE` operation, after recovery, either the old table and associated triggers and status will be intact, or the new table will be active.

In older MariaDB versions one could get leftover '#sql-alter..', '#sql-backup..' or 'table_name.frm' files if the system crashed during the `ALTER TABLE` operation.

See [Atomic DDL](#) for more information.

Replication

MariaDB starting with [10.8.0](#)

Before [MariaDB 10.8.0](#), `ALTER TABLE` got fully executed on the primary first, and only then was it replicated and started executing on replicas. From [MariaDB 10.8.0](#), `ALTER TABLE` gets replicated and starts executing on replicas when it *starts* executing on the primary, not when it *finishes*. This way the replication lag caused by a heavy `ALTER TABLE` can be completely eliminated ([MDEV-11675](#)).

Examples

Adding a new column:

```
ALTER TABLE t1 ADD x INT;
```

Dropping a column:

```
ALTER TABLE t1 DROP x;
```

Modifying the type of a column:

```
ALTER TABLE t1 MODIFY x bigint unsigned;
```

Changing the name and type of a column:

```
ALTER TABLE t1 CHANGE a b bigint unsigned auto_increment;
```

Combining multiple clauses in a single ALTER TABLE statement, separated by commas:

```
ALTER TABLE t1 DROP x, ADD x2 INT, CHANGE y y2 INT;
```

Changing the storage engine and adding a comment:

```
ALTER TABLE t1
  ENGINE = InnoDB
  COMMENT = 'First of three tables containing usage info';
```

Rebuilding the table (the previous example will also rebuild the table if it was already InnoDB):

```
ALTER TABLE t1 FORCE;
```

Dropping an index:

```
ALTER TABLE rooms DROP INDEX u;
```

Adding a unique index:

```
ALTER TABLE rooms ADD UNIQUE INDEX u(room_number);
```

From MariaDB 10.5.3, adding a primary key for an application-time period table with a WITHOUT OVERLAPS constraint:

```
ALTER TABLE rooms ADD PRIMARY KEY(room_number, p) WITHOUT OVERLAPS;
```

See Also

- [CREATE TABLE](#)
- [DROP TABLE](#)
- [Character Sets and Collations](#)
- [SHOW CREATE TABLE](#)
- [Instant ADD COLUMN for InnoDB](#)

H3Dr1.2.1.1.2 ALTER DATABASE

Modifies a database, changing its overall characteristics.

Syntax

```
ALTER {DATABASE | SCHEMA} [db_name]
      alter_specification ...
ALTER {DATABASE | SCHEMA} db_name
      UPGRADE DATA DIRECTORY NAME

alter_specification:
  [DEFAULT] CHARACTER SET [=] charset_name
  | [DEFAULT] COLLATE [=] collation_name
  | COMMENT [=] 'comment'
```

Contents

1. [Syntax](#)
2. [Description](#)
 1. [COMMENT](#)
3. [Examples](#)
4. [See Also](#)

Description

`ALTER DATABASE` enables you to change the overall characteristics of a database. These characteristics are stored in the `db.opt` file in the database directory. To use `ALTER DATABASE`, you need the `ALTER` privilege on the database. `ALTER SCHEMA` is a synonym for `ALTER DATABASE`.

The `CHARACTER SET` clause changes the default database character set. The `COLLATE` clause changes the default database collation. See [Character Sets and Collations](#) for more.

You can see what character sets and collations are available using, respectively, the `SHOW CHARACTER SET` and `SHOW COLLATION` statements.

Changing the default character set/collation of a database does not change the character set/collation of any [stored procedures](#) or [stored functions](#) that were previously created, and relied on the defaults. These need to be dropped and recreated in order to apply the character set/collation changes.

The database name can be omitted from the first syntax, in which case the statement applies to the default database.

The syntax that includes the `UPGRADE DATA DIRECTORY NAME` clause was added in MySQL 5.1.23. It updates the name of the directory associated with the database to use the encoding implemented in MySQL 5.1 for mapping database names to database directory names (see [Identifier to File Name Mapping](#)). This clause is for use under these conditions:

- It is intended when upgrading MySQL to 5.1 or later from older versions.
- It is intended to update a database directory name to the current encoding format if the name contains special characters that need encoding.
- The statement is used by `mysqlcheck` (as invoked by `mysql_upgrade`).

For example, if a database in MySQL 5.0 has a name of `a-b-c`, the name contains instance of the `'-` character. In 5.0, the database directory is also named `a-b-c`, which is not necessarily safe for all file systems. In MySQL 5.1 and up, the same database name is encoded as `a@002db@002dc` to produce a file system-neutral directory name.

When a MySQL installation is upgraded to MySQL 5.1 or later from an older version, the server displays a name such as `a-b-c` (which is in the old format) as `#mysql50#a-b-c`, and you must refer to the name using the `#mysql50#` prefix. Use `UPGRADE DATA DIRECTORY NAME` in this case to explicitly tell the server to re-encode the database directory name to the current encoding format:

```
ALTER DATABASE `#mysql50#a-b-c` UPGRADE DATA DIRECTORY NAME;
```

After executing this statement, you can refer to the database as `a-b-c` without the special `#mysql50#` prefix.

COMMENT

MariaDB starting with [10.5.0](#)

From [MariaDB 10.5.0](#), it is possible to add a comment of a maximum of 1024 bytes. If the comment length exceeds this length, an error/warning code 4144 is thrown. The database comment is also added to the `db.opt` file, as well as to the [information_schema.schemata table](#).

Examples

```
ALTER DATABASE test CHARACTER SET='utf8' COLLATE='utf8_bin';
```

From [MariaDB 10.5.0](#):

```
ALTER DATABASE p COMMENT='Presentations';
```

See Also

- [CREATE DATABASE](#)
- [DROP DATABASE](#)
- [SHOW CREATE DATABASE](#)
- [SHOW DATABASES](#)
- [Character Sets and Collations](#)
- [Information Schema SCHEMATA Table](#)

H3Dr1.2.1.1.3 ALTER EVENT

Modifies one or more characteristics of an existing event.

Syntax

```
ALTER  
  [DEFINER = { user | CURRENT_USER }]  
  EVENT event_name  
  [ON SCHEDULE schedule]  
  [ON COMPLETION [NOT] PRESERVE]  
  [RENAME TO new_event_name]  
  [ENABLE | DISABLE | DISABLE ON SLAVE]  
  [COMMENT 'comment']  
  [DO sql_statement]
```

Contents

1. [Syntax](#)
2. [Description](#)
3. [Examples](#)
4. [See Also](#)

Description

The `ALTER EVENT` statement is used to change one or more of the characteristics of an existing [event](#) without the need to drop and recreate it. The syntax for each of the `DEFINER` , `ON SCHEDULE` , `ON COMPLETION` , `COMMENT` , `ENABLE` / `DISABLE` , and `DO` clauses is exactly the same as when used with [CREATE EVENT](#).

This statement requires the `EVENT` privilege. When a user executes a successful `ALTER EVENT` statement, that user becomes the definer for the affected event.

(In MySQL 5.1.11 and earlier, an event could be altered only by its definer, or by a user having the `SUPER` privilege.)

`ALTER EVENT` works only with an existing event:

```
ALTER EVENT no_such_event ON SCHEDULE EVERY '2:3' DAY_HOUR;  
ERROR 1539 (HY000): Unknown event 'no_such_event'
```

Examples

```
ALTER EVENT myevent  
  ON SCHEDULE AT CURRENT_TIMESTAMP + INTERVAL 2 HOUR  
  DO  
    UPDATE myschema.mytable SET mycol = mycol + 1;
```

See Also

- [Events Overview](#)
- [CREATE EVENT](#)
- [SHOW CREATE EVENT](#)
- [DROP EVENT](#)

H3Dr1.2.1.1.4 ALTER FUNCTION

Syntax

```
ALTER FUNCTION func_name [characteristic ...]

characteristic:
  { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }
  | COMMENT 'string'
```

Contents

1. [Syntax](#)
2. [Description](#)
3. [Example](#)
4. [See Also](#)

Description

This statement can be used to change the characteristics of a stored function. More than one change may be specified in an `ALTER FUNCTION` statement. However, you cannot change the parameters or body of a stored function using this statement; to make such changes, you must drop and re-create the function using [DROP FUNCTION](#) and [CREATE FUNCTION](#).

You must have the `ALTER ROUTINE` privilege for the function. (That privilege is granted automatically to the function creator.) If binary logging is enabled, the `ALTER FUNCTION` statement might also require the `SUPER` privilege, as described in [Binary Logging of Stored Routines](#).

Example

```
ALTER FUNCTION hello SQL SECURITY INVOKER;
```

See Also

- [CREATE FUNCTION](#)
- [SHOW CREATE FUNCTION](#)
- [DROP FUNCTION](#)
- [SHOW FUNCTION STATUS](#)
- [Information Schema ROUTINES Table](#)

H3Dr1.2.1.1.5 ALTER LOGFILE GROUP

Syntax

```
ALTER LOGFILE GROUP logfile_group
  ADD UNDOFILE 'file_name'
  [INITIAL_SIZE [=] size]
  [WAIT]
  ENGINE [=] engine_name
```

The `ALTER LOGFILE GROUP` statement is not supported by MariaDB. It was originally inherited from MySQL NDB Cluster. See [MDEV-19295](#) for more information.

H3Dr1.2.1.1.6 ALTER PROCEDURE

Syntax

```
ALTER PROCEDURE proc_name [characteristic ...]

characteristic:
  { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }
  | COMMENT 'string'
```

Description

This statement can be used to change the characteristics of a [stored procedure](#). More than one change may be specified in an `ALTER PROCEDURE` statement. However, you cannot change the parameters or body of a stored procedure using this statement. To make such changes, you must drop and re-create the procedure using either [CREATE OR REPLACE PROCEDURE](#) (since [MariaDB 10.1.3](#)) or [DROP PROCEDURE](#) and [CREATE PROCEDURE](#) ([MariaDB 10.1.2](#) and before).

You must have the `ALTER ROUTINE` privilege for the procedure. By default, that privilege is granted automatically to the procedure creator. See [Stored Routine Privileges](#).

Example

```
ALTER PROCEDURE simpleproc SQL SECURITY INVOKER;
```

See Also

- [Stored Procedure Overview](#)
- [CREATE PROCEDURE](#)
- [SHOW CREATE PROCEDURE](#)
- [DROP PROCEDURE](#)
- [SHOW CREATE PROCEDURE](#)
- [SHOW PROCEDURE STATUS](#)
- [Stored Routine Privileges](#)
- [Information Schema ROUTINES Table](#)

H3Dr1.2.1.1.7 ALTER SEQUENCE

H3Dr1.2.1.1.8 ALTER SERVER

Syntax

```
ALTER SERVER server_name
    OPTIONS (option [, option] ...)
```

Contents

1. [Syntax](#)
2. [Description](#)
3. [Examples](#)
4. [See Also](#)

Description

Alters the server information for `server_name`, adjusting the specified options as per the [CREATE SERVER](#) command. The corresponding fields in the [mysql.servers table](#) are updated accordingly. This statement requires the [SUPER](#) privilege or, from MariaDB 10.5.2, the [FEDERATED ADMIN](#) privilege.

ALTER SERVER is not written to the [binary log](#), irrespective of the [binary log format](#) being used. From MariaDB 10.1.13, Galera replicates the [CREATE SERVER](#), ALTER SERVER and [DROP SERVER](#) statements.

Examples

```
ALTER SERVER s OPTIONS (USER 'sally');
```

See Also

- [CREATE SERVER](#)
- [DROP SERVER](#)
- [Spider Storage Engine](#)

H3Dr1.2.1.1.9 ALTER TABLESPACE

The `ALTER TABLESPACE` statement is not supported by MariaDB. It was originally inherited from MySQL NDB Cluster. In MySQL 5.7 and later, the statement is also supported for InnoDB. However, MariaDB has chosen not to include that specific feature. See [MDEV-19294](#) for more information.

H3Dr1.2.1.1.10 ALTER USER

H3Dr1.2.1.1.11 ALTER VIEW

Syntax

```
ALTER
    [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
    [DEFINER = { user | CURRENT_USER }]
    [SQL SECURITY { DEFINER | INVOKER }]
    VIEW view_name [(column_list)]
    AS select_statement
    [WITH [CASCDED | LOCAL] CHECK OPTION]
```

Contents

1. [Syntax](#)
2. [Description](#)
3. [Example](#)
4. [See Also](#)

Description

This statement changes the definition of a [view](#), which must exist. The syntax is similar to that for [CREATE VIEW](#) and the effect is the same as for `CREATE OR REPLACE VIEW` if the view exists. This statement requires the `CREATE VIEW` and `DROP` [privileges](#) for the view, and some privilege for each column referred to in the `SELECT` statement. `ALTER VIEW` is allowed only to the definer or users with the `SUPER` privilege.

Example

```
ALTER VIEW v AS SELECT a, a*3 AS a2 FROM t;
```

See Also

- [CREATE VIEW](#)
- [DROP VIEW](#)
- [SHOW CREATE VIEW](#)
- [INFORMATION SCHEMA VIEWS Table](#)

H3Dr1.2.1.2 ANALYZE TABLE

Syntax

```
ANALYZE [NO_WRITE_TO_BINLOG | LOCAL] TABLE tbl_name [,tbl_name ...]
[PERSISTENT FOR [ALL|COLUMNS ([col_name [,col_name ...]])]
[INDEXES ([index_name [,index_name ...]])]]
```

Contents

1. [Syntax](#)
2. [Description](#)
3. [Engine-Independent Statistics](#)
4. [See Also](#)

Description

`ANALYZE TABLE` analyzes and stores the key distribution for a table ([index statistics](#)). This statement works with [MyISAM](#), [Aria](#) and [InnoDB](#) tables. During the analysis, InnoDB will allow reads/writes, and MyISAM/Aria reads/inserts. For MyISAM tables, this statement is equivalent to using `myisamchk --analyze`.

For more information on how the analysis works within InnoDB, see [InnoDB Limitations](#).

MariaDB uses the stored key distribution to decide the order in which tables should be joined when you perform a join on something other than a constant. In addition, key distributions can be used when deciding which indexes to use for a specific table within a query.

This statement requires `SELECT` and `INSERT` [privileges](#) for the table.

By default, `ANALYZE TABLE` statements are written to the [binary log](#) and will be replicated. The `NO_WRITE_TO_BINLOG` keyword (`LOCAL` is an alias) will ensure the statement is not written to the binary log.

From [MariaDB 10.3.19](#), `ANALYZE TABLE` statements are not logged to the binary log if `read_only` is

set. See also [Read-Only Replicas](#).

`ANALYZE TABLE` is also supported for partitioned tables. You can use `ALTER TABLE ... ANALYZE PARTITION` to analyze one or more partitions.

The [Aria](#) storage engine supports [progress reporting](#) for the `ANALYZE TABLE` statement.

Engine-Independent Statistics

`ANALYZE TABLE` supports [engine-independent statistics](#). See [Engine-Independent Table Statistics: Collecting Statistics with the ANALYZE TABLE Statement](#) for more information.

See Also

- [Index Statistics](#)
- [InnoDB Persistent Statistics](#)
- [Progress Reporting](#)
- [Engine-independent Statistics](#)
- [Histogram-based Statistics](#)
- [ANALYZE Statement](#)

H3Dr1.2.1.3 CHECK TABLE

Syntax

```
CHECK TABLE tbl_name [, tbl_name] ... [option] ...
option = {FOR UPGRADE | QUICK | FAST | MEDIUM | EXTENDED | CHANGED}
```

Description

`CHECK TABLE` checks a table or tables for errors. `CHECK TABLE` works for [Archive](#), [Aria](#), [CSV](#), [InnoDB](#), and [MyISAM](#) tables. For Aria and MyISAM tables, the key statistics are updated as well. For CSV, see also [Checking and Repairing CSV Tables](#).

As an alternative, [myisamchk](#) is a commandline tool for checking MyISAM tables when the tables are not being accessed.

For checking [dynamic columns](#) integrity, `COLUMN_CHECK()` can be used.

`CHECK TABLE` can also check views for problems, such as tables that are referenced in the view definition that no longer exist.

`CHECK TABLE` is also supported for partitioned tables. You can use `ALTER TABLE ... CHECK PARTITION` to check one or more partitions.

The meaning of the different options are as follows - note that this can vary a bit between storage engines:

FOR UPGRADE	Do a very quick check if the storage format for the table has changed so that one needs to do a REPAIR. This is only needed when one upgrades between major versions of MariaDB or MySQL. This is usually done by running mysql_upgrade .
FAST	Only check tables that has not been closed properly or are marked as corrupt. Only supported by the MyISAM and Aria engines. For other engines the table is checked normally
CHANGED	Check only tables that has changed since last REPAIR / CHECK. Only supported by the MyISAM and Aria engines. For other engines the table is checked normally.
QUICK	Do a fast check. For MyISAM and Aria engine this means we skip checking the delete link chain which may take some time.

MEDIUM	Scan also the data files. Checks integrity between data and index files with checksums. In most cases this should find all possible errors.
EXTENDED	Does a full check to verify every possible error. For MyISAM and Aria we verify for each row that all its keys exists and points to the row. This may take a long time on big tables!

For most cases running `CHECK TABLE` without options or `MEDIUM` should be good enough.

The [Aria](#) storage engine supports [progress reporting](#) for this statement.

If you want to know if two tables are identical, take a look at [CHECKSUM TABLE](#).

InnoDB

If `CHECK TABLE` finds an error in an InnoDB table, MariaDB might shutdown to prevent the error propagation. In this case, the problem will be reported in the error log. Otherwise the table or an index might be marked as corrupted, to prevent use. This does not happen with some minor problems, like a wrong number of entries in a secondary index. Those problems are reported in the output of `CHECK TABLE`.

Each tablespace contains a header with metadata. This header is not checked by this statement.

During the execution of `CHECK TABLE`, other threads may be blocked.

H3Dr1.2.1.4 CHECK VIEW

Syntax

```
CHECK VIEW view_name
```

Description

The `CHECK VIEW` statement was introduced in [MariaDB 10.0.18](#) to assist with fixing [MDEV-6916](#), an issue introduced in [MariaDB 5.2](#) where the view algorithms were swapped. It checks whether the view algorithm is correct. It is run as part of [mysql_upgrade](#), and should not normally be required in regular use.

See Also

- [REPAIR VIEW](#)

H3Dr1.2.1.5 CHECKSUM TABLE

Syntax

```
CHECKSUM TABLE tbl_name [, tbl_name] ... [ QUICK | EXTENDED ]
```

Contents

1. [Syntax](#)
2. [Description](#)
3. [Differences Between MariaDB and MySQL](#)

Description

`CHECKSUM TABLE` reports a table checksum. This is very useful if you want to know if two tables are the

same (for example on a master and slave).

With `QUICK`, the live table checksum is reported if it is available, or `NULL` otherwise. This is very fast. A live checksum is enabled by specifying the `CHECKSUM=1` table option when you [create the table](#); currently, this is supported only for [Aria](#) and [MyISAM](#) tables.

With `EXTENDED`, the entire table is read row by row and the checksum is calculated. This can be very slow for large tables.

If neither `QUICK` nor `EXTENDED` is specified, MariaDB returns a live checksum if the table storage engine supports it and scans the table otherwise.

`CHECKSUM TABLE` requires the [SELECT privilege](#) for the table.

For a nonexistent table, `CHECKSUM TABLE` returns `NULL` and generates a warning.

The table row format affects the checksum value. If the row format changes, the checksum will change. This means that when a table created with a MariaDB/MySQL version is upgraded to another version, the checksum value will probably change.

Two identical tables should always match to the same checksum value; however, also for non-identical tables there is a very slight chance that they will return the same value as the hashing algorithm is not completely collision-free.

Differences Between MariaDB and MySQL

`CHECKSUM TABLE` may give a different result as MariaDB doesn't ignore `NULL`s in the columns as MySQL 5.1 does (Later MySQL versions should calculate checksums the same way as MariaDB). You can get the 'old style' checksum in MariaDB by starting mysqld with the `--old` option. Note however that the MyISAM and Aria storage engines in MariaDB are using the new checksum internally, so if you are using `--old`, the `CHECKSUM` command will be slower as it needs to calculate the checksum row by row. Starting from MariaDB Server 10.9, `--old` is deprecated and will be removed in a future release. Set `--old-mode` or `OLD_MODE` to `COMPAT_5_1_CHECKSUM` to get 'old style' checksum.

H3Dr1.2.1.6 CREATE TABLE

Syntax

```
CREATE [OR REPLACE] [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
    (create_definition,...) [table_options]... [partition_options]
CREATE [OR REPLACE] [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
    [(create_definition,...)] [table_options]... [partition_options]
    select_statement
CREATE [OR REPLACE] [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
    { LIKE old_table_name | (LIKE old_table_name) }

select_statement:
    [IGNORE | REPLACE] [AS] SELECT ...   (Some legal select statement)
```

Contents

1. [Syntax](#)
2. [Description](#)
3. [Privileges](#)
4. [CREATE OR REPLACE](#)
 1. [Things to be Aware of With CREATE OR REPLACE](#)
5. [CREATE TABLE IF NOT EXISTS](#)
6. [CREATE TEMPORARY TABLE](#)
7. [CREATE TABLE ... LIKE](#)
8. [CREATE TABLE ... SELECT](#)
9. [Column Definitions](#)
 1. [NULL and NOT NULL](#)
 2. [DEFAULT Column Option](#)
 3. [AUTO_INCREMENT Column Option](#)
 4. [ZEROFILL Column Option](#)
 5. [PRIMARY KEY Column Option](#)
 6. [UNIQUE KEY Column Option](#)
 7. [COMMENT Column Option](#)
 8. [REF_SYSTEM_ID](#)
 9. [Generated Columns](#)
10. [COMPRESSED](#)
11. [INVISIBLE](#)
12. [WITH SYSTEM VERSIONING Column Option](#)
13. [WITHOUT SYSTEM VERSIONING Column Option](#)
10. [Index Definitions](#)
 1. [Index Categories](#)
 1. Plain Indexes
 2. PRIMARY KEY
 3. UNIQUE
 4. FOREIGN KEY
 5. FULLTEXT
 6. SPATIAL
 2. [Index Options](#)
 1. KEY_BLOCK_SIZE Index Option
 2. Index Types
 3. WITH PARSER Index Option
 4. COMMENT Index Option
 5. CLUSTERING Index Option
 6. IGNORED / NOT IGNORED
11. [Periods](#)
12. [Constraint Expressions](#)
13. [Table Options](#)
 1. [\[STORAGE\] ENGINE](#)

- 2. [AUTO_INCREMENT](#)
 - 3. [AVG_ROW_LENGTH](#)
 - 4. [\[DEFAULT\] CHARACTER SET/CHARSET](#)
 - 5. [CHECKSUM/TABLE_CHECKSUM](#)
 - 6. [\[DEFAULT\] COLLATE](#)
 - 7. [COMMENT](#)
 - 8. [CONNECTION](#)
 - 9. [DATA DIRECTORY/INDEX DIRECTORY](#)
 - 10. [DELAY_KEY_WRITE](#)
 - 11. [ENCRYPTED](#)
 - 12. [ENCRYPTION_KEY_ID](#)
 - 13. [IETF_QUOTES](#)
 - 14. [INSERT_METHOD](#)
 - 15. [KEY_BLOCK_SIZE](#)
 - 16. [MIN_ROWS/MAX_ROWS](#)
 - 17. [PACK_KEYS](#)
 - 18. [PAGE_CHECKSUM](#)
 - 19. [PAGE_COMPRESSED](#)
 - 20. [PAGE_COMPRESSION_LEVEL](#)
 - 21. [PASSWORD](#)
 - 22. [RAID_TYPE](#)
 - 23. [ROW_FORMAT](#)
 - 1. Supported MyISAM Row Formats
 - 2. Supported Aria Row Formats
 - 3. Supported InnoDB Row Formats
 - 4. Other Storage Engines and ROW_FORMAT
 - 24. [SEQUENCE](#)
 - 25. [STATS_AUTO_RECALC](#)
 - 26. [STATS_PERSISTENT](#)
 - 27. [STATS_SAMPLE_PAGES](#)
 - 28. [TRANSACTIONAL](#)
 - 29. [UNION](#)
 - 30. [WITH SYSTEM VERSIONING](#)
- 14. [Partitions](#)
 - 15. [Sequences](#)
 - 16. [Atomic DDL](#)
 - 17. [Examples](#)
 - 18. [See Also](#)

Description

Use the `CREATE TABLE` statement to create a table with the given name.

In its most basic form, the `CREATE TABLE` statement provides a table name followed by a list of columns, indexes, and constraints. By default, the table is created in the default database. Specify a database with `db_name.tbl_name`. If you quote the table name, you must quote the database name and table name separately as ``db_name`.`tbl_name``. This is particularly useful for `CREATE TABLE ... SELECT`, because it allows to create a table into a database, which contains data from other databases. See [Identifier Qualifiers](#).

If a table with the same name exists, error 1050 results. Use `IF NOT EXISTS` to suppress this error and issue a note instead. Use `SHOW WARNINGS` to see notes.

The `CREATE TABLE` statement automatically commits the current transaction, except when using the `TEMPORARY` keyword.

For valid identifiers to use as table names, see [Identifier Names](#).

Note: if the `default_storage_engine` is set to ColumnStore then it needs setting on all UMs. Otherwise when the tables using the default engine are replicated across UMs they will use the wrong engine. You should therefore not use this option as a session variable with ColumnStore.

[Microsecond precision](#) can be between 0-6. If no precision is specified it is assumed to be 0, for backward compatibility reasons.

Privileges

Executing the `CREATE TABLE` statement requires the [CREATE](#) privilege for the table or the database.

CREATE OR REPLACE

If the `OR REPLACE` clause is used and the table already exists, then instead of returning an error, the server will drop the existing table and replace it with the newly defined table.

This syntax was originally added to make [replication](#) more robust if it has to rollback and repeat statements such as `CREATE ... SELECT` on replicas.

```
CREATE OR REPLACE TABLE table_name (a int);
```

is basically the same as:

```
DROP TABLE IF EXISTS table_name;
CREATE TABLE table_name (a int);
```

with the following exceptions:

- If `table_name` was locked with [LOCK TABLES](#) it will continue to be locked after the statement.
- Temporary tables are only dropped if the `TEMPORARY` keyword was used. (With [DROP TABLE](#), temporary tables are preferred to be dropped before normal tables).

Things to be Aware of With CREATE OR REPLACE

- The table is dropped first (if it existed), after that the `CREATE` is done. Because of this, if the `CREATE` fails, then the table will not exist anymore after the statement. If the table was used with `LOCK TABLES` it will be unlocked.
- One can't use `OR REPLACE` together with `IF EXISTS`.
- Slaves in replication will by default use `CREATE OR REPLACE` when replicating `CREATE` statements that don't use `IF EXISTS`. This can be changed by setting the variable [slave-ddl-exec-mode](#) to `STRICT`.

CREATE TABLE IF NOT EXISTS

If the `IF NOT EXISTS` clause is used, then the table will only be created if a table with the same name does not already exist. If the table already exists, then a warning will be triggered by default.

CREATE TEMPORARY TABLE

Use the `TEMPORARY` keyword to create a temporary table that is only available to the current session. Temporary tables are dropped when the session ends. Temporary table names are specific to the session. They will not conflict with other temporary tables from other sessions even if they share the same name. They will shadow names of non-temporary tables or views, if they are identical. A temporary table can have the same name as a non-temporary table which is located in the same database. In that case, their name will reference the temporary table when used in SQL statements. You must have the [CREATE TEMPORARY TABLES](#) privilege on the database to create temporary tables. If no storage engine is specified, the [default_tmp_storage_engine](#) setting will determine the engine.

[ROCKSDB](#) temporary tables cannot be created by setting the [default_tmp_storage_engine](#) system variable, or using `CREATE TEMPORARY TABLE LIKE`. Before [MariaDB 10.7](#), they could be specified, but would silently fail, and a MyISAM table would be created instead. From [MariaDB 10.7](#) an error is returned. Explicitly creating a temporary table with `ENGINE=ROCKSDB` has never been permitted.

CREATE TABLE ... LIKE

Use the `LIKE` clause instead of a full table definition to create a table with the same definition as another table, including columns, indexes, and table options. Foreign key definitions, as well as any DATA DIRECTORY or INDEX DIRECTORY table options specified on the original table, will not be created.

CREATE TABLE ... SELECT

You can create a table containing data from other tables using the `CREATE ... SELECT` statement. Columns will be created in the table for each field returned by the `SELECT` query.

You can also define some columns normally and add other columns from a `SELECT`. You can also create columns in the normal way and assign them some values using the query, this is done to force a certain type or other field characteristics. The columns that are not named in the query will be placed before the others. For example:

```
CREATE TABLE test (a INT NOT NULL, b CHAR(10)) ENGINE=MyISAM  
    SELECT 5 AS b, c, d FROM another_table;
```

Remember that the query just returns data. If you want to use the same indexes, or the same columns attributes ([NOT] NULL, DEFAULT, AUTO_INCREMENT) in the new table, you need to specify them manually. Types and sizes are not automatically preserved if no data returned by the `SELECT` requires the full size, and VARCHAR could be converted into CHAR. The `CAST()` function can be used to force the new table to use certain types.

Aliases (AS) are taken into account, and they should always be used when you `SELECT` an expression (function, arithmetical operation, etc).

If an error occurs during the query, the table will not be created at all.

If the new table has a primary key or UNIQUE indexes, you can use the `IGNORE` or `REPLACE` keywords to handle duplicate key errors during the query. `IGNORE` means that the newer values must not be inserted an identical value exists in the index. `REPLACE` means that older values must be overwritten.

If the columns in the new table are more than the rows returned by the query, the columns populated by the query will be placed after other columns. Note that if the strict `SQL_MODE` is on, and the columns that are not names in the query do not have a `DEFAULT` value, an error will raise and no rows will be copied.

`Concurrent inserts` are not used during the execution of a `CREATE ... SELECT`.

If the table already exists, an error similar to the following will be returned:

```
ERROR 1050 (42S01): Table 't' already exists
```

If the `IF NOT EXISTS` clause is used and the table exists, a note will be produced instead of an error.

To insert rows from a query into an existing table, `INSERT ... SELECT` can be used.

Column Definitions

```

create_definition:
{ col_name column_definition | index_definition | period_definition | CHECK (expr) }

column_definition:
data_type [NOT NULL | NULL] [DEFAULT default_value | (expression)]
[ON UPDATE [NOW | CURRENT_TIMESTAMP] [(precision)]]
[AUTO_INCREMENT] [ZEROFILL] [UNIQUE [KEY] | [PRIMARY] KEY]
[INVISIBLE] [{WITH|WITHOUT} SYSTEM VERSIONING]
[COMMENT 'string'] [REF_SYSTEM_ID = value]
[reference_definition]
| data_type [GENERATED ALWAYS]
AS { { ROW {START|END} } | { (expression) [VIRTUAL | PERSISTENT | STORED] } }
[UNIQUE [KEY]] [COMMENT 'string']

constraint_definition:
CONSTRAINT [constraint_name] CHECK (expression)

```

Note: Until MariaDB 10.4, MariaDB accepts the shortcut format with a REFERENCES clause only in ALTER TABLE and CREATE TABLE statements, but that syntax does nothing. For example:

```
CREATE TABLE b(for_key INT REFERENCES a(not_key));
```

MariaDB simply parses it without returning any error or warning, for compatibility with other DBMS's. Before MariaDB 10.2.1 this was also true for CHECK constraints. However, only the syntax described below creates foreign keys.

From MariaDB 10.5, MariaDB will attempt to apply the constraint. See [Foreign Keys examples](#).

Each definition either creates a column in the table or specifies and index or constraint on one or more columns. See [Indexes](#) below for details on creating indexes.

Create a column by specifying a column name and a data type, optionally followed by column options. See [Data Types](#) for a full list of data types allowed in MariaDB.

NULL and NOT NULL

Use the `NONE` or `NOT NONE` options to specify that values in the column may or may not be `NONE`, respectively. By default, values may be `NONE`. See also [NONE Values in MariaDB](#).

DEFAULT Column Option

MariaDB starting with 10.2.1

The `DEFAULT` clause was enhanced in MariaDB 10.2.1. Some enhancements include

- `BLOB` and `TEXT` columns now support `DEFAULT`.
- The `DEFAULT` clause can now be used with an expression or function.

Specify a default value using the `DEFAULT` clause. If you don't specify `DEFAULT` then the following rules apply:

- If the column is not defined with `NOT NULL`, `AUTO_INCREMENT` or `TIMESTAMP`, an explicit `DEFAULT NULL` will be added. Note that in MySQL and in MariaDB before 10.1.6, you may get an explicit `DEFAULT` for primary key parts, if not specified with `NOT NULL`.

The default value will be used if you [INSERT](#) a row without specifying a value for that column, or if you specify `DEFAULT` for that column. Before MariaDB 10.2.1 you couldn't usually provide an expression or function to evaluate at insertion time. You had to provide a constant default value instead. The one exception is that you may use `CURRENT_TIMESTAMP` as the default value for a `TIMESTAMP` column to use the current timestamp at insertion time.

`CURRENT_TIMESTAMP` may also be used as the default value for a `DATETIME`

From MariaDB 10.2.1 you can use most functions in `DEFAULT`. Expressions should have parentheses around them. If you use a non deterministic function in `DEFAULT` then all inserts to the table will be replicated in [row mode](#). You can even refer to earlier columns in the `DEFAULT` expression (excluding `AUTO_INCREMENT` columns):

```
CREATE TABLE t1 (a int DEFAULT (1+1), b int DEFAULT (a+1));
CREATE TABLE t2 (a bigint primary key DEFAULT UUID_SHORT());
```

The `DEFAULT` clause cannot contain any [stored functions](#) or [subqueries](#), and a column used in the clause must already have been defined earlier in the statement.

Since MariaDB 10.2.1, it is possible to assign `BLOB` or `TEXT` columns a `DEFAULT` value. In earlier versions, assigning a default to these columns was not possible.

MariaDB starting with 10.3.3

Starting from 10.3.3 you can also use `DEFAULT (NEXT VALUE FOR sequence)`

AUTO_INCREMENT Column Option

Use [AUTO_INCREMENT](#) to create a column whose value can be set automatically from a simple counter. You can only use `AUTO_INCREMENT` on a column with an integer type. The column must be a key, and there can only be one `AUTO_INCREMENT` column in a table. If you insert a row without specifying a value for that column (or if you specify `0`, `NULL`, or `DEFAULT` as the value), the actual value will be taken from the counter, with each insertion incrementing the counter by one. You can still insert a value explicitly. If you insert a value that is greater than the current counter value, the counter is set based on the new value. An `AUTO_INCREMENT` column is implicitly `NOT NULL`. Use [LAST_INSERT_ID](#) to get the `AUTO_INCREMENT` value most recently used by an [INSERT](#) statement.

ZEROFILL Column Option

If the `ZEROFILL` column option is specified for a column using a [numeric](#) data type, then the column will be set to `UNSIGNED` and the spaces used by default to pad the field are replaced with zeros. `ZEROFILL` is ignored in expressions or as part of a [UNION](#). `ZEROFILL` is a non-standard MySQL and MariaDB enhancement.

PRIMARY KEY Column Option

Use `PRIMARY KEY` to make a column a primary key. A primary key is a special type of a unique key. There can be at most one primary key per table, and it is implicitly `NOT NULL`.

Specifying a column as a unique key creates a unique index on that column. See the [Index Definitions](#) section below for more information.

UNIQUE KEY Column Option

Use `UNIQUE KEY` (or just `UNIQUE`) to specify that all values in the column must be distinct from each other. Unless the column is `NOT NULL`, there may be multiple rows with `NULL` in the column.

Specifying a column as a unique key creates a unique index on that column. See the [Index Definitions](#) section below for more information.

COMMENT Column Option

You can provide a comment for each column using the `COMMENT` clause. The maximum length is 1024 characters. Use the `SHOW FULL COLUMNS` statement to see column comments.

REF_SYSTEM_ID

`REF_SYSTEM_ID` can be used to specify Spatial Reference System IDs for spatial data type columns.

Generated Columns

A generated column is a column in a table that cannot explicitly be set to a specific value in a [DML query](#). Instead, its value is automatically generated based on an expression. This expression might generate the value based on the values of other columns in the table, or it might generate the value by calling [built-in functions](#) or [user-defined functions \(UDFs\)](#).

There are two types of generated columns:

- `PERSISTENT` or `STORED` : This type's value is actually stored in the table.
- `VIRTUAL` : This type's value is not stored at all. Instead, the value is generated dynamically when the table is queried. This type is the default.

Generated columns are also sometimes called computed columns or virtual columns.

For a complete description about generated columns and their limitations, see [Generated \(Virtual and Persistent/Stored\) Columns](#).

COMPRESSED

MariaDB starting with [10.3.3](#)

Certain columns may be compressed. See [Storage-Engine Independent Column Compression](#).

INVISIBLE

MariaDB starting with [10.3.3](#)

Columns may be made invisible, and hidden in certain contexts. See [Invisible Columns](#).

WITH SYSTEM VERSIONING Column Option

MariaDB starting with [10.3.4](#)

Columns may be explicitly marked as included from system versioning. See [System-versioned tables](#) for details.

WITHOUT SYSTEM VERSIONING Column Option

MariaDB starting with [10.3.4](#)

Columns may be explicitly marked as excluded from system versioning. See [System-versioned tables](#) for details.

Index Definitions

```

index_definition:
  {{INDEX|KEY}} [index_name] [index_type] (index_col_name,...) [index_option] ...
  {{{}|}} {FULLTEXT|SPATIAL} [INDEX|KEY] [index_name] (index_col_name,...) [index_option]
...
  {{{}|}} [CONSTRAINT [symbol]] PRIMARY KEY [index_type] (index_col_name,...)
[index_option] ...
  {{{}|}} [CONSTRAINT [symbol]] UNIQUE [INDEX|KEY] [index_name] [index_type]
(index_col_name,...) [index_option] ...
  {{{}|}} [CONSTRAINT [symbol]] FOREIGN KEY [index_name] (index_col_name,...)
reference_definition

index_col_name:
  col_name [(length)] [ASC | DESC]

index_type:
  USING {BTREE | HASH | RTREE}

index_option:
  [ KEY_BLOCK_SIZE [=] value
  {{{}|}} index_type
  {{{}|}} WITH PARSER parser_name
  {{{}|}} COMMENT 'string'
  {{{}|}} CLUSTERING={YES| NO} ]
  [ IGNORED | NOT IGNORED ]

reference_definition:
  REFERENCES tbl_name (index_col_name,...)
    [MATCH FULL | MATCH PARTIAL | MATCH SIMPLE]
    [ON DELETE reference_option]
    [ON UPDATE reference_option]

reference_option:
  RESTRICT | CASCADE | SET NULL | NO ACTION

```

`INDEX` and `KEY` are synonyms.

Index names are optional, if not specified an automatic name will be assigned. Index name are needed to drop indexes and appear in error messages when a constraint is violated.

Index Categories

Plain Indexes

Plain indexes are regular indexes that are not unique, and are not acting as a primary key or a foreign key. They are also not the "specialized" `FULLTEXT` or `SPATIAL` indexes.

See [Getting Started with Indexes: Plain Indexes](#) for more information.

PRIMARY KEY

For `PRIMARY KEY` indexes, you can specify a name for the index, but it is ignored, and the name of the index is always `PRIMARY`. From MariaDB 10.3.18 and MariaDB 10.4.8, a warning is explicitly issued if a name is specified. Before then, the name was silently ignored.

See [Getting Started with Indexes: Primary Key](#) for more information.

UNIQUE

The `UNIQUE` keyword means that the index will not accept duplicated values, except for NULLs. An error will raise if you try to insert duplicate values in a `UNIQUE` index.

For `UNIQUE` indexes, you can specify a name for the constraint, using the `CONSTRAINT` keyword. That name will be used in error messages.

See [Getting Started with Indexes: Unique Index](#) for more information.

FOREIGN KEY

For `FOREIGN KEY` indexes, a reference definition must be provided.

For `FOREIGN KEY` indexes, you can specify a name for the constraint, using the `CONSTRAINT` keyword. That name will be used in error messages.

First, you have to specify the name of the target (parent) table and a column or a column list which must be indexed and whose values must match to the foreign key's values. The `MATCH` clause is accepted to improve the compatibility with other DBMS's, but has no meaning in MariaDB. The `ON DELETE` and `ON UPDATE` clauses specify what must be done when a `DELETE` (or a `REPLACE`) statements attempts to delete a referenced row from the parent table, and when an `UPDATE` statement attempts to modify the referenced foreign key columns in a parent table row, respectively. The following options are allowed:

- `RESTRICT` : The delete/update operation is not performed. The statement terminates with a 1451 error (`SQLSTATE '2300'`).
- `NO ACTION` : Synonym for `RESTRICT`.
- `CASCADE` : The delete/update operation is performed in both tables.
- `SET NULL` : The update or delete goes ahead in the parent table, and the corresponding foreign key fields in the child table are set to `NULL`. (They must not be defined as `NOT NULL` for this to succeed).
- `SET DEFAULT` : This option is currently implemented only for the PBXT storage engine, which is disabled by default and no longer maintained. It sets the child table's foreign key fields to their `DEFAULT` values when the referenced parent table key entries are updated or deleted.

If either clause is omitted, the default behavior for the omitted clause is `RESTRICT`.

See [Foreign Keys](#) for more information.

FULLTEXT

Use the `FULLTEXT` keyword to create full-text indexes.

See [Full-Text Indexes](#) for more information.

SPATIAL

Use the `SPATIAL` keyword to create geometric indexes.

See [SPATIAL INDEX](#) for more information.

Index Options

KEY_BLOCK_SIZE Index Option

The `KEY_BLOCK_SIZE` index option is similar to the [KEY_BLOCK_SIZE](#) table option.

With the [InnoDB](#) storage engine, if you specify a non-zero value for the `KEY_BLOCK_SIZE` table option for the whole table, then the table will implicitly be created with the `ROW_FORMAT` table option set to `COMPRESSED`. However, this does not happen if you just set the `KEY_BLOCK_SIZE` index option for one or more indexes in the table. The [InnoDB](#) storage engine ignores the `KEY_BLOCK_SIZE` index option. However, the `SHOW CREATE TABLE` statement may still report it for the index.

For information about the `KEY_BLOCK_SIZE` index option, see the [KEY_BLOCK_SIZE](#) table option below.

Index Types

Each storage engine supports some or all index types. See [Storage Engine Index Types](#) for details on permitted index types for each storage engine.

Different index types are optimized for different kind of operations:

- `BTREE` is the default type, and normally is the best choice. It is supported by all storage engines. It can be used to compare a column's value with a value using the `=`, `>`, `>=`, `<`, `<=`, `BETWEEN`, and `LIKE` operators. `BTREE` can also be used to find `NULL` values. Searches against an index prefix are possible.
- `HASH` is only supported by the `MEMORY` storage engine. `HASH` indexes can only be used for `=`, `<=`, and `>=` comparisons. It can not be used for the `ORDER BY` clause. Searches against an index prefix are not possible.
- `RTREE` is the default for [SPATIAL](#) indexes, but if the storage engine does not support it `BTREE`

can be used.

Index columns names are listed between parenthesis. After each column, a prefix length can be specified. If no length is specified, the whole column will be indexed. `ASC` and `DESC` can be specified for compatibility with other DBMS's, but have no meaning in MariaDB.

WITH PARSER Index Option

The `WITH PARSER` index option only applies to [FULLTEXT](#) indexes and contains the fulltext parser name. The fulltext parser must be an installed plugin.

COMMENT Index Option

A comment of up to 1024 characters is permitted with the `COMMENT` index option.

The `COMMENT` index option allows you to specify a comment with user-readable text describing what the index is for. This information is not used by the server itself.

CLUSTERING Index Option

The `CLUSTERING` index option is only valid for tables using the [TokuDB](#) storage engine.

IGNORED / NOT IGNORED

MariaDB starting with [10.6.0](#)

From [MariaDB 10.6.0](#), indexes can be specified to be ignored by the optimizer. See [Ignored Indexes](#).

Periods

MariaDB starting with [10.3.4](#)

```
period_definition:  
PERIOD FOR SYSTEM_TIME (start_column_name, end_column_name)
```

MariaDB supports a subset of the standard syntax for periods. At the moment it's only used for creating [System-versioned tables](#). Both columns must be created, must be either of a `TIMESTAMP(6)` or `BIGINT UNSIGNED` type, and be generated as `ROW START` and `ROW END` accordingly. See [System-versioned tables](#) for details.

The table must also have the `WITH SYSTEM VERSIONING` clause.

Constraint Expressions

MariaDB starting with [10.2.1](#)

[MariaDB 10.2.1](#) introduced new ways to define a constraint.

Note: Before [MariaDB 10.2.1](#), constraint expressions were accepted in the syntax but ignored.

[MariaDB 10.2.1](#) introduced two ways to define a constraint:

- `CHECK(expression)` given as part of a column definition.
- `CONSTRAINT [constraint_name] CHECK (expression)`

Before a row is inserted or updated, all constraints are evaluated in the order they are defined. If any constraints fail, then the row will not be updated. One can use most deterministic functions in a constraint, including [UDFs](#).

```
create table t1 (a int check(a>0) ,b int check (b> 0), constraint abc check (a>b));
```

If you use the second format and you don't give a name to the constraint, then the constraint will get a auto generated name. This is done so that you can later delete the constraint with [ALTER TABLE DROP constraint_name](#).

One can disable all constraint expression checks by setting the variable `check_constraint_checks` to `OFF`. This is useful for example when loading a table that violates some constraints that you want to later find and fix in SQL.

See [CONSTRAINT](#) for more information.

Table Options

For each individual table you create (or alter), you can set some table options. The general syntax for setting options is:

`<OPTION_NAME> = <option_value>, [<OPTION_NAME> = <option_value> ...]`

The equal sign is optional.

Some options are supported by the server and can be used for all tables, no matter what storage engine they use; other options can be specified for all storage engines, but have a meaning only for some engines. Also, engines can [extend CREATE TABLE with new options](#).

If the `IGNORE_BAD_TABLE_OPTIONS` [SQL_MODE](#) is enabled, wrong table options generate a warning; otherwise, they generate an error.

```
table_option:  
  [STORAGE] ENGINE [=] engine_name  
  | AUTO_INCREMENT [=] value  
  | AVG_ROW_LENGTH [=] value  
  | [DEFAULT] CHARACTER SET [=] charset\_name  
  | CHECKSUM [=] {0 | 1}  
  | [DEFAULT] COLLATE [=] collation\_name  
  | COMMENT [=] 'string'  
  | CONNECTION [=] 'connect_string'  
  | DATA DIRECTORY [=] 'absolute path to directory'  
  | DELAY_KEY_WRITE [=] {0 | 1}  
  | ENCRYPTED [=] {YES | NO}  
  | ENCRYPTION_KEY_ID [=] value  
  | IETF_QUOTES [=] {YES | NO}  
  | INDEX DIRECTORY [=] 'absolute path to directory'  
  | INSERT_METHOD [=] {NO | FIRST | LAST}  
  | KEY_BLOCK_SIZE [=] value  
  | MAX_ROWS [=] value  
  | MIN_ROWS [=] value  
  | PACK_KEYS [=] {0 | 1 | DEFAULT}  
  | PAGE_CHECKSUM [=] {0 | 1}  
  | PAGE_COMPRESSED [=] {0 | 1}  
  | PAGE_COMPRESSION_LEVEL [=] {0 .. 9}  
  | PASSWORD [=] 'string'  
  | ROW_FORMAT [=] {DEFAULT|DYNAMIC|FIXED|COMPRESSED|REDUNDANT|COMPACT|PAGE}  
  | SEQUENCE [=] {0|1}  
  | STATS_AUTO_RECALC [=] {DEFAULT|0|1}  
  | STATS_PERSISTENT [=] {DEFAULT|0|1}  
  | STATS_SAMPLE_PAGES [=] {DEFAULT|value}  
  | TABLESPACE tablespace_name  
  | TRANSACTIONAL [=] {0 | 1}  
  | UNION [=] (tbl_name[,tbl_name]...)  
  | WITH SYSTEM VERSIONING
```

[STORAGE] ENGINE

`[STORAGE] ENGINE` specifies a [storage engine](#) for the table. If this option is not used, the default storage engine is used instead. That is, the `default_storage_engine` session option value if it is set, or the value specified for the `--default-storage-engine` [mysqld startup option](#), or the default storage engine,

[InnoDB](#). If the specified storage engine is not installed and active, the default value will be used, unless the `NO_ENGINE_SUBSTITUTION` [SQL MODE](#) is set (default). This is only true for `CREATE TABLE`, not for `ALTER TABLE`. For a list of storage engines that are present in your server, issue a [SHOW ENGINES](#).

AUTO_INCREMENT

`AUTO_INCREMENT` specifies the initial value for the [AUTO_INCREMENT](#) primary key. This works for MyISAM, Aria, InnoDB, MEMORY, and ARCHIVE tables. You can change this option with `ALTER TABLE`, but in that case the new value must be higher than the highest value which is present in the `AUTO_INCREMENT` column. If the storage engine does not support this option, you can insert (and then delete) a row having the wanted value - 1 in the `AUTO_INCREMENT` column.

AVG_ROW_LENGTH

`AVG_ROW_LENGTH` is the average rows size. It only applies to tables using [MyISAM](#) and [Aria](#) storage engines that have the `ROW_FORMAT` table option set to `FIXED` format.

MyISAM uses `MAX_ROWS` and `AVG_ROW_LENGTH` to decide the maximum size of a table (default: 256TB, or the maximum file size allowed by the system).

[DEFAULT] CHARACTER SET/CHARSET

`[DEFAULT] CHARACTER SET` (or `[DEFAULT] CHARSET`) is used to set a default character set for the table. This is the character set used for all columns where an explicit character set is not specified. If this option is omitted or `DEFAULT` is specified, database's default character set will be used. See [Setting Character Sets and Collations](#) for details on setting the [character sets](#).

CHECKSUM/TABLE_CHECKSUM

`CHECKSUM` (or `TABLE_CHECKSUM`) can be set to 1 to maintain a live checksum for all table's rows. This makes write operations slower, but `CHECKSUM TABLE` will be very fast. This option is only supported for [MyISAM](#) and [Aria tables](#).

[DEFAULT] COLLATE

`[DEFAULT] COLLATE` is used to set a default collation for the table. This is the collation used for all columns where an explicit character set is not specified. If this option is omitted or `DEFAULT` is specified, database's default option will be used. See [Setting Character Sets and Collations](#) for details on setting the [collations](#).

COMMENT

`COMMENT` is a comment for the table. The maximum length is 2048 characters. Also used to define table parameters when creating a [Spider](#) table.

CONNECTION

`CONNECTION` is used to specify a server name or a connection string for a [Spider](#), [CONNECT](#), [Federated](#) or [FederatedX](#) table.

DATA DIRECTORY/INDEX DIRECTORY

`DATA DIRECTORY` and `INDEX DIRECTORY` are supported for MyISAM and Aria, and `DATA DIRECTORY` is also supported by InnoDB if the `innodb_file_per_table` server system variable is enabled, but only in `CREATE TABLE`, not in `ALTER TABLE`. So, carefully choose a path for InnoDB tables at creation time, because it cannot be changed without dropping and re-creating the table. These options specify the paths for data files and index files, respectively. If these options are omitted, the database's directory will be used to store data files and index files. Note that these table options do not work for [partitioned](#) tables (use the partition options instead), or if the server has been invoked with the `--skip-symbolic-links startup option`. To avoid the overwriting of old files with the same name that could be present in the directories,

you can use the `--keep_files_on_create` option (an error will be issued if files already exist). These options are ignored if the `NO_DIR_IN_CREATE` SQL MODE is enabled (useful for replication slaves). Also note that symbolic links cannot be used for InnoDB tables.

`DATA DIRECTORY` works by creating symlinks from where the table would normally have been (inside the `datadir`) to where the option specifies. For security reasons, to avoid bypassing the privilege system, the server does not permit symlinks inside the datadir. Therefore, `DATA DIRECTORY` cannot be used to specify a location inside the datadir. An attempt to do so will result in an error `1210 (HY000) Incorrect arguments to DATA DIRECTORY`.

DELAY_KEY_WRITE

`DELAY_KEY_WRITE` is supported by MyISAM and Aria, and can be set to 1 to speed up write operations. In that case, when data are modified, the indexes are not updated until the table is closed. Writing the changes to the index file altogether can be much faster. However, note that this option is applied only if the `delay_key_write` server variable is set to 'ON'. If it is 'OFF' the delayed index writes are always disabled, and if it is 'ALL' the delayed index writes are always used, disregarding the value of `DELAY_KEY_WRITE`.

ENCRYPTED

The `ENCRYPTED` table option can be used to manually set the encryption status of an InnoDB table. See [InnoDB Encryption](#) for more information.

Aria does not support the `ENCRYPTED` table option. See [MDEV-18049](#).

See [Data-at-Rest Encryption](#) for more information.

ENCRYPTION_KEY_ID

The `ENCRYPTION_KEY_ID` table option can be used to manually set the encryption key of an InnoDB table. See [InnoDB Encryption](#) for more information.

Aria does not support the `ENCRYPTION_KEY_ID` table option. See [MDEV-18049](#).

See [Data-at-Rest Encryption](#) for more information.

IETF_QUOTES

For the CSV storage engine, the `IETF_QUOTES` option, when set to `YES`, enables IETF-compatible parsing of embedded quote and comma characters. Enabling this option for a table improves compatibility with other tools that use CSV, but is not compatible with MySQL CSV tables, or MariaDB CSV tables created without this option. Disabled by default.

INSERT_METHOD

`INSERT_METHOD` is only used with MERGE tables. This option determines in which underlying table the new rows should be inserted. If you set it to 'NO' (which is the default) no new rows can be added to the table (but you will still be able to perform `INSERT`s directly against the underlying tables). `FIRST` means that the rows are inserted into the first table, and `LAST` means that they are inserted into the last table.

KEY_BLOCK_SIZE

`KEY_BLOCK_SIZE` is used to determine the size of key blocks, in bytes or kilobytes. However, this value is just a hint, and the storage engine could modify or ignore it. If `KEY_BLOCK_SIZE` is set to 0, the storage engine's default value will be used.

With the InnoDB storage engine, if you specify a non-zero value for the `KEY_BLOCK_SIZE` table option for the whole table, then the table will implicitly be created with the `ROW_FORMAT` table option set to `COMPRESSED`.

MIN_ROWS/MAX_ROWS

`MIN_ROWS` and `MAX_ROWS` let the storage engine know how many rows you are planning to store as a minimum and as a maximum. These values will not be used as real limits, but they help the storage engine to optimize the table. `MIN_ROWS` is only used by MEMORY storage engine to decide the minimum memory that is always allocated. `MAX_ROWS` is used to decide the minimum size for indexes.

PACK_KEYS

`PACK_KEYS` can be used to determine whether the indexes will be compressed. Set it to 1 to compress all keys. With a value of 0, compression will not be used. With the `DEFAULT` value, only long strings will be compressed. Uncompressed keys are faster.

PAGE_CHECKSUM

`PAGE_CHECKSUM` is only applicable to [Aria](#) tables, and determines whether indexes and data should use page checksums for extra safety.

PAGE_COMPRESSED

`PAGE_COMPRESSED` is used to enable [InnoDB page compression](#) for [InnoDB](#) tables.

PAGE_COMPRESSION_LEVEL

`PAGE_COMPRESSION_LEVEL` is used to set the compression level for [InnoDB page compression](#) for [InnoDB](#) tables. The table must also have the `PAGE_COMPRESSED` table option set to `1`.

Valid values for `PAGE_COMPRESSION_LEVEL` are 1 (the best speed) through 9 (the best compression), .

PASSWORD

`PASSWORD` is unused.

RAID_TYPE

`RAID_TYPE` is an obsolete option, as the raid support has been disabled since MySQL 5.0.

ROW_FORMAT

The `ROW_FORMAT` table option specifies the row format for the data file. Possible values are engine-dependent.

Supported MyISAM Row Formats

For [MyISAM](#), the supported row formats are:

- `FIXED`
- `DYNAMIC`
- `COMPRESSED`

The `COMPRESSED` row format can only be set by the [myisampack](#) command line tool.

See [MyISAM Storage Formats](#) for more information.

Supported Aria Row Formats

For [Aria](#), the supported row formats are:

- `PAGE`
- `FIXED`
- `DYNAMIC` .

See [Aria Storage Formats](#) for more information.

Supported InnoDB Row Formats

For [InnoDB](#), the supported row formats are:

- COMPACT
- REDUNDANT
- COMPRESSED
- DYNAMIC .

If the `ROW_FORMAT` table option is set to `FIXED` for an InnoDB table, then the server will either return an error or a warning depending on the value of the [innodb_strict_mode](#) system variable. If the [innodb_strict_mode](#) system variable is set to `OFF`, then a warning is issued, and MariaDB will create the table using the default row format for the specific MariaDB server version. If the [innodb_strict_mode](#) system variable is set to `ON`, then an error will be raised.

See [InnoDB Storage Formats](#) for more information.

Other Storage Engines and ROW_FORMAT

Other storage engines do not support the `ROW_FORMAT` table option.

SEQUENCE

MariaDB starting with [10.3](#)

If the table is a [sequence](#), then it will have the `SEQUENCE` set to `1`.

STATS_AUTO_RECALC

`STATS_AUTO_RECALC` indicates whether to automatically recalculate persistent statistics (see `STATS_PERSISTENT`, below) for an InnoDB table. If set to `1`, statistics will be recalculated when more than 10% of the data has changed. When set to `0`, stats will be recalculated only when an [ANALYZE TABLE](#) is run. If set to `DEFAULT`, or left out, the value set by the [innodb_stats_auto_recalc](#) system variable applies. See [InnoDB Persistent Statistics](#).

STATS_PERSISTENT

`STATS_PERSISTENT` indicates whether the InnoDB statistics created by [ANALYZE TABLE](#) will remain on disk or not. It can be set to `1` (on disk), `0` (not on disk, the pre-MariaDB 10 behavior), or `DEFAULT` (the same as leaving out the option), in which case the value set by the [innodb_stats_persistent](#) system variable will apply. Persistent statistics stored on disk allow the statistics to survive server restarts, and provide better query plan stability. See [InnoDB Persistent Statistics](#).

STATS_SAMPLE_PAGES

`STATS_SAMPLE_PAGES` indicates how many pages are used to sample index statistics. If `0` or `DEFAULT`, the default value, the [innodb_stats_sample_pages](#) value is used. See [InnoDB Persistent Statistics](#).

TRANSACTIONAL

`TRANSACTIONAL` is only applicable for Aria tables. In future Aria tables created with this option will be fully transactional, but currently this provides a form of crash protection. See [Aria Storage Engine](#) for more details.

UNION

`UNION` must be specified when you create a MERGE table. This option contains a comma-separated list of MyISAM tables which are accessed by the new table. The list is enclosed between parenthesis.

Example: `UNION = (t1,t2)`

WITH SYSTEM VERSIONING

`WITH SYSTEM VERSIONING` is used for creating [System-versioned tables](#).

Partitions

```
partition_options:
    PARTITION BY
        { [LINEAR] HASH(expr)
        | [LINEAR] KEY(column_list)
        | RANGE(expr)
        | LIST(expr)
        | SYSTEM_TIME [INTERVAL time_quantity time_unit] [LIMIT num] }
    [PARTITIONS num]
    [SUBPARTITION BY
        { [LINEAR] HASH(expr)
        | [LINEAR] KEY(column_list) }
    [SUBPARTITIONS num]
    ]
    [(partition_definition [, partition_definition] ...)]

partition_definition:
    PARTITION partition_name
        [VALUES {LESS THAN {(expr) | MAXVALUE} | IN (value_list)}]
        [[STORAGE] ENGINE [=] engine_name]
        [COMMENT [=] 'comment_text' ]
        [DATA DIRECTORY [=] 'data_dir']
        [INDEX DIRECTORY [=] 'index_dir']
        [MAX_ROWS [=] max_number_of_rows]
        [MIN_ROWS [=] min_number_of_rows]
        [TABLESPACE [=] tablespace_name]
        [NODEGROUP [=] node_group_id]
        [(subpartition_definition [, subpartition_definition] ...)]

subpartition_definition:
    SUBPARTITION logical_name
        [[STORAGE] ENGINE [=] engine_name]
        [COMMENT [=] 'comment_text' ]
        [DATA DIRECTORY [=] 'data_dir']
        [INDEX DIRECTORY [=] 'index_dir']
        [MAX_ROWS [=] max_number_of_rows]
        [MIN_ROWS [=] min_number_of_rows]
        [TABLESPACE [=] tablespace_name]
        [NODEGROUP [=] node_group_id]
```

If the `PARTITION BY` clause is used, the table will be [partitioned](#). A partition method must be explicitly indicated for partitions and subpartitions. Partition methods are:

- `[LINEAR] HASH` creates a hash key which will be used to read and write rows. The partition function can be any valid SQL expression which returns an `INTEGER` number. Thus, it is possible to use the `HASH` method on an integer column, or on functions which accept integer columns as an argument. However, `VALUES LESS THAN` and `VALUES IN` clauses can not be used with `HASH`. An example:

```
CREATE TABLE t1 (a INT, b CHAR(5), c DATETIME)
    PARTITION BY HASH ( YEAR(c) );
```

`[LINEAR] HASH` can be used for subpartitions, too.

- `[LINEAR] KEY` is similar to `HASH`, but the index has an even distribution of data. Also, the expression can only be a column or a list of columns. `VALUES LESS THAN` and `VALUES IN` clauses can not be used with `KEY`.
- `RANGE` partitions the rows using on a range of values, using the `VALUES LESS THAN` operator. `VALUES IN` is not allowed with `RANGE`. The partition function can be any valid SQL expression which returns a single value.
- `LIST` assigns partitions based on a table's column with a restricted set of possible values. It is similar to `RANGE`, but `VALUES IN` must be used for at least 1 columns, and `VALUES LESS THAN` is

disallowed.

- `SYSTEM_TIME` partitioning is used for [System-versioned tables](#) to store historical data separately from current data.

Only `HASH` and `KEY` can be used for subpartitions, and they can be `[LINEAR]`.

It is possible to define up to 1024 partitions and subpartitions.

The number of defined partitions can be optionally specified as `PARTITION count`. This can be done to avoid specifying all partitions individually. But you can also declare each individual partition and, additionally, specify a `PARTITIONS count` clause; in the case, the number of `PARTITION`s must equal `count`.

Also see [Partitioning Types Overview](#).

Sequences

MariaDB starting with [10.3](#)

`CREATE TABLE` can also be used to create a [SEQUENCE](#). See [CREATE SEQUENCE](#) and [Sequence Overview](#).

Atomic DDL

MariaDB starting with [10.6.1](#)

[MariaDB 10.6.1](#) supports [Atomic DDL](#). `CREATE TABLE` is atomic, except for `CREATE OR REPLACE`, which is only crash safe.

Examples

```
create table if not exists test (
    a bigint auto_increment primary key,
    name varchar(128) charset utf8,
    key name (name(32))
) engine=InnoDB default charset latin1;
```

This example shows a couple of things:

- Usage of `IF NOT EXISTS`; If the table already existed, it will not be created. There will not be any error for the client, just a warning.
- How to create a `PRIMARY KEY` that is [automatically generated](#).
- How to specify a table-specific [character set](#) and another for a column.
- How to create an index (`name`) that is only partly indexed (to save space).

The following clauses will work from [MariaDB 10.2.1](#) only.

```
CREATE TABLE t1(
    a int DEFAULT (1+1),
    b int DEFAULT (a+1),
    expires DATETIME DEFAULT(NOW() + INTERVAL 1 YEAR),
    x BLOB DEFAULT USER(),
);
```

See Also

- [Identifier Names](#)
- [ALTER TABLE](#)
- [DROP TABLE](#)
- [Character Sets and Collations](#)
- [SHOW CREATE TABLE](#)
- Storage engines can add their own [attributes for columns, indexes and tables](#).

- Variable [slave-ddl-exec-mode](#).

H3Dr1.2.1.7 DELETE

Contents

1. [Syntax](#)
2. [Description](#)
 1. [PARTITION](#)
 2. [FOR PORTION OF](#)
 3. [RETURNING](#)
 4. [Same Source and Target Table](#)
 5. [DELETE HISTORY](#)
3. [Examples](#)
 1. [Deleting from the Same Source and Target](#)
4. [See Also](#)

Syntax

Single-table syntax:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
      FROM tbl_name [PARTITION (partition_list)]
      [FOR PORTION OF period FROM expr1 TO expr2]
      [WHERE where_condition]
      [ORDER BY ...]
      [LIMIT row_count]
      [RETURNING select_expr
      [, select_expr ...]]
```

Multiple-table syntax:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
      tbl_name[.*] [, tbl_name[.*]] ...
      FROM table_references
      [WHERE where_condition]
```

Or:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
      FROM tbl_name[.*] [, tbl_name[.*]] ...
      USING table_references
      [WHERE where_condition]
```

Trimming history:

```
DELETE HISTORY
      FROM tbl_name [PARTITION (partition_list)]
      [BEFORE SYSTEM_TIME [TIMESTAMP|TRANSACTION] expression]
```

Description

Option	Description
LOW_PRIORITY	Wait until all SELECT's are done before starting the statement. Used with storage engines that uses table locking (MyISAM, Aria etc). See HIGH_PRIORITY and LOW_PRIORITY clauses for details.

QUICK	Signal the storage engine that it should expect that a lot of rows are deleted. The storage engine can do things to speed up the DELETE like ignoring merging of data blocks until all rows are deleted from the block (instead of when a block is half full). This speeds up things at the expense of lost space in data blocks. At least MyISAM and Aria support this feature.
IGNORE	Don't stop the query even if a not-critical error occurs (like data overflow). See How IGNORE works for a full description.

For the single-table syntax, the `DELETE` statement deletes rows from `tbl_name` and returns a count of the number of deleted rows. This count can be obtained by calling the `ROW_COUNT()` function. The `WHERE` clause, if given, specifies the conditions that identify which rows to delete. With no `WHERE` clause, all rows are deleted. If the `ORDER BY` clause is specified, the rows are deleted in the order that is specified. The `LIMIT` clause places a limit on the number of rows that can be deleted.

For the multiple-table syntax, `DELETE` deletes from each `tbl_name` the rows that satisfy the conditions. In this case, `ORDER BY` and `LIMIT` cannot be used. A `DELETE` can also reference tables which are located in different databases; see [Identifier Qualifiers](#) for the syntax.

`where_condition` is an expression that evaluates to true for each row to be deleted. It is specified as described in [SELECT](#).

Currently, you cannot delete from a table and select from the same table in a subquery.

You need the `DELETE` privilege on a table to delete rows from it. You need only the `SELECT` privilege for any columns that are only read, such as those named in the `WHERE` clause. See [GRANT](#).

As stated, a `DELETE` statement with no `WHERE` clause deletes all rows. A faster way to do this, when you do not need to know the number of deleted rows, is to use `TRUNCATE TABLE`. However, within a transaction or if you have a lock on the table, `TRUNCATE TABLE` cannot be used whereas `DELETE` can. See [TRUNCATE TABLE](#), and [LOCK](#).

PARTITION

See [Partition Pruning and Selection](#) for details.

FOR PORTION OF

MariaDB starting with [10.4.3](#)

See [Application Time Periods - Deletion by Portion](#).

RETURNING

It is possible to return a resultset of the deleted rows for a single table to the client by using the syntax

`DELETE ... RETURNING select_expr [, select_expr2 ...]`

Any of SQL expression that can be calculated from a single row fields is allowed. Subqueries are allowed. The AS keyword is allowed, so it is possible to use aliases.

The use of aggregate functions is not allowed. `RETURNING` cannot be used in multi-table `DELETE`s.

MariaDB starting with [10.3.1](#)

Same Source and Target Table

Until [MariaDB 10.3.1](#), deleting from a table with the same source and target was not possible. From [MariaDB 10.3.1](#), this is now possible. For example:

```
DELETE FROM t1 WHERE c1 IN (SELECT b.c1 FROM t1 b WHERE b.c2=0);
```

MariaDB starting with [10.3.4](#)

DELETE HISTORY

One can use `DELETE HISTORY` to delete historical information from [System-versioned tables](#).

Examples

How to use the [ORDER BY](#) and [LIMIT](#) clauses:

```
DELETE FROM page_hit ORDER BY timestamp LIMIT 1000000;
```

How to use the [RETURNING](#) clause:

```
DELETE FROM t RETURNING f1;
+-----+
| f1   |
+-----+
|    5 |
|   50 |
| 500 |
+-----+
```

The following statement joins two tables: one is only used to satisfy a WHERE condition, but no row is deleted from it; rows from the other table are deleted, instead.

```
DELETE post FROM blog INNER JOIN post WHERE blog.id = post.blog_id;
```

Deleting from the Same Source and Target

```
CREATE TABLE t1 (c1 INT, c2 INT);
DELETE FROM t1 WHERE c1 IN (SELECT b.c1 FROM t1 b WHERE b.c2=0);
```

Until [MariaDB 10.3.1](#), this returned:

```
ERROR 1093 (HY000): Table 't1' is specified twice, both as a target for 'DELETE'
and as a separate source for
```

From [MariaDB 10.3.1](#):

```
Query OK, 0 rows affected (0.00 sec)
```

See Also

- [How IGNORE works](#)
- [SELECT](#)
- [ORDER BY](#)
- [LIMIT](#)
- [REPLACE ... RETURNING](#)
- [INSERT ... RETURNING](#)
- [Returning clause](#) (video)

H3Dr1.2.1.8 DROP TABLE

Syntax

```
DROP [TEMPORARY] TABLE [IF EXISTS] /*COMMENT TO SAVE*/
tbl_name [, tbl_name] ...
[WAIT n|NOWAIT]
[RESTRICT | CASCADE]
```

Contents

1. [Syntax](#)
2. [Description](#)
 1. [WAIT/NOWAIT](#)
3. [DROP TABLE in replication](#)
4. [Dropping an Internal #sql... Table](#)
5. [Dropping All Tables in a Database](#)
6. [Atomic DROP TABLE](#)
7. [Examples](#)
8. [Notes](#)
9. [See Also](#)

Description

`DROP TABLE` removes one or more tables. You must have the `DROP` privilege for each table. All table data and the table definition are removed, as well as [triggers](#) associated to the table, so be careful with this statement! If any of the tables named in the argument list do not exist, MariaDB returns an error indicating by name which non-existing tables it was unable to drop, but it also drops all of the tables in the list that do exist.

Important: When a table is dropped, user privileges on the table are not automatically dropped. See [GRANT](#).

If another thread is using the table in an explicit transaction or an autocommit transaction, then the thread acquires a [metadata lock \(MDL\)](#) on the table. The `DROP TABLE` statement will wait in the "Waiting for table metadata lock" [thread state](#) until the MDL is released. MDLs are released in the following cases:

- If an MDL is acquired in an explicit transaction, then the MDL will be released when the transaction ends.
- If an MDL is acquired in an autocommit transaction, then the MDL will be released when the statement ends.
- Transactional and non-transactional tables are handled the same.

Note that for a partitioned table, `DROP TABLE` permanently removes the table definition, all of its partitions, and all of the data which was stored in those partitions. It also removes the partitioning definition (.par) file associated with the dropped table.

For each referenced table, `DROP TABLE` drops a temporary table with that name, if it exists. If it does not exist, and the `TEMPORARY` keyword is not used, it drops a non-temporary table with the same name, if it exists. The `TEMPORARY` keyword ensures that a non-temporary table will not accidentally be dropped.

Use `IF EXISTS` to prevent an error from occurring for tables that do not exist. A `NOTE` is generated for each non-existent table when using `IF EXISTS`. See [SHOW WARNINGS](#).

If a [foreign key](#) references this table, the table cannot be dropped. In this case, it is necessary to drop the foreign key first.

`RESTRICT` and `CASCADE` are allowed to make porting from other database systems easier. In MariaDB, they do nothing.

The comment before the table names (`/*COMMENT TO SAVE*/`) is stored in the [binary log](#). That feature can be used by replication tools to send their internal messages.

It is possible to specify table names as `db_name.tab_name`. This is useful to delete tables from multiple databases with one statement. See [Identifier Qualifiers](#) for details.

The [DROP privilege](#) is required to use `DROP TABLE` on non-temporary tables. For temporary tables, no privilege is required, because such tables are only visible for the current session.

Note: `DROP TABLE` automatically commits the current active transaction, unless you use the `TEMPORARY` keyword.

MariaDB starting with [10.5.4](#)

From [MariaDB 10.5.4](#), `DROP TABLE` reliably deletes table remnants inside a storage engine even if the `.frm` file is missing. Before then, a missing `.frm` file would result in the statement failing.

MariaDB starting with [10.3.1](#)

WAIT/NOWAIT

Set the lock wait timeout. See [WAIT and NOWAIT](#).

DROP TABLE in replication

`DROP TABLE` has the following characteristics in [replication](#):

- `DROP TABLE IF EXISTS` are always logged.
- `DROP TABLE` without `IF EXISTS` for tables that don't exist are not written to the [binary log](#).
- Dropping of `TEMPORARY` tables are prefixed in the log with `TEMPORARY`. These drops are only logged when running [statement](#) or [mixed mode](#) replication.
- One `DROP TABLE` statement can be logged with up to 3 different `DROP` statements:
 - `DROP TEMPORARY TABLE list_of_non_transactional_temporary_tables`
 - `DROP TEMPORARY TABLE list_of_transactional_temporary_tables`
 - `DROP TABLE list_of_normal_tables`

`DROP TABLE` on the primary is treated on the replica as `DROP TABLE IF EXISTS`. You can change that by setting [slave-ddl-exec-mode](#) to `STRICT`.

Dropping an Internal #sql-... Table

From [MariaDB 10.6](#), `DROP TABLE` is atomic and the following does not apply. Until [MariaDB 10.5](#), if the [mariadb/mysqld process](#) is killed during an `ALTER TABLE` you may find a table named `#sql-...` in your data directory. In [MariaDB 10.3](#), InnoDB tables with this prefix will be deleted automatically during startup. From [MariaDB 10.4](#), these temporary tables will always be deleted automatically.

If you want to delete one of these tables explicitly you can do so by using the following syntax:

```
DROP TABLE `#mysql150##sql-...`;
```

When running an `ALTER TABLE...ALGORITHM=INPLACE` that rebuilds the table, InnoDB will create an internal `#sql-ib` table. Until [MariaDB 10.3.2](#), for these tables, the `.frm` file will be called something else. In order to drop such a table after a server crash, you must rename the `#sql*.frm` file to match the `#sql-ib*.ibd` file.

From [MariaDB 10.3.3](#), the same name as the `.frm` file is used for the intermediate copy of the table. The `#sql-ib` names are used by `TRUNCATE` and delayed `DROP`.

From [MariaDB 10.2.19](#) and [MariaDB 10.3.10](#), the `#sql-ib` tables will be deleted automatically.

Dropping All Tables in a Database

The best way to drop all tables in a database is by executing `DROP DATABASE`, which will drop the database itself, and all tables in it.

However, if you want to drop all tables in the database, but you also want to keep the database itself and any other non-table objects in it, then you would need to execute `DROP TABLE` to drop each individual table. You can construct these `DROP TABLE` commands by querying the [TABLES](#) table in the `information_schema` database. For example:

```
SELECT CONCAT('DROP TABLE IF EXISTS `', TABLE_SCHEMA, '`.`', TABLE_NAME, '`;')
FROM information_schema.TABLES
WHERE TABLE_SCHEMA = 'mydb';
```

Atomic DROP TABLE

MariaDB starting with [10.6.1](#)

From [MariaDB 10.6](#), `DROP TABLE` for a single table is atomic ([MDEV-25180](#)) for most engines, including InnoDB, MyRocks, MyISAM and Aria.

This means that if there is a crash (server down or power outage) during `DROP TABLE`, all tables that have been processed so far will be completely dropped, including related trigger files and status entries, and the [binary log](#) will include a `DROP TABLE` statement for the dropped tables. Tables for which the drop had not started will be left intact.

In older MariaDB versions, there was a small chance that, during a server crash happening in the middle of `DROP TABLE`, some storage engines that were using multiple storage files, like [MyISAM](#), could have only a part of its internal files dropped.

In [MariaDB 10.5](#), `DROP TABLE` was extended to be able to delete a table that was only partly dropped ([MDEV-11412](#)) as explained above. Atomic `DROP TABLE` is the final piece to make `DROP TABLE` fully reliable.

Dropping multiple tables is crash-safe.

See [Atomic DDL](#) for more information.

Examples

```
DROP TABLE Employees, Customers;
```

Notes

Beware that `DROP TABLE` can drop both tables and [sequences](#). This is mainly done to allow old tools like [mysqldump](#) to work with sequences.

See Also

- [CREATE TABLE](#)
- [ALTER TABLE](#)
- [SHOW CREATE TABLE](#)
- [DROP SEQUENCE](#)
- Variable [slave-ddl-exec-mode](#).

H3Dr1.2.1.9 Installing System Tables (`mysql_install_db`)

`mysql_install_db` initializes the MariaDB data directory and creates the [system tables](#) in the [mysql](#) database, if they do not exist. MariaDB uses these tables to manage [privileges](#), [roles](#), and [plugins](#). It also uses them to provide the data for the [help](#) command in the [mysql](#) client.

`mysql_install_db` works by starting MariaDB Server's `mysqld` process in [--bootstrap](#) mode and sending commands to create the [system tables](#) and their content.

There is a version specifically for Windows, [mysql_install_db.exe](#).

To invoke `mysql_install_db`, use the following syntax:

```
mysql_install_db --user=mysql
```

For the options supported by `mysql_install_db`, see [mysql_install_db: Options](#).

For the option groups read by `mysql_install_db`, see [mysql_install_db: Option Groups](#).

See [mysql_install_db: Installing System Tables](#) for information on the installation process.

See [mysql_install_db: Troubleshooting Issues](#) for information on how to troubleshoot the installation

process.

See Also

- [mysql_install_db](#)
- The Windows version of mysql_install_db : [mysql_install_db.exe](#)

H3Dr1.2.1.10 mysqlcheck

MariaDB starting with [10.4.6](#)

From [MariaDB 10.4.6](#), mariadb-check is a symlink to mysqlcheck .

MariaDB starting with [10.5.2](#)

From [MariaDB 10.5.2](#), mariadb-check is the name of the tool, with mysqlcheck a symlink .

Contents

1. [Using mysqlcheck](#)
 1. [Options](#)
 2. [Option Files](#)
 1. [Option Groups](#)
2. [Notes](#)
 1. [Default Values](#)
 2. [mysqlcheck and auto-repair](#)
 3. [mysqlcheck and all-databases](#)
 4. [mysqlcheck and verbose](#)

mysqlcheck is a maintenance tool that allows you to check, repair, analyze and optimize multiple tables from the command line.

It is essentially a commandline interface to the [CHECK TABLE](#), [REPAIR TABLE](#), [ANALYZE TABLE](#) and [OPTIMIZE TABLE](#) commands, and so, unlike [myisamchk](#) and [aria_chk](#), requires the server to be running.

This tool does not work with partitioned tables.

Using mysqlcheck

```
./client/mysqlcheck [OPTIONS] database [tables]
```

OR

```
./client/mysqlcheck [OPTIONS] --databases DB1 [DB2 DB3...]
```

OR

```
./client/mysqlcheck [OPTIONS] --all-databases
```

mysqlcheck can be used to CHECK (-c, -m, -C), REPAIR (-r), ANALYZE (-a), or OPTIMIZE (-o) tables. Some of the options (like -e or -q) can be used at the same time. Not all options are supported by all storage engines.

The -c, -r, -a and -o options are exclusive to each other.

The option --check will be used by default, if no other options were specified. You can change the default behavior by making a symbolic link to the binary, or copying it somewhere with another name, the alternatives are:

[mysqlrepair](#)

The default option will be -r (--repair)

<code>mysqlanalyze</code>	The default option will be <code>-a (--analyze)</code>
<code>mysqloptimize</code>	The default option will be <code>-o (--optimize)</code>

Options

`mysqlcheck` supports the following options:

Option	Description
<code>-A , --all-databases</code>	Check all the databases. This is the same as <code>--databases</code> with all databases selected.
<code>-1 , --all-in-1</code>	Instead of issuing one query for each table, use one query per database, naming all tables in the database in a comma-separated list.
<code>-a , --analyze</code>	Analyze given tables.
<code>--auto-repair</code>	If a checked table is corrupted, automatically fix it. Repairing will be done after all tables have been checked.
<code>--character-sets-dir=name</code>	Directory where character set files are installed.
<code>-c , --check</code>	Check table for errors.
<code>-C , --check-only-changed</code>	Check only tables that have changed since last check or haven't been closed properly.
<code>-g , --check-upgrade</code>	Check tables for version-dependent changes. May be used with <code>--auto-repair</code> to correct tables requiring version-dependent updates. Automatically enables the <code>--fix-db-names</code> and <code>--fix-table-names</code> options. Used when upgrading
<code>--compress</code>	Compress all information sent between the client and server if both support compression.
<code>-B , --databases</code>	Check several databases. Note that normally <code>mysql/check</code> treats the first argument as a database name, and following arguments as table names. With this option, no tables are given, and all name arguments are regarded as database names.
<code>-# , --debug[=name]</code>	Output debug log. Often this is 'd:t:o,filename'.
<code>--debug-check</code>	Check memory and open file usage at exit.
<code>--debug-info</code>	Print some debug info at exit.
<code>--default-auth=plugin</code>	Default authentication client-side plugin to use.
<code>--default-character-set=name</code>	Set the default character set .
<code>-e , --extended</code>	If you are using this option with <code>--check</code> , it will ensure that the table is 100 percent consistent, but will take a long time. If you are using this option with <code>--repair</code> , it will force using the old, slow, repair with keycache method, instead of the much faster repair by sorting.
<code>-F , --fast</code>	Check only tables that haven't been closed properly.
<code>--fix-db-names</code>	Convert database names to the format used since MySQL 5.1. Only database names that contain special characters are affected. Used when upgrading from an old MySQL version.
<code>--fix-table-names</code>	Convert table names (including views) to the format used since MySQL 5.1. Only table names that contain special characters are affected. Used when upgrading from an old MySQL version.
<code>--flush</code>	Flush each table after check. This is useful if you don't want to have the checked tables take up space in the caches after the check.

<code>-f , --force</code>	Continue even if we get an SQL error.
<code>-? , --help</code>	Display this help message and exit.
<code>-h name , --host=name</code>	Connect to the given host.
<code>-m , --medium-check</code>	Faster than extended-check, but only finds 99.99 percent of all errors. Should be good enough for most cases.
<code>-o , --optimize</code>	Optimize tables.
<code>-p , --password[=name]</code>	Password to use when connecting to the server. If you use the short option form (<code>-p</code>), you cannot have a space between the option and the password. If you omit the password value following the <code>--password</code> or <code>-p</code> option on the command line, mysqlcheck prompts for one. Specifying a password on the command line should be considered insecure. You can use an option file to avoid giving the password on the command line.
<code>-z , --persistent</code>	When using <code>ANALYZE TABLE</code> (<code>--analyze</code>), uses the <code>PERSISTENT FOR ALL</code> option, which forces Engine-independent Statistics for this table to be updated. Added in MariaDB 10.1.10
<code>-W , --pipe</code>	On Windows, connect to the server via a named pipe. This option applies only if the server supports named-pipe connections.
<code>--plugin-dir</code>	Directory for client-side plugins.
<code>-P num , --port=num</code>	Port number to use for connection or 0 for default to, in order of preference, my.cnf, \$MYSQL_TCP_PORT, /etc/services, built-in default (3306).
<code>--process-tables</code>	Perform the requested operation (check, repair, analyze, optimize) on tables. Enabled by default. Use <code>--skip-process-tables</code> to disable.
<code>--process-views[=val]</code>	Perform the requested operation (only CHECK VIEW or REPAIR VIEW). Possible values are NO, YES (correct the checksum, if necessary, add the mariadb-version field), UPGRADE_FROM_MYSQL (same as YES and toggle the algorithm MERGE<->TEMPTABLE).
<code>--protocol=name</code>	The connection protocol (tcp, socket, pipe, memory) to use for connecting to the server. Useful when other connection parameters would cause a protocol to be used other than the one you want.
<code>-q , --quick</code>	If you are using this option with <code>CHECK TABLE</code> , it prevents the check from scanning the rows to check for wrong links. This is the fastest check. If you are using this option with <code>REPAIR TABLE</code> , it will try to repair only the index tree. This is the fastest repair method for a table.
<code>-r , --repair</code>	Can fix almost anything except unique keys that aren't unique.
<code>--shared-memory-base-name</code>	Shared-memory name to use for Windows connections using shared memory to a local server (started with the <code>--shared-memory</code> option). Case-sensitive.
<code>-s , --silent</code>	Print only error messages.
<code>--skip-database</code>	Don't process the database (case-sensitive) specified as argument.
<code>-S name , --socket=name</code>	For connections to localhost, the Unix socket file to use, or, on Windows, the name of the named pipe to use.
<code>--ssl</code>	Enables TLS . TLS is also enabled even without setting this option when certain other TLS options are set. Starting with MariaDB 10.2 , the <code>--ssl</code> option will not enable verifying the server certificate by default. In order to verify the server certificate, the user must specify the <code>--ssl-verify-server-cert</code> option.

--ssl-ca=name	Defines a path to a PEM file that should contain one or more X509 certificates for trusted Certificate Authorities (CAs) to use for TLS . This option requires that you use the absolute path, not a relative path. See Secure Connections Overview: Certificate Authorities (CAs) for more information. This option implies the --ssl option.
--ssl-capath=name	Defines a path to a directory that contains one or more PEM files that should each contain one X509 certificate for a trusted Certificate Authority (CA) to use for TLS . This option requires that you use the absolute path, not a relative path. The directory specified by this option needs to be run through the openssl rehash command. See Secure Connections Overview: Certificate Authorities (CAs) for more information. This option is only supported if the client was built with OpenSSL or yaSSL. If the client was built with GnuTLS or Schannel, then this option is not supported. See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used on which platforms. This option implies the --ssl option.
--ssl-cert=name	Defines a path to the X509 certificate file to use for TLS . This option requires that you use the absolute path, not a relative path. This option implies the --ssl option.
--ssl-cipher=name	List of permitted ciphers or cipher suites to use for TLS . This option implies the --ssl option.
--ssl-crl=name	Defines a path to a PEM file that should contain one or more revoked X509 certificates to use for TLS . This option requires that you use the absolute path, not a relative path. See Secure Connections Overview: Certificate Revocation Lists (CRLs) for more information. This option is only supported if the client was built with OpenSSL or Schannel. If the client was built with yaSSL or GnuTLS, then this option is not supported. See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used on which platforms.
--ssl-crlpath=name	Defines a path to a directory that contains one or more PEM files that should each contain one revoked X509 certificate to use for TLS . This option requires that you use the absolute path, not a relative path. The directory specified by this option needs to be run through the openssl rehash command. See Secure Connections Overview: Certificate Revocation Lists (CRLs) for more information. This option is only supported if the client was built with OpenSSL. If the client was built with yaSSL, GnuTLS, or Schannel, then this option is not supported. See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used on which platforms.
--ssl-key=name	Defines a path to a private key file to use for TLS . This option requires that you use the absolute path, not a relative path. This option implies the --ssl option.
--ssl-verify-server-cert	Enables server certificate verification . This option is disabled by default.
--tables	Overrides the --databases or -B option such that all name arguments following the option are regarded as table names.
--use-frm	For repair operations on MyISAM tables, get table structure from .frm file, so the table can be repaired even if the .MYI header is corrupted.
-u , --user=name	User for login if not current user.
-v , --verbose	Print info about the various stages. You can give this option several times to get even more information. See mysqlcheck and verbose , below.
-V , --version	Output version information and exit.
--write-binlog	Write ANALYZE, OPTIMIZE and REPAIR TABLE commands to the binary log . Enabled by default; use --skip-write-binlog when commands should not be sent to replication slaves.

Option Files

In addition to reading options from the command-line, `mysqlcheck` can also read options from [option files](#)

>If an unknown option is provided to `mysqlcheck` in an option file, then it is ignored.

The following options relate to how MariaDB command-line tools handles option files. They must be given as the first argument on the command-line:

Option	Description
<code>--print-defaults</code>	Print the program argument list and exit.
<code>--no-defaults</code>	Don't read default options from any option file.
<code>--defaults-file=#</code>	Only read default options from the given file #.
<code>--defaults-extra-file=#</code>	Read this file after the global files are read.
<code>--defaults-group-suffix=#</code>	In addition to the default option groups, also read option groups with this suffix.

In MariaDB 10.2 and later, `mysqlcheck` is linked with MariaDB Connector/C. However, MariaDB Connector/C does not yet handle the parsing of option files for this client. That is still performed by the server option file parsing code. See MDEV-19035 for more information.

Option Groups

`mysqlcheck` reads options from the following option groups from option files:

Group	Description
<code>[mysqlcheck]</code>	Options read by <code>mysqlcheck</code> , which includes both MariaDB Server and MySQL Server.
<code>[mariadb-check]</code>	Options read by <code>mysqlcheck</code> . Available starting with MariaDB 10.4.6.
<code>[client]</code>	Options read by all MariaDB and MySQL client programs, which includes both MariaDB and MySQL clients. For example, <code>mysqldump</code> .
<code>[client-server]</code>	Options read by all MariaDB client programs and the MariaDB Server. This is useful for options like socket and port, which is common between the server and the clients.
<code>[client-mariadb]</code>	Options read by all MariaDB client programs.

Notes

Default Values

To see the default values for the options and also to see the arguments you get from configuration files you can do:

```
./client/mysqlcheck --print-defaults  
./client/mysqlcheck --help
```

mysqlcheck and auto-repair

When running `mysqlcheck` with `--auto-repair` (as done by `mysql_upgrade`), `mysqlcheck` will first check all tables and then in a separate phase repair those that failed the check.

mysqlcheck and all-databases

`mysqlcheck --all-databases` will ignore the internal log tables `general_log` and `slow_log` as these can't be checked, repaired or optimized.

mysqlcheck and verbose

Using one `--verbose` option will give you more information about what mysqlcheck is doing.

Using two `--verbose` options will also give you connection information.

If you use three `--verbose` options you will also get, on stdout, all `ALTER`, `RENAME`, and `CHECK` commands that mysqlcheck executes.

H3Dr1.2.1.11 OPTIMIZE TABLE

Syntax

```
OPTIMIZE [NO_WRITE_TO_BINLOG | LOCAL] TABLE
    tbl_name [, tbl_name] ...
    [WAIT n | NOWAIT]
```

Contents

1. [Syntax](#)
2. [Description](#)
 1. [WAIT/NOWAIT](#)
 2. [Defragmenting](#)
 3. [Updating an InnoDB fulltext index](#)
 4. [Defragmenting InnoDB tablespaces](#)
3. [See Also](#)

Description

`OPTIMIZE TABLE` has two main functions. It can either be used to defragment tables, or to update the InnoDB fulltext index.

MariaDB starting with [10.3.0](#)

WAIT/NOWAIT

Set the lock wait timeout. See [WAIT](#) and [NOWAIT](#).

Defragmenting

`OPTIMIZE TABLE` works for [InnoDB](#) (before MariaDB 10.1.1, only if the `innodb_file_per_table` server system variable is set), [Aria](#), [MyISAM](#) and [ARCHIVE](#) tables, and should be used if you have deleted a large part of a table or if you have made many changes to a table with variable-length rows (tables that have `VARCHAR`, `VARBINARY`, `BLOB`, or `TEXT` columns). Deleted rows are maintained in a linked list and subsequent `INSERT` operations reuse old row positions.

This statement requires [SELECT](#) and [INSERT](#) privileges for the table.

By default, `OPTIMIZE TABLE` statements are written to the [binary log](#) and will be [replicated](#). The `NO_WRITE_TO_BINLOG` keyword (`LOCAL` is an alias) will ensure the statement is not written to the binary log.

From [MariaDB 10.3.19](#), `OPTIMIZE TABLE` statements are not logged to the binary log if `read_only` is set. See also [Read-Only Replicas](#).

`OPTIMIZE TABLE` is also supported for partitioned tables. You can use `ALTER TABLE ... OPTIMIZE PARTITION` to optimize one or more partitions.

You can use `OPTIMIZE TABLE` to reclaim the unused space and to defragment the data file. With other storage engines, `OPTIMIZE TABLE` does nothing by default, and returns this message: "The storage engine for the table doesn't support optimize". However, if the server has been started with the `--skip-new` option, `OPTIMIZE TABLE` is linked to `ALTER TABLE`, and recreates the table. This operation frees the unused space and updates index statistics.

The [Aria](#) storage engine supports [progress reporting](#) for this statement.

If a [MyISAM](#) table is fragmented, [concurrent inserts](#) will not be performed until an `OPTIMIZE TABLE`

statement is executed on that table, unless the [concurrent_insert](#) server system variable is set to `ALWAYS`.

Updating an InnoDB fulltext index

When rows are added or deleted to an InnoDB [fulltext index](#), the index is not immediately re-organized, as this can be an expensive operation. Change statistics are stored in a separate location. The fulltext index is only fully re-organized when an `OPTIMIZE TABLE` statement is run.

By default, an `OPTIMIZE TABLE` will defragment a table. In order to use it to update fulltext index statistics, the [innodb_optimize_fulltext_only](#) system variable must be set to `1`. This is intended to be a temporary setting, and should be reset to `0` once the fulltext index has been re-organized.

Since fulltext re-organization can take a long time, the [innodb_ft_num_word_optimize](#) variable limits the re-organization to a number of words (2000 by default). You can run multiple `OPTIMIZE` statements to fully re-organize the index.

Defragmenting InnoDB tablespaces

[MariaDB 10.1.1](#) merged the Facebook/Kakao defragmentation patch, allowing one to use `OPTIMIZE TABLE` to defragment InnoDB tablespaces. For this functionality to be enabled, the [innodb_defragment](#) system variable must be enabled. No new tables are created and there is no need to copy data from old tables to new tables. Instead, this feature loads `n` pages (determined by [innodb-defragment-n-pages](#)) and tries to move records so that pages would be full of records and then frees pages that are fully empty after the operation. Note that tablespace files (including `ibdata1`) will not shrink as the result of defragmentation, but one will get better memory utilization in the InnoDB buffer pool as there are fewer data pages in use.

See [Defragmenting InnoDB Tablespaces](#) for more details.

See Also

- [Optimize Table in InnoDB with ALGORITHM set to INPLACE](#)
- [Optimize Table in InnoDB with ALGORITHM set to NOCOPY](#)
- [Optimize Table in InnoDB with ALGORITHM set to INSTANT](#)

H3Dr1.2.1.12 RENAME TABLE

Syntax

```
RENAME TABLE[S] [IF EXISTS] tbl_name  
[WAIT n | NOWAIT]  
TO new_tbl_name  
[, tbl_name2 TO new_tbl_name2] ...
```

Contents

1. [Syntax](#)
2. [Description](#)
 1. [IF EXISTS](#)
 2. [WAIT/NOWAIT](#)
 3. [Privileges](#)
 4. [Atomic RENAME TABLE](#)

Description

This statement renames one or more tables or [views](#), but not the privileges associated with them.

IF EXISTS

MariaDB starting with 10.5.2 [↗](#)

If this directive is used, one will not get an error if the table to be renamed doesn't exist.

The rename operation is done atomically, which means that no other session can access any of the tables while the rename is running. For example, if you have an existing table `old_table`, you can create another table `new_table` that has the same structure but is empty, and then replace the existing table with the empty one as follows (assuming that `backup_table` does not already exist):

```
CREATE TABLE new_table (...);  
RENAME TABLE old_table TO backup_table, new_table TO old_table;
```

`tbl_name` can optionally be specified as `db_name . tbl_name`. See [Identifier Qualifiers ↗](#). This allows to use `RENAME` to move a table from a database to another (as long as they are on the same filesystem):

```
RENAME TABLE db1.t TO db2.t;
```

Note that moving a table to another database is not possible if it has some [triggers ↗](#). Trying to do so produces the following error:

```
ERROR 1435 (HY000): Trigger in wrong schema
```

Also, views cannot be moved to another database:

```
ERROR 1450 (HY000): Changing schema from 'old_db' to 'new_db' is not allowed.
```

Multiple tables can be renamed in a single statement. The presence or absence of the optional `s` (`RENAME TABLE` or `RENAME TABLES`) has no impact, whether a single or multiple tables are being renamed.

If a `RENAME TABLE` renames more than one table and one renaming fails, all renames executed by the same statement are rolled back.

Renames are always executed in the specified order. Knowing this, it is also possible to swap two tables' names:

```
RENAME TABLE t1 TO tmp_table,  
      t2 TO t1,  
      tmp_table TO t2;
```

WAIT/NOWAIT

MariaDB starting with 10.3.0 [↗](#)

Set the lock wait timeout. See [WAIT and NOWAIT ↗](#).

Privileges

Executing the `RENAME TABLE` statement requires the `DROP`, `CREATE` and `INSERT` privileges for the table or the database.

Atomic RENAME TABLE

MariaDB starting with 10.6.1 [↗](#)

From [MariaDB 10.6 ↗](#), `RENAME TABLE` is atomic for most engines, including InnoDB, MyRocks, MyISAM and Aria ([MDEV-23842 ↗](#)). This means that if there is a crash (server down or power outage) during `RENAME TABLE`, all tables will revert to their original names and any changes to trigger files will be reverted.

In older MariaDB version there was a small chance that, during a server crash happening in the middle of `RENAME TABLE`, some tables could have been renamed (in the worst case partly) while others would

not be renamed.

See [Atomic DDL](#) for more information.

H3Dr1.2.1.13 REPAIR TABLE

Syntax

```
REPAIR [NO_WRITE_TO_BINLOG | LOCAL] TABLE
tbl_name [, tbl_name] ...
[QUICK] [EXTENDED] [USE_FRM]
```

Description

`REPAIR TABLE` repairs a possibly corrupted table. By default, it has the same effect as

```
myisamchk --recover tbl_name
```

or

```
aria_chk --recover tbl_name
```

See [aria_chk](#) and [myisamchk](#) for more.

`REPAIR TABLE` works for [Archive](#), [Aria](#), [CSV](#) and [MyISAM](#) tables. For [InnoDB](#), see [recovery modes](#). For CSV, see also [Checking and Repairing CSV Tables](#). For Archive, this statement also improves compression. If the storage engine does not support this statement, a warning is issued.

This statement requires [SELECT](#) and [INSERT](#) privileges for the table.

By default, `REPAIR TABLE` statements are written to the [binary log](#) and will be [replicated](#). The `NO_WRITE_TO_BINLOG` keyword (`LOCAL` is an alias) will ensure the statement is not written to the binary log.

From [MariaDB 10.3.19](#), `REPAIR TABLE` statements are not logged to the binary log if [read_only](#) is set. See also [Read-Only Replicas](#).

When an index is recreated, the storage engine may use a configurable buffer in the process. Incrementing the buffer speeds up the index creation. [Aria](#) and [MyISAM](#) allocate a buffer whose size is defined by [aria_sort_buffer_size](#) or [myisam_sort_buffer_size](#), also used for [ALTER TABLE](#).

`REPAIR TABLE` is also supported for partitioned tables. However, the `USE_FRM` option cannot be used with this statement on a partitioned table.

[ALTER TABLE ... REPAIR PARTITION](#) can be used to repair one or more partitions.

The [Aria](#) storage engine supports [progress reporting](#) for this statement.

H3Dr1.2.1.14 REPAIR VIEW

Syntax

```
REPAIR [NO_WRITE_TO_BINLOG | LOCAL] VIEW view_name[, view_name] ... [FROM MYSQL]
```

Contents

1. [Syntax](#)
2. [Description](#)
3. [See Also](#)

Description

The `REPAIR VIEW` statement was introduced to assist with fixing [MDEV-6916](#), an issue introduced in [MariaDB 5.2](#) where the view algorithms were swapped compared to their MySQL on disk representation. It checks whether the view algorithm is correct. It is run as part of `mysql_upgrade`, and should not normally be required in regular use.

By default it corrects the checksum and if necessary adds the mariadb-version field. If the optional `FROM MYSQL` clause is used, and no mariadb-version field is present, the MERGE and TEMPTABLE algorithms are toggled.

By default, `REPAIR VIEW` statements are written to the [binary log](#) and will be [replicated](#). The `NO_WRITE_TO_BINLOG` keyword (`LOCAL` is an alias) will ensure the statement is not written to the binary log.

See Also

- [CHECK VIEW](#)

H3Dr1.2.1.15 REPLACE

Syntax

```
REPLACE [LOW_PRIORITY | DELAYED]
[INTO] tbl_name [PARTITION (partition_list)] [(col,...)]
{VALUES | VALUE} ({expr | DEFAULT},...),(...),...
[RETURNING select_expr
 [, select_expr ...]]
```

Or:

```
REPLACE [LOW_PRIORITY | DELAYED]
[INTO] tbl_name [PARTITION (partition_list)]
SET col={expr | DEFAULT}, ...
[RETURNING select_expr
 [, select_expr ...]]
```

Or:

```
REPLACE [LOW_PRIORITY | DELAYED]
[INTO] tbl_name [PARTITION (partition_list)] [(col,...)]
SELECT ...
[RETURNING select_expr
 [, select_expr ...]]
```

Contents

1. [Syntax](#)
2. [Description](#)
 1. [PARTITION](#)
 2. [REPLACE RETURNING](#)
 1. [Examples](#)
3. [Examples](#)
4. [See Also](#)

Description

`REPLACE` works exactly like [INSERT](#), except that if an old row in the table has the same value as a new row for a `PRIMARY KEY` or a `UNIQUE` index, the old row is deleted before the new row is inserted. If the table has more than one `UNIQUE` keys, it is possible that the new row conflicts with more than one row. In this case, all conflicting rows will be deleted.

The table name can be specified in the form `db_name . tbl_name` or, if a default database is selected, in the form `tbl_name` (see [Identifier Qualifiers](#)). This allows to use `REPLACE ... SELECT` to copy rows between different databases.

MariaDB starting with [10.5.0](#)

The RETURNING clause was introduced in [MariaDB 10.5.0](#)

Basically it works like this:

```
BEGIN;
SELECT 1 FROM t1 WHERE key=# FOR UPDATE;
IF found-row
    DELETE FROM t1 WHERE key=# ;
ENDIF
INSERT INTO t1 VALUES (...);
END;
```

The above can be replaced with:

```
REPLACE INTO t1 VALUES (...)
```

`REPLACE` is a MariaDB/MySQL extension to the SQL standard. It either inserts, or deletes and inserts. For other MariaDB/MySQL extensions to standard SQL --- that also handle duplicate values --- see [IGNORE](#) and [INSERT ON DUPLICATE KEY UPDATE](#).

Note that unless the table has a `PRIMARY KEY` or `UNIQUE` index, using a `REPLACE` statement makes no sense. It becomes equivalent to `INSERT`, because there is no index to be used to determine whether a new row duplicates another.

Values for all columns are taken from the values specified in the `REPLACE` statement. Any missing columns are set to their default values, just as happens for `INSERT`. You cannot refer to values from the current row and use them in the new row. If you use an assignment such as `'SET col = col + 1'`, the reference to the column name on the right hand side is treated as `DEFAULT(col)`, so the assignment is equivalent to `'SET col = DEFAULT(col) + 1'`.

To use `REPLACE`, you must have both the `INSERT` and `DELETE` privileges for the table.

There are some gotchas you should be aware of, before using `REPLACE`:

- If there is an [AUTO_INCREMENT](#) field, a new value will be generated.
- If there are foreign keys, `ON DELETE` action will be activated by `REPLACE`.
- [Triggers](#) on `DELETE` and `INSERT` will be activated by `REPLACE`.

To avoid some of these behaviors, you can use `INSERT ... ON DUPLICATE KEY UPDATE`.

This statement activates `INSERT` and `DELETE` triggers. See [Trigger Overview](#) for details.

PARTITION

See [Partition Pruning and Selection](#) for details.

REPLACE RETURNING

`REPLACE ... RETURNING` returns a resultset of the replaced rows.

This returns the listed columns for all the rows that are replaced, or alternatively, the specified `SELECT` expression. Any SQL expressions which can be calculated can be used in the select expression for the `RETURNING` clause, including virtual columns and aliases, expressions which use various operators such as bitwise, logical and arithmetic operators, string functions, date-time functions, numeric functions, control flow functions, secondary functions and stored functions. Along with this, statements which have subqueries and prepared statements can also be used.

Examples

Simple REPLACE statement

```
REPLACE INTO t2 VALUES (1,'Leopard'),(2,'Dog') RETURNING id2, id2+id2
as Total ,id2|id2, id2&&id2;
+-----+-----+
| id2 | Total | id2|id2 | id2&&id2 |
+-----+-----+
| 1 | 2 | 1 | 1 |
| 2 | 4 | 2 | 1 |
+-----+-----+
```

Using stored functions in RETURNING

```
DELIMITER |
CREATE FUNCTION f(arg INT) RETURNS INT
BEGIN
    RETURN (SELECT arg+arg);
END |

DELIMITER ;
PREPARE stmt FROM "REPLACE INTO t2 SET id2=3, animal2='Fox' RETURNING f2(id2),
UPPER(animal2)";

EXECUTE stmt;
+-----+
| f2(id2) | UPPER(animal2) |
+-----+
| 6 | FOX |
+-----+
```

Subqueries in the statement

```
REPLACE INTO t1 SELECT * FROM t2 RETURNING (SELECT id2 FROM t2 WHERE
id2 IN (SELECT id2 FROM t2 WHERE id2=1)) AS new_id;
+-----+
| new_id |
+-----+
| 1 |
| 1 |
| 1 |
| 1 |
+-----+
```

Subqueries in the RETURNING clause that return more than one row or column cannot be used..

Aggregate functions cannot be used in the RETURNING clause. Since aggregate functions work on a set of values and if the purpose is to get the row count, ROW_COUNT() with SELECT can be used, or it can be used in REPLACE...SEL== Description

REPLACE ... RETURNING returns a resultset of the replaced rows.

This returns the listed columns for all the rows that are replaced, or alternatively, the specified SELECT expression. Any SQL expressions which can be calculated can be used in the select expression for the RETURNING clause, including virtual columns and aliases, expressions which use various operators such as bitwise, logical and arithmetic operators, string functions, date-time functions, numeric functions, control flow functions, secondary functions and stored functions. Along with this, statements which have subqueries and prepared statements can also be used.

Examples

Simple REPLACE statement

```

REPLACE INTO t2 VALUES (1,'Leopard'),(2,'Dog') RETURNING id2, id2+id2
as Total ,id2|id2, id2&&id2;
+-----+-----+-----+
| id2 | Total | id2|id2 | id2&&id2 |
+-----+-----+-----+
|   1 |      2 |       1 |       1 |
|   2 |      4 |       2 |       1 |
+-----+-----+-----+

```

Using stored functions in RETURNING

```

DELIMITER |
CREATE FUNCTION f(arg INT) RETURNS INT
BEGIN
    RETURN (SELECT arg+arg);
END |

DELIMITER ;
PREPARE stmt FROM "REPLACE INTO t2 SET id2=3, animal2='Fox' RETURNING f2(id2),
UPPER(animal2)";

EXECUTE stmt;
+-----+-----+
| f2(id2) | UPPER(animal2) |
+-----+-----+
|       6 | FOX           |
+-----+-----+

```

Subqueries in the statement

```

REPLACE INTO t1 SELECT * FROM t2 RETURNING (SELECT id2 FROM t2 WHERE
id2 IN (SELECT id2 FROM t2 WHERE id2=1)) AS new_id;
+-----+
| new_id |
+-----+
|   1 |
|   1 |
|   1 |
|   1 |
+-----+

```

Subqueries in the RETURNING clause that return more than one row or column cannot be used..

Aggregate functions cannot be used in the RETURNING clause. Since aggregate functions work on a set of values and if the purpose is to get the row count, ROW_COUNT() with SELECT can be used, or it can be used in REPLACE...SELECT...RETURNING if the table in the RETURNING clause is not the same as the REPLACE table. ECT...RETURNING if the table in the RETURNING clause is not the same as the REPLACE table.

See Also

- [INSERT ↗](#)
- [HIGH_PRIORITY and LOW_PRIORITY clauses ↗](#)
- [INSERT DELAYED ↗](#) for details on the `DELAYED` clause

H3Dr1.2.1.16 SHOW COLUMNS

Syntax

```

SHOW [FULL] {COLUMNS | FIELDS} FROM tbl_name [FROM db_name]
[LIKE 'pattern' | WHERE expr]

```

Contents

1. [Syntax](#)
2. [Description](#)
3. [Examples](#)
4. [See Also](#)

Description

`SHOW COLUMNS` displays information about the columns in a given table. It also works for views. The `LIKE` clause, if present on its own, indicates which column names to match. The `WHERE` and `LIKE` clauses can be given to select rows using more general conditions, as discussed in [Extended SHOW](#).

If the data types differ from what you expect them to be based on a `CREATE TABLE` statement, note that MariaDB sometimes changes data types when you create or alter a table. The conditions under which this occurs are described in the [Silent Column Changes](#) article.

The `FULL` keyword causes the output to include the column collation and comments, as well as the privileges you have for each column.

You can use `db_name.tbl_name` as an alternative to the `tbl_name FROM db_name` syntax. In other words, these two statements are equivalent:

```
SHOW COLUMNS FROM mytable FROM mydb;
SHOW COLUMNS FROM mydb.mytable;
```

`SHOW COLUMNS` displays the following values for each table column:

Field indicates the column name.

Type indicates the column data type.

Collation indicates the collation for non-binary string columns, or `NULL` for other columns. This value is displayed only if you use the `FULL` keyword.

The `Null` field contains `YES` if `NULL` values can be stored in the column, `NO` if not.

The **Key** field indicates whether the column is indexed:

- If **Key** is empty, the column either is not indexed or is indexed only as a secondary column in a multiple-column, non-unique index.
- If **Key** is `PRI`, the column is a `PRIMARY KEY` or is one of the columns in a multiple-column `PRIMARY KEY`.
- If **Key** is `UNI`, the column is the first column of a unique-valued index that cannot contain `NULL` values.
- If **Key** is `MUL`, multiple occurrences of a given value are allowed within the column. The column is the first column of a non-unique index or a unique-valued index that can contain `NULL` values.

If more than one of the **Key** values applies to a given column of a table, **Key** displays the one with the highest priority, in the order `PRI`, `UNI`, `MUL`.

A `UNIQUE` index may be displayed as `PRI` if it cannot contain `NULL` values and there is no `PRIMARY KEY` in the table. A `UNIQUE` index may display as `MUL` if several columns form a composite `UNIQUE` index; although the combination of the columns is unique, each column can still hold multiple occurrences of a given value.

The **Default** field indicates the default value that is assigned to the column.

The **Extra** field contains any additional information that is available about a given column.

Value	Description
<code>AUTO_INCREMENT</code>	The column was created with the <code>AUTO_INCREMENT</code> keyword.
<code>PERSISTENT</code>	The column was created with the <code>PERSISTENT</code> keyword. (New in 5.3)
<code>VIRTUAL</code>	The column was created with the <code>VIRTUAL</code> keyword. (New in 5.3)
<code>on update CURRENT_TIMESTAMP</code>	The column is a <code>TIMESTAMP</code> column that is automatically updated on <code>INSERT</code> and <code>UPDATE</code> .

Privileges indicates the privileges you have for the column. This value is displayed only if you use the `FULL` keyword.

Comment indicates any comment the column has. This value is displayed only if you use the `FULL` keyword.

`SHOW FIELDS` is a synonym for `SHOW COLUMNS`. Also [DESCRIBE](#) and [EXPLAIN](#) can be used as shortcuts.

You can also list a table's columns with:

```
mysqlshow db_name tbl_name
```

See the [mysqlshow](#) command for more details.

The [DESCRIBE](#) statement provides information similar to `SHOW COLUMNS`. The `information_schema.COLUMNS` table provides similar, but more complete, information.

The `SHOW CREATE TABLE`, `SHOW TABLE STATUS`, and `SHOW INDEX` statements also provide information about tables.

Examples

```
SHOW COLUMNS FROM city;
+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra       |
+-----+-----+-----+-----+-----+
| Id    | int(11) | NO   | PRI | NULL    | auto_increment |
| Name  | char(35) | NO   |     |          |               |
| Country | char(3) | NO   | UNI |          |               |
| District | char(20) | YES  | MUL |          |               |
| Population | int(11) | NO   |     | 0        |               |
+-----+-----+-----+-----+-----+
```

```
SHOW COLUMNS FROM employees WHERE Type LIKE 'Varchar%';
+-----+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra       |
+-----+-----+-----+-----+-----+
| first_name | varchar(30) | NO   | MUL | NULL    |             |
| last_name  | varchar(40)  | NO   |     | NULL    |             |
| position   | varchar(25)  | NO   |     | NULL    |             |
| home_address | varchar(50) | NO   |     | NULL    |             |
| home_phone  | varchar(12)  | NO   |     | NULL    |             |
| employee_code | varchar(25) | NO   | UNI | NULL    |             |
+-----+-----+-----+-----+-----+
```

See Also

- [DESCRIBE](#)
- [mysqlshow](#)
- [SHOW CREATE TABLE](#)
- [SHOW TABLE STATUS](#)
- [SHOW INDEX](#)
- [Extended SHOW](#)
- [Silent Column Changes](#)

H3Dr1.2.1.17 SHOW CREATE TABLE

Syntax

```
SHOW CREATE TABLE tbl_name
```

Contents

1. [Syntax](#)
2. [Description](#)
3. [Examples](#)
4. [See Also](#)

Description

Shows the [CREATE TABLE](#) statement that created the given table. The statement requires the [SELECT](#) privilege  for the table. This statement also works with [views](#)  and [SEQUENCE](#) .

`SHOW CREATE TABLE` quotes table and column names according to the value of the `sql_quote_show_create`  server system variable.

Certain [SQL_MODE](#)  values can result in parts of the original [CREATE](#) statement not being included in the output. MariaDB-specific table options, column options, and index options are not included in the output of this statement if the [NO_TABLE_OPTIONS](#) , [NO_FIELD_OPTIONS](#)  and [NO_KEY_OPTIONS](#)  [SQL_MODE](#)  flags are used. All MariaDB-specific table attributes are also not shown when a non-MariaDB/MySQL emulation mode is used, which includes [ANSI](#) , [DB2](#) , [POSTGRESQL](#) , [MSSQL](#) , [MAXDB](#)  or [ORACLE](#) .

Invalid table options, column options and index options are normally commented out (note, that it is possible to create a table with invalid options, by altering a table of a different engine, where these options were valid). To have them uncommented, enable the [IGNORE_BAD_TABLE_OPTIONS](#)  [SQL_MODE](#) . Remember that replaying a [CREATE TABLE](#) statement with uncommented invalid options will fail with an error, unless the [IGNORE_BAD_TABLE_OPTIONS](#)  [SQL_MODE](#)  is in effect.

Note that `SHOW CREATE TABLE` is not meant to provide metadata about a table. It provides information about how the table was declared, but the real table structure could differ a bit. For example, if an index has been declared as `HASH`, the `CREATE TABLE` statement returned by `SHOW CREATE TABLE` will declare that index as `HASH`; however, it is possible that the index is in fact a `BTREE`, because the storage engine does not support `HASH`.

MariaDB starting with 10.2.1

MariaDB 10.2.1  permits [TEXT](#)  and [BLOB](#)  data types to be assigned a [DEFAULT](#) value. As a result, from MariaDB 10.2.1 , `SHOW CREATE TABLE` will append a `DEFAULT NULL` to nullable `TEXT` or `BLOB` fields if no specific default is provided.

MariaDB starting with 10.2.2

From MariaDB 10.2.2 , numbers are no longer quoted in the `DEFAULT` clause in `SHOW CREATE` statement. Previously, MariaDB quoted numbers.

Examples

```
SHOW CREATE TABLE t\G
*****
* 1. row *****
      Table: t
Create Table: CREATE TABLE `t` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `s` char(60) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

With `sql_quote_show_create`  off:

```
SHOW CREATE TABLE t\G
***** 1. row *****
    Table: t
Create Table: CREATE TABLE t (
    id int(11) NOT NULL AUTO_INCREMENT,
    s char(60) DEFAULT NULL,
    PRIMARY KEY (id)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

Unquoted numeric DEFAULTs, from [MariaDB 10.2.2](#):

```
CREATE TABLE td (link TINYINT DEFAULT 1);

SHOW CREATE TABLE td\G
***** 1. row *****
    Table: td
Create Table: CREATE TABLE `td` (
    `link` tinyint(4) DEFAULT 1
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

Quoted numeric DEFAULTs, until [MariaDB 10.2.1](#):

```
CREATE TABLE td (link TINYINT DEFAULT 1);

SHOW CREATE TABLE td\G
***** 1. row *****
    Table: td
Create Table: CREATE TABLE `td` (
    `link` tinyint(4) DEFAULT '1'
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

[SQL_MODE](#) impacting the output:

```
SELECT @@sql_mode;
+-
| @@sql_mode
+-
| STRICT_TRANS_TABLES,ERROR_FOR_DIVISION_BY_ZERO,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION
+-

CREATE TABLE `t1` (
    `id` int(11) NOT NULL AUTO_INCREMENT,
    `msg` varchar(100) DEFAULT NULL,
    PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
;

SHOW CREATE TABLE t1\G
***** 1. row *****
    Table: t1
Create Table: CREATE TABLE `t1` (
    `id` int(11) NOT NULL AUTO_INCREMENT,
    `msg` varchar(100) DEFAULT NULL,
    PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1

SET SQL_MODE=ORACLE;

SHOW CREATE TABLE t1\G
***** 1. row *****
    Table: t1
Create Table: CREATE TABLE "t1" (
    "id" int(11) NOT NULL,
    "msg" varchar(100) DEFAULT NULL,
    PRIMARY KEY ("id")
```

See Also

- [SHOW CREATE SEQUENCE](#)
- [SHOW CREATE VIEW](#)

H3Dr1.2.1.18 SHOW INDEX

Syntax

```
SHOW {INDEX | INDEXES | KEYS}
  FROM tbl_name [FROM db_name]
  [WHERE expr]
```

Contents

1. [Syntax](#)
2. [Description](#)
3. [Examples](#)
4. [See Also](#)

Description

`SHOW INDEX` returns table index information. The format resembles that of the `SQLStatistics` call in ODBC.

You can use `db_name.tbl_name` as an alternative to the `tbl_name FROM db_name` syntax. These two statements are equivalent:

```
SHOW INDEX FROM mytable FROM mydb;
SHOW INDEX FROM mydb.mytable;
```

`SHOW KEYS` and `SHOW INDEXES` are synonyms for `SHOW INDEX`.

You can also list a table's indexes with the [mariadb-show/mysqlshow](#) command:

```
mysqlshow -k db_name tbl_name
```

The [information_schema.STATISTICS](#) table stores similar information.

The following fields are returned by `SHOW INDEX`.

Field	Description
Table	Table name
Non_unique	1 if the index permits duplicate values, 0 if values must be unique.
Key_name	Index name. The primary key is always named PRIMARY.
Seq_in_index	The column's sequence in the index, beginning with 1.
Column_name	Column name.
Collation	Either A, if the column is sorted in ascending order in the index, or NULL if it's not sorted.
Cardinality	Estimated number of unique values in the index. The cardinality statistics are calculated at various times, and can help the optimizer make improved decisions.
Sub_part	NULL if the entire column is included in the index, or the number of included characters if not.
Packed	NULL if the index is not packed, otherwise how the index is packed.
Null	NULL if NULL values are permitted in the column, an empty string if NULLs are not permitted.

Index_type	The index type, which can be BTREE , FULLTEXT , HASH or RTREE . See Storage Engine Index Types .
Comment	Other information, such as whether the index is disabled.
Index_comment	Contents of the COMMENT attribute when the index was created.
Ignored	Whether or not an index will be ignored by the optimizer. See Ignored Indexes . From MariaDB 10.6.0 .

The WHERE and LIKE clauses can be given to select rows using more general conditions, as discussed in [Extended SHOW](#).

Examples

```

CREATE TABLE IF NOT EXISTS `employees_example` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `first_name` varchar(30) NOT NULL,
  `last_name` varchar(40) NOT NULL,
  `position` varchar(25) NOT NULL,
  `home_address` varchar(50) NOT NULL,
  `home_phone` varchar(12) NOT NULL,
  `employee_code` varchar(25) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `employee_code` (`employee_code`),
  KEY `first_name` (`first_name`,`last_name`)
) ENGINE=Aria;

INSERT INTO `employees_example`(`first_name`, `last_name`, `position`, `home_address`,
`home_phone`, `employee_code`)
VALUES
('Mustapha', 'Mond', 'Chief Executive Officer', '692 Promiscuous Plaza', '326-555-3492',
'MM1'),
('Henry', 'Foster', 'Store Manager', '314 Savage Circle', '326-555-3847', 'HF1'),
('Bernard', 'Marx', 'Cashier', '1240 Ambient Avenue', '326-555-8456', 'BM1'),
('Lenina', 'Crowne', 'Cashier', '281 Bumblepuppy Boulevard', '328-555-2349', 'LC1'),
('Fanny', 'Crowne', 'Restocker', '1023 Bokanovsky Lane', '326-555-6329', 'FC1'),
('Helmholtz', 'Watson', 'Janitor', '944 Soma Court', '329-555-2478', 'HW1');

```

```

SHOW INDEXES FROM employees_example\G
*****
 1. row *****
      Table: employees_example
      Non_unique: 0
      Key_name: PRIMARY
      Seq_in_index: 1
      Column_name: id
      Collation: A
      Cardinality: 6
      Sub_part: NULL
      Packed: NULL
      Null:
      Index_type: BTREE
      Comment:
Index_comment:
      Ignored: NO
*****
 2. row *****
      Table: employees_example
      Non_unique: 0
      Key_name: employee_code
      Seq_in_index: 1
      Column_name: employee_code
      Collation: A
      Cardinality: 6
      Sub_part: NULL
      Packed: NULL
      Null:
      Index_type: BTREE
      Comment:
Index_comment:
      Ignored: NO
*****
 3. row *****
      Table: employees_example
      Non_unique: 1
      Key_name: first_name
      Seq_in_index: 1
      Column_name: first_name
      Collation: A
      Cardinality: NULL
      Sub_part: NULL
      Packed: NULL
      Null:
      Index_type: BTREE
      Comment:
Index_comment:
      Ignored: NO
*****
 4. row *****
      Table: employees_example
      Non_unique: 1
      Key_name: first_name
      Seq_in_index: 2
      Column_name: last_name
      Collation: A
      Cardinality: NULL
      Sub_part: NULL
      Packed: NULL
      Null:
      Index_type: BTREE
      Comment:
Index_comment:
      Ignored: NO

```

See Also

- [Ignored Indexes](#)

H3Dr1.2.1.19 TRUNCATE TABLE

Syntax

```
TRUNCATE [TABLE] tbl_name  
[WAIT n | NOWAIT]
```

Contents

1. [Syntax](#)
2. [Description](#)
 1. [WAIT/NOWAIT](#)
 2. [Oracle-mode](#)
 3. [Performance](#)
3. [See Also](#)

Description

`TRUNCATE TABLE` empties a table completely. It requires the `DROP` privilege. See [GRANT](#).

`tbl_name` can also be specified in the form `db_name . tbl_name` (see [Identifier Qualifiers](#)).

Logically, `TRUNCATE TABLE` is equivalent to a `DELETE` statement that deletes all rows, but there are practical differences under some circumstances.

`TRUNCATE TABLE` will fail for an [InnoDB table](#) if any FOREIGN KEY constraints from other tables reference the table, returning the error:

```
ERROR 1701 (42000): Cannot truncate a table referenced in a foreign key constraint
```

Foreign Key constraints between columns in the same table are permitted.

For an InnoDB table, if there are no FOREIGN KEY constraints, InnoDB performs fast truncation by dropping the original table and creating an empty one with the same definition, which is much faster than deleting rows one by one. The [AUTO_INCREMENT](#) counter is reset by `TRUNCATE TABLE`, regardless of whether there is a FOREIGN KEY constraint.

The count of rows affected by `TRUNCATE TABLE` is accurate only when it is mapped to a `DELETE` statement.

For other storage engines, `TRUNCATE TABLE` differs from `DELETE` in the following ways:

- Truncate operations drop and re-create the table, which is much faster than deleting rows one by one, particularly for large tables.
- Truncate operations cause an implicit commit.
- Truncation operations cannot be performed if the session holds an active table lock.
- Truncation operations do not return a meaningful value for the number of deleted rows. The usual result is "0 rows affected," which should be interpreted as "no information."
- As long as the table format file `tbl_name.frm` is valid, the table can be re-created as an empty table with `TRUNCATE TABLE`, even if the data or index files have become corrupted.
- The table handler does not remember the last used [AUTO_INCREMENT](#) value, but starts counting from the beginning. This is true even for MyISAM and InnoDB, which normally do not reuse sequence values.
- When used with partitioned tables, `TRUNCATE TABLE` preserves the partitioning; that is, the data and index files are dropped and re-created, while the partition definitions (.par) file is unaffected.
- Since truncation of a table does not make any use of `DELETE`, the `TRUNCATE` statement does not invoke `ON DELETE` triggers.
- `TRUNCATE TABLE` will only reset the values in the [Performance Schema summary tables](#) to zero or null, and will not remove the rows.

For the purposes of binary logging and [replication](#), `TRUNCATE TABLE` is treated as `DROP TABLE` followed by `CREATE TABLE` (DDL rather than DML).

`TRUNCATE TABLE` does not work on [views](#). Currently, `TRUNCATE TABLE` drops all historical records from a [system-versioned table](#).

MariaDB starting with [10.3.0](#)

WAIT/NOWAIT

Set the lock wait timeout. See [WAIT](#) and [NOWAIT](#).

Oracle-mode

[Oracle-mode](#) from [MariaDB 10.3](#) permits the optional keywords REUSE STORAGE or DROP STORAGE to be used.

```
TRUNCATE [TABLE] tbl_name [{DROP | REUSE} STORAGE] [WAIT n | NOWAIT]
```

These have no effect on the operation.

Performance

`TRUNCATE TABLE` is faster than [DELETE](#), because it drops and re-creates a table.

With [InnoDB](#), `TRUNCATE TABLE` is slower if `innodb_file_per_table=ON` is set (the default). This is because `TRUNCATE TABLE` unlinks the underlying tablespace file, which can be an expensive operation. See [MDEV-8069](#) for more details.

The performance issues with `innodb_file_per_table=ON` can be exacerbated in cases where the [InnoDB buffer pool](#) is very large and `innodb_adaptive_hash_index=ON` is set. In that case, using `DROP TABLE` followed by `CREATE TABLE` instead of `TRUNCATE TABLE` may perform better. Setting `innodb_adaptive_hash_index=OFF` (it defaults to ON before [MariaDB 10.5](#)) can also help. In [MariaDB 10.2](#) only, from [MariaDB 10.2.19](#), this performance can also be improved by setting `innodb_safe_truncate=OFF`. See [MDEV-9459](#) for more details.

Setting `innodb_adaptive_hash_index=OFF` can also improve `TRUNCATE TABLE` performance in general. See [MDEV-16796](#) for more details.

See Also

- [TRUNCATE function](#)
- [innodb_safe_truncate](#) system variable
- Oracle mode from [MariaDB 10.3](#)

H3Dr1.2.1.20 UPDATE

Syntax

Single-table syntax:

```
UPDATE [LOW_PRIORITY] [IGNORE] table_reference
[PARTITION (partition_list)]
[FOR PORTION OF period FROM expr1 TO expr2]
SET col1={expr1|DEFAULT} [,col2={expr2|DEFAULT}] ...
[WHERE where_condition]
[ORDER BY ...]
[LIMIT row_count]
```

Multiple-table syntax:

```
UPDATE [LOW_PRIORITY] [IGNORE] table_references
SET col1={expr1|DEFAULT} [, col2={expr2|DEFAULT}] ...
[WHERE where_condition]
```

Contents

1. [Syntax](#)
2. [Description](#)
 1. [PARTITION](#)
 2. [FOR PORTION OF](#)
 3. [UPDATE Statements With the Same Source and Target](#)
3. [Example](#)
4. [See Also](#)

Description

For the single-table syntax, the `UPDATE` statement updates columns of existing rows in the named table with new values. The `SET` clause indicates which columns to modify and the values they should be given. Each value can be given as an expression, or the keyword `DEFAULT` to set a column explicitly to its default value. The `WHERE` clause, if given, specifies the conditions that identify which rows to update. With no `WHERE` clause, all rows are updated. If the `ORDER BY` clause is specified, the rows are updated in the order that is specified. The `LIMIT` clause places a limit on the number of rows that can be updated.

Until MariaDB 10.3.2, for the multiple-table syntax, `UPDATE` updates rows in each table named in `table_references` that satisfy the conditions. In this case, `ORDER BY` and `LIMIT` cannot be used. This restriction was lifted in MariaDB 10.3.2 and both clauses can be used with multiple-table updates. An `UPDATE` can also reference tables which are located in different databases; see [Identifier Qualifiers](#) for the syntax.

`where_condition` is an expression that evaluates to true for each row to be updated.

`table_references` and `where_condition` are as specified as described in [SELECT](#).

For single-table updates, assignments are evaluated in left-to-right order, while for multi-table updates, there is no guarantee of a particular order. If the `SIMULTANEOUS_ASSIGNMENT` `sql_mode` (available from MariaDB 10.3.5) is set, `UPDATE` statements evaluate all assignments simultaneously.

You need the `UPDATE` privilege only for columns referenced in an `UPDATE` that are actually updated. You need only the `SELECT` privilege for any columns that are read but not modified. See [GRANT](#).

The `UPDATE` statement supports the following modifiers:

- If you use the `LOW_PRIORITY` keyword, execution of the `UPDATE` is delayed until no other clients are reading from the table. This affects only storage engines that use only table-level locking (MyISAM, MEMORY, MERGE). See [HIGH_PRIORITY and LOW_PRIORITY clauses](#) for details.
- If you use the `IGNORE` keyword, the update statement does not abort even if errors occur during the update. Rows for which duplicate-key conflicts occur are not updated. Rows for which columns are updated to values that would cause data conversion errors are updated to the closest valid values instead.

PARTITION

See [Partition Pruning and Selection](#) for details.

FOR PORTION OF

MariaDB starting with 10.4.3

See [Application Time Periods - Updating by Portion](#).

UPDATE Statements With the Same Source and Target

MariaDB starting with 10.3.2

From MariaDB 10.3.2, UPDATE statements may have the same source and target.

For example, given the following table:

```
DROP TABLE t1;
CREATE TABLE t1 (c1 INT, c2 INT);
INSERT INTO t1 VALUES (10,10), (20,20);
```

Until MariaDB 10.3.1, the following UPDATE statement would not work:

```
UPDATE t1 SET c1=c1+1 WHERE c2=(SELECT MAX(c2) FROM t1);
ERROR 1093 (HY000): Table 't1' is specified twice,
both as a target for 'UPDATE' and as a separate source for data
```

From MariaDB 10.3.2, the statement executes successfully:

```
UPDATE t1 SET c1=c1+1 WHERE c2=(SELECT MAX(c2) FROM t1);

SELECT * FROM t1;
+-----+-----+
| c1   | c2    |
+-----+-----+
| 10   | 10   |
| 21   | 20   |
+-----+-----+
```

Example

Single-table syntax:

```
UPDATE table_name SET column1 = value1, column2 = value2 WHERE id=100;
```

Multiple-table syntax:

```
UPDATE tab1, tab2 SET tab1.column1 = value1, tab1.column2 = value2 WHERE tab1.id = tab2.id;
```

See Also

- [How IGNORE works](#)
- [SELECT](#)
- [ORDER BY](#)
- [LIMIT](#)
- [Identifier Qualifiers](#)

H3Dr1.2.1.21 IGNORE

The `IGNORE` option tells the server to ignore some common errors.

`IGNORE` can be used with the following statements:

- [DELETE](#)
- [INSERT](#) (see also [INSERT IGNORE](#))
- [LOAD DATA INFILE](#)
- [UPDATE](#)
- [ALTER TABLE](#)
- [CREATE TABLE ... SELECT](#)
- [INSERT ... SELECT](#)

The logic used:

- Variables out of ranges are replaced with the maximum/minimum value.
- [SQL_MODEs](#) `STRICT_TRANS_TABLES`, `STRICT_ALL_TABLES`, `NO_ZERO_IN_DATE`, `NO_ZERO_DATE` are ignored.
- Inserting `NULL` in a `NOT NULL` field will insert 0 (in a numerical field), 0000-00-00 (in a date field) or an empty string (in a character field).

- Rows that cause a duplicate key error or break a foreign key constraint are not inserted, updated, or deleted.

The following errors are ignored:

Error number	Symbolic error name	Description
1022	ER_DUP_KEY	Can't write; duplicate key in table '%s'
1048	ER_BAD_NULL_ERROR	Column '%s' cannot be null
1062	ER_DUP_ENTRY	Duplicate entry '%s' for key %d
1242	ER_SUBQUERY_NO_1_ROW	Subquery returns more than 1 row
1264	ER_WARN_DATA_OUT_OF_RANGE	Out of range value for column '%s' at row %ld
1265	WARN_DATA_TRUNCATED	Data truncated for column '%s' at row %ld
1292	ER_TRUNCATED_WRONG_VALUE	Truncated incorrect %s value: '%s'
1366	ER_TRUNCATED_WRONG_VALUE_FOR_FIELD	Incorrect integer value
1369	ER_VIEW_CHECK_FAILED	CHECK OPTION failed '%s.%s'
1451	ER_ROW_IS_REFERENCED_2	Cannot delete or update a parent row
1452	ER_NO_REFERENCED_ROW_2	Cannot add or update a child row: a foreign key constraint fails (%s)
1526	ER_NO_PARTITION_FOR_GIVEN_VALUE	Table has no partition for value %s
1586	ER_DUP_ENTRY_WITH_KEY_NAME	Duplicate entry '%s' for key '%s'
1591	ER_NO_PARTITION_FOR_GIVEN_VALUE_SILENT	Table has no partition for some existing values
1748	ER_ROW_DOES_NOT_MATCH_GIVEN_PARTITION_SET	Found a row not matching the given partition set

Ignored errors normally generate a warning.

A property of the `IGNORE` clause consists in causing transactional engines and non-transactional engines (like XtraDB and Aria) to behave the same way. For example, normally a multi-row insert which tries to violate a `UNIQUE` constraint is completely rolled back on XtraDB/InnoDB, but might be partially executed on Aria. With the `IGNORE` clause, the statement will be partially executed in both engines.

Duplicate key errors also generate warnings. The `OLD_MODE` server variable can be used to prevent this.

H3Dr1.2.1.22 System-Versioned Tables

H3Dr1.2.2 ANALYZE and EXPLAIN Statements



ANALYZE FORMAT=JSON

Mix of the EXPLAIN FORMAT=JSON and ANALYZE statement features.



ANALYZE FORMAT=JSON Examples

Examples with ANALYZE FORMAT=JSON.



ANALYZE Statement

Invokes the optimizer, executes the statement, and then produces EXPLAIN output.



EXPLAIN

EXPLAIN returns information about index usage, as well as being a synonym for DESCRIBE.



EXPLAIN ANALYZE

Old implementation, now ANALYZE statement



EXPLAIN FORMAT=JSON

Variant of EXPLAIN that produces output in JSON form



SHOW EXPLAIN

Shows an execution plan for a running query.



Using Buffer UPDATE Algorithm

Explanation of UPDATE's "Using Buffer" algorithm.

H3Dr1.2.2.1 ANALYZE FORMAT=JSON

Contents

1. [Basic Execution Data](#)
2. [Advanced Execution Data](#)
3. [Data About Individual Query Plan Nodes](#)
4. [Use Cases](#)

`ANALYZE FORMAT=JSON` is a mix of the `EXPLAIN FORMAT=JSON` and `ANALYZE` statement features. The `ANALYZE FORMAT=JSON $statement` will execute `$statement`, and then print the output of `EXPLAIN FORMAT=JSON`, amended with data from the query execution.

Basic Execution Data

You can get the following also from tabular `ANALYZE` statement form:

- `r_rows` is provided for any node that reads rows. It shows how many rows were read, on average
- `r_filtered` is provided whenever there is a condition that is checked. It shows the percentage of rows left after checking the condition.

Advanced Execution Data

The most important data not available in the regular tabular `ANALYZE` statement are:

- `r_loops` field. This shows how many times the node was executed. Most query plan elements have this field.
- `r_total_time_ms` field. It shows how much time in total was spent executing this node. If the node has subnodes, their execution time is included.
- `r_buffer_size` field. Query plan nodes that make use of buffers report the size of buffer that was used.

Data About Individual Query Plan Nodes

- `filesort` node reports whether sorting was done with `LIMIT n` parameter, and how many rows were in the sort result.
- `block-nl-join` node has `r_loops` field, which allows to tell whether `Using join buffer` was efficient
- `range-checked-for-each-record` reports counters that show the result of the check.
- `expression-cache` is used for subqueries, and it reports how many times the cache was used, and what cache hit ratio was.
- `union_result` node has `r_rows` so one can see how many rows were produced after UNION operation
- and so forth

Use Cases

See [Examples of ANALYZE FORMAT=JSON](#).

H3Dr1.2.2.2 ANALYZE FORMAT=JSON Examples

Example #1

Customers who have ordered more than 1M goods.

```
ANALYZE FORMAT=JSON
SELECT COUNT(*)
FROM customer
WHERE
    (SELECT SUM(o_totalprice) FROM orders WHERE o_custkey=c_custkey) > 1000*1000;
```

The query takes 40 seconds over cold cache

```

EXPLAIN: {
  "query_block": {
    "select_id": 1,
    "r_loops": 1,
    "r_total_time_ms": 39872,
    "table": {
      "table_name": "customer",
      "access_type": "index",
      "key": "i_c_nationkey",
      "key_length": "5",
      "used_key_parts": ["c_nationkey"],
      "r_loops": 1,
      "rows": 150303,
      "r_rows": 150000,
      "r_total_time_ms": 270.3,
      "filtered": 100,
      "r_filtered": 60.691,
      "attached_condition": "((subquery#2) > <cache>((1000 * 1000)))",
      "using_index": true
    },
    "subqueries": [
      {
        "query_block": {
          "select_id": 2,
          "r_loops": 150000,
          "r_total_time_ms": 39531,
          "table": {
            "table_name": "orders",
            "access_type": "ref",
            "possible_keys": ["i_o_custkey"],
            "key": "i_o_custkey",
            "key_length": "5",
            "used_key_parts": ["o_custkey"],
            "ref": ["dbt3sf1.customer.c_custkey"],
            "r_loops": 150000,
            "rows": 7,
            "r_rows": 10,
            "r_total_time_ms": 39208,
            "filtered": 100,
            "r_filtered": 100
          }
        }
      }
    ]
  }
}

```

`ANALYZE` shows that 39.2 seconds were spent in the subquery, which was executed 150K times (for every row of outer table).

H3Dr1.2.2.3 ANALYZE Statement

Contents

1. [Description](#)
2. [Command Output](#)
3. [Interpreting the Output](#)
 1. [Joins](#)
 2. [Meaning of NULL in r_rows and r_filtered](#)
4. [ANALYZE FORMAT=JSON](#)
5. [Notes](#)
6. [See Also](#)

Description

The `ANALYZE` statement is similar to the `EXPLAIN` statement. `ANALYZE` statement will invoke the optimizer, execute the statement, and then produce `EXPLAIN` output instead of the result set. The `EXPLAIN` output will be annotated with statistics from statement execution.

This lets one check how close the optimizer's estimates about the query plan are to the reality. `ANALYZE` produces an overview, while the `ANALYZE FORMAT=JSON` command provides a more detailed view of the query plan and the query execution.

The syntax is

```
ANALYZE explainable_statement;
```

where the statement is any statement for which one can run `EXPLAIN`.

Command Output

Consider an example:

```
ANALYZE SELECT * FROM tbl1
WHERE key1
    BETWEEN 10 AND 200 AND
    col1 LIKE 'foo%'\G
```

```
***** 1. row *****
    id: 1
    select_type: SIMPLE
        table: tbl1
        type: range
possible_keys: key1
    key: key1
    key_len: 5
    ref: NULL
    rows: 181
    r_rows: 181
    filtered: 100.00
    r_filtered: 10.50
    Extra: Using index condition; Using where
```

Compared to `EXPLAIN`, `ANALYZE` produces two extra columns:

- `r_rows` is an observation-based counterpart of the `rows` column. It shows how many rows were actually read from the table.
- `r_filtered` is an observation-based counterpart of the `filtered` column. It shows which fraction of rows was left after applying the WHERE condition.

Interpreting the Output

Joins

Let's consider a more complicated example.

```
ANALYZE SELECT *
FROM orders, customer
WHERE
    customer.c_custkey=orders.o_custkey AND
    customer.c_acctbal < 0 AND
    orders.o_totalprice > 200*1000
```

+-----+-----+-----+-----+-----+-----+	id select_type table type possible_keys key key_len ref	+-----+-----+-----+-----+-----+-----+
-+-----+-----+-----+-----+-----+-----+	rows r_rows filtered r_filtered Extra	-+-----+-----+-----+-----+-----+
1 SIMPLE customer ALL PRIMARY,... NULL NULL NULL	+-----+-----+-----+-----+-----+-----+	
149095 150000 18.08 9.13 Using where	+-----+-----+-----+-----+-----+-----+	
1 SIMPLE orders ref i_o_custkey i_o_custkey 5	+-----+-----+-----+-----+-----+-----+	
customer.c_custkey 7 10 100.00 30.03 Using where	+-----+-----+-----+-----+-----+-----+	

Here, one can see that

- For table customer, **customer.rows=149095, customer.r_rows=150000**. The estimate for number of rows we will read was fairly precise
- **customer.filtered=18.08, customer.r_filtered=9.13**. The optimizer somewhat overestimated the number of records that will match selectivity of condition attached to `customer` table (in general, when you have a full scan and r_filtered is less than 15%, it's time to consider adding an appropriate index).
- For table orders, **orders.rows=7, orders.r_rows=10**. This means that on average, there are 7 orders for a given c_custkey, but in our case there were 10, which is close to the expectation (when this number is consistently far from the expectation, it may be time to run ANALYZE TABLE, or even edit the table statistics manually to get better query plans).
- **orders.filtered=100, orders.r_filtered=30.03**. The optimizer didn't have any way to estimate which fraction of records will be left after it checks the condition that is attached to table orders (it's orders.o_totalprice > 200*1000). So, it used 100%. In reality, it is 30%. 30% is typically not selective enough to warrant adding new indexes. For joins with many tables, it might be worth to collect and use [column statistics](#) for columns in question, this may help the optimizer to pick a better query plan.

Meaning of NULL in r_rows and r_filtered

Let's modify the previous example slightly

```
ANALYZE SELECT *
FROM orders, customer
WHERE
    customer.c_custkey=orders.o_custkey AND
    customer.c_acctbal < -0 AND
    customer.c_comment LIKE '%foo%' AND
    orders.o_totalprice > 200*1000;
```

+-----+-----+-----+-----+-----+-----+	id select_type table type possible_keys key key_len ref	+-----+-----+-----+-----+-----+-----+
-+-----+-----+-----+-----+-----+-----+	rows r_rows filtered r_filtered Extra	-+-----+-----+-----+-----+-----+
1 SIMPLE customer ALL PRIMARY,... NULL NULL NULL	+-----+-----+-----+-----+-----+-----+	
149095 150000 18.08 0.00 Using where	+-----+-----+-----+-----+-----+-----+	
1 SIMPLE orders ref i_o_custkey i_o_custkey 5	+-----+-----+-----+-----+-----+-----+	
customer.c_custkey 7 NULL 100.00 NULL Using where	+-----+-----+-----+-----+-----+-----+	

Here, one can see that `orders.r_rows=NULL` and `orders.r_filtered=NULL`. This means that table `orders` was not scanned even once. Indeed, we can also see `customer.r_filtered=0.00`. This shows that a part of WHERE attached to table `customer` was never satisfied (or, satisfied in less than 0.01% of cases).

ANALYZE FORMAT=JSON

`ANALYZE FORMAT=JSON` produces JSON output. It produces much more information than tabular `ANALYZE`.

Notes

- `ANALYZE UPDATE` or `ANALYZE DELETE` will actually make updates/deletes (`ANALYZE SELECT` will perform the select operation and then discard the resultset).
- PostgreSQL has a similar command, `EXPLAIN ANALYZE`.
- The [EXPLAIN in the slow query log](#) feature allows MariaDB to have `ANALYZE` output of slow queries printed into the [slow query log](#) (see [MDEV-6388](#)).

See Also

- [ANALYZE FORMAT=JSON](#)
- [ANALYZE TABLE](#)
- JIRA task for `ANALYZE` statement, [MDEV-406](#)

H3Dr1.2.2.4 EXPLAIN

Syntax

```
EXPLAIN tbl_name [col_name | wild]
```

Or

```
EXPLAIN [EXTENDED | PARTITIONS | FORMAT=JSON]
{SELECT select_options | UPDATE update_options | DELETE delete_options}
```

Contents

1. [Syntax](#)
2. [Description](#)
 1. [Columns in EXPLAIN ... SELECT](#)
 1. "Select_type" Column
 2. "Type" Column
 3. "Extra" Column
 2. [EXPLAIN EXTENDED](#)
3. [Examples](#)
 1. [Example of ref_or_null Optimization](#)
4. [See Also](#)

Description

The `EXPLAIN` statement can be used either as a synonym for `DESCRIBE` or as a way to obtain information about how MariaDB executes a `SELECT`, `UPDATE` or `DELETE` statement:

- '`EXPLAIN tbl_name`' is synonymous with '[DESCRIBE](#) `tbl_name`' or '`SHOW COLUMNS FROM` `tbl_name`'.
- When you precede a `SELECT`, `UPDATE` or a `DELETE` statement with the keyword `EXPLAIN`, MariaDB displays information from the optimizer about the query execution plan. That is, MariaDB explains how it would process the `SELECT`, `UPDATE` or `DELETE`, including information about how

tables are joined and in which order. EXPLAIN EXTENDED can be used to provide additional information.

- EXPLAIN PARTITIONS is useful only when examining queries involving partitioned tables. For details, see [Partition pruning and selection](#).
- ANALYZE statement performs the query as well as producing EXPLAIN output, and provides actual as well as estimated statistics.
- EXPLAIN output can be printed in the slow query log. See [EXPLAIN in the Slow Query Log](#) for details.

SHOW EXPLAIN shows the output of a running statement. In some cases, its output can be closer to reality than EXPLAIN.

The ANALYZE statement runs a statement and returns information about its execution plan. It also shows additional columns, to check how much the optimizer's estimation about filtering and found rows are close to reality.

There is an online [EXPLAIN Analyzer](#) that you can use to share EXPLAIN and EXPLAIN EXTENDED output with others.

EXPLAIN can acquire metadata locks in the same way that SELECT does, as it needs to know table metadata and, sometimes, data as well.

Columns in EXPLAIN ... SELECT

Column name	Description
id	Sequence number that shows in which order tables are joined.
select_type	What kind of SELECT the table comes from.
table	Alias name of table. Materialized temporary tables for sub queries are named <subquery#>
type	How rows are found from the table (join type).
possible_keys	keys in table that could be used to find rows in the table
key	The name of the key that is used to retrieve rows. NULL is no key was used.
key_len	How many bytes of the key that was used (shows if we are using only parts of the multi-column key).
ref	The reference that is used as the key value.
rows	An estimate of how many rows we will find in the table for each key lookup.
Extra	Extra information about this join.

Here are descriptions of the values for some of the more complex columns in EXPLAIN ... SELECT :

"Select_type" Column

The select_type column can have the following values:

Value	Description	Comment
DEPENDENT SUBQUERY	The SUBQUERY is DEPENDENT .	
DEPENDENT UNION	The UNION is DEPENDENT .	
DERIVED	The SELECT is DERIVED from the PRIMARY .	
MATERIALIZED	The SUBQUERY is MATERIALIZED .	Materialized tables will be populated at first access and will be accessed by the primary key (= one key lookup). Number of rows in EXPLAIN shows the cost of populating the table

PRIMARY	The SELECT is a PRIMARY one.	
SIMPLE	The SELECT is a SIMPLE one.	
SUBQUERY	The SELECT is a SUBQUERY of the PRIMARY .	
UNCACHEABLE SUBQUERY	The SUBQUERY is UNCACHEABLE .	
UNCACHEABLE UNION	The UNION is UNCACHEABLE .	
UNION	The SELECT is a UNION of the PRIMARY .	
UNION RESULT	The result of the UNION .	
LATERAL DERIVED	The SELECT uses a Lateral Derived optimization	

"Type" Column

This column contains information on how the table is accessed.

Value	Description
ALL	A full table scan is done for the table (all rows are read). This is bad if the table is large and the table is joined against a previous table! This happens when the optimizer could not find any usable index to access rows.
const	There is only one possibly matching row in the table. The row is read before the optimization phase and all columns in the table are treated as constants.
eq_ref	A unique index is used to find the rows. This is the best possible plan to find the row.
fulltext	A fulltext index is used to access the rows.
index_merge	A 'range' access is done for several index and the found rows are merged. The key column shows which keys are used.
index_subquery	This is similar as ref, but used for sub queries that are transformed to key lookups.
index	A full scan over the used index. Better than ALL but still bad if index is large and the table is joined against a previous table.
range	The table will be accessed with a key over one or more value ranges.
ref_or_null	Like 'ref' but in addition another search for the 'null' value is done if the first value was not found. This happens usually with sub queries.
ref	A non unique index or prefix of an unique index is used to find the rows. Good if the prefix doesn't match many rows.
system	The table has 0 or 1 rows.
unique_subquery	This is similar as eq_ref, but used for sub queries that are transformed to key lookups

"Extra" Column

This column consists of one or more of the following values, separated by ';'.

Note that some of these values are detected after the optimization phase.

The optimization phase can do the following changes to the WHERE clause:

- Add the expressions from the ON and USING clauses to the WHERE clause.
- Constant propagation: If there is column=constant , replace all column instances with this constant.
- Replace all columns from 'const' tables with their values.
- Remove the used key columns from the WHERE (as this will be tested as part of the key lookup).
- Remove impossible constant sub expressions. For example WHERE '(a=1 and a=2) OR b=1' becomes 'b=1' .
- Replace columns with other columns that has identical values: Example: WHERE a=b and a=c may be treated as 'WHERE a=b and a=c and b=c' .
- Add extra conditions to detect impossible row conditions earlier. This happens mainly with OUTER JOIN where we in some cases add detection of NULL values in the WHERE (Part of 'Not exists' optimization). This can cause an unexpected 'Using where' in the Extra column.
- For each table level we remove expressions that have already been tested when we read the previous row. Example: When joining tables t1 with t2 using the following WHERE 't1.a=1 and t1.a=t2.b' , we don't have to test 't1.a=1' when checking rows in t2 as we already know that this expression is true.

Value	Description
const row not found	The table was a system table (a table with should exactly one row), but no row was found.
Distinct	If distinct optimization (remove duplicates) was used. This is marked only for the last table in the SELECT .
Full scan on NULL key	The table is a part of the sub query and if the value that is used to match the sub query will be NULL , we will do a full table scan.
Impossible HAVING	The used HAVING clause is always false so the SELECT will return no rows.
Impossible WHERE noticed after reading const tables.	The used WHERE clause is always false so the SELECT will return no rows. This case was detected after we had read all 'const' tables and used the column values as constant in the WHERE clause. For example: WHERE const_column=5 and const_column had a value of 4.
Impossible WHERE	The used WHERE clause is always false so the SELECT will return no rows. For example: WHERE 1=2
No matching min/max row	During early optimization of MIN() / MAX() values it was detected that no row could match the WHERE clause. The MIN() / MAX() function will return NULL .
no matching row in const table	The table was a const table (a table with only one possible matching row), but no row was found.
No tables used	The SELECT was a sub query that did not use any tables. For example a there was no FROM clause or a FROM DUAL clause.
Not exists	Stop searching after more row if we find one single matching row. This optimization is used with LEFT JOIN where one is explicitly searching for rows that doesn't exists in the LEFT JOIN TABLE . Example: SELECT * FROM t1 LEFT JOIN t2 on (...) WHERE t2.not_null_column IS NULL . As t2.not_null_column can only be NULL if there was no matching row for on condition, we can stop searching if we find a single matching row.
Open_frm_only	For information_schema tables. Only the frm (table definition file was opened) was opened for each matching row.
Open_full_table	For information_schema tables. A full table open for each matching row is done to retrieve the requested information. (Slow)
Open_trigger_only	For information_schema tables. Only the trigger file definition was opened for each matching row.

Range checked for each record (index map: ...)	This only happens when there was no good default index to use but there may some index that could be used when we can treat all columns from previous table as constants. For each row combination the optimizer will decide which index to use (if any) to fetch a row from this table. This is not fast, but faster than a full table scan that is the only other choice. The index map is a bitmask that shows which index are considered for each row condition.
Scanned 0/1/all databases	For <code>information_schema</code> tables. Shows how many times we had to do a directory scan.
Select tables optimized away	All tables in the join was optimized away. This happens when we are only using <code>COUNT(*)</code> , <code>MIN()</code> and <code>MAX()</code> functions in the <code>SELECT</code> and we where able to replace all of these with constants.
Skip_open_table	For <code>information_schema</code> tables. The queried table didn't need to be opened.
unique row not found	The table was detected to be a const table (a table with only one possible matching row) during the early optimization phase, but no row was found.
Using filesort	Filesort is needed to resolve the query. This means an extra phase where we first collect all columns to sort, sort them with a disk based merge sort and then use the sorted set to retrieve the rows in sorted order. If the column set is small, we store all the columns in the sort file to not have to go to the database to retrieve them again.
Using index	Only the index is used to retrieve the needed information from the table. There is no need to perform an extra seek to retrieve the actual record.
Using index condition	Like 'Using where' but the where condition is pushed down to the table engine for internal optimization at the index level.
Using index condition(BKA)	Like 'Using index condition' but in addition we use batch key access to retrieve rows.
Using index for group-by	The index is being used to resolve a <code>GROUP BY</code> or <code>DISTINCT</code> query. The rows are not read. This is very efficient if the table has a lot of identical index entries as duplicates are quickly jumped over.
Using intersect(...)	For <code>index_merge</code> joins. Shows which index are part of the intersect.
Using join buffer	We store previous row combinations in a row buffer to be able to match each row against all of the rows combinations in the join buffer at one go.
Using sort_union(...)	For <code>index_merge</code> joins. Shows which index are part of the union.
Using temporary	A temporary table is created to hold the result. This typically happens if you are using <code>GROUP BY</code> , <code>DISTINCT</code> or <code>ORDER BY</code> .
Using where	A <code>WHERE</code> expression (in additional to the possible key lookup) is used to check if the row should be accepted. If you don't have 'Using where' together with a join type of <code>ALL</code> , you are probably doing something wrong!
Using where with pushed condition	Like 'Using where' but the where condition is pushed down to the table engine for internal optimization at the row level.
Using buffer	The <code>UPDATE</code> statement will first buffer the rows, and then run the updates, rather than do updates on the fly. See Using Buffer UPDATE Algorithm for a detailed explanation.

EXPLAIN EXTENDED

The `EXTENDED` keyword adds another column, *filtered*, to the output. This is a percentage estimate of the table rows that will be filtered by the condition.

An `EXPLAIN EXTENDED` will always throw a warning, as it adds extra *Message* information to a subsequent `SHOW WARNINGS` statement. This includes what the `SELECT` query would look like after optimizing and rewriting rules are applied and how the optimizer qualifies columns and tables.

Examples

As synonym for DESCRIBE or SHOW COLUMNS FROM :

```
DESCRIBE city;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| Id    | int(11) | NO | PRI | NULL | auto_increment |
| Name  | char(35) | YES |     | NULL |           |
| Country | char(3) | NO | UNI |       |           |
| District | char(20) | YES | MUL |       |           |
| Population | int(11) | YES |     | NULL |           |
+-----+-----+-----+-----+-----+
```

A simple set of examples to see how EXPLAIN can identify poor index usage:

```
CREATE TABLE IF NOT EXISTS `employees_example` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `first_name` varchar(30) NOT NULL,
  `last_name` varchar(40) NOT NULL,
  `position` varchar(25) NOT NULL,
  `home_address` varchar(50) NOT NULL,
  `home_phone` varchar(12) NOT NULL,
  `employee_code` varchar(25) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `employee_code` (`employee_code`),
  KEY `first_name` (`first_name`,`last_name`)
) ENGINE=Aria;

INSERT INTO `employees_example` (`first_name`, `last_name`, `position`, `home_address`,
`home_phone`, `employee_code`)
VALUES
  ('Mustapha', 'Mond', 'Chief Executive Officer', '692 Promiscuous Plaza', '326-555-3492',
'MM1'),
  ('Henry', 'Foster', 'Store Manager', '314 Savage Circle', '326-555-3847', 'HF1'),
  ('Bernard', 'Marx', 'Cashier', '1240 Ambient Avenue', '326-555-8456', 'BM1'),
  ('Lenina', 'Crowne', 'Cashier', '281 Bumblepuppy Boulevard', '328-555-2349', 'LC1'),
  ('Fanny', 'Crowne', 'Restocker', '1023 Bokanovsky Lane', '326-555-6329', 'FC1'),
  ('Helmholtz', 'Watson', 'Janitor', '944 Soma Court', '329-555-2478', 'HW1');

SHOW INDEXES FROM employees_example;
+-----+-----+-----+-----+-----+
| Table          | Non_unique | Key_name      | Seq_in_index | Column_name   | | | |
| Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment |
|               |             |            |            |           |             |         |                 |
+-----+-----+-----+-----+-----+
| employees_example |      0 | PRIMARY      |          1 | id          | A
| employees_example |      7 | NULL | NULL | BTREE |           |
| employees_example |      0 | employee_code |          1 | employee_code | A
| employees_example |      7 | NULL | NULL | BTREE |           |
| employees_example |      1 | first_name   |          1 | first_name  | A
|               | NULL | NULL | NULL | BTREE |           |
| employees_example |      1 | first_name   |          2 | last_name   | A
|               | NULL | NULL | NULL | BTREE |           |
+-----+-----+-----+-----+-----+
```

SELECT on a primary key:

```

EXPLAIN SELECT * FROM employees_example WHERE id=1;
+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref |
| rows | Extra |
+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | employees_example | const | PRIMARY | PRIMARY | 4 |
const | 1 |
+-----+-----+-----+-----+-----+-----+-----+

```

The type is *const*, which means that only one possible result could be returned. Now, returning the same record but searching by their phone number:

```

EXPLAIN SELECT * FROM employees_example WHERE home_phone='326-555-3492';
+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref |
| rows | Extra |
+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | employees_example | ALL | NULL | NULL | NULL | NULL |
6 | Using where |
+-----+-----+-----+-----+-----+-----+-----+

```

Here, the type is *All*, which means no index could be used. Looking at the rows count, a full table scan (all six rows) had to be performed in order to retrieve the record. If it's a requirement to search by phone number, an index will have to be created.

[SHOW EXPLAIN](#) example:

```

SHOW EXPLAIN FOR 1;
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | tbl | index | NULL | a | 5 | NULL | 1000107 | Using index |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.00 sec)

```

Example of ref_or_null Optimization

```

SELECT * FROM table_name
WHERE key_column=expr OR key_column IS NULL;

```

`ref_or_null` is something that often happens when you use subqueries with `NOT IN` as then one has to do an extra check for `NULL` values if the first value didn't have a matching row.

See Also

- [SHOW EXPLAIN](#)
- [Ignored Indexes ↗](#)

H3Dr1.2.2.5 EXPLAIN ANALYZE

The syntax for the `EXPLAIN ANALYZE` feature was changed to `ANALYZE statement`, available since MariaDB 10.1.0 ↗. See [ANALYZE statement](#).

H3Dr1.2.2.6 EXPLAIN FORMAT=JSON

Contents

1. [Synopsis](#)
2. [Output is different from MySQL](#)
3. [Output Format](#)
4. [See Also](#)

Synopsis

`EXPLAIN FORMAT=JSON` is a variant of `EXPLAIN` command that produces output in JSON form. The output always has one row which has only one column titled " JSON ". The contents are a JSON representation of the query plan, formatted for readability:

```
EXPLAIN FORMAT=JSON SELECT * FROM t1 WHERE col1=1\G
```

```
***** 1. row *****  
EXPLAIN: {  
  "query_block": {  
    "select_id": 1,  
    "table": {  
      "table_name": "t1",  
      "access_type": "ALL",  
      "rows": 1000,  
      "filtered": 100,  
      "attached_condition": "(t1.col1 = 1)"  
    }  
  }  
}
```

Output is different from MySQL

The output of MariaDB's `EXPLAIN FORMAT=JSON` is different from `EXPLAIN FORMAT=JSON` in MySQL. The reasons for that are:

- MySQL's output has deficiencies. Some are listed here: [EXPLAIN FORMAT=JSON in MySQL](#).
- The output of MySQL's `EXPLAIN FORMAT=JSON` is not defined. Even MySQL Workbench has trouble parsing it (see this [blog post](#)).
- MariaDB has query optimizations that MySQL does not have. Ergo, MariaDB generates query plans that MySQL does not generate.

A (as yet incomplete) list of how MariaDB's output is different from MySQL can be found here: [EXPLAIN FORMAT=JSON differences from MySQL](#).

Output Format

TODO: MariaDB's output format description.

See Also

- `ANALYZE FORMAT=JSON` produces output like `EXPLAIN FORMAT=JSON`, but amended with the data from query execution.

H3Dr1.2.2.7 SHOW EXPLAIN

Contents

1. [Syntax](#)
2. [Description](#)
 1. [Possible Errors](#)
 2. [Differences Between SHOW EXPLAIN and EXPLAIN Outputs](#)
 1. [Background](#)
 2. [List of Recorded Differences](#)
 3. [Required Permissions](#)
 3. [See Also](#)

Syntax

```
SHOW EXPLAIN FOR <thread_id>;
```

Description

The `SHOW EXPLAIN` command allows one to get an [EXPLAIN](#) (that is, a description of a query plan) of a query running in a certain thread.

```
SHOW EXPLAIN FOR <thread_id>;
```

will produce an `EXPLAIN` output for the query that thread number `thread_id` is running. The thread id can be obtained with [SHOW PROCESSLIST](#).

```
SHOW EXPLAIN FOR 1;
+-----+-----+-----+-----+-----+-----+-----+-----+
| id   | select_type | table | type   | possible_keys | key    | key_len | ref    | rows   | Ext
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1    | SIMPLE      | tbl   | index  | NULL        | a      | 5       | NULL   | 1000107 | Using
+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.00 sec)
```

The output is always accompanied with a warning which shows the query the target thread is running (this shows what the `EXPLAIN` is for):

```
SHOW WARNINGS;
+-----+
| Level | Code | Message          |
+-----+
| Note  | 1003 | select sum(a) from tbl |
+-----+
1 row in set (0.00 sec)
```

Possible Errors

The output can be only produced if the target thread is *currently* running a query, which has a ready query plan. If this is not the case, the output will be:

```
SHOW EXPLAIN FOR 2;
ERROR 1932 (HY000): Target is not running an EXPLAINable command
```

You will get this error when:

- the target thread is not running a command for which one can run `EXPLAIN`
- the target thread is running a command for which one can run `EXPLAIN`, but
 - there is no query plan yet (for example, tables are open and locks are acquired before the query plan is produced)

Differences Between SHOW EXPLAIN and EXPLAIN Outputs

Background

In MySQL, `EXPLAIN` execution takes a slightly different route from the way the real query (typically the `SELECT`) is optimized. This is unfortunate, and has caused a number of bugs in `EXPLAIN`. (For example, see [MDEV-326](#), [MDEV-410](#), and [Ip:1013343](#). [Ip:992942](#) is not directly about `EXPLAIN`, but it also would not have existed if MySQL didn't try to delete parts of a query plan in the middle of the query)

`SHOW EXPLAIN` examines a running `SELECT`, and hence its output may be slightly different from what `EXPLAIN SELECT` would produce. We did our best to make sure that either the difference is negligible, or `SHOW EXPLAIN`'s output is closer to reality than `EXPLAIN`'s output.

List of Recorded Differences

- `SHOW EXPLAIN` may have `Extra='no matching row in const table'`, where `EXPLAIN` would produce `Extra='Impossible WHERE ...'`
- For queries with subqueries, `SHOW EXPLAIN` may print `select_type==PRIMARY` where regular `EXPLAIN` used to print `select_type==SIMPLE`, or vice versa.

Required Permissions

Running `SHOW EXPLAIN` requires the same permissions as running `SHOW PROCESSLIST` would.

See Also

- [EXPLAIN ANALYZE](#), which will perform a query and outputs enhanced `EXPLAIN` results.
- It is also possible to [save EXPLAIN into the slow query log](#).

H3Dr1.2.2.8 Using Buffer UPDATE Algorithm

This article explains the `UPDATE` statement's *Using Buffer* algorithm.

Take the following table and query:

Name	Salary
Babatunde	1000
Jolana	1050
Pankaja	1300

```
UPDATE employees SET salary = salary+100 WHERE salary < 2000;
```

Suppose the `employees` table has an index on the `salary` column, and the optimizer decides to use a range scan on that index.

The optimizer starts a range scan on the `salary` index. We find the first record *Babatunde, 1000*. If we do an on-the-fly update, we immediately instruct the storage engine to change this record to be *Babatunde, 1000+100=1100*.

Then we proceed to search for the next record, and find *Jolana, 1050*. We instruct the storage engine to update it to be *Jolana, 1050+100=1150*.

Then we proceed to search for the next record ... and what happens next depends on the storage engine. In some storage engines, data changes are visible immediately, so we will find the *Babatunde, 1100* record that we wrote at the first step, modifying it again, giving Babatunde an undeserved raise. Then we will see Babatunde again and again, looping continually.

In order to prevent such situations, the optimizer checks whether the `UPDATE` statement is going to change key values for the keys it is using. In that case, it will use a different algorithm:

1. Scan everyone with "salary<2000", remembering the rowids of the rows in a buffer.
2. Read the buffer and apply the updates.

This way, each row will be updated only once.

The Using buffer [EXPLAIN](#) output indicates that the buffer as described above will be used.

H3Dr1.2.3 BACKUP Commands

Commands used by backup tools



BACKUP STAGE

Commands to be used by a MariaDB backup tool.



BACKUP LOCK

Blocks a table from DDL statements.



Mariabackup and BACKUP STAGE Commands

How Mariabackup could use BACKUP STAGE commands.



Storage Snapshots and BACKUP STAGE Commands

How storage snapshots could use BACKUP STAGE commands.

H3Dr1.2.3.1 BACKUP STAGE

MariaDB starting with [10.4.1](#)

The BACKUP STAGE commands were introduced in MariaDB 10.4.1.

Contents

1. [Syntax](#)
2. [Goals with BACKUP STAGE Commands](#)
3. [BACKUP STAGE Commands](#)
 1. [BACKUP STAGE START](#)
 2. [BACKUP STAGE FLUSH](#)
 3. [BACKUP STAGE BLOCK_DDL](#)
 4. [BACKUP STAGE BLOCK_COMMIT](#)
 5. [BACKUP STAGE END](#)
4. [Using BACKUP STAGE Commands with Backup Tools](#)
 1. [Using BACKUP STAGE Commands with Mariabackup](#)
 2. [Using BACKUP STAGE Commands with Storage Snapshots](#)
5. [Privileges](#)
6. [Notes](#)
7. [See Also](#)

The BACKUP STAGE commands are a set of commands to make it possible to make an efficient external backup tool.

Syntax

```
BACKUP STAGE [START | FLUSH | BLOCK_DDL | BLOCK_COMMIT | END ]
```

In the following text, a transactional table means InnoDB or "InnoDB-like engine with redo log that can lock redo purges and can be copied without locks by an outside process".

Goals with BACKUP STAGE Commands

- To be able to do a majority of the backup with the minimum possible server locks. Especially for

transactional tables (InnoDB, MyRocks etc) there is only need for a very short block of new commits while copying statistics and log tables.

- DDL are only needed to be blocked for a very short duration of the backup while [mariabackup](#) is copying the tables affected by DDL during the initial part of the backup.
- Most non transactional tables (those that are not in use) will be copied during BACKUP STAGE START . The exceptions are system statistic and log tables that are not blocked during the backup until BLOCK_COMMIT .
- Should work efficiently with backup tools that use disk snapshots.
- Should work as efficiently as possible for all table types that store data on the local disks.
- As little copying as possible under higher level stages/locks. For example, .frm (dictionary) and .trn (trigger) files should be copying while copying the table data.

BACKUP STAGE Commands

BACKUP STAGE START

The START stage is designed for the following tasks:

- Blocks purge of redo files for storage engines that needs this (Aria)
- Start logging of DDL commands into 'datadir'/ddl.log. This may take a short time as the command has to wait until there are no active DDL commands.

BACKUP STAGE FLUSH

The FLUSH stage is designed for the following tasks:

- FLUSH all changes for inactive non-transactional tables, except for statistics and log tables.
- Close all tables that are not in use, to ensure they are marked as closed for the backup.
- BLOCK all new write locks for all non transactional tables (except statistics and log tables). The command will not wait for tables that are in use by read-only transactions.

DDLS don't have to be blocked at this stage as they can't cause the table to be in an inconsistent state. This is true also for non-transactional tables.

BACKUP STAGE BLOCK_DDL

The BLOCK_DDL stage is designed for the following tasks:

- Wait for all statements using write locked non-transactional tables to end.
- Blocks [CREATE TABLE](#), [DROP TABLE](#), [TRUNCATE TABLE](#), and [RENAME TABLE](#).
- Blocks also start off a **new** [ALTER TABLE](#) and the **final rename phase** of [ALTER TABLE](#). Running ALTER TABLES are not blocked.

BACKUP STAGE BLOCK_COMMIT

The BLOCK_COMMIT stage is designed for the following tasks:

- Lock the binary log and commit/rollback to ensure that no changes are committed to any tables. If there are active commits or data to be copied to the binary log this will be allowed to finish. Active transactions will not affect BLOCK_COMMIT .
- This doesn't lock temporary tables that are not used by replication. However these will be blocked when it's time to write to the binary log.
- Lock system log tables and statistics tables, flush them and mark them closed.

When the BLOCK_COMMIT 's stages return, this is the 'backup time'. Everything committed will be in the backup and everything not committed will roll back.

Transactional engines will continue to do changes to the redo log during the BLOCK COMMIT stage, but this is not important as all of these will roll back later as the changes will not be committed.

BACKUP STAGE END

The END stage is designed for the following tasks:

- End DDL logging

- Free resources

Using BACKUP STAGE Commands with Backup Tools

Using BACKUP STAGE Commands with Mariabackup

The `BUSCUP STAGE` commands are a set of commands to make it possible to make an efficient external backup tool. How [Mariabackup](#) uses these commands depends on whether you are using the version that is bundled with MariaDB Community Server or the version that is bundled with [MariaDB Enterprise Server](#). See [Mariabackup and BACKUP STAGE Commands](#) for some examples on how [Mariabackup](#) uses these commands.

If you would like to use a version of [Mariabackup](#) that uses the `BUSCUP STAGE` commands in an efficient way, then one option is to use [MariaDB Enterprise Backup](#) that is bundled with [MariaDB Enterprise Server](#).

Using BACKUP STAGE Commands with Storage Snapshots

The `BUSCUP STAGE` commands are a set of commands to make it possible to make an efficient external backup tool. These commands could even be used by tools that perform backups by taking a snapshot of a file system, SAN, or some other kind of storage device. See [Storage Snapshots and BACKUP STAGE Commands](#) for some examples on how to use each `BUSCUP STAGE` command in an efficient way.

Privileges

`BUSCUP STAGE` requires the [RELOAD](#) privilege.

Notes

- Only one connection can run `BUSCUP STAGE START`. If a second connection tries, it will wait until the first one has executed `BUSCUP STAGE END`.
- If the user skips a `BUSCUP STAGE`, then all intermediate backup stages will automatically be run. This will allow us to add new stages within the `BUSCUP STAGE` hierarchy in the future with even more precise locks without causing problems for tools using an earlier version of the `BUSCUP STAGE` implementation.
- One can use the [max_statement_time](#) or [lock_wait_timeout](#) system variables to ensure that a `BUSCUP STAGE` command doesn't block the server too long.
- DDL logging will only be available in [MariaDB Enterprise Server](#) 10.2 and later.

See Also

- [BACKUP LOCK](#) Locking a table from DDL's.
- [MDEV-5336](#). Implement `BUSCUP STAGE` for safe external backups.

H3Dr1.2.3.2 BACKUP LOCK

MariaDB starting with [10.4.2](#)

The `BUSCUP LOCK` command was introduced in [MariaDB 10.4.2](#).

Contents

1. [Syntax](#)
2. [Usage in a Backup Tool](#)
3. [Privileges](#)
4. [Notes](#)
5. [Implementation](#)
6. [See Also](#)

BACKUP LOCK blocks a table from DDL statements. This is mainly intended to be used by tools like [mariabackup](#) that need to ensure there are no DDLs on a table while the table files are opened. For example, for an Aria table that stores data in 3 files with extensions .frm, .MAI and .MAD. Normal read/write operations can continue as normal.

Syntax

To lock a table:

```
BACKUP LOCK table_name
```

To unlock a table:

```
BACKUP UNLOCK
```

Usage in a Backup Tool

```
BACKUP LOCK [database.]table_name;
- Open all files related to a table (for example, t.frm, t.MAI and t.MYD)
BACKUP UNLOCK;
- Copy data
- Close files
```

This ensures that all files are from the same generation, that is created at the same time by the MariaDB server. This works, because the open files will point to the original table files which will not be affected if there is any ALTER TABLE while copying the files.

Privileges

BACKUP LOCK requires the [RELOAD](#) privilege.

Notes

- The idea is that the BACKUP LOCK should be held for as short a time as possible by the backup tool. The time to take an uncontested lock is very short! One can easily do 50,000 locks/unlocks per second on low end hardware.
- One should use different connections for [BACKUP STAGE](#) commands and BACKUP LOCK .

Implementation

- Internally, BACKUP LOCK is implemented by taking an MDL_SHARED_HIGH_PRIO MDL lock on the table object, which protects the table from any DDL operations.

See Also

- [BACKUP STAGE](#)
- [MDEV-17309](#) - BACKUP LOCK: DDL locking of tables during backup

H3Dr1.2.3.3 Mariabackup and BACKUP STAGE Commands

H3Dr1.2.3.4 Storage Snapshots and BACKUP STAGE Commands

MariaDB starting with [10.4.1](#)

The `BACKUP STAGE` commands were introduced in [MariaDB 10.4.1](#).

Contents

1. [Generic Backup Process with Storage Snapshots](#)

The `BACKUP STAGE` commands are a set of commands to make it possible to make an efficient external backup tool. These commands could even be used by tools that perform backups by taking a snapshot of a file system, SAN, or some other kind of storage device.

Generic Backup Process with Storage Snapshots

A tool that backs up MariaDB by taking a snapshot of a file system, SAN, or some other kind of storage device could use each `BACKUP STAGE` command in the following way:

- First, execute the following:

```
BACKUP STAGE START  
BACKUP STAGE BLOCK_COMMIT
```

- Then, take the snapshot.
- Then, execute the following:

```
BACKUP STAGE END
```

The above ensures that all non-transactional tables are properly flushed to disk before the snapshot is done. Using `BACKUP STAGE` commands is also more efficient than using the [FLUSH TABLES WITH READ LOCK](#) command as the above set of commands will not block or be blocked by write operations to transactional tables.

Note that when the backup is completed, one should delete all files with the "#sql" prefix, as these are files used by concurrent running `ALTER TABLE`. Note that InnoDB will on server restart automatically delete any tables with the "#sql" prefix.

H3Dr1.2.4 FLUSH Commands

Commands to reset (flush) various caches in MariaDB.



FLUSH

Clear or reload various internal caches.



FLUSH QUERY CACHE

Defragmenting the query cache



FLUSH TABLES FOR EXPORT

Flushes changes to disk for specific tables.

There are [2 related questions](#).

H3Dr1.2.4.1 FLUSH

Syntax

```
FLUSH [NO_WRITE_TO_BINLOG | LOCAL]
      flush_option [, flush_option] ...
```

or when flushing tables:

```
FLUSH [NO_WRITE_TO_BINLOG | LOCAL] TABLES [table_list] [table_flush_option]
```

Contents

1. [Syntax](#)
2. [Description](#)
3. [FLUSH RELAY LOGS](#)
 1. [Compatibility with MySQL](#)
4. [FLUSH STATUS](#)
 1. [Global Status Variables that Support FLUSH STATUS](#)
5. [The different usage of FLUSH TABLES](#)
 1. [The purpose of FLUSH TABLES](#)
 2. [The purpose of FLUSH TABLES WITH READ LOCK](#)
 3. [The purpose of FLUSH TABLES table_list](#)
 4. [The purpose of FLUSH TABLES table_list WITH READ LOCK](#)
6. [Implementation of FLUSH TABLES commands in MariaDB 10.4.8 and above](#)
 1. [Implementation of FLUSH TABLES](#)
 2. [Implementation of FLUSH TABLES WITH READ LOCK](#)
 3. [Implementation of FLUSH TABLES table_list](#)
 4. [Implementation of FLUSH TABLES table_list FOR EXPORT](#)
7. [FLUSH SSL](#)
8. [Reducing Memory Usage](#)

where `table_list` is a list of tables separated by `,` (comma).

Description

The `FLUSH` statement clears or reloads various internal caches used by MariaDB. To execute `FLUSH`, you must have the `RELOAD` privilege. See [GRANT](#).

The `RESET` statement is similar to `FLUSH`. See [RESET](#).

You cannot issue a `FLUSH` statement from within a [stored function](#) or a [trigger](#). Doing so within a stored procedure is permitted, as long as it is not called by a stored function or trigger. See [Stored Routine Limitations](#), [Stored Function Limitations](#) and [Trigger Limitations](#).

If a listed table is a view, an error like the following will be produced:

```
ERROR 1347 (HY000): 'test.v' is not BASE TABLE
```

By default, `FLUSH` statements are written to the [binary log](#) and will be [replicated](#). The `NO_WRITE_TO_BINLOG` keyword (`LOCAL` is an alias) will ensure the statement is not written to the binary log.

The different flush options are:

Option	Description
CHANGED_PAGE_BITMAPS	XtraDB only . Internal command used for backup purposes. See the Information Schema CHANGED_PAGE_BITMAPS Table .
CLIENT_STATISTICS	Reset client statistics (see SHOW CLIENT_STATISTICS).
DES_KEY_FILE	Reloads the DES key file (Specified with the --des-key-file startup option).
HOSTS	Flush the hostname cache (used for converting ip to host names and for unblocking blocked hosts. See max_connect_errors)
INDEX_STATISTICS	Reset index statistics (see SHOW INDEX_STATISTICS).
[ERROR ENGINE GENERAL SLOW BINARY RELAY] LOGS	Close and reopen the specified log type, or all log types if none are specified. FLUSH RELAY LOGS [connection-name] can be used to flush the relay logs for a specific connection. Only one connection can be specified per FLUSH command. See Multi-source replication . FLUSH ENGINE LOGS will delete all unneeded Aria redo logs. Since MariaDB 10.1.30 and MariaDB 10.2.11 , FLUSH BINARY LOGS DELETE_DOMAIN_ID=(list-of-domains) can be used to discard obsolete GTID domains from the server's binary log state . In order for this to be successful, no event group from the listed GTID domains can be present in existing binary log files . If some still exist, then they must be purged prior to executing this command. If the command completes successfully, then it also rotates the binary log.
MASTER	Deprecated option, use RESET MASTER instead.
PRIVILEGES	Reload all privileges from the privilege tables in the mysql database. If the server is started with --skip-grant-table option, this will activate the privilege tables again.
QUERY CACHE	Defragment the query cache to better utilize its memory . If you want to reset the query cache, you can do it with RESET QUERY CACHE .
QUERY_RESPONSE_TIME	See the QUERY_RESPONSE_TIME plugin.
SLAVE	Deprecated option, use RESET REPLICA or RESET SLAVE instead.
SSL	Used to dynamically reinitialize the server's TLS context by reloading the files defined by several TLS system variables . See FLUSH SSL for more information. This command was first added in MariaDB 10.4.1 .
STATUS	Resets all server status variables that can be reset to 0. Not all global status variables support this, so not all global values are reset. See FLUSH STATUS for more information.
TABLE	Close tables given as options or all open tables if no table list was used. From MariaDB 10.4.1 , using without any table list will only close tables not in use, and tables not locked by the FLUSH TABLES connection. If there are no locked tables, FLUSH TABLES will be instant and will not cause any waits, as it no longer waits for tables in use. When a table list is provided, from MariaDB 10.4.1 , the server will wait for the end of any transactions that are using the tables. Previously, FLUSH TABLES only waited for the statements to complete.
TABLES	Same as FLUSH TABLE .
TABLES ... FOR EXPORT	For InnoDB tables, flushes table changes to disk to permit binary table copies while the server is running. See FLUSH TABLES ... FOR EXPORT for more.
TABLES WITH READ LOCK	Closes all open tables. New tables are only allowed to be opened with read locks until an UNLOCK TABLES is given.
TABLES WITH READ LOCK AND DISABLE CHECKPOINT	As TABLES WITH READ LOCK but also disable all checkpoint writes by transactional table engines. This is useful when doing a disk snapshot of all tables.
TABLE_STATISTICS	Reset table statistics (see SHOW TABLE_STATISTICS).

USER_RESOURCES	Resets all per hour user resources . This enables clients that have exhausted their resources to connect again.
USER_STATISTICS	Reset user statistics (see SHOW USER_STATISTICS).
USER_VARIABLES	Reset user variables (see User-defined variables).

You can also use the [mysqladmin](#) client to flush things. Use `mysqladmin --help` to examine what flush commands it supports.

FLUSH RELAY LOGS

```
FLUSH RELAY LOGS 'connection_name';
```

Compatibility with MySQL

MariaDB starting with [10.7.0](#)

The `FOR CHANNEL` keyword was added for MySQL compatibility. This is identical as using the `channel_name` directly after the `FLUSH` command .

For example, one can now use:

```
FLUSH RELAY LOGS FOR CHANNEL 'connection_name';
```

FLUSH STATUS

Server status variables [can](#) be reset by executing the following:

```
FLUSH STATUS;
```

Global Status Variables that Support FLUSH STATUS

Not all global status variables support being reset by `FLUSH STATUS` . Currently, the following status variables are reset by `FLUSH STATUS` :

- [Aborted_clients](#)
- [Aborted_connects](#)
- [Binlog_cache_disk_use](#)
- [Binlog_cache_use](#)
- [Binlog_stmt_cache_disk_use](#)
- [Binlog_stmt_cache_use](#)
- [Connection_errors_accept](#)
- [Connection_errors_internal](#)
- [Connection_errors_max_connections](#)
- [Connection_errors_peer_address](#)
- [Connection_errors_select](#)
- [Connection_errors_tcpwrap](#)
- [Created_tmp_files](#)
- [Delayed_errors](#)
- [Delayed_writes](#)
- [Feature_check_constraint](#)
- [Feature_delay_key_write](#)
- [Max_used_connections](#)
- [Opened_plugin_libraries](#)
- [Performance_schema_accounts_lost](#)
- [Performance_schema_cond_instances_lost](#)
- [Performance_schema_digest_lost](#)
- [Performance_schema_file_handles_lost](#)
- [Performance_schema_file_instances_lost](#)

- Performance_schema_hosts_lost ↗
- Performance_schema_locker_lost ↗
- Performance_schema_mutex_instances_lost ↗
- Performance_schema_rwlock_instances_lost ↗
- Performance_schema_session_connect_attrs_lost ↗
- Performance_schema_socket_instances_lost ↗
- Performance_schema_stage_classes_lost ↗
- Performance_schema_statement_classes_lost ↗
- Performance_schema_table_handles_lost ↗
- Performance_schema_table_instances_lost ↗
- Performance_schema_thread_instances_lost ↗
- Performance_schema_users_lost ↗
- Qcache_hits ↗
- Qcache_inserts ↗
- Qcache_lowmem_prunes ↗
- Qcache_not_cached ↗
- Rpl_semi_sync_master_no_times ↗
- Rpl_semi_sync_master_no_tx ↗
- Rpl_semi_sync_master_timefunc_failures ↗
- Rpl_semi_sync_master_wait_pos_backtraverse ↗
- Rpl_semi_sync_master_yes_tx ↗
- Rpl_transactions_multi_engine ↗
- Server_audit_writes_failed ↗
- Slave_retried_transactions ↗
- Slow_launch_threads ↗
- Ssl_accept_renegotiates ↗
- Ssl_accepts ↗
- Ssl_callback_cache_hits ↗
- Ssl_client_connects ↗
- Ssl_connect_renegotiates ↗
- Ssl_ctx_verify_depth ↗
- Ssl_ctx_verify_mode ↗
- Ssl_finished_accepts ↗
- Ssl_finished_connects ↗
- Ssl_session_cache_hits ↗
- Ssl_session_cache_misses ↗
- Ssl_session_cache_overflows ↗
- Ssl_session_cache_size ↗
- Ssl_session_cache_timeouts ↗
- Ssl_sessions_reused ↗
- Ssl_used_session_cache_entries ↗
- Subquery_cache_hit ↗
- Subquery_cache_miss ↗
- Table_locks_immediate ↗
- Table_locks_waited ↗
- Tc_log_max_pages_used ↗
- Tc_log_page_waits ↗
- Transactions_gtid_foreign_engine ↗
- Transactions_multi_engine ↗

The different usage of FLUSH TABLES

The purpose of FLUSH TABLES

The purpose of FLUSH TABLES is to clean up the open table cache and table definition cache from not in use tables. This frees up memory and file descriptors. Normally this is not needed as the caches works on a FIFO bases, but can be useful if the server seams to use up to much memory for some reason.

The purpose of FLUSH TABLES WITH READ LOCK

FLUSH TABLES WITH READ LOCK is useful if you want to take a backup of some tables. When FLUSH TABLES WITH READ LOCK returns, all write access to tables are blocked and all tables are marked as

'properly closed' on disk. The tables can still be used for read operations.

The purpose of `FLUSH TABLES table_list`

`FLUSH TABLES table_list` is useful if you want to copy a table object/files to or from the server. This command puts a lock that stops new users of the table and will wait until everyone has stopped using the table. The table is then removed from the table definition and table cache.

Note that it's up to the user to ensure that no one is accessing the table between `FLUSH TABLES` and the table is copied to or from the server. This can be secured by using [LOCK TABLES](#).

If there are any tables locked by the connection that is using `FLUSH TABLES` all the locked tables will be closed as part of the flush and reopened and relocked before `FLUSH TABLES` returns. This allows one to copy the table after `FLUSH TABLES` returns without having any writes on the table. For now this works works with most tables, except InnoDB as InnoDB may do background purges on the table even while it's write locked.

The purpose of `FLUSH TABLES table_list WITH READ LOCK`

`FLUSH TABLES table_list WITH READ LOCK` should work as `FLUSH TABLES WITH READ LOCK`, but only those tables that are listed will be properly closed. However in practice this works exactly like `FLUSH TABLES WITH READ LOCK` as the `FLUSH` command has anyway to wait for all WRITE operations to end because we are depending on a global read lock for this code. In the future we should consider fixing this to instead use meta data locks.

Implementation of `FLUSH TABLES` commands in MariaDB 10.4.8 and above

Implementation of `FLUSH TABLES`

- Free memory and file descriptors not in use

Implementation of `FLUSH TABLES WITH READ LOCK`

- Lock all tables read only for simple old style backup.
- All background writes are suspended and tables are marked as closed.
- No statement requiring table changes are allowed for any user until `UNLOCK TABLES`.

Instead of using `FLUSH TABLE WITH READ LOCK` one should in most cases instead use [BACKUP STAGE BLOCK_COMMIT](#).

Implementation of `FLUSH TABLES table_list`

- Free memory and file descriptors for tables not in use from table list.
- Lock given tables as read only.
- Wait until all translations has ended that uses any of the given tables.
- Wait until all background writes are suspended and tables are marked as closed.

Implementation of `FLUSH TABLES table_list FOR EXPORT`

- Free memory and file descriptors for tables not in use from table list
- Lock given tables as read.
- Wait until all background writes are suspended and tables are marked as closed.
- Check that all tables supports `FOR EXPORT`
- No changes to these tables allowed until `UNLOCK TABLES`

This is basically the same behavior as in old MariaDB version if one first lock the tables, then do `FLUSH TABLES`. The tables will be copyable until `UNLOCK TABLES`.

FLUSH SSL

MariaDB starting with [10.4](#)

The `FLUSH SSL` command was first added in [MariaDB 10.4](#).

In [MariaDB 10.4](#) and later, the `FLUSH SSL` command can be used to dynamically reinitialize the server's [TLS](#) context. This is most useful if you need to replace a certificate that is about to expire without restarting the server.

This operation is performed by reloading the files defined by the following [TLS system variables](#):

- `ssl_cert`
- `ssl_key`
- `ssl_ca`
- `ssl_capath`
- `ssl_crl`
- `ssl_crlpath`

These [TLS system variables](#) are not dynamic, so their values can **not** be changed without restarting the server.

If you want to dynamically reinitialize the server's [TLS](#) context, then you need to change the certificate and key files at the relevant paths defined by these [TLS system variables](#), without actually changing the values of the variables. See [MDEV-19341](#) for more information.

Reducing Memory Usage

To flush some of the global caches that take up memory, you could execute the following command:

```
FLUSH LOCAL HOSTS,  
      QUERY CACHE,  
      TABLE_STATISTICS,  
      INDEX_STATISTICS,  
      USER_STATISTICS;
```

H3Dr1.2.4.2 FLUSH QUERY CACHE

Description

You can defragment the [query cache](#) to better utilize its memory with the `FLUSH QUERY CACHE` statement. The statement does not remove any queries from the cache.

The [RESET QUERY CACHE](#) statement removes all query results from the query cache. The [FLUSH TABLES](#) statement also does this.

H3Dr1.2.4.3 FLUSH TABLES FOR EXPORT

Syntax

```
FLUSH TABLES table_name [, table_name] FOR EXPORT
```

Contents

1. [Syntax](#)
2. [Description](#)
3. [Example](#)
4. [See Also](#)

Description

`FLUSH TABLES ... FOR EXPORT` flushes changes to the specified tables to disk so that binary copies can be made while the server is still running. This works for [Archive](#), [Aria](#), [CSV](#), [InnoDB](#), [MyISAM](#), [MERGE](#), and [XtraDB](#) tables.

The table is read locked until one has issued [UNLOCK TABLES](#).

If a storage engine does not support `FLUSH TABLES FOR EXPORT`, a 1031 error ([SQLSTATE 'HY000'](#)) is produced.

If `FLUSH TABLES ... FOR EXPORT` is in effect in the session, the following statements will produce an error if attempted:

- `FLUSH TABLES WITH READ LOCK`
- `FLUSH TABLES ... WITH READ LOCK`
- `FLUSH TABLES ... FOR EXPORT`
- Any statement trying to update any table

If any of the following statements is in effect in the session, attempting `FLUSH TABLES ... FOR EXPORT` will produce an error.

- `FLUSH TABLES ... WITH READ LOCK`
- `FLUSH TABLES ... FOR EXPORT`
- `LOCK TABLES ... READ`
- `LOCK TABLES ... WRITE`

`FLUSH FOR EXPORT` is not written to the [binary log](#).

This statement requires the [RELOAD](#) and the [LOCK TABLES](#) privileges.

If one of the specified tables cannot be locked, none of the tables will be locked.

If a table does not exist, an error like the following will be produced:

```
ERROR 1146 (42S02): Table 'test.xxx' doesn't exist
```

If a table is a view, an error like the following will be produced:

```
ERROR 1347 (HY000): 'test.v' is not BASE TABLE
```

Example

```
FLUSH TABLES test.t1 FOR EXPORT;
# Copy files related to the table (see below)
UNLOCK TABLES;
```

For a full description, please see [copying MariaDB tables](#).

See Also

- [Copying Tables Between Different MariaDB Databases and MariaDB Servers](#)
- [Copying Transportable InnoDB Tablespaces](#)
- [mysampack](#) - Compressing the MyISAM data file for easier distribution.
- [aria_pack](#) - Compressing the Aria data file for easier distribution

H3Dr1.2.5 Replication Commands

The terms *master* and *slave* have historically been used in replication, but the terms *primary* and *replica* are now preferred. The old terms are used still used in parts of the documentation, and in MariaDB commands, although [MariaDB 10.5](#) has begun the process of renaming. The documentation process is ongoing. See [MDEV-18777](#) to follow progress on this effort.



CHANGE MASTER TO

Set or change replica parameters for connecting to the primary.



START SLAVE

Start replica threads.



STOP SLAVE

Stop replica threads.



RESET REPLICA/SLAVE

Forget replica connection information and start a new relay log file.



SET GLOBAL SQL_SLAVE_SKIP_COUNTER

Skips a number of events from the primary.



SHOW RELAYLOG EVENTS

Show events in the relay log.



SHOW SLAVE STATUS

Show status for one or all primaries.



SHOW MASTER STATUS

Status information about the binary log.



SHOW SLAVE HOSTS

Display replicas currently registered with the primary.



RESET MASTER

Delete binary log files.

H3Dr1.2.5.1 CHANGE MASTER TO

The terms *master* and *slave* have historically been used in replication, but the terms *primary* and *replica* are now preferred. The old terms are used still used in parts of the documentation, and in MariaDB commands, although MariaDB 10.5 [has](#) begun the process of renaming. The documentation process is ongoing. See [MDEV-18777](#) to follow progress on this effort.

Syntax

```
CHANGE MASTER [ 'connection_name' ] TO master_def [, master_def] ...
    [FOR CHANNEL 'channel_name']

master_def:
    MASTER_BIND = 'interface_name'
    | MASTER_HOST = 'host_name'
    | MASTER_USER = 'user_name'
    | MASTER_PASSWORD = 'password'
    | MASTER_PORT = port_num
    | MASTER_CONNECT_RETRY = interval
    | MASTER_HEARTBEAT_PERIOD = interval
    | MASTER_LOG_FILE = 'master_log_name'
    | MASTER_LOG_POS = master_log_pos
    | RELAY_LOG_FILE = 'relay_log_name'
    | RELAY_LOG_POS = relay_log_pos
    | MASTER_DELAY = interval
    | MASTER_SSL = {0|1}
    | MASTER_SSL_CA = 'ca_file_name'
    | MASTER_SSL_CAPATH = 'ca_directory_name'
    | MASTER_SSL_CERT = 'cert_file_name'
    | MASTER_SSL_CRL = 'crl_file_name'
    | MASTER_SSL_CRLPATH = 'crl_directory_name'
    | MASTER_SSL_KEY = 'key_file_name'
    | MASTER_SSL_CIPHER = 'cipher_list'
    | MASTER_SSL_VERIFY_SERVER_CERT = {0|1}
    | MASTER_USE_GTID = {current_pos|slave_pos|no}
    | IGNORE_SERVER_IDS = (server_id_list)
    | DO_DOMAIN_IDS = ([N,...])
    | IGNORE_DOMAIN_IDS = ([N,...])
```

Contents

1. [Syntax](#)
2. [Description](#)
3. [Multi-Source Replication](#)
 1. [default_master_connection](#)
 2. [connection_name](#)
4. [Options](#)
 1. [Connection Options](#)
 1. [MASTER_USER](#)
 2. [MASTER_PASSWORD](#)
 3. [MASTER_HOST](#)
 4. [MASTER_PORT](#)
 5. [MASTER_CONNECT_RETRY](#)
 6. [MASTER_BIND](#)
 7. [MASTER_HEARTBEAT_PERIOD](#)
 2. [TLS Options](#)
 1. [MASTER_SSL](#)
 2. [MASTER_SSL_CA](#)
 3. [MASTER_SSL_CAPATH](#)
 4. [MASTER_SSL_CERT](#)
 5. [MASTER_SSL_CRL](#)
 6. [MASTER_SSL_CRLPATH](#)
 7. [MASTER_SSL_KEY](#)
 8. [MASTER_SSL_CIPHER](#)
 9. [MASTER_SSL_VERIFY_SERVER_C](#)
 3. [Binary Log Options](#)
 1. [MASTER_LOG_FILE](#)
 2. [MASTER_LOG_POS](#)
 4. [Relay Log Options](#)
 1. [RELAY_LOG_FILE](#)
 2. [RELAY_LOG_POS](#)
 5. [GTID Options](#)
 1. [MASTER_USE_GTID](#)
 6. [Replication Filter Options](#)
 1. [IGNORE_SERVER_IDS](#)
 2. [DO_DOMAIN_IDS](#)
 3. [IGNORE_DOMAIN_IDS](#)
 7. [Delayed Replication Options](#)
 1. [MASTER_DELAY](#)
 5. [Changing Option Values](#)
 6. [Option Persistence](#)
 7. [GTID Persistence](#)
 8. [Creating a Slave from a Backup](#)
 9. [Example](#)
 10. [See Also](#)

Description

The `CHANGE MASTER` statement sets the options that a [replica](#) uses to connect to and replicate from a [primary](#).

MariaDB starting with [10.7.0](#)

The `FOR CHANNEL` keyword was added for MySQL compatibility. This is identical to using the `channel_name` directly after `CHANGE MASTER`.

Multi-Source Replication

If you are using [multi-source replication](#), then you need to specify a connection name when you execute `CHANGE MASTER`. There are two ways to do this:

- Setting the [default_master_connection](#) system variable prior to executing `CHANGE MASTER`.

- Setting the `connection_name` parameter when executing `CHANGE MASTER`.

default_master_connection

```
SET default_master_connection = 'gandalf';
STOP SLAVE;
CHANGE MASTER TO
    MASTER_PASSWORD='new3cret';
START SLAVE;
```

connection_name

```
STOP SLAVE 'gandalf';
CHANGE MASTER 'gandalf' TO
    MASTER_PASSWORD='new3cret';
START SLAVE 'gandalf';
```

Options

Connection Options

MASTER_USER

The `MASTER_USER` option for `CHANGE MASTER` defines the user account that the [replica](#) will use to connect to the [primary](#).

This user account will need the [REPLICATION SLAVE](#) privilege (or, from [MariaDB 10.5.1](#), the [REPLICATION REPLICA](#) on the primary).

For example:

```
STOP SLAVE;
CHANGE MASTER TO
    MASTER_USER='rep1',
    MASTER_PASSWORD='new3cret';
START SLAVE;
```

The maximum length of the `MASTER_USER` string is 96 characters until [MariaDB 10.5](#), and 128 characters from [MariaDB 10.6](#).

MASTER_PASSWORD

The `MASTER_USER` option for `CHANGE MASTER` defines the password that the [replica](#) will use to connect to the [primary](#) as the user account defined by the `MASTER_USER` option.

For example:

```
STOP SLAVE;
CHANGE MASTER TO
    MASTER_PASSWORD='new3cret';
START SLAVE;
```

The maximum length of the `MASTER_PASSWORD` string is 32 characters.

MASTER_HOST

The `MASTER_HOST` option for `CHANGE MASTER` defines the hostname or IP address of the [primary](#).

If you set the value of the `MASTER_HOST` option to the empty string, then that is not the same as not setting the option's value at all. If you set the value of the `MASTER_HOST` option to the empty string, then the `CHANGE MASTER` command will fail with an error. In [MariaDB 5.3](#) and before, if you set the value of the `MASTER_HOST` option to the empty string, then the `CHANGE MASTER` command would succeed, but the

subsequent [START SLAVE](#) command would fail.

For example:

```
STOP SLAVE;
CHANGE MASTER TO
    MASTER_HOST='dbserver1.example.com',
    MASTER_USER='repl',
    MASTER_PASSWORD='new3cret',
    MASTER_USE_GTID=slave_pos;
START SLAVE;
```

If you set the value of the `MASTER_HOST` option in a `CHANGE MASTER` command, then the replica assumes that the primary is different from before, even if you set the value of this option to the same value it had previously. In this scenario, the replica will consider the old values for the primary's [binary log](#) file name and position to be invalid for the new primary. As a side effect, if you do not explicitly set the values of the `MASTER_LOG_FILE` and `MASTER_LOG_POS` options in the statement, then the statement will be implicitly appended with `MASTER_LOG_FILE=''` and `MASTER_LOG_POS=4`. However, if you enable [GTID](#) mode for replication by setting the `MASTER_USE_GTID` option to some value other than `no` in the statement, then these values will effectively be ignored anyway.

Replicas cannot connect to primaries using Unix socket files or Windows named pipes. The replica must connect to the primary using TCP/IP.

The maximum length of the `MASTER_HOST` string is 60 characters until [MariaDB 10.5](#), and 255 characters from [MariaDB 10.6](#).

MASTER_PORT

The `MASTER_PORT` option for `CHANGE MASTER` defines the TCP/IP port of the [primary](#).

For example:

```
STOP SLAVE;
CHANGE MASTER TO
    MASTER_HOST='dbserver1.example.com',
    MASTER_PORT=3307,
    MASTER_USER='repl',
    MASTER_PASSWORD='new3cret',
    MASTER_USE_GTID=slave_pos;
START SLAVE;
```

If you set the value of the `MASTER_PORT` option in a `CHANGE MASTER` command, then the replica assumes that the primary is different from before, even if you set the value of this option to the same value it had previously. In this scenario, the replica will consider the old values for the primary's [binary log](#) file name and position to be invalid for the new primary. As a side effect, if you do not explicitly set the values of the `MASTER_LOG_FILE` and `MASTER_LOG_POS` options in the statement, then the statement will be implicitly appended with `MASTER_LOG_FILE=''` and `MASTER_LOG_POS=4`. However, if you enable [GTID](#) mode for replication by setting the `MASTER_USE_GTID` option to some value other than `no` in the statement, then these values will effectively be ignored anyway.

Replicas cannot connect to primaries using Unix socket files or Windows named pipes. The replica must connect to the primary using TCP/IP.

MASTER_CONNECT_RETRY

The `MASTER_CONNECT_RETRY` option for `CHANGE MASTER` defines how many seconds that the replica will

wait between connection retries. The default is 60.

```
STOP SLAVE;
CHANGE MASTER TO
    MASTER_CONNECT_RETRY=20;
START SLAVE;
```

The number of connection attempts is limited by the [master_retry_count](#) option. It can be set either on the command-line or in a server [option group](#) in an [option file](#) prior to starting up the server. For example:

```
[mariadb]
...
master_retry_count=4294967295
```

MASTER_BIND

The `MASTER_BIND` option for `CHANGE MASTER` is only supported by MySQL 5.6.2 and later and by MySQL NDB Cluster 7.3.1 and later. This option is not supported by MariaDB. See [MDEV-19248](#) for more information.

The `MASTER_BIND` option for `CHANGE MASTER` can be used on replicas that have multiple network interfaces to choose which network interface the replica will use to connect to the primary.

MASTER_HEARTBEAT_PERIOD

The `MASTER_HEARTBEAT_PERIOD` option for `CHANGE MASTER` can be used to set the interval in seconds between replication heartbeats. Whenever the primary's [binary log](#) is updated with an event, the waiting period for the next heartbeat is reset.

This option's *interval* argument has the following characteristics:

- It is a decimal value with a range of 0 to 4294967 seconds.
- It has a resolution of hundredths of a second.
- Its smallest valid non-zero value is 0.001.
- Its default value is the value of the [slave_net_timeout](#) system variable divided by 2.
- If it's set to 0, then heartbeats are disabled.

Heartbeats are sent by the primary only if there are no unsent events in the binary log file for a period longer than the interval.

If the `RESET SLAVE` statement is executed, then the heartbeat interval is reset to the default.

If the [slave_net_timeout](#) system variable is set to a value that is lower than the current heartbeat interval, then a warning will be issued.

TLS Options

The TLS options are used for providing information about [TLS](#). The options can be set even on replicas that are compiled without TLS support. The TLS options are saved to either the default `master.info` file or the file that is configured by the [master_info_file](#) option, but these TLS options are ignored unless the replica supports TLS.

See [Replication with Secure Connections](#) for more information.

MASTER_SSL

The `MASTER_SSL` option for `CHANGE MASTER` tells the replica whether to force [TLS](#) for the connection. The valid values are 0 or 1.

For example:

```
STOP SLAVE;
CHANGE MASTER TO
    MASTER_SSL=1;
START SLAVE;
```

MASTER_SSL_CA

The `MASTER_SSL_CA` option for `CHANGE MASTER` defines a path to a PEM file that should contain one or more X509 certificates for trusted Certificate Authorities (CAs) to use for [TLS](#). This option requires that you use the absolute path, not a relative path. This option implies the [MASTER_SSL](#) option.

For example:

```
STOP SLAVE;
CHANGE MASTER TO
    MASTER_SSL_CERT='/etc/my.cnf.d/certificates/server-cert.pem',
    MASTER_SSL_KEY='/etc/my.cnf.d/certificates/server-key.pem',
    MASTER_SSL_CA='/etc/my.cnf.d/certificates/ca.pem',
    MASTER_SSL_VERIFY_SERVER_CERT=1;
START SLAVE;
```

See [Secure Connections Overview: Certificate Authorities \(CAs\)](#) for more information.

The maximum length of `MASTER_SSL_CA` string is 511 characters.

MASTER_SSL_CAPATH

The `MASTER_SSL_CAPATH` option for `CHANGE MASTER` defines a path to a directory that contains one or more PEM files that should each contain one X509 certificate for a trusted Certificate Authority (CA) to use for [TLS](#). This option requires that you use the absolute path, not a relative path. The directory specified by this option needs to be run through the [openssl rehash](#) command. This option implies the [MASTER_SSL](#) option.

For example:

```
STOP SLAVE;
CHANGE MASTER TO
    MASTER_SSL_CERT='/etc/my.cnf.d/certificates/server-cert.pem',
    MASTER_SSL_KEY='/etc/my.cnf.d/certificates/server-key.pem',
    MASTER_SSL_CAPATH='/etc/my.cnf.d/certificates/ca/',
    MASTER_SSL_VERIFY_SERVER_CERT=1;
START SLAVE;
```

See [Secure Connections Overview: Certificate Authorities \(CAs\)](#) for more information.

The maximum length of `MASTER_SSL_CA_PATH` string is 511 characters.

MASTER_SSL_CERT

The `MASTER_SSL_CERT` option for `CHANGE MASTER` defines a path to the X509 certificate file to use for [TLS](#). This option requires that you use the absolute path, not a relative path. This option implies the [MASTER_SSL](#) option.

For example:

```
STOP SLAVE;
CHANGE MASTER TO
    MASTER_SSL_CERT='/etc/my.cnf.d/certificates/server-cert.pem',
    MASTER_SSL_KEY='/etc/my.cnf.d/certificates/server-key.pem',
    MASTER_SSL_CA='/etc/my.cnf.d/certificates/ca.pem',
    MASTER_SSL_VERIFY_SERVER_CERT=1;
START SLAVE;
```

The maximum length of `MASTER_SSL_CERT` string is 511 characters.

MASTER_SSL_CRL

The `MASTER_SSL_CRL` option for `CHANGE MASTER` defines a path to a PEM file that should contain one or more revoked X509 certificates to use for [TLS](#). This option requires that you use the absolute path, not a relative path.

This option is only supported if the server was built with OpenSSL. If the server was built with yaSSL, then this option is not supported. See [TLS and Cryptography Libraries Used by MariaDB](#) for more information about which libraries are used on which platforms.

For example:

```
STOP SLAVE;
CHANGE MASTER TO
    MASTER_SSL_CERT='/etc/my.cnf.d/certificates/server-cert.pem',
    MASTER_SSL_KEY='/etc/my.cnf.d/certificates/server-key.pem',
    MASTER_SSL_CA='/etc/my.cnf.d/certificates/ca.pem',
    MASTER_SSL_VERIFY_SERVER_CERT=1,
    MASTER_SSL_CRL='"/etc/my.cnf.d/certificates/crl.pem';
START SLAVE;
```

See [Secure Connections Overview: Certificate Revocation Lists \(CRLs\)](#) for more information.

The maximum length of `MASTER_SSL_CRL` string is 511 characters.

MASTER_SSL_CRLPATH

The `MASTER_SSL_CRLPATH` option for `CHANGE MASTER` defines a path to a directory that contains one or more PEM files that should each contain one revoked X509 certificate to use for [TLS](#). This option requires that you use the absolute path, not a relative path. The directory specified by this variable needs to be run through the [openssl rehash](#) command.

This option is only supported if the server was built with OpenSSL. If the server was built with yaSSL, then this option is not supported. See [TLS and Cryptography Libraries Used by MariaDB](#) for more information about which libraries are used on which platforms.

For example:

```
STOP SLAVE;
CHANGE MASTER TO
    MASTER_SSL_CERT='/etc/my.cnf.d/certificates/server-cert.pem',
    MASTER_SSL_KEY='/etc/my.cnf.d/certificates/server-key.pem',
    MASTER_SSL_CA='/etc/my.cnf.d/certificates/ca.pem',
    MASTER_SSL_VERIFY_SERVER_CERT=1,
    MASTER_SSL_CRLPATH='"/etc/my.cnf.d/certificates/crl/';
START SLAVE;
```

See [Secure Connections Overview: Certificate Revocation Lists \(CRLs\)](#) for more information.

The maximum length of `MASTER_SSL_CRL_PATH` string is 511 characters.

MASTER_SSL_KEY

The `MASTER_SSL_KEY` option for `CHANGE MASTER` defines a path to a private key file to use for [TLS](#). This option requires that you use the absolute path, not a relative path. This option implies the `MASTER_SSL` option.

For example:

```
STOP SLAVE;
CHANGE MASTER TO
    MASTER_SSL_CERT='/etc/my.cnf.d/certificates/server-cert.pem',
    MASTER_SSL_KEY='/etc/my.cnf.d/certificates/server-key.pem',
    MASTER_SSL_CA='/etc/my.cnf.d/certificates/ca.pem',
    MASTER_SSL_VERIFY_SERVER_CERT=1;
START SLAVE;
```

The maximum length of `MASTER_SSL_KEY` string is 511 characters.

MASTER_SSL_CIPHER

The `MASTER_SSL_CIPHER` option for `CHANGE MASTER` defines the list of permitted ciphers or cipher suites to use for [TLS](#). Besides cipher names, if MariaDB was compiled with OpenSSL, this option could be set to "SSLv3" or "TLSv1.2" to allow all SSLv3 or all TLSv1.2 ciphers. Note that the TLSv1.3 ciphers cannot be excluded when using OpenSSL, even by using this option. See [Using TLSv1.3](#) for details. This option implies the `MASTER_SSL` option.

For example:

```
STOP SLAVE;
CHANGE MASTER TO
    MASTER_SSL_CERT='/etc/my.cnf.d/certificates/server-cert.pem',
    MASTER_SSL_KEY='/etc/my.cnf.d/certificates/server-key.pem',
    MASTER_SSL_CA='/etc/my.cnf.d/certificates/ca.pem',
    MASTER_SSL_VERIFY_SERVER_CERT=1,
    MASTER_SSL_CIPHER='TLSv1.2';
START SLAVE;
```

The maximum length of `MASTER_SSL_CIPHER` string is 511 characters.

`MASTER_SSL_VERIFY_SERVER_CERT`

The `MASTER_SSL_VERIFY_SERVER_CERT` option for `CHANGE MASTER` enables [server certificate verification](#). This option is disabled by default.

For example:

```
STOP SLAVE;
CHANGE MASTER TO
    MASTER_SSL_CERT='/etc/my.cnf.d/certificates/server-cert.pem',
    MASTER_SSL_KEY='/etc/my.cnf.d/certificates/server-key.pem',
    MASTER_SSL_CA='/etc/my.cnf.d/certificates/ca.pem',
    MASTER_SSL_VERIFY_SERVER_CERT=1;
START SLAVE;
```

See [Secure Connections Overview: Server Certificate Verification](#) for more information.

Binary Log Options

These options are related to the [binary log](#) position on the primary.

`MASTER_LOG_FILE`

The `MASTER_LOG_FILE` option for `CHANGE MASTER` can be used along with `MASTER_LOG_POS` to specify the coordinates at which the [replica's I/O thread](#) should begin reading from the primary's [binary logs](#) the next time the thread starts.

For example:

```
STOP SLAVE;
CHANGE MASTER TO
    MASTER_LOG_FILE='master2-bin.001',
    MASTER_LOG_POS=4;
START SLAVE;
```

The `MASTER_LOG_FILE` and `MASTER_LOG_POS` options cannot be specified if the `RELAY_LOG_FILE` and `RELAY_LOG_POS` options were also specified.

The `MASTER_LOG_FILE` and `MASTER_LOG_POS` options are effectively ignored if you enable [GTID](#) mode for replication by setting the `MASTER_USE_GTID` option to some value other than `no` in the statement.

`MASTER_LOG_POS`

The `MASTER_LOG_POS` option for `CHANGE MASTER` can be used along with `MASTER_LOG_FILE` to specify the coordinates at which the [replica's I/O thread](#) should begin reading from the primary's [binary logs](#) the next time the thread starts.

For example:

```
STOP SLAVE;
CHANGE MASTER TO
    MASTER_LOG_FILE='master2-bin.001',
    MASTER_LOG_POS=4;
START SLAVE;
```

The `MASTER_LOG_FILE` and `MASTER_LOG_POS` options cannot be specified if the `RELAY_LOG_FILE` and `RELAY_LOG_POS` options were also specified.

The `MASTER_LOG_FILE` and `MASTER_LOG_POS` options are effectively ignored if you enable `GTID` mode for replication by setting the `MASTER_USE_GTID` option to some value other than `no` in the statement.

Relay Log Options

These options are related to the [relay log](#) position on the replica.

RELAY_LOG_FILE

The `RELAY_LOG_FILE` option for `CHANGE MASTER` can be used along with the `RELAY_LOG_POS` option to specify the coordinates at which the [replica's SQL thread](#) should begin reading from the [relay log](#) the next time the thread starts.

The `CHANGE MASTER` statement usually deletes all [relay log](#) files. However, if the `RELAY_LOG_FILE` and/or `RELAY_LOG_POS` options are specified, then existing [relay log](#) files are kept.

When you want to change the [relay log](#) position, you only need to stop the [replica's SQL thread](#). The [replica's I/O thread](#) can continue running. The `STOP SLAVE` and `START SLAVE` statements support the `SQL_THREAD` option for this scenario. For example:

```
STOP SLAVE SQL_THREAD;
CHANGE MASTER TO
    RELAY_LOG_FILE='slave-relay-bin.006',
    RELAY_LOG_POS=4025;
START SLAVE SQL_THREAD;
```

When the value of this option is changed, the metadata about the [replica's SQL thread's](#) position in the [relay logs](#) will also be changed in the `relay-log.info` file or the file that is configured by the `relay_log_info_file` system variable.

The `RELAY_LOG_FILE` and `RELAY_LOG_POS` options cannot be specified if the `MASTER_LOG_FILE` and `MASTER_LOG_POS` options were also specified.

RELAY_LOG_POS

The `RELAY_LOG_POS` option for `CHANGE MASTER` can be used along with the `RELAY_LOG_FILE` option to specify the coordinates at which the [replica's SQL thread](#) should begin reading from the [relay log](#) the next time the thread starts.

The `CHANGE MASTER` statement usually deletes all [relay log](#) files. However, if the `RELAY_LOG_FILE` and/or `RELAY_LOG_POS` options are specified, then existing [relay log](#) files are kept.

When you want to change the [relay log](#) position, you only need to stop the [replica's SQL thread](#). The [replica's I/O thread](#) can continue running. The `STOP SLAVE` and `START SLAVE` statements support the `SQL_THREAD` option for this scenario. For example:

```
STOP SLAVE SQL_THREAD;
CHANGE MASTER TO
    RELAY_LOG_FILE='slave-relay-bin.006',
    RELAY_LOG_POS=4025;
START SLAVE SQL_THREAD;
```

When the value of this option is changed, the metadata about the [replica's SQL thread's](#) position in the [relay logs](#) will also be changed in the `relay-log.info` file or the file that is configured by the `relay_log_info_file` system variable.

The `RELAY_LOG_FILE` and `RELAY_LOG_POS` options cannot be specified if the `MASTER_LOG_FILE` and `MASTER_LOG_POS` options were also specified.

GTID Options

MASTER_USE_GTID

The `MASTER_USE_GTID` option for `CHANGE MASTER` can be used to configure the replica to use the [global transaction ID \(GTID\)](#) when connecting to a primary. The possible values are:

- `current_pos` - Replicate in [GTID](#) mode and use [gtid_current_pos](#) as the position to start downloading transactions from the primary.
- `slave_pos` - Replicate in [GTID](#) mode and use [gtid_slave_pos](#) as the position to start downloading transactions from the primary. From [MariaDB 10.5.1](#), `replica_pos` is an alias for `slave_pos`.
- `no` - Don't replicate in [GTID](#) mode.

For example:

```
STOP SLAVE;
CHANGE MASTER TO
    MASTER_USE_GTID = current_pos;
START SLAVE;
```

Or:

```
STOP SLAVE;
SET GLOBAL gtid_slave_pos='0-1-153';
CHANGE MASTER TO
    MASTER_USE_GTID = slave_pos;
START SLAVE;
```

Replication Filter Options

Also see [Replication filters](#).

IGNORE_SERVER_IDS

The `IGNORE_SERVER_IDS` option for `CHANGE MASTER` can be used to configure a [replica](#) to ignore [binary log](#) events that originated from certain servers. Filtered [binary log](#) events will not get logged to the replica's [relay log](#), and they will not be applied by the replica.

The option's value can be specified by providing a comma-separated list of [server_id](#) values. For example:

```
STOP SLAVE;
CHANGE MASTER TO
    IGNORE_SERVER_IDS = (3,5);
START SLAVE;
```

If you would like to clear a previously set list, then you can set the value to an empty list. For example:

```
STOP SLAVE;
CHANGE MASTER TO
    IGNORE_SERVER_IDS = ();
START SLAVE;
```

DO_DOMAIN_IDS

The `DO_DOMAIN_IDS` option for `CHANGE MASTER` can be used to configure a replica to only apply binary log events if the transaction's GTID is in a specific gtid_domain_id value. Filtered binary log events will not get logged to the replica's relay log, and they will not be applied by the replica.

The option's value can be specified by providing a comma-separated list of gtid_domain_id values. Duplicate values are automatically ignored. For example:

```
STOP SLAVE;
CHANGE MASTER TO
    DO_DOMAIN_IDS = (1,2);
START SLAVE;
```

If you would like to clear a previously set list, then you can set the value to an empty list. For example:

```
STOP SLAVE;
CHANGE MASTER TO
    DO_DOMAIN_IDS = ();
START SLAVE;
```

The `DO_DOMAIN_IDS` option and the `IGNORE_DOMAIN_IDS` option cannot both be set to non-empty values at the same time. If you want to set the `DO_DOMAIN_IDS` option, and the `IGNORE_DOMAIN_IDS` option was previously set, then you need to clear the value of the `IGNORE_DOMAIN_IDS` option. For example:

```
STOP SLAVE;
CHANGE MASTER TO
    IGNORE_DOMAIN_IDS = (),
    DO_DOMAIN_IDS = (1,2);
START SLAVE;
```

The `DO_DOMAIN_IDS` option can only be specified if the replica is replicating in GTID mode. Therefore, the `MASTER_USE_GTID` option must also be set to some value other than `no` in order to use this option.

IGNORE_DOMAIN_IDS

The `IGNORE_DOMAIN_IDS` option for `CHANGE MASTER` can be used to configure a replica to ignore binary log events if the transaction's GTID is in a specific gtid_domain_id value. Filtered binary log events will not get logged to the replica's relay log, and they will not be applied by the replica.

The option's value can be specified by providing a comma-separated list of gtid_domain_id values. Duplicate values are automatically ignored. For example:

```
STOP SLAVE;
CHANGE MASTER TO
    IGNORE_DOMAIN_IDS = (1,2);
START SLAVE;
```

If you would like to clear a previously set list, then you can set the value to an empty list. For example:

```
STOP SLAVE;
CHANGE MASTER TO
    IGNORE_DOMAIN_IDS = ();
START SLAVE;
```

The `DO_DOMAIN_IDS` option and the `IGNORE_DOMAIN_IDS` option cannot both be set to non-empty values at the same time. If you want to set the `IGNORE_DOMAIN_IDS` option, and the `DO_DOMAIN_IDS` option was previously set, then you need to clear the value of the `DO_DOMAIN_IDS` option. For example:

```
STOP SLAVE;
CHANGE MASTER TO
    DO_DOMAIN_IDS = (),
    IGNORE_DOMAIN_IDS = (1,2);
START SLAVE;
```

The `IGNORE_DOMAIN_IDS` option can only be specified if the replica is replicating in [GTID](#) mode. Therefore, the `MASTER_USE_GTID` option must also be set to some value other than `no` in order to use this option.

Delayed Replication Options

MASTER_DELAY

MariaDB starting with [10.2.3](#)

The `MASTER_DELAY` option for `CHANGE MASTER` was first added in [MariaDB 10.2.3](#) to enable [delayed replication](#).

The `MASTER_DELAY` option for `CHANGE MASTER` can be used to enable [delayed replication](#). This option specifies the time in seconds (at least) that a replica should lag behind the primary up to a maximum value of 2147483647, or about 68 years. Before executing an event, the replica will first wait, if necessary, until the given time has passed since the event was created on the primary. The result is that the replica will reflect the state of the primary some time back in the past. The default is zero, no delay.

```
STOP SLAVE;
CHANGE MASTER TO
    MASTER_DELAY=3600;
START SLAVE;
```

Changing Option Values

If you don't specify a given option when executing the `CHANGE MASTER` statement, then the option keeps its old value in most cases. Most of the time, there is no need to specify the options that do not need to change. For example, if the password for the user account that the replica uses to connect to its primary has changed, but no other options need to change, then you can just change the `MASTER_PASSWORD` option by executing the following commands:

```
STOP SLAVE;
CHANGE MASTER TO
    MASTER_PASSWORD='new3cret';
START SLAVE;
```

There are some cases where options are implicitly reset, such as when the `MASTER_HOST` and `MASTER_PORT` options are changed.

Option Persistence

The values of the [MASTER_LOG_FILE](#) and [MASTER_LOG_POS](#) options (i.e. the [binary log](#) position on the primary) and most other options are written to either the default `master.info` file or the file that is configured by the [master_info_file](#) option. The replica's I/O thread keeps this [binary log](#) position updated as it downloads events only when [MASTER_USE_GTID](#) option is set to `No`. Otherwise the file is not updated on a per event basis.

The [master_info_file](#) option can be set either on the command-line or in a server [option group](#) in an [option file](#) prior to starting up the server. For example:

```
[mariadb]
...
master_info_file=/mariadb/myserver1-master.info
```

The values of the [RELAY_LOG_FILE](#) and [RELAY_LOG_POS](#) options (i.e. the [relay log](#) position) are written to either the default `relay-log.info` file or the file that is configured by the [relay_log_info_file](#) system variable. The replica's [SQL thread](#) keeps this [relay log](#) position updated as it applies events.

The [relay_log_info_file](#) system variable can be set either on the command-line or in a server [option group](#) in an [option file](#) prior to starting up the server. For example:

```
[mariadb]
...
relay_log_info_file=/mariadb/myserver1-relay-log.info
```

GTID Persistence

If the replica is replicating [binary log](#) events that contain [GTIDs](#), then the replica's [SQL thread](#) will write every GTID that it applies to the [mysql.gtid_slave_pos](#) table. This GTID can be inspected and modified through the [gtid_slave_pos](#) system variable.

If the replica has the [log_slave_updates](#) system variable enabled and if the replica has the [binary log](#) enabled, then every write by the replica's [SQL thread](#) will also go into the replica's [binary log](#). This means that [GTIDs](#) of replicated transactions would be reflected in the value of the [gtid_binlog_pos](#) system variable.

Creating a Slave from a Backup

The `CHANGE MASTER` statement is useful for setting up a replica when you have a backup of the primary and you also have the [binary log](#) position or [GTID](#) position corresponding to the backup.

After restoring the backup on the replica, you could execute something like this to use the [binary log](#) position:

```
CHANGE MASTER TO
  MASTER_LOG_FILE='master2-bin.001',
  MASTER_LOG_POS=4;
START SLAVE;
```

Or you could execute something like this to use the [GTID](#) position:

```
SET GLOBAL gtid_slave_pos='0-1-153';
CHANGE MASTER TO
  MASTER_USE_GTID=slave_pos;
START SLAVE;
```

See [Setting up a Replication Slave with Mariabackup](#) for more information on how to do this with [Mariabackup](#).

Example

The following example changes the primary and primary's binary log coordinates. This is used when you want to set up the replica to replicate the primary:

```
CHANGE MASTER TO
MASTER_HOST='master2.mycompany.com',
MASTER_USER='replication',
MASTER_PASSWORD='bigs3cret',
MASTER_PORT=3306,
MASTER_LOG_FILE='master2-bin.001',
MASTER_LOG_POS=4,
MASTER_CONNECT_RETRY=10;
START SLAVE;
```

See Also

- [Setting up replication](#)
- [START SLAVE](#)
- [Multi-source replication](#)
- [RESET SLAVE](#). Removes a connection created with `CHANGE MASTER TO`.
- [Global Transaction ID](#)

H3Dr1.2.5.2 START SLAVE

The terms *master* and *slave* have historically been used in replication, but the terms *primary* and *replica* are now preferred. The old terms are used still used in parts of the documentation, and in MariaDB commands, although MariaDB 10.5.2 has begun the process of renaming. The documentation process is ongoing. See [MDEV-18777](#) to follow progress on this effort.

Syntax

```
START SLAVE ["connection_name"] [thread_type [, thread_type] ... ] [FOR CHANNEL  
"connection_name"]  
START SLAVE ["connection_name"] [SQL_THREAD] UNTIL  
    MASTER_LOG_FILE = 'log_name', MASTER_LOG_POS = log_pos [FOR CHANNEL "connection_name"]  
START SLAVE ["connection_name"] [SQL_THREAD] UNTIL  
    RELAY_LOG_FILE = 'log_name', RELAY_LOG_POS = log_pos [FOR CHANNEL "connection_name"]  
START SLAVE ["connection_name"] [SQL_THREAD] UNTIL  
    MASTER_GTID_POS = <GTID position> [FOR CHANNEL "connection_name"]  
START ALL SLAVES [thread_type [, thread_type]]  
  
START REPLICA ["connection_name"] [thread_type [, thread_type] ... ] -- from 10.5.1  
START REPLICA ["connection_name"] [SQL_THREAD] UNTIL  
    MASTER_LOG_FILE = 'log_name', MASTER_LOG_POS = log_pos -- from 10.5.1  
START REPLICA ["connection_name"] [SQL_THREAD] UNTIL  
    RELAY_LOG_FILE = 'log_name', RELAY_LOG_POS = log_pos -- from 10.5.1  
START REPLICA ["connection_name"] [SQL_THREAD] UNTIL  
    MASTER_GTID_POS = <GTID position> -- from 10.5.1  
START ALL REPLICAS [thread_type [, thread_type]] -- from 10.5.1  
  
thread_type: IO_THREAD | SQL_THREAD
```

Contents

1. [Syntax](#)
2. [Description](#)
 1. [START SLAVE UNTIL](#)
 2. [connection_name](#)
 3. [START ALL SLAVES](#)
 4. [START REPLICA](#)
 3. [See Also](#)

Description

START SLAVE (START REPLICA from [MariaDB 10.5.1](#)) with no thread_type options starts both of the replica threads (see [replication](#)). The I/O thread reads events from the primary server and stores them in the [relay log](#). The SQL thread reads events from the relay log and executes them. START SLAVE requires the [SUPER](#) privilege, or, from [MariaDB 10.5.2](#), the [REPLICATION SLAVE ADMIN](#) privilege.

If START SLAVE succeeds in starting the replica threads, it returns without any error. However, even in that case, it might be that the replica threads start and then later stop (for example, because they do not manage to connect to the primary or read its [binary log](#), or some other problem). START SLAVE does not warn you about this. You must check the replica's [error log](#) for error messages generated by the replica threads, or check that they are running satisfactorily with [SHOW SLAVE STATUS](#) ([SHOW REPLICAS STATUS](#) from [MariaDB 10.5.1](#)).

START SLAVE UNTIL

START SLAVE UNTIL refers to the SQL_THREAD replica position at which the SQL_THREAD replication will halt. If SQL_THREAD isn't specified both threads are started.

START SLAVE UNTIL master_gtid_pos=xxx is also supported. See [Global Transaction ID/START SLAVE UNTIL master_gtid_pos=xxx](#) for more details.

connection_name

If there is only one nameless primary, or the default primary (as specified by the [default_master_connection](#) system variable) is intended, `connection_name` can be omitted. If provided, the `START SLAVE` statement will apply to the specified primary. `connection_name` is case-insensitive.

MariaDB starting with [10.7.0](#)

The `FOR CHANNEL` keyword was added for MySQL compatibility. This is identical as using the `channel_name` directly after `START SLAVE`.

START ALL SLAVES

`START ALL SLAVES` starts all configured replicas (replicas with `master_host` not empty) that were not started before. It will give a `note` for all started connections. You can check the notes with [SHOW WARNINGS](#).

START REPLICA

MariaDB starting with [10.5.1](#)

`START REPLICA` is an alias for `START SLAVE` from [MariaDB 10.5.1](#).

See Also

- [Setting up replication](#).
- [CHANGE MASTER TO](#) is used to create and change connections.
- [STOP SLAVE](#) is used to stop a running connection.
- [RESET SLAVE](#) is used to reset parameters for a connection and also to permanently delete a primary connection.

H3Dr1.2.5.3 STOP SLAVE

The terms *master* and *slave* have historically been used in replication, but the terms *primary* and *replica* are now preferred. The old terms are used still used in parts of the documentation, and in MariaDB commands, although MariaDB 10.5 has begun the process of renaming. The documentation process is ongoing. See [MDEV-18777](#) to follow progress on this effort.

Syntax

```
STOP SLAVE ["connection_name"] [thread_type [, thread_type] ... ] [FOR CHANNEL "connection_n  
STOP ALL SLAVES [thread_type [, thread_type]]  
STOP REPLICA ["connection_name"] [thread_type [, thread_type] ... ] -- from 10.5.1  
STOP ALL REPLICAS [thread_type [, thread_type]] -- from 10.5.1  
thread_type: IO_THREAD | SQL_THREAD
```

Contents

1. [Syntax](#)
2. [Description](#)
 1. [STOP ALL SLAVES](#)
 2. [connection_name](#)
 3. [STOP REPLICA](#)
3. [See Also](#)

Description

Stops the replica threads. `STOP SLAVE` requires the [SUPER](#) privilege, or, from [MariaDB 10.5.2](#), the [REPLICATION SLAVE ADMIN](#) privilege.

Like [START SLAVE](#), this statement may be used with the `IO_THREAD` and `SQL_THREAD` options to name the thread or threads to be stopped. In almost all cases, one never need to use the `thread_type` options.

`STOP SLAVE` waits until any current replication event group affecting one or more non-transactional tables has finished executing (if there is any such replication group), or until the user issues a [KILL QUERY](#) or [KILL CONNECTION](#) statement.

Note that `STOP SLAVE` doesn't delete the connection permanently. Next time you execute [START SLAVE](#) or the MariaDB server restarts, the replica connection is restored with its [original arguments](#). If you want to delete a connection, you should execute [RESET SLAVE](#).

STOP ALL SLAVES

`STOP ALL SLAVES` stops all your running replicas. It will give you a `note` for every stopped connection. You can check the notes with [SHOW WARNINGS](#).

connection_name

The `connection_name` option is used for [multi-source replication](#).

If there is only one nameless master, or the default master (as specified by the [default_master_connection](#) system variable) is intended, `connection_name` can be omitted. If provided, the `STOP SLAVE` statement will apply to the specified master. `connection_name` is case-insensitive.

MariaDB starting with [10.7.0](#)

The `FOR CHANNEL` keyword was added for MySQL compatibility. This is identical as using the `channel_name` directly after `STOP SLAVE`.

STOP REPLICA

MariaDB starting with [10.5.1](#)

`STOP REPLICA` is an alias for `STOP SLAVE` from [MariaDB 10.5.1](#).

See Also

- [CHANGE MASTER TO](#) is used to create and change connections.
- [START SLAVE](#) is used to start a predefined connection.
- [RESET SLAVE](#) is used to reset parameters for a connection and also to permanently delete a master connection.

H3Dr1.2.5.4 RESET REPLICA/SLAVE

The terms *master* and *slave* have historically been used in replication, but the terms *primary* and *replica* are now preferred. The old terms are used still used in parts of the documentation, and in MariaDB commands, although MariaDB 10.5 has begun the process of renaming. The

Syntax

```
RESET REPLICA ["connection_name"] [ALL] [FOR CHANNEL "connection_name"] -- from MariaDB  
10.5.1  
RESET SLAVE ["connection_name"] [ALL] [FOR CHANNEL "connection_name"]
```

Contents

1. [Syntax](#)
2. [Description](#)
 1. [connection_name](#)
 2. [RESET REPLICA](#)
3. [See Also](#)

Description

RESET REPLICA/SLAVE makes the replica forget its [replication](#) position in the master's [binary log](#). This statement is meant to be used for a clean start. It deletes the master.info and relay-log.info files, all the [relay log](#) files, and starts a new relay log file. To use RESET REPLICA/SLAVE, the replica threads must be stopped (use [STOP REPLICA/SLAVE](#) if necessary).

Note: All relay log files are deleted, even if they have not been completely executed by the slave SQL thread. (This is a condition likely to exist on a replication slave if you have issued a [STOP REPLICA/SLAVE](#) statement or if the slave is highly loaded.)

Note: `RESET REPLICA` does not reset the global `gtid_slave_pos` variable. This means that a replica server configured with `CHANGE MASTER TO MASTER_USE_GTID=slave_pos` will not receive events with GTIDs occurring before the state saved in `gtid_slave_pos`. If the intent is to reprocess these events, `gtid_slave_pos` must be manually reset, e.g. by executing `set global gtid_slave_pos=""`.

Connection information stored in the master.info file is immediately reset using any values specified in the corresponding startup options. This information includes values such as master host, master port, master user, and master password. If the replica SQL thread was in the middle of replicating temporary tables when it was stopped, and RESET REPLICA/SLAVE is issued, these replicated temporary tables are deleted on the slave.

The `ALL` also resets the `PORT`, `HOST`, `USER` and `PASSWORD` parameters for the slave. If you are using a connection name, it will permanently delete it and it will not show up anymore in [SHOW ALL REPLICAS/SLAVE STATUS](#).

connection_name

The `connection_name` option is used for [multi-source replication](#).

If there is only one nameless primary, or the default primary (as specified by the `default_master_connection` system variable) is intended, `connection_name` can be omitted. If provided, the `RESET REPLICA/SLAVE` statement will apply to the specified primary. `connection_name` is case-insensitive.

MariaDB starting with [10.7.0](#)

The `FOR CHANNEL` keyword was added for MySQL compatibility. This is identical as using the `channel_name` directly after `RESET REPLICA`.

RESET REPLICA

MariaDB starting with [10.5.1](#)

RESET REPLICA is an alias for RESET SLAVE from [MariaDB 10.5.1](#).

See Also

- [STOP REPLICA/SLAVE](#) stops the replica, but it can be restarted with [START REPLICA/SLAVE](#) or after next MariaDB server restart.

H3Dr1.2.5.5 SET GLOBAL SQL_SLAVE_SKIP_COUNTER

Syntax

```
SET GLOBAL sql_slave_skip_counter = N
```

Contents

1. [Syntax](#)
2. [Description](#)
3. [Example](#)
4. [Multiple Replication Domains](#)
5. [See Also](#)

Description

This statement skips the next `N` events from the primary. This is useful for recovering from [replication](#) stops caused by a statement.

If multi-source replication is used, this statement applies to the default connection. It could be necessary to change the value of the [default_master_connection](#) system variable.

Note that, if the event is a [transaction](#), the whole transaction will be skipped. With non-transactional engines, an event is always a single statement.

This statement is valid only when the replica threads are not running. Otherwise, it produces an error.

The statement does not automatically restart the replica threads.

Example

```
SHOW SLAVE STATUS \G
...
SET GLOBAL sql_slave_skip_counter = 1;
START SLAVE;
```

Multi-source replication:

```
SET @@default_master_connection = 'master_01';
SET GLOBAL SQL_SLAVE_SKIP_COUNTER = 1;
START SLAVE;
```

Multiple Replication Domains

`sql_slave_skip_counter` can't be used to skip transactions on a replica if [GTID replication](#) is in use and if [gtid_slave_pos](#) contains multiple [gtid_domain_id](#) values. In that case, you'll get an error like the following:

ERROR 1966 (HY000): When using parallel replication and GTID with multiple replication domains, @@sql_slave_skip_counter can not be used. Instead, setting @gtid_slave_pos explicitly can be used to skip to after a given GTID position.

In order to skip transactions in cases like this, you will have to manually change [gtid_slave_pos](#).

See Also

- [Selectively Skipping Replication of Binlog Events](#)

H3Dr1.2.5.6 SHOW RELAYLOG EVENTS

The terms *master* and *slave* have historically been used in replication, but the terms *primary* and *replica* are now preferred. The old terms are used still used in parts of the documentation, and in MariaDB commands, although MariaDB 10.5 has begun the process of renaming. The documentation process is ongoing. See [MDEV-18777](#) to follow progress on this effort.

Syntax

```
SHOW RELAYLOG [ 'connection_name' ] EVENTS  
[ IN 'log_name' ] [FROM pos] [LIMIT [offset,] row_count]  
[ FOR CHANNEL 'channel_name' ]
```

Description

On [replicas](#), this command shows the events in the [relay log](#). If 'log_name' is not specified, the first relay log is shown.

Syntax for the `LIMIT` clause is the same as for [SELECT ... LIMIT](#).

Using the `LIMIT` clause is highly recommended because the `SHOW RELAYLOG EVENTS` command returns the complete contents of the relay log, which can be quite large.

This command does not return events related to setting user and system variables. If you need those, use [mariadb-binlog/mysqlbinlog](#).

On the primary, this command does nothing.

Requires the `REPLICA MONITOR` privilege (>= MariaDB 10.5.9), the `REPLICATION SLAVE ADMIN` privilege (>= MariaDB 10.5.2) or the `REPLICATION SLAVE` privilege (<= MariaDB 10.5.1).

connection_name

If there is only one nameless primary, or the default primary (as specified by the `default_master_connection` system variable) is intended, `connection_name` can be omitted. If provided, the `SHOW RELAYLOG` statement will apply to the specified primary. `connection_name` is case-insensitive.

MariaDB starting with 10.7.0

The `FOR CHANNEL` keyword was added for MySQL compatibility. This is identical as using the `channel_name` directly after `SHOW RELAYLOG`.

H3Dr1.2.5.7 SHOW SLAVE STATUS

Syntax

```
SHOW SLAVE ["connection_name"] STATUS [FOR CHANNEL "connection_name"]
SHOW REPLICA ["connection_name"] STATUS -- From MariaDB 10.5.1
```

or

```
SHOW ALL SLAVES STATUS
SHOW ALL REPLICAS STATUS -- From MariaDB 10.5.1
```

Contents

1. [Syntax](#)
2. [Description](#)
 1. [Multi-Source](#)
 2. [Column Descriptions](#)
 3. [SHOW REPLICA STATUS](#)
3. [Examples](#)
4. [See Also](#)

Description

This statement is to be run on a replica and provides status information on essential parameters of the [replica](#) threads.

This statement requires the [SUPER](#) privilege, the [REPLICATION_CLIENT](#) privilege, or, from MariaDB 10.5.2, the [REPLICATION SLAVE ADMIN](#) privilege, or, from MariaDB 10.5.9, the [REPLICA MONITOR](#) privilege.

Multi-Source

The `FULL` and `"connection_name"` options allow you to connect to [many primaries at the same time](#).

`ALL SLAVES` (or `ALL REPLICAS` from [MariaDB 10.5.1](#)) gives you a list of all connections to the primary nodes.

The rows will be sorted according to `Connection_name`.

If you specify a `connection_name`, you only get the information about that connection. If `connection_name` is not used, then the name set by `default_master_connection` is used. If the connection name doesn't exist you will get an error: `There is no master connection for 'xxx'`.

MariaDB starting with [10.7.0](#)

The `FOR CHANNEL` keyword was added for MySQL compatibility. This is identical as using the `channel_name` directly after `SHOW SLAVE`.

Column Descriptions

Name	Description	Added
Connection_name	Name of the primary connection. Returned with <code>SHOW ALL SLAVES STATUS</code> (or <code>SHOW ALL REPLICAS STATUS</code> from MariaDB 10.5.1) only.	
Slave_SQL_State	State of SQL thread. Returned with <code>SHOW ALL SLAVES STATUS</code> (or <code>SHOW ALL REPLICAS STATUS</code> from MariaDB 10.5.1) only. See Slave SQL Thread States .	
Slave_IO_State	State of I/O thread. See Slave I/O Thread States .	
Master_host	Master host that the replica is connected to.	

Master_user	Account user name being used to connect to the primary.	
Master_port	The port being used to connect to the primary.	
Connect_Retry	Time in seconds between retries to connect. The default is 60. The CHANGE MASTER TO statement can set this. The master-retry-count option determines the maximum number of reconnection attempts.	
Master_Log_File	Name of the primary binary log file that the I/O thread is currently reading from.	
Read_Master_Log_Pos	Position up to which the I/O thread has read in the current primary binary log file.	
Relay_Log_File	Name of the relay log file that the SQL thread is currently processing.	
Relay_Log_Pos	Position up to which the SQL thread has finished processing in the current relay log file.	
Relay_Master_Log_File	Name of the primary binary log file that contains the most recent event executed by the SQL thread.	
Slave_IO_Running	Whether the replica I/O thread is running and connected (<code>Yes</code>), running but not connected to a primary (<code>Connecting</code>) or not running (<code>No</code>).	
Slave_SQL_Running	Whether or not the SQL thread is running.	
Replicate_Do_DB	Databases specified for replicating with the replicate_do_db option.	
Replicate_Ignore_DB	Databases specified for ignoring with the replicate_ignore_db option.	
Replicate_Do_Table	Tables specified for replicating with the replicate_do_table option.	
Replicate_Ignore_Table	Tables specified for ignoring with the replicate_ignore_table option.	
Replicate_Wild_Do_Table	Tables specified for replicating with the replicate_wild_do_table option.	
Replicate_Wild_Ignore_Table	Tables specified for ignoring with the replicate_wild_ignore_table option.	
Last_Error	Alias for <code>Last_SQL_Error</code> (see below)	
Skip_Counter	Number of events that a replica skips from the master, as recorded in the sql_slave_skip_counter system variable.	
Exec_Master_Log_Pos	Position up to which the SQL thread has processed in the current master binary log file. Can be used to start a new replica from a current replica with the CHANGE MASTER TO ... MASTER_LOG_POS option.	
Relay_Log_Space	Total size of all relay log files combined.	
Until_Condition		
Until_Log_File	The <code>MASTER_LOG_FILE</code> value of the START SLAVE UNTIL condition.	
Until_Log_Pos	The <code>MASTER_LOG_POS</code> value of the START SLAVE UNTIL condition.	

Master_SSL_Allowed	Whether an SSL connection is permitted (Yes), not permitted (No) or permitted but without the replica having SSL support enabled (Ignored)	
Master_SSL_CA_File	The MASTER_SSL_CA option of the <code>CHANGE MASTER TO</code> statement.	
Master_SSL_CA_Path	The MASTER_SSL_CAPATH option of the <code>CHANGE MASTER TO</code> statement.	
Master_SSL_Cert	The MASTER_SSL_CERT option of the <code>CHANGE MASTER TO</code> statement.	
Master_SSL_Cipher	The MASTER_SSL_CIPHER option of the <code>CHANGE MASTER TO</code> statement.	
Master_SSL_Key	The MASTER_SSL_KEY option of the <code>CHANGE MASTER TO</code> statement.	
Seconds_Behind_Master	Difference between the timestamp logged on the master for the event that the replica is currently processing, and the current timestamp on the replica. Zero if the replica is not currently processing an event. With <code>parallel replication</code> , <code>seconds_behind_master</code> is updated only after transactions commit.	
Master_SSL_Verify_Server_Cert	The MASTER_SSL_VERIFY_SERVER_CERT option of the <code>CHANGE MASTER TO</code> statement.	
Last_IO_Errorno	Error code of the most recent error that caused the I/O thread to stop (also recorded in the replica's error log). 0 means no error. <code>RESET SLAVE</code> or <code>RESET MASTER</code> will reset this value.	
Last_IO_Error	Error message of the most recent error that caused the I/O thread to stop (also recorded in the replica's error log). An empty string means no error. <code>RESET SLAVE</code> or <code>RESET MASTER</code> will reset this value.	
Last_SQL_Errorno	Error code of the most recent error that caused the SQL thread to stop (also recorded in the replica's error log). 0 means no error. <code>RESET SLAVE</code> or <code>RESET MASTER</code> will reset this value.	
Last_SQL_Error	Error message of the most recent error that caused the SQL thread to stop (also recorded in the replica's error log). An empty string means no error. <code>RESET SLAVE</code> or <code>RESET MASTER</code> will reset this value.	
Replicate_Ignore_Server_Ids	List of <code>server_ids</code> that are currently being ignored for replication purposes, or an empty string for none, as specified in the IGNORE_SERVER_IDS option of the <code>CHANGE MASTER TO</code> statement.	
Master_Server_Id	The master's <code>server_id</code> value.	
Master_SSL_Crl	The MASTER_SSL_CRL option of the <code>CHANGE MASTER TO</code> statement.	
Master_SSL_Crlpath	The MASTER_SSL_CRLPATH option of the <code>CHANGE MASTER TO</code> statement.	
Using_Gtid	Whether or not global transaction ID's are being used for replication (can be No , Slave_Pos , or Current_Pos).	
Gtid_IO_Pos	Current global transaction ID value.	
Retried_transactions	Number of retried transactions for this connection. Returned with <code>SHOW ALL SLAVES STATUS</code> only.	

Max_relay_log_size	Max relay log size for this connection. Returned with <code>SHOW ALL SLAVES STATUS</code> only.	
Executed_log_entries	How many log entries the replica has executed. Returned with <code>SHOW ALL SLAVES STATUS</code> only.	
Slave_received_heartbeats	How many heartbeats we have got from the master. Returned with <code>SHOW ALL SLAVES STATUS</code> only.	
Slave_heartbeat_period	How often to request a heartbeat packet from the master (in seconds). Returned with <code>SHOW ALL SLAVES STATUS</code> only.	
Gtid_Slave_Pos	GTID of the last event group replicated on a replica server, for each replication domain, as stored in the gtid_slave_pos system variable. Returned with <code>SHOW ALL SLAVES STATUS</code> only.	
SQL_Delay	Value specified by <code>MASTER_DELAY</code> in <code>CHANGE MASTER</code> (or 0 if none).	MariaDB 10.2.3
SQL_Remaining_Delay	When the replica is delaying the execution of an event due to <code>MASTER_DELAY</code> , this is the number of seconds of delay remaining before the event will be applied. Otherwise, the value is <code>NULL</code> .	MariaDB 10.2.3
Slave_SQL_Running_State	The state of the SQL driver threads, same as in <code>SHOW PROCESSLIST</code> . When the replica is delaying the execution of an event due to <code>MASTER_DELAY</code> , this field displays: "Waiting until <code>MASTER_DELAY</code> seconds after master executed event".	MariaDB 10.2.3
Slave_DDL_Groups	This status variable counts the occurrence of DDL statements. This is a replica-side counter for optimistic parallel replication.	MariaDB 10.3.7
Slave_Non_Transactional_Groups	This status variable counts the occurrence of non-transactional event groups. This is a replica-side counter for optimistic parallel replication.	MariaDB 10.3.7
Slave_Transactional_Groups	This status variable counts the occurrence of transactional event groups. This is a replica-side counter for optimistic parallel replication.	MariaDB 10.3.7

SHOW REPLICAS STATUS

MariaDB starting with [10.5.1](#)

`SHOW REPLICAS STATUS` is an alias for `SHOW SLAVE STATUS` from [MariaDB 10.5.1](#).

Examples

If you issue this statement using the [mysql](#) client, you can use a `\G` statement terminator rather than a semicolon to obtain a more readable vertical layout.

```
SHOW SLAVE STATUS\G
***** 1. row *****
Slave_IO_State: Waiting for master to send event
Master_Host: db01.example.com
Master_User: replicant
Master_Port: 3306
Connect_Retry: 60
Master_Log_File: mariadb-bin.000010
Read_Master_Log_Pos: 548
Relay_Log_File: relay-bin.000004
Relay_Log_Pos: 837
Relay_Master_Log_File: mariadb-bin.000010
Slave_IO_Running: Yes
Slave_SQL_Running: Yes
Replicate_Do_DB:
Replicate_Ignore_DB:
Replicate_Do_Table:
Replicate_Ignore_Table:
Replicate_Wild_Do_Table:
Replicate_Wild_Ignore_Table:
Last_Error:
Skip_Counter: 0
Last_Erno: 0
Exec_Master_Log_Pos: 548
Relay_Log_Space: 1497
Until_Condition: None
Until_Log_File:
Until_Log_Pos: 0
Master_SSL_Allowed: No
Master_SSL_CA_File:
Master_SSL_CA_Path:
Master_SSL_Cert:
Master_SSL_Cipher:
Master_SSL_Key:
Seconds_Behind_Master: 0
Master_SSL_Verify_Server_Cert: No
Last_IO_Erno: 0
Last_IO_Error:
Last_SQL_Erno: 0
Last_SQL_Error:
Replicate_Ignore_Server_Ids:
Master_Server_Id: 101
Master_SSL_Crl:
Master_SSL_Crlpath:
Using_Gtid: No
Gtid_IO_Pos:
```

```

SHOW ALL SLAVES STATUS\G
*****
1. row *****
Connection_name:
Slave_SQL_State: Slave has read all relay log; waiting for the slave I/O
thread to update it
Slave_IO_State: Waiting for master to send event
Master_Host: db01.example.com
Master_User: replicant
Master_Port: 3306
Connect_Retry: 60
Master_Log_File: mariadb-bin.000010
Read_Master_Log_Pos: 3608
Relay_Log_File: relay-bin.000004
Relay_Log_Pos: 3897
Relay_Master_Log_File: mariadb-bin.000010
Slave_IO_Running: Yes
Slave_SQL_Running: Yes
Replicate_Do_DB:
Replicate_Ignore_DB:
Replicate_Do_Table:
Replicate_Ignore_Table:
Replicate_Wild_Do_Table:
Replicate_Wild_Ignore_Table:
Last_Error:
Skip_Counter: 0
Exec_Master_Log_Pos: 3608
Relay_Log_Space: 4557
Until_Condition: None
Until_Log_File:
Until_Log_Pos: 0
Master_SSL_Allowed: No
Master_SSL_CA_File:
Master_SSL_CA_Path:
Master_SSL_Cert:
Master_SSL_Cipher:
Master_SSL_Key:
Seconds_Behind_Master: 0
Master_SSL_Verify_Server_Cert: No
Last_IO_Error:
Last_SQL_Error:
Last_SQL_Erno: 0
Last_SQL_Error:
Replicate_Ignore_Servers:
Master_Server_Id: 101
Master_SSL_Crl:
Master_SSL_Crlpath:
Using_Gtid: No
Gtid_IO_Pos:
Retried_transactions: 0
Max_relay_log_size: 104857600
Executed_log_entries: 40
Slave_received_heartbeats: 11
Slave_heartbeat_period: 1800.000
Gtid_Slave_Pos: 0-101-2320

```

You can also access some of the variables directly from status variables:

```
SET @@default_master_connection="test" ;
show status like "%slave%"

Variable_name      Value
Com_show_slave_hosts      0
Com_show_slave_status      0
Com_start_all_slaves      0
Com_start_slave 0
Com_stop_all_slaves      0
Com_stop_slave 0
Rpl_semi_sync_slave_status      OFF
Slave_connections      0
Slave_heartbeat_period    1800.000
Slave_open_temp_tables    0
Slave_received_heartbeats    0
Slave_retried_transactions    0
Slave_running  OFF
Slaves_connected      0
Slaves_running   1
```

See Also

- [MariaDB replication](#)

H3Dr1.2.5.8 SHOW MASTER STATUS

Syntax

```
SHOW MASTER STATUS
SHOW BINLOG STATUS -- From MariaDB 10.5.2
```

Description

Provides status information about the [binary log](#) files of the primary.

This statement requires the [SUPER](#) privilege, the [REPLICATION_CLIENT](#) privilege, or, from [MariaDB 10.5.2](#), the [BINLOG MONITOR](#) privilege.

To see information about the current GTIDs in the binary log, use the [gtid_binlog_pos](#) variable.

`SHOW MASTER STATUS` was renamed to `SHOW BINLOG STATUS` in [MariaDB 10.5.2](#), but the old name remains an alias for compatibility purposes.

Example

```
SHOW MASTER STATUS;
+-----+-----+-----+-----+
| File          | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+-----+-----+-----+-----+
| mariadb-bin.000016 |      475 |           |           |
+-----+-----+-----+-----+
SELECT @@global.gtid_binlog_pos;
+-----+
| @@global.gtid_binlog_pos |
+-----+
| 0-1-2               |
+-----+
```

See Also

- [MariaDB replication](#)

- [Using and Maintaining the Binary Log](#)
- [The gtid_binlog_pos variable](#)

H3Dr1.2.5.9 SHOW SLAVE HOSTS

Contents

1. [Syntax](#)
2. [Description](#)
 1. [SHOW REPLICAS HOSTS](#)
3. [See Also](#)

Syntax

```
SHOW SLAVE HOSTS
SHOW REPLICAS HOSTS -- from MariaDB 10.5.1
```

Description

This command is run on the primary and displays a list of replicas that are currently registered with it. Only replicas started with the `--report-host=host_name` option are visible in this list.

The list is displayed on any server (not just the primary server). The output looks like this:

```
SHOW SLAVE HOSTS;
+-----+-----+-----+-----+
| Server_id | Host      | Port | Master_id |
+-----+-----+-----+-----+
| 192168010 | iconnect2 | 3306 | 192168011 |
| 1921680101 | athena    | 3306 | 192168011 |
+-----+-----+-----+-----+
```

- **Server_id** : The unique server ID of the replica server, as configured in the server's option file, or on the command line with [--server-id=value](#).
- **Host** : The host name of the replica server, as configured in the server's option file, or on the command line with `--report-host=host_name`. Note that this can differ from the machine name as configured in the operating system.
- **Port** : The port the replica server is listening on.
- **Master_id** : The unique server ID of the primary server that the replica server is replicating from.

Some MariaDB and MySQL versions report another variable, [rpl_recovery_rank](#). This variable was never used, and was eventually removed in [MariaDB 10.1.2](#).

Requires the [REPLICATION MASTER ADMIN](#) privilege (>= [MariaDB 10.5.2](#)) or the [REPLICATION SLAVE](#) privilege (<= [MariaDB 10.5.1](#)).

SHOW REPLICAS HOSTS

MariaDB starting with [10.5.1](#)

`SHOW REPLICAS HOSTS` is an alias for `SHOW SLAVE HOSTS` from [MariaDB 10.5.1](#).

See Also

- [MariaDB replication](#)
- [Replication threads](#)
- [SHOW PROCESSLIST](#). In `SHOW PROCESSLIST` output, replica threads are identified by `Binlog Dump`

H3Dr1.2.5.10 RESET MASTER

RESET MASTER [TO #]

Deletes all [binary log](#) files listed in the index file, resets the binary log index file to be empty, and creates a new binary log file with a suffix of .000001.

If TO # is given, then the first new binary log file will start from number #.

This statement is for use only when the master is started for the first time, and should never be used if any slaves are actively [replicating](#) from the binary log.

See Also

- The [PURGE BINARY LOGS](#) statement is intended for use in active replication.

H3Dr1.2.6 Plugin SQL Statements

[Plugin](#) commands.



SHOW PLUGINS

Display information about installed plugins.



SHOW PLUGINS SONAME

Information about all available plugins, installed or not.



INSTALL PLUGIN

Install a plugin.



UNINSTALL PLUGIN

Remove a single installed plugin.



INSTALL SONAME

Installs all plugins from a given library.



UNINSTALL SONAME

Remove all plugins belonging to a specified library.



mysql_plugin

Tool for enabling or disabling plugins.

H3Dr1.2.6.1 SHOW PLUGINS

Syntax

```
SHOW PLUGINS;
```

Contents

1. [Syntax](#)
2. [Description](#)
3. [Examples](#)
4. [See Also](#)

Description

SHOW PLUGINS displays information about installed [plugins](#). The Library column indicates the plugin library - if it is NULL , the plugin is built-in and cannot be uninstalled.

The [PLUGINS](#) table in the `information_schema` database contains more detailed information.

For specific information about storage engines (a particular type of plugin), see the `information_schema.ENGINES` table and the `SHOW ENGINES` statement.

Examples

SHOW PLUGINS;					
Name	Status	Type	Library	License	
binlog	ACTIVE	STORAGE ENGINE	NULL	GPL	
mysql_native_password	ACTIVE	AUTHENTICATION	NULL	GPL	
mysql_old_password	ACTIVE	AUTHENTICATION	NULL	GPL	
MRG_MyISAM	ACTIVE	STORAGE ENGINE	NULL	GPL	
MyISAM	ACTIVE	STORAGE ENGINE	NULL	GPL	
CSV	ACTIVE	STORAGE ENGINE	NULL	GPL	
MEMORY	ACTIVE	STORAGE ENGINE	NULL	GPL	
FEDERATED	ACTIVE	STORAGE ENGINE	NULL	GPL	
PERFORMANCE_SCHEMA	ACTIVE	STORAGE ENGINE	NULL	GPL	
Aria	ACTIVE	STORAGE ENGINE	NULL	GPL	
InnoDB	ACTIVE	STORAGE ENGINE	NULL	GPL	
INNODB_TRX	ACTIVE	INFORMATION SCHEMA	NULL	GPL	
...					
INNODB_SYS_FOREIGN	ACTIVE	INFORMATION SCHEMA	NULL	GPL	
INNODB_SYS_FOREIGN_COLS	ACTIVE	INFORMATION SCHEMA	NULL	GPL	
SPHINX	ACTIVE	STORAGE ENGINE	NULL	GPL	
ARCHIVE	ACTIVE	STORAGE ENGINE	NULL	GPL	
BLACKHOLE	ACTIVE	STORAGE ENGINE	NULL	GPL	
FEEDBACK	DISABLED	INFORMATION SCHEMA	NULL	GPL	
partition	ACTIVE	STORAGE ENGINE	NULL	GPL	
pam	ACTIVE	AUTHENTICATION	auth_pam.so	GPL	

See Also

- [List of Plugins](#)
- [Plugin Overview](#)
- [INFORMATION_SCHEMA.PLUGINS Table](#)
- [INSTALL PLUGIN](#)
- [INFORMATION_SCHEMA.ALL_PLUGINS Table](#) (all plugins, installed or not)
- [INSTALL SONAME](#)
- [UNINSTALL PLUGIN](#)
- [UNINSTALL SONAME](#)

H3Dr1.2.6.2 SHOW PLUGINS SONAME

Syntax

```
SHOW PLUGINS SONAME { library | LIKE 'pattern' | WHERE expr };
```

Description

`SHOW PLUGINS SONAME` displays information about compiled-in and all server plugins in the `plugin_dir` directory, including plugins that haven't been installed.

Examples

```
SHOW PLUGINS SONAME 'ha_example.so';
+-----+-----+-----+-----+-----+
| Name      | Status       | Type          | Library      | License     |
+-----+-----+-----+-----+-----+
| EXAMPLE   | NOT INSTALLED | STORAGE ENGINE | ha_example.so | GPL          |
| UNUSABLE  | NOT INSTALLED | DAEMON        | ha_example.so | GPL          |
+-----+-----+-----+-----+
```

There is also a corresponding `information_schema` table, called [ALL_PLUGINS](#), which contains more complete information.

H3Dr1.2.6.3 INSTALL PLUGIN

Syntax

```
INSTALL PLUGIN [IF NOT EXISTS] plugin_name SONAME 'plugin_library'
```

Contents

1. [Syntax](#)
2. [Description](#)
 1. [IF NOT EXISTS](#)
3. [Examples](#)
4. [See Also](#)

Description

This statement installs an individual [plugin](#) from the specified library. To install the whole library (which could be required), use [INSTALL SONAME](#). See also [Installing a Plugin](#).

`plugin_name` is the name of the plugin as defined in the plugin declaration structure contained in the library file. Plugin names are not case sensitive. For maximal compatibility, plugin names should be limited to ASCII letters, digits, and underscore, because they are used in C source files, shell command lines, M4 and Bourne shell scripts, and SQL environments.

`plugin_library` is the name of the shared library that contains the plugin code. The file name extension can be omitted (which makes the statement look the same on all architectures).

The shared library must be located in the plugin directory (that is, the directory named by the [plugin_dir](#) system variable). The library must be in the plugin directory itself, not in a subdirectory. By default, `plugin_dir` is plugin directory under the directory named by the `pkglibdir` configuration variable, but it can be changed by setting the value of `plugin_dir` at server startup. For example, set its value in a `my.cnf` file:

```
[mysqld]
plugin_dir=/path/to/plugin/directory
```

If the value of `plugin_dir` is a relative path name, it is taken to be relative to the MySQL base directory (the value of the `basedir` system variable).

`INSTALL PLUGIN` adds a line to the `mysql.plugin` table that describes the plugin. This table contains the plugin name and library file name.

`INSTALL PLUGIN` causes the server to read option (`my.cnf`) files just as during server startup. This enables the plugin to pick up any relevant options from those files. It is possible to add plugin options to an option file even before loading a plugin (if the loose prefix is used). It is also possible to uninstall a plugin, edit `my.cnf`, and install the plugin again. Restarting the plugin this way enables it to the new option values without a server restart.

`INSTALL PLUGIN` also loads and initializes the plugin code to make the plugin available for use. A plugin is initialized by executing its initialization function, which handles any setup that the plugin must perform before it can be used.

To use `INSTALL PLUGIN`, you must have the [INSERT privilege](#) for the `mysql.plugin` table.

At server startup, the server loads and initializes any plugin that is listed in the `mysql.plugin` table. This means that a plugin is installed with `INSTALL PLUGIN` only once, not every time the server starts. Plugin loading at startup does not occur if the server is started with the `--skip-grant-tables` option.

When the server shuts down, it executes the de-initialization function for each plugin that is loaded so that the plugin has a chance to perform any final cleanup.

If you need to load plugins for a single server startup when the `--skip-grant-tables` option is given (which tells the server not to read system tables), use the `--plugin-load mysqld` option [option](#).

MariaDB starting with [10.4.0](#)

IF NOT EXISTS

When the `IF NOT EXISTS` clause is used, MariaDB will return a note instead of an error if the specified plugin already exists. See [SHOW WARNINGS](#).

Examples

```
INSTALL PLUGIN sphinx SONAME 'ha_sphinx.so';
```

The extension can also be omitted:

```
INSTALL PLUGIN innodb SONAME 'ha_xtradb';
```

From [MariaDB 10.4.0](#):

```
INSTALL PLUGIN IF NOT EXISTS example SONAME 'ha_example';
Query OK, 0 rows affected (0.104 sec)
```

```
INSTALL PLUGIN IF NOT EXISTS example SONAME 'ha_example';
Query OK, 0 rows affected, 1 warning (0.000 sec)
```

```
SHOW WARNINGS;
+-----+-----+
| Level | Code | Message           |
+-----+-----+
| Note  | 1968 | Plugin 'example' already installed |
+-----+-----+
```

See Also

- [List of Plugins](#)
- [Plugin Overview](#)
- [INFORMATION_SCHEMA.PLUGINS Table](#)
- `mysql_plugin`
- `SHOW PLUGINS`
- `INSTALL SONAME`
- `UNINSTALL PLUGIN`
- `UNINSTALL SONAME`

H3Dr1.2.6.4 UNINSTALL PLUGIN

Syntax

```
UNINSTALL PLUGIN [IF EXISTS] plugin_name
```

Contents

1. [Syntax](#)
2. [Description](#)
 1. [IF EXISTS](#)
3. [Examples](#)
4. [See Also](#)

Description

This statement removes a single installed plugin. To uninstall the whole library which contains the plugin, use [UNINSTALL SONAME](#). You cannot uninstall a plugin if any table that uses it is open.

`plugin_name` must be the name of some plugin that is listed in the [mysql.plugin](#) table. The server executes the plugin's deinitialization function and removes the row for the plugin from the `mysql.plugin` table, so that subsequent server restarts will not load and initialize the plugin. [UNINSTALL PLUGIN](#) does not remove the plugin's shared library file.

To use [UNINSTALL PLUGIN](#), you must have the [DELETE](#) privilege for the [mysql.plugin](#) table.

MariaDB starting with [10.4.0](#):

IF EXISTS

If the `IF EXISTS` clause is used, MariaDB will return a note instead of an error if the plugin does not exist. See [SHOW WARNINGS](#).

Examples

`UNINSTALL PLUGIN example;`

From [MariaDB 10.4.0](#):

```
UNINSTALL PLUGIN IF EXISTS example;
Query OK, 0 rows affected (0.099 sec)

UNINSTALL PLUGIN IF EXISTS example;
Query OK, 0 rows affected, 1 warning (0.000 sec)

SHOW WARNINGS;
+-----+-----+
| Level | Code  | Message          |
+-----+-----+
| Note  | 1305  | PLUGIN example does not exist |
+-----+-----+
```

See Also

- [Plugin Overview](#)
- [mysql_plugin](#)
- [INSTALL PLUGIN](#)
- [List of Plugins](#)

H3Dr1.2.6.5 INSTALL SONAME

Syntax

`INSTALL SONAME 'plugin_library'`

Contents

1. [Syntax](#)
2. [Description](#)
3. [Examples](#)
4. [See Also](#)

Description

This statement is a variant of [INSTALL PLUGIN](#). It installs **all** [plugins](#) from a given `plugin_library`. See [INSTALL PLUGIN](#) for details.

`plugin_library` is the name of the shared library that contains the plugin code. The file name extension (for example, `libmyplugin.so` or `libmyplugin.dll`) can be omitted (which makes the statement look the same on all architectures).

The shared library must be located in the plugin directory (that is, the directory named by the `plugin_dir` system variable). The library must be in the plugin directory itself, not in a subdirectory. By default, `plugin_dir` is plugin directory under the directory named by the `pkglibdir` configuration variable, but it can be changed by setting the value of `plugin_dir` at server startup. For example, set its value in a `my.cnf` file:

```
[mysqld]
plugin_dir=/path/to/plugin/directory
```

If the value of `plugin_dir` is a relative path name, it is taken to be relative to the MySQL base directory (the value of the `basedir` system variable).

`INSTALL SONAME` adds one or more lines to the `mysql.plugin` table that describes the plugin. This table contains the plugin name and library file name.

`INSTALL SONAME` causes the server to read option (`my.cnf`) files just as during server startup. This enables the plugin to pick up any relevant options from those files. It is possible to add plugin options to an option file even before loading a plugin (if the loose prefix is used). It is also possible to uninstall a plugin, edit `my.cnf`, and install the plugin again. Restarting the plugin this way enables it to the new option values without a server restart.

`INSTALL SONAME` also loads and initializes the plugin code to make the plugin available for use. A plugin is initialized by executing its initialization function, which handles any setup that the plugin must perform before it can be used.

To use `INSTALL SONAME`, you must have the [INSERT privilege](#) for the `mysql.plugin` table.

At server startup, the server loads and initializes any plugin that is listed in the `mysql.plugin` table. This means that a plugin is installed with `INSTALL SONAME` only once, not every time the server starts. Plugin loading at startup does not occur if the server is started with the `--skip-grant-tables` option.

When the server shuts down, it executes the de-initialization function for each plugin that is loaded so that the plugin has a chance to perform any final cleanup.

If you need to load plugins for a single server startup when the `--skip-grant-tables` option is given (which tells the server not to read system tables), use the `--plugin-load mysql option`.

If you need to install only one plugin from a library, use the [INSTALL PLUGIN](#) statement.

Examples

To load the XtraDB storage engine and all of its `information_schema` tables with one statement, use

```
INSTALL SONAME 'ha_xtradb';
```

This statement can be used instead of `INSTALL PLUGIN` even when the library contains only one plugin:

```
INSTALL SONAME 'ha_sequence';
```

See Also

- [List of Plugins ↗](#)
- [Plugin Overview ↗](#)
- [SHOW PLUGINS](#)
- [INSTALL PLUGIN](#)
- [UNINSTALL PLUGIN](#)
- [UNINSTALL SONAME](#)
- [SHOW PLUGINS](#)
- [INFORMATION_SCHEMA.PLUGINS Table ↗](#)
- [mysql_plugin](#)

H3Dr1.2.6.6 UNINSTALL SONAME

Syntax

```
UNINSTALL SONAME [IF EXISTS] 'plugin_library'
```

Contents

1. [Syntax](#)
2. [Description](#)
 1. [IF EXISTS](#)
3. [Examples](#)
4. [See Also](#)

Description

This statement is a variant of [UNINSTALL PLUGIN](#) statement, that removes all [plugins ↗](#) belonging to a specified `plugin_library`. See [UNINSTALL PLUGIN](#) for details.

`plugin_library` is the name of the shared library that contains the plugin code. The file name extension (for example, `libmyplugin.so` or `libmyplugin.dll`) can be omitted (which makes the statement look the same on all architectures).

To use `UNINSTALL SONAME`, you must have the [DELETE privilege](#) for the `mysql.plugin` table.

MariaDB starting with [10.4.0 ↗](#)

IF EXISTS

If the `IF EXISTS` clause is used, MariaDB will return a note instead of an error if the plugin library does not exist. See [SHOW WARNINGS](#).

Examples

To uninstall the XtraDB plugin and all of its `information_schema` tables with one statement, use

```
UNINSTALL SONAME 'ha_xtradb';
```

From [MariaDB 10.4.0 ↗](#):

```

UNINSTALL SONAME IF EXISTS 'ha_example';
Query OK, 0 rows affected (0.099 sec)

UNINSTALL SONAME IF EXISTS 'ha_example';
Query OK, 0 rows affected, 1 warning (0.000 sec)

SHOW WARNINGS;
+-----+-----+
| Level | Code | Message           |
+-----+-----+
| Note  | 1305 | SONAME ha_example.so does not exist |
+-----+-----+

```

See Also

- [INSTALL SONAME](#)
- [SHOW PLUGINS](#)
- [INSTALL PLUGIN](#)
- [UNINSTALL PLUGIN](#)
- [SHOW PLUGINS](#)
- [INFORMATION_SCHEMA.PLUGINS Table ↗](#)
- [mysql_plugin](#)
- [List of Plugins ↗](#)

H3Dr1.2.6.7 mysql_plugin

H3Dr1.2.7 SET Commands



SET

Set a variable value.



SET CHARACTER SET

Maps all strings sent between the current client and the server with the given mapping.



SET GLOBAL SQL_SLAVE_SKIP_COUNTER

Skips a number of events from the primary.



SET NAMES

The character set used to send statements to the server, and results back to the client.



SET PASSWORD

Assign password to an existing MariaDB user.



SET ROLE

Enable a role.



SET SQL_LOG_BIN

Set binary logging for the current connection.



SET STATEMENT

Set variable values on a per-query basis



SET TRANSACTION

Sets the transaction isolation level.



SET Variable

Used to insert a value into a variable with a code block.

There are 1 related questions ↗.

H3Dr1.2.7.1 SET

Syntax

```
SET variable_assignment [, variable_assignment] ...  
  
variable_assignment:  
    user_var_name = expr  
    | [GLOBAL | SESSION] system_var_name = expr  
    | @@global. | @@session. | @@system_var_name = expr
```

Contents

1. [Syntax](#)
2. [Description](#)
 1. [GLOBAL / SESSION](#)
 2. [DEFAULT](#)
3. [Examples](#)
4. [See Also](#)

One can also set a user variable in any expression with this syntax:

```
user_var_name:= expr
```

Description

The `SET` statement assigns values to different types of variables that affect the operation of the server or your client. Older versions of MySQL employed `SET OPTION`, but this syntax was deprecated in favor of `SET` without `OPTION`, and was removed in [MariaDB 10.0](#).

Changing a system variable by using the `SET` statement does not make the change permanently. To do so, the change must be made in a [configuration file](#).

For setting variables on a per-query basis, see [SET STATEMENT](#).

See [SHOW VARIABLES](#) for documentation on viewing server system variables.

See [Server System Variables](#) for a list of all the system variables.

GLOBAL / SESSION

When setting a system variable, the scope can be specified as either `GLOBAL` or `SESSION`.

A global variable change affects all new sessions. It does not affect any currently open sessions, including the one that made the change.

A session variable change affects the current session only.

If the variable has a session value, not specifying either `GLOBAL` or `SESSION` will be the same as specifying `SESSION`. If the variable only has a global value, not specifying `GLOBAL` or `SESSION` will apply to the change to the global value.

DEFAULT

Setting a global variable to `DEFAULT` will restore it to the server default, and setting a session variable to `DEFAULT` will restore it to the current global value.

Examples

- [innodb_sync_spin_loops](#) is a global variable.
- [skip_parallel_replication](#) is a session variable.

- `max_error_count` is both global and session.

```
SELECT VARIABLE_NAME, SESSION_VALUE, GLOBAL_VALUE FROM
INFORMATION_SCHEMA.SYSTEM_VARIABLES WHERE
VARIABLE_NAME IN ('max_error_count', 'skip_parallel_replication', 'innodb_sync_spin_loops'
+-----+-----+-----+
| VARIABLE_NAME | SESSION_VALUE | GLOBAL_VALUE |
+-----+-----+-----+
| MAX_ERROR_COUNT | 64 | 64 |
| SKIP_PARALLEL_REPLICATION | OFF | NULL |
| INNODB_SYNC_SPIN_LOOPS | NULL | 30 |
+-----+-----+-----+
```

Setting the session values:

```
SET max_error_count=128;Query OK, 0 rows affected (0.000 sec)

SET skip_parallel_replication=ON;Query OK, 0 rows affected (0.000 sec)

SET innodb_sync_spin_loops=60;
ERROR 1229 (HY000): Variable 'innodb_sync_spin_loops' is a GLOBAL variable
and should be set with SET GLOBAL

SELECT VARIABLE_NAME, SESSION_VALUE, GLOBAL_VALUE FROM
INFORMATION_SCHEMA.SYSTEM_VARIABLES WHERE
VARIABLE_NAME IN ('max_error_count', 'skip_parallel_replication', 'innodb_sync_spin_loops'
+-----+-----+-----+
| VARIABLE_NAME | SESSION_VALUE | GLOBAL_VALUE |
+-----+-----+-----+
| MAX_ERROR_COUNT | 128 | 64 |
| SKIP_PARALLEL_REPLICATION | ON | NULL |
| INNODB_SYNC_SPIN_LOOPS | NULL | 30 |
+-----+-----+-----+
```

Setting the global values:

```
SET GLOBAL max_error_count=256;

SET GLOBAL skip_parallel_replication=ON;
ERROR 1228 (HY000): Variable 'skip_parallel_replication' is a SESSION variable
and can't be used with SET GLOBAL

SET GLOBAL innodb_sync_spin_loops=120;

SELECT VARIABLE_NAME, SESSION_VALUE, GLOBAL_VALUE FROM
INFORMATION_SCHEMA.SYSTEM_VARIABLES WHERE
VARIABLE_NAME IN ('max_error_count', 'skip_parallel_replication', 'innodb_sync_spin_loops')
+-----+-----+-----+
| VARIABLE_NAME | SESSION_VALUE | GLOBAL_VALUE |
+-----+-----+-----+
| MAX_ERROR_COUNT | 128 | 256 |
| SKIP_PARALLEL_REPLICATION | ON | NULL |
| INNODB_SYNC_SPIN_LOOPS | NULL | 120 |
+-----+-----+-----+
```

`SHOW VARIABLES` will by default return the session value unless the variable is global only.

```

SHOW VARIABLES LIKE 'max_error_count';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| max_error_count | 128 |
+-----+-----+

SHOW VARIABLES LIKE 'skip_parallel_replication';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| skip_parallel_replication | ON |
+-----+-----+

SHOW VARIABLES LIKE 'innodb_sync_spin_loops';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| innodb_sync_spin_loops | 120 |
+-----+-----+

```

Using the inplace syntax:

```

SELECT (@a:=1);
+-----+
| (@a:=1) |
+-----+
|      1 |
+-----+

SELECT @a;
+-----+
| @a    |
+-----+
|      1 |
+-----+

```

See Also

- [Using last_value\(\) to return data of used rows](#)
- [SET STATEMENT](#)
- [SET Variable](#)
- [SET Data Type](#)
- [DECLARE Variable](#)

H3Dr1.2.7.2 SET CHARACTER SET

Syntax

```

SET {CHARACTER SET | CHARSET}
{charset_name | DEFAULT}

```

Description

Sets the [character_set_client](#) and [character_set_results](#) session system variables to the specified character set and [collation_connection](#) to the value of [collation_database](#), which implicitly sets [character_set_connection](#) to the value of [character_set_database](#).

This maps all strings sent between the current client and the server with the given mapping.

Example

```

SHOW VARIABLES LIKE 'character_set\_%';
+-----+-----+
| Variable_name      | Value   |
+-----+-----+
| character_set_client | utf8    |
| character_set_connection | utf8    |
| character_set_database | latin1  |
| character_set_filesystem | binary  |
| character_set_results | utf8    |
| character_set_server | latin1  |
| character_set_system | utf8    |
+-----+-----+

SHOW VARIABLES LIKE 'collation%';
+-----+-----+
| Variable_name      | Value           |
+-----+-----+
| collation_connection | utf8_general_ci |
| collation_database   | latin1_swedish_ci |
| collation_server     | latin1_swedish_ci |
+-----+-----+

SET CHARACTER SET utf8mb4;

SHOW VARIABLES LIKE 'character_set\_%';
+-----+-----+
| Variable_name      | Value   |
+-----+-----+
| character_set_client | utf8mb4 |
| character_set_connection | latin1  |
| character_set_database | latin1  |
| character_set_filesystem | binary  |
| character_set_results | utf8mb4 |
| character_set_server | latin1  |
| character_set_system | utf8    |
+-----+-----+

SHOW VARIABLES LIKE 'collation%';
+-----+-----+
| Variable_name      | Value           |
+-----+-----+
| collation_connection | latin1_swedish_ci |
| collation_database   | latin1_swedish_ci |
| collation_server     | latin1_swedish_ci |
+-----+-----+

```

See Also

- [SET NAMES](#)

H3Dr1.2.7.3 SET GLOBAL SQL_SLAVE_SKIP_COUNTER

H3Dr1.2.7.4 SET NAMES

Syntax

```

SET NAMES {'charset_name'
           [COLLATE 'collation_name'] | DEFAULT}

```

Contents

1. [Syntax](#)
2. [Description](#)
3. [Examples](#)
4. [See Also](#)

Description

Sets the [character_set_client](#), [character_set_connection](#), [character_set_results](#) and, implicitly, the [collation_connection](#) session system variables to the specified character set and collation.

This determines which [character set](#) the client will use to send statements to the server, and the server will use for sending results back to the client.

`ucs2`, `utf16`, and `utf32` are not valid character sets for `SET NAMES`, as they cannot be used as client character sets.

The collation clause is optional. If not defined (or if `DEFAULT` is specified), the [default collation for the character set](#) will be used.

Quotes are optional for the character set or collation clauses.

Examples

```

SELECT VARIABLE_NAME, SESSION_VALUE
  FROM INFORMATION_SCHEMA.SYSTEM_VARIABLES WHERE
VARIABLE_NAME LIKE 'character_set_c%' OR
VARIABLE_NAME LIKE 'character_set_re%' OR
VARIABLE_NAME LIKE 'collation_c%';
+-----+-----+
| VARIABLE_NAME | SESSION_VALUE |
+-----+-----+
| CHARACTER_SET_RESULTS | utf8 |
| CHARACTER_SET_CONNECTION | utf8 |
| CHARACTER_SET_CLIENT | utf8 |
| COLLATION_CONNECTION | utf8_general_ci |
+-----+-----+


SET NAMES big5;

SELECT VARIABLE_NAME, SESSION_VALUE
  FROM INFORMATION_SCHEMA.SYSTEM_VARIABLES WHERE
VARIABLE_NAME LIKE 'character_set_c%' OR
VARIABLE_NAME LIKE 'character_set_re%' OR
VARIABLE_NAME LIKE 'collation_c%';
+-----+-----+
| VARIABLE_NAME | SESSION_VALUE |
+-----+-----+
| CHARACTER_SET_RESULTS | big5 |
| CHARACTER_SET_CONNECTION | big5 |
| CHARACTER_SET_CLIENT | big5 |
| COLLATION_CONNECTION | big5_chinese_ci |
+-----+-----+


SET NAMES 'latin1' COLLATE 'latin1_bin';

SELECT VARIABLE_NAME, SESSION_VALUE
  FROM INFORMATION_SCHEMA.SYSTEM_VARIABLES WHERE
VARIABLE_NAME LIKE 'character_set_c%' OR
VARIABLE_NAME LIKE 'character_set_re%' OR
VARIABLE_NAME LIKE 'collation_c%';
+-----+-----+
| VARIABLE_NAME | SESSION_VALUE |
+-----+-----+
| CHARACTER_SET_RESULTS | latin1 |
| CHARACTER_SET_CONNECTION | latin1 |
| CHARACTER_SET_CLIENT | latin1 |
| COLLATION_CONNECTION | latin1_bin |
+-----+-----+


SET NAMES DEFAULT;

SELECT VARIABLE_NAME, SESSION_VALUE
  FROM INFORMATION_SCHEMA.SYSTEM_VARIABLES WHERE
VARIABLE_NAME LIKE 'character_set_c%' OR
VARIABLE_NAME LIKE 'character_set_re%' OR
VARIABLE_NAME LIKE 'collation_c%';
+-----+-----+
| VARIABLE_NAME | SESSION_VALUE |
+-----+-----+
| CHARACTER_SET_RESULTS | latin1 |
| CHARACTER_SET_CONNECTION | latin1 |
| CHARACTER_SET_CLIENT | latin1 |
| COLLATION_CONNECTION | latin1_swedish_ci |
+-----+-----+

```

See Also

- [SET CHARACTER SET](#)
- [Character Sets and Collations ↗](#)

H3Dr1.2.7.6 SET ROLE

H3Dr1.2.7.7 SET SQL_LOG_BIN

Syntax

```
SET [SESSION] sql_log_bin = {0|1}
```

Description

Sets the [sql_log_bin](#) system variable, which disables or enables [binary logging](#) for the current connection, if the client has the [SUPER privilege](#). The statement is refused with an error if the client does not have that privilege.

Before [MariaDB 5.5](#) and before MySQL 5.6 one could also set `sql_log_bin` as a global variable. This was disabled as this was too dangerous as it could damage replication.

H3Dr1.2.7.8 SET STATEMENT

Contents

1. [Syntax](#)
2. [Description](#)
3. [Examples](#)
4. [Limitations](#)
5. [Source](#)

MariaDB starting with [10.1.2](#)

Per-query variables were introduced in [MariaDB 10.1.2](#)

`SET STATEMENT` can be used to set the value of a system variable for the duration of the statement. It is also possible to set multiple variables.

Syntax

```
SET STATEMENT var1=value1 [, var2=value2, ...]  
FOR <statement>
```

where `varN` is a system variable (list of allowed variables is provided below), and `valueN` is a constant literal.

Description

```
SET STATEMENT var1=value1 FOR stmt
```

is roughly equivalent to

```
SET @save_value=@var1;  
SET SESSION var1=value1;  
stmt;  
SET SESSION var1=@save_value;
```

The server parses the whole statement before executing it, so any variables set in this fashion that affect the parser may not have the expected effect. Examples include the charset variables, `sql_mode=ansi_quotes`, etc.

Examples

One can limit statement execution time `max_statement_time` :

```
SET STATEMENT max_statement_time=1000 FOR SELECT ... ;
```

One can switch on/off individual optimizations:

```
SET STATEMENT optimizer_switch='materialization=off' FOR SELECT ....;
```

It is possible to enable MRR/BKA for a query:

```
SET STATEMENT join_cache_level=6, optimizer_switch='mrr=on' FOR SELECT ...
```

Note that it makes no sense to try to set a session variable inside a `SET STATEMENT` :

```
#USELESS STATEMENT
SET STATEMENT sort_buffer_size = 100000 for SET SESSION sort_buffer_size = 200000;
```

For the above, after setting `sort_buffer_size` to 200000 it will be reset to its original state (the state before the `SET STATEMENT` started) after the statement execution.

Limitations

There are a number of variables that cannot be set on per-query basis. These include:

- `autocommit`
- `character_set_client`
- `character_set_connection`
- `character_set_filesystem`
- `collation_connection`
- `default_master_connection`
- `debug_sync`
- `interactive_timeout`
- `gtid_domain_id`
- `last_insert_id`
- `log_slow_filter`
- `log_slow_rate_limit`
- `log_slow_verbosity`
- `long_query_time`
- `min_examined_row_limit`
- `profiling`
- `profiling_history_size`
- `query_cache_type`
- `rand_seed1`
- `rand_seed2`
- `skip_replication`
- `slow_query_log`
- `sql_log_off`
- `tx_isolation`
- `wait_timeout`

Source

- The feature was originally implemented as a Google Summer of Code 2009 project by Joseph Lukas.
- Percona Server 5.6 included it as [Per-query variable statement](#)
- MariaDB ported the patch and fixed *many* bugs. The task in MariaDB Jira is [MDEV-5231](#).

H3Dr1.2.7.9 SET TRANSACTION

Syntax

```
SET [GLOBAL | SESSION] TRANSACTION  
    transaction_property [, transaction_property] ...  
  
transaction_property:  
    ISOLATION LEVEL level  
    | READ WRITE  
    | READ ONLY  
  
level:  
    REPEATABLE READ  
    | READ COMMITTED  
    | READ UNCOMMITTED  
    | SERIALIZABLE
```

Contents

1. [Syntax](#)
2. [Description](#)
 1. [Isolation Level](#)
 2. [Isolation Levels](#)
 1. [READ UNCOMMITTED](#)
 2. [READ COMMITTED](#)
 3. [REPEATABLE READ](#)
 4. [SERIALIZABLE](#)
 3. [Access Mode](#)
 3. [Examples](#)

Description

This statement sets the transaction isolation level or the transaction access mode globally, for the current session, or for the next transaction:

- With the `GLOBAL` keyword, the statement sets the default transaction level globally for all subsequent sessions. Existing sessions are unaffected.
- With the `SESSION` keyword, the statement sets the default transaction level for all subsequent transactions performed within the current session.
- Without any `SESSION` or `GLOBAL` keyword, the statement sets the isolation level for the next (not started) transaction performed within the current session.

A change to the global default isolation level requires the `SUPER` privilege. Any session is free to change its session isolation level (even in the middle of a transaction), or the isolation level for its next transaction.

Isolation Level

To set the global default isolation level at server startup, use the `--transaction-isolation=level` option on the command line or in an option file. Values of level for this option use dashes rather than spaces, so the allowable values are `READ-UNCOMMITTED`, `READ-COMMITTED`, `REPEATABLE-READ`, or `SERIALIZABLE`. For example, to set the default isolation level to `REPEATABLE READ`, use these lines in the `[mysqld]` section of an option file:

```
[mysqld]  
transaction-isolation = REPEATABLE-READ
```

To determine the global and session transaction isolation levels at runtime, check the value of the `tx_isolation` system variable:

```
SELECT @@GLOBAL.tx_isolation, @@tx_isolation;
```

InnoDB supports each of the translation isolation levels described here using different locking strategies. The default level is `REPEATABLE READ`. For additional information about InnoDB record-level locks and how it uses them to execute various types of statements, see [InnoDB Lock Modes](#), and <http://dev.mysql.com/doc/refman/en/innodb-locks-set.html>.

Isolation Levels

The following sections describe how MariaDB supports the different transaction levels.

READ UNCOMMITTED

`SELECT` statements are performed in a non-locking fashion, but a possible earlier version of a row might be used. Thus, using this isolation level, such reads are not consistent. This is also called a "dirty read." Otherwise, this isolation level works like `READ COMMITTED`.

READ COMMITTED

A somewhat Oracle-like isolation level with respect to consistent (non-locking) reads: Each consistent read, even within the same transaction, sets and reads its own fresh snapshot. See <http://dev.mysql.com/doc/refman/en/innodb-consistent-read.html>.

For locking reads (`SELECT` with `FOR UPDATE` or `LOCK IN SHARE MODE`), InnoDB locks only index records, not the gaps before them, and thus allows the free insertion of new records next to locked records. For `UPDATE` and `DELETE` statements, locking depends on whether the statement uses a unique index with a unique search condition (such as `WHERE id = 100`), or a range-type search condition (such as `WHERE id > 100`). For a unique index with a unique search condition, InnoDB locks only the index record found, not the gap before it. For range-type searches, InnoDB locks the index range scanned, using gap locks or next-key (gap plus index-record) locks to block insertions by other sessions into the gaps covered by the range. This is necessary because "phantom rows" must be blocked for MySQL replication and recovery to work.

Note: If the `READ COMMITTED` isolation level is used or the `innodb_locks_unsafe_for_binlog` system variable is enabled, there is no InnoDB gap locking except for `foreign-key` constraint checking and duplicate-key checking. Also, record locks for non-matching rows are released after MariaDB has evaluated the `WHERE` condition. If you use `READ COMMITTED` or enable `innodb_locks_unsafe_for_binlog`, you must use row-based binary logging.

REPEATABLE READ

This is the default isolation level for InnoDB. For consistent reads, there is an important difference from the `READ COMMITTED` isolation level: All consistent reads within the same transaction read the snapshot established by the first read. This convention means that if you issue several plain (non-locking) `SELECT` statements within the same transaction, these `SELECT` statements are consistent also with respect to each other. See <http://dev.mysql.com/doc/refman/en/innodb-consistent-read.html>.

For locking reads (`SELECT` with `FOR UPDATE` or `LOCK IN SHARE MODE`), `UPDATE`, and `DELETE` statements, locking depends on whether the statement uses a unique index with a unique search condition, or a range-type search condition. For a unique index with a unique search condition, InnoDB locks only the index record found, not the gap before it. For other search conditions, InnoDB locks the index range scanned, using gap locks or next-key (gap plus index-record) locks to block insertions by other sessions into the gaps covered by the range.

This is the minimum isolation level for non-distributed [XA transactions](#).

SERIALIZABLE

This level is like REPEATABLE READ, but InnoDB implicitly converts all plain `SELECT` statements to `SELECT ... LOCK IN SHARE MODE` if `autocommit` is disabled. If autocommit is enabled, the `SELECT` is its own transaction. It therefore is known to be read only and can be serialized if performed as a consistent (non-locking) read and need not block for other transactions. (This means that to force a plain `SELECT` to block if other transactions have modified the selected rows, you should disable `autocommit`.)

Distributed [XA transactions](#) should always use this isolation level.

Access Mode

The access mode specifies whether the transaction is allowed to write data or not. By default, transactions are in `READ WRITE` mode (see the [tx_read_only](#) system variable). `READ ONLY` mode allows the storage engine to apply optimizations that cannot be used for transactions which write data. The only exception to this rule is that read only transactions can perform DDL statements on temporary tables.

It is not permitted to specify both `READ WRITE` and `READ ONLY` in the same statement.

`READ WRITE` and `READ ONLY` can also be specified in the [START TRANSACTION](#) statement, in which case the specified mode is only valid for one transaction.

Examples

```
SET GLOBAL TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```

Attempting to set the isolation level within an existing transaction without specifying `GLOBAL` or `SESSION`.

```
START TRANSACTION;
```

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```

```
ERROR 1568 (25001): Transaction characteristics can't be changed while a transaction is in p
```

H3Dr1.2.7.10 SET Variable

Syntax

```
SET var_name = expr [, var_name = expr] ...
```

Contents

1. [Syntax](#)
2. [Description](#)
3. [See Also](#)

Description

The `SET` statement in [stored programs](#) is an extended version of the general `SET` statement.

Referenced variables may be ones declared inside a stored program, global system variables, or user-defined variables.

The `SET` statement in stored programs is implemented as part of the pre-existing `SET` syntax. This allows an extended syntax of `SET a=x, b=y, ...` where different variable types (locally declared variables, global and session server variables, user-defined variables) can be mixed. This also allows combinations of local variables and some options that make sense only for system variables; in that case, the options are recognized but ignored.

`SET` can be used with both [local variables](#) and [user-defined variables](#).

When setting several variables using the columns returned by a query, [SELECT INTO](#) should be preferred.

To set many variables to the same value, the [LAST_VALUE\(\)](#) function can be used.

Below is an example of how a user-defined variable may be set:

```
SET @x = 1;
```

See Also

- SET
- SET STATEMENT
- DECLARE Variable ↗

H3Dr1.2.8 SHOW

Articles on the various SHOW commands.



About SHOW

General information about the SHOW statement.



Extended Show

Extended SHOW with WHERE and LIKE.



SHOW AUTHORS

Information about the people who work on MariaDB.



SHOW BINARY LOGS

SHOW BINARY LOGS lists all binary logs on the server.



SHOW BINLOG EVENTS

Show events in the binary log.



SHOW CHARACTER SET

Available character sets.



SHOW CLIENT_STATISTICS

Statistics about client connections.



SHOW COLLATION

Supported collations.



SHOW COLUMNS

Column information.



SHOW CONTRIBUTORS

Companies and people who financially contribute to MariaDB.



SHOW CREATE DATABASE

Shows the CREATE DATABASE statement that created the database.



SHOW CREATE EVENT

Displays the CREATE EVENT statement needed to re-create a given event



SHOW CREATE FUNCTION

Statement that created the function.



SHOW CREATE PACKAGE

Show the CREATE statement that creates the given package specification.



SHOW CREATE PACKAGE BODY

Show the CREATE statement that creates the given package body (i.e. implementation).



SHOW CREATE PROCEDURE

Returns the string used for creating a stored procedure.



SHOW CREATE SEQUENCE

Shows the CREATE SEQUENCE statement that created the sequence.



SHOW CREATE TABLE

Shows the CREATE TABLE statement that created the table.

-  **SHOW CREATE TRIGGER**
Shows the CREATE TRIGGER statement used to create the trigger
-  **SHOW CREATE USER**
Show the CREATE USER statement for a specified user.
-  **SHOW CREATE VIEW**
Show the CREATE VIEW statement that created a view.
-  **SHOW DATABASES**
Lists the databases on the server.
-  **SHOW ENGINE**
Show storage engine information.
-  **SHOW ENGINE INNODB STATUS**
Display extensive InnoDB information.
-  **SHOW ENGINES**
Server storage engine info
-  **SHOW ERRORS**
Displays errors.
-  **SHOW EVENTS**
Shows information about events
-  **SHOW EXPLAIN**
Shows an execution plan for a running query.
-  **SHOW FUNCTION CODE**
Representation of the internal implementation of the stored function
-  **SHOW FUNCTION STATUS**
Stored function characteristics
-  **SHOW GRANTS**
View GRANT statements.
-  **SHOW INDEX**
Information about table indexes.
-  **SHOW INDEX_STATISTICS**
Index usage statistics.
-  **SHOW INNODB STATUS (removed)**
Removed synonym for SHOW ENGINE INNODB STATUS ↗
-  **SHOW LOCALES**
View locales information.
-  **SHOW MASTER STATUS**
Status information about the binary log.
-  **SHOW OPEN TABLES**
List non-temporary open tables.
-  **SHOW PACKAGE BODY STATUS**
Returns characteristics of stored package bodies (implementations).



SHOW PACKAGE STATUS

Returns characteristics of stored package specifications.



SHOW PLUGINS

Display information about installed plugins.



SHOW PLUGINS SONAME

Information about all available plugins, installed or not.



SHOW PRIVILEGES

Shows the list of supported system privileges.



SHOW PROCEDURE CODE

Display internal implementation of a stored procedure.



SHOW PROCEDURE STATUS

Stored procedure characteristics.



SHOW PROCESSLIST

Running threads and information about them.



SHOW PROFILE

Display statement resource usage



SHOW PROFILES

Show statement resource usage



SHOW QUERY_RESPONSE_TIME

Retrieving information from the QUERY_RESPONSE_TIME plugin.



SHOW RELAYLOG EVENTS

Show events in the relay log.



SHOW SLAVE HOSTS

Display replicas currently registered with the primary.



SHOW SLAVE STATUS

Show status for one or all primaries.



SHOW STATUS

Server status information.



SHOW TABLE STATUS

SHOW TABLES with information about non-temporary tables.



SHOW TABLES

List of non-temporary tables, views or sequences.



SHOW TABLE_STATISTICS

Table usage statistics.



SHOW TRIGGERS

Shows currently-defined triggers



SHOW USER_STATISTICS

User activity statistics.



SHOW VARIABLES

Displays the values of system variables.



SHOW WARNINGS

Displays errors, warnings and notes.



SHOW WSREP_MEMBERSHIP

Galera node cluster membership information.



SHOW WSREP_STATUS

Galera node cluster status information.

H3Dr1.2.8.1 About SHOW

SHOW has many forms that provide information about databases, tables, columns, or status information about the server. These include:

- SHOW AUTHORS
- SHOW CHARACTER SET [like_or_where]
- SHOW COLLATION [like_or_where]
- SHOW [FULL] COLUMNS FROM tbl_name [FROM db_name] [like_or_where]
- SHOW CONTRIBUTORS
- SHOW CREATE DATABASE db_name
- SHOW CREATE EVENT event_name
- SHOW CREATE PACKAGE package_name
- SHOW CREATE PACKAGE BODY package_name
- SHOW CREATE PROCEDURE proc_name
- SHOW CREATE TABLE tbl_name
- SHOW CREATE TRIGGER trigger_name
- SHOW CREATE VIEW view_name
- SHOW DATABASES [like_or_where]
- SHOW ENGINE engine_name {STATUS | MUTEX}
- SHOW [STORAGE] ENGINES
- SHOW ERRORS [LIMIT [offset,] row_count]
- SHOW [FULL] EVENTS
- SHOW FUNCTION CODE func_name
- SHOW FUNCTION STATUS [like_or_where]
- SHOW GRANTS FOR user
- SHOW INDEX FROM tbl_name [FROM db_name]
- SHOW INNODB STATUS ↗
- SHOW OPEN TABLES [FROM db_name] [like_or_where]
- SHOW PLUGINS
- SHOW PROCEDURE CODE proc_name
- SHOW PROCEDURE STATUS [like_or_where]
- SHOW PRIVILEGES
- SHOW [FULL] PROCESSLIST
- SHOW PROFILE [types] [FOR QUERY n] [OFFSET n] [LIMIT n]
- SHOW PROFILES
- SHOW [GLOBAL | SESSION] STATUS [like_or_where]
- SHOW TABLE STATUS [FROM db_name] [like_or_where]
- SHOW TABLES [FROM db_name] [like_or_where]
- SHOW TRIGGERS [FROM db_name] [like_or_where]
- SHOW [GLOBAL | SESSION] VARIABLES [like_or_where]
- SHOW WARNINGS [LIMIT [offset,] row_count]

```
like_or_where:  
    LIKE 'pattern'  
    | WHERE expr
```

If the syntax for a given SHOW statement includes a LIKE 'pattern' part, 'pattern' is a string that can contain the SQL "%" and "_" wildcard characters. The pattern is useful for restricting statement output to matching values.

Several SHOW statements also accept a WHERE clause that provides more flexibility in specifying which rows to display. See [Extended Show](#).

H3Dr1.2.8.2 Extended Show

Contents

1. Examples

The following `SHOW` statements can be extended by using a `WHERE` clause and a `LIKE` clause to refine the results:

- `SHOW CHARACTER SET`
- `SHOW COLLATION`
- `SHOW COLUMNS`
- `SHOW DATABASES`
- `SHOW FUNCTION STATUS`
- `SHOW INDEX`
- `SHOW OPEN TABLES`
- `SHOW PACKAGE STATUS`
- `SHOW PACKAGE BODY STATUS`
- `SHOW INDEX`
- `SHOW PROCEDURE STATUS`
- `SHOW STATUS`
- `SHOW TABLE STATUS`
- `SHOW TABLES`
- `SHOW TRIGGERS`
- `SHOW VARIABLES`

As with a regular `SELECT`, the `WHERE` clause can be used for the specific columns returned, and the `LIKE` clause with the regular wildcards.

Examples

```
SHOW TABLES;  
+-----+  
| Tables_in_test |  
+-----+  
| animal_count   |  
| animals        |  
| are_the_mooses_loose |  
| aria_test2     |  
| t1             |  
| view1          |  
+-----+
```

Showing the tables beginning with `a` only.

```
SHOW TABLES WHERE Tables_in_test LIKE 'a%';  
+-----+  
| Tables_in_test |  
+-----+  
| animal_count   |  
| animals        |  
| are_the_mooses_loose |  
| aria_test2     |  
+-----+
```

Variables whose name starts with `aria` and with a value of greater than 8192:

```
SHOW VARIABLES WHERE Variable_name LIKE 'aria%' AND Value >8192;
+-----+-----+
| Variable_name | Value |
+-----+-----+
| aria_checkpoint_log_activity | 1048576 |
| aria_log_file_size | 1073741824 |
| aria_max_sort_file_size | 9223372036853727232 |
| aria_pagecache_buffer_size | 134217728 |
| aria_sort_buffer_size | 134217728 |
+-----+-----+
```

Shortcut, just returning variables whose name begins with *aria*.

```
SHOW VARIABLES LIKE 'aria%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| aria_block_size | 8192 |
| aria_checkpoint_interval | 30 |
| aria_checkpoint_log_activity | 1048576 |
| aria_force_start_after_recovery_failures | 0 |
| aria_group_commit | none |
| aria_group_commit_interval | 0 |
| aria_log_file_size | 1073741824 |
| aria_log_purge_type | immediate |
| aria_max_sort_file_size | 9223372036853727232 |
| aria_page_checksum | ON |
| aria_pagecache_age_threshold | 300 |
| aria_pagecache_buffer_size | 134217728 |
| aria_pagecache_division_limit | 100 |
| aria_recover | NORMAL |
| aria_repair_threads | 1 |
| aria_sort_buffer_size | 134217728 |
| aria_stats_method | nulls_unequal |
| aria_sync_log_dir | NEWFILE |
| aria_used_for_temp_tables | ON |
+-----+-----+
```

H3Dr1.2.8.3 SHOW AUTHORS

Syntax

```
SHOW AUTHORS
```

Description

The `SHOW AUTHORS` statement displays information about the people who work on MariaDB. For each author, it displays Name, Location, and Comment values. All columns are encoded as latin1.

These include:

- First the active people in MariaDB are listed.
- Then the active people in MySQL.
- Last the people that have contributed to MariaDB/MySQL in the past.

The order is somewhat related to importance of the contribution given to the MariaDB project, but this is not 100% accurate. There is still room for improvement and debate...

Example

```
SHOW AUTHORS\G
***** 1. row *****
Name: Michael (Monty) Widenius
```

Location: Tusby, Finland
Comment: Lead developer and main author
***** 2. row *****
 Name: Sergei Golubchik
Location: Kerpen, Germany
Comment: Architect, Full-text search, precision math, plugin framework, merges etc
***** 3. row *****
 Name: Igor Babaev
Location: Bellevue, USA
Comment: Optimizer, keycache, core work
***** 4. row *****
 Name: Sergey Petrunia
Location: St. Petersburg, Russia
Comment: Optimizer
***** 5. row *****
 Name: Oleksandr Byelkin
Location: Lugansk, Ukraine
Comment: Query Cache (4.0), Subqueries (4.1), Views (5.0)
***** 6. row *****
 Name: Timour Katchaounov
Location: Sofia , Bulgaria
Comment: Optimizer
***** 7. row *****
 Name: Kristian Nielsen
Location: Copenhagen, Denmark
Comment: Replication, Async client protocol, General buildbot stuff
***** 8. row *****
 Name: Alexander (Bar) Barkov
Location: Izhevsk, Russia
Comment: Unicode and character sets
***** 9. row *****
 Name: Alexey Botchkov (Holyfoot)
Location: Izhevsk, Russia
Comment: GIS extensions, embedded server, precision math
***** 10. row *****
 Name: Daniel Bartholomew
Location: Raleigh, USA
Comment: MariaDB documentation, Buildbot, releases
***** 11. row *****
 Name: Colin Charles
Location: Selangor, Malesia
Comment: MariaDB documentation, talks at a LOT of conferences
***** 12. row *****
 Name: Sergey Vojtovich
Location: Izhevsk, Russia
Comment: initial implementation of plugin architecture, maintained native storage engines (MyISAM, MEMORY, ARCHIVE, etc), rewrite of table cache
***** 13. row *****
 Name: Vladislav Vaintrob
Location: Mannheim, Germany
Comment: MariaDB Java connector, new thread pool, Windows optimizations
***** 14. row *****
 Name: Elena Stepanova
Location: Sankt Petersburg, Russia
Comment: QA, test cases
***** 15. row *****
 Name: Georg Richter
Location: Heidelberg, Germany
Comment: New LGPL C connector, PHP connector
***** 16. row *****
 Name: Jan Lindström
Location: Ylämylly, Finland
Comment: Working on InnoDB
***** 17. row *****
 Name: Lixun Peng
Location: Hangzhou, China
Comment: Multi Source replication
***** 18. row *****
 Name: Olivier Bertrand
Location: Paris, France
Comment: CONNECT storage engine
***** 19. row *****
 Name: Kentoku Shiba

```

Name: Kentaro Shiba
Location: Tokyo, Japan
Comment: Spider storage engine, metadata_lock_info Information schema
***** 20. row *****
    Name: Percona
    Location: CA, USA
    Comment: XtraDB, microslow patches, extensions to slow log
***** 21. row *****
    Name: Vicentiu Ciorbaru
    Location: Bucharest, Romania
    Comment: Roles
***** 22. row *****
    Name: Sudheera Palihakkara
    Location:
    Comment: PCRE Regular Expressions
***** 23. row *****
    Name: Pavel Ivanov
    Location: USA
    Comment: Some patches and bug fixes
***** 24. row *****
    Name: Konstantin Osipov
    Location: Moscow, Russia
    Comment: Prepared statements (4.1), Cursors (5.0), GET_LOCK (10.0)
***** 25. row *****
    Name: Ian Gilfillan
    Location: South Africa
    Comment: MariaDB documentation
***** 26. row *****
    Name: Federico Razolli
    Location: Italy
    Comment: MariaDB documentation Italian translation
***** 27. row *****
    Name: Guilhem Bichot
    Location: Bordeaux, France
    Comment: Replication (since 4.0)
***** 28. row *****
    Name: Andrei Elkin
    Location: Espoo, Finland
    Comment: Replication
***** 29. row *****
    Name: Dmitri Lenev
    Location: Moscow, Russia
    Comment: Time zones support (4.1), Triggers (5.0)
***** 30. row *****
    Name: Marc Alff
    Location: Denver, CO, USA
    Comment: Signal, Resignal, Performance schema
***** 31. row *****
    Name: Mikael Ronström
    Location: Stockholm, Sweden
    Comment: NDB Cluster, Partitioning, online alter table
***** 32. row *****
    Name: Ingo Strüwing
    Location: Berlin, Germany
    Comment: Bug fixing in MyISAM, Merge tables etc
***** 33. row *****
    Name: Marko Mäkelä
    Location: Helsinki, Finland
    Comment: InnoDB core developer
...

```

See Also

- [SHOW CONTRIBUTORS](#). This list all members and sponsors of the MariaDB Foundation and other sponsors.

H3Dr1.2.8.4 SHOW BINARY LOGS

Syntax

```
SHOW BINARY LOGS  
SHOW MASTER LOGS
```

Description

Lists the [binary log](#) files on the server. This statement is used as part of the procedure described in [PURGE BINARY LOGS](#), that shows how to determine which logs can be purged.

This statement requires the [SUPER](#) privilege, the [REPLICATION_CLIENT](#) privilege, or, from [MariaDB 10.5.2](#), the [BINLOG MONITOR](#) privilege.

Examples

```
SHOW BINARY LOGS;  
+-----+-----+  
| Log_name      | File_size |  
+-----+-----+  
| mariadb-bin.000001 |    19039 |  
| mariadb-bin.000002 |  717389 |  
| mariadb-bin.000003 |     300 |  
| mariadb-bin.000004 |     333 |  
| mariadb-bin.000005 |     899 |  
| mariadb-bin.000006 |     125 |  
| mariadb-bin.000007 |  18907 |  
| mariadb-bin.000008 |  19530 |  
| mariadb-bin.000009 |     151 |  
| mariadb-bin.000010 |     151 |  
| mariadb-bin.000011 |     125 |  
| mariadb-bin.000012 |     151 |  
| mariadb-bin.000013 |     151 |  
| mariadb-bin.000014 |     125 |  
| mariadb-bin.000015 |     151 |  
| mariadb-bin.000016 |     314 |  
+-----+-----+
```

H3Dr1.2.8.5 SHOW BINLOG EVENTS

Syntax

```
SHOW BINLOG EVENTS  
[IN 'log_name'] [FROM pos] [LIMIT [offset,] row_count]
```

Description

Shows the events in the [binary log](#). If you do not specify 'log_name', the first binary log is displayed.

Requires the [BINLOG MONITOR](#) privilege (>= [MariaDB 10.5.2](#)) or the [REPLICATION SLAVE](#) privilege (<= [MariaDB 10.5.1](#)).

Example

```

SHOW BINLOG EVENTS IN 'mysql_sandbox10019-bin.000002';
+-----+-----+-----+-----+-----+
| Log_name          | Pos | Event_type      | Server_id | End_log_pos | Info
|                   |
+-----+-----+-----+-----+-----+
| mysql_sandbox10019-bin.000002 | 4 | Format_desc    | 1 | 248 |
Server ver: 10.0.19-MariaDB-log, Binlog ver: 4 |
| mysql_sandbox10019-bin.000002 | 248 | Gtid_list      | 1 | 273 | []
|
| mysql_sandbox10019-bin.000002 | 273 | Binlog_checkpoint | 1 | 325 |
mysql_sandbox10019-bin.000002
| mysql_sandbox10019-bin.000002 | 325 | Gtid           | 1 | 363 | GTID
0-1-1
| mysql_sandbox10019-bin.000002 | 363 | Query          | 1 | 446 |
CREATE DATABASE blog
| mysql_sandbox10019-bin.000002 | 446 | Gtid           | 1 | 484 | GTID
0-1-2
| mysql_sandbox10019-bin.000002 | 484 | Query          | 1 | 571 | use
`blog`; CREATE TABLE bb (id INT)
+-----+-----+-----+-----+-----+

```

H3Dr1.2.8.6 SHOW CHARACTER SET

Syntax

```

SHOW CHARACTER SET
[LIKE 'pattern' | WHERE expr]

```

Contents

1. [Syntax](#)
2. [Description](#)
3. [Examples](#)
4. [See Also](#)

Description

The `SHOW CHARACTER SET` statement shows all available [character sets](#). The `LIKE` clause, if present on its own, indicates which character set names to match. The `WHERE` and `LIKE` clauses can be given to select rows using more general conditions, as discussed in [Extended SHOW](#).

The same information can be queried from the [Information Schema CHARACTER_SETS](#) table.

See [Setting Character Sets and Collations](#) for details on specifying the character set at the server, database, table and column levels.

Examples

```

SHOW CHARACTER SET LIKE 'latin%';
+-----+-----+-----+-----+
| Charset | Description          | Default collation | Maxlen |
+-----+-----+-----+-----+
| latin1  | cp1252 West European   | latin1_swedish_ci | 1 |
| latin2  | ISO 8859-2 Central European | latin2_general_ci | 1 |
| latin5  | ISO 8859-9 Turkish       | latin5_turkish_ci | 1 |
| latin7  | ISO 8859-13 Baltic       | latin7_general_ci | 1 |
+-----+-----+-----+-----+

```

```
SHOW CHARACTER SET WHERE Maxlen LIKE '2';
+-----+-----+-----+-----+
| Charset | Description          | Default collation | Maxlen |
+-----+-----+-----+-----+
| big5    | Big5 Traditional Chinese | big5_chinese_ci   |      2 |
| sjis   | Shift-JIS Japanese     | sjis_japanese_ci |      2 |
| euckr  | EUC-KR Korean          | euckr_korean_ci  |      2 |
| gb2312 | GB2312 Simplified Chinese | gb2312_chinese_ci |      2 |
| gbk    | GBK Simplified Chinese | gbk_chinese_ci   |      2 |
| ucs2   | UCS-2 Unicode           | ucs2_general_ci  |      2 |
| cp932  | SJIS for Windows Japanese | cp932_japanese_ci |      2 |
+-----+-----+-----+-----+
```

See Also

- [Supported Character Sets and Collations](#)
- [Setting Character Sets and Collations](#)
- [Information Schema CHARACTER_SETS](#)

H3Dr1.2.8.7 SHOW CLIENT_STATISTICS

Syntax

```
SHOW CLIENT_STATISTICS
```

Description

The `SHOW CLIENT_STATISTICS` statement is part of the [User Statistics](#) feature. It was removed as a separate statement in [MariaDB 10.1.1](#), but effectively replaced by the generic `SHOW information_schema_table` statement. The [information_schema.CLIENT_STATISTICS](#) table holds statistics about client connections.

The `userstat` system variable must be set to 1 to activate this feature. See the [User Statistics](#) and [information_schema.CLIENT_STATISTICS](#) articles for more information.

Example

```
SHOW CLIENT_STATISTICS\G
*****
1. row *****
Client: localhost
Total_connections: 35
Concurrent_connections: 0
Connected_time: 708
    Busy_time: 2.555797999999985
    Cpu_time: 0.0412374000000002
Bytes_received: 3883
    Bytes_sent: 21595
Binlog_bytes_written: 0
    Rows_read: 18
    Rows_sent: 115
    Rows_deleted: 0
    Rows_inserted: 0
    Rows_updated: 0
Select_commands: 70
Update_commands: 0
Other_commands: 0
Commit_transactions: 1
Rollback_transactions: 0
    Denied_connections: 0
    Lost_connections: 0
    Access_denied: 0
Empty_queries: 35
```

H3Dr1.2.8.8 SHOW COLLATION

H3Dr1.2.8.9 SHOW COLUMNS

H3Dr1.2.8.10 SHOW CONTRIBUTORS

Syntax

```
SHOW CONTRIBUTORS
```

Description

The `SHOW CONTRIBUTORS` statement displays information about the companies and people who financially contribute to MariaDB. For each contributor, it displays `Name`, `Location`, and `Comment` values. All columns are encoded as `latin1`.

It displays all [members and sponsors of the MariaDB Foundation](#) as well as other financial contributors.

Example

SHOW CONTRIBUTORS;		
Name	Location	Comment
Booking.com	https://www.booking.com	Founding member, Platinum Sponsor of the MariaDB Foundation
Alibaba Cloud Foundation	https://www.alibabacloud.com/	Platinum Sponsor of the MariaDB Foundation
Tencent Cloud Foundation	https://cloud.tencent.com	Platinum Sponsor of the MariaDB Foundation
Microsoft Foundation	https://microsoft.com/	Platinum Sponsor of the MariaDB Foundation
MariaDB Corporation of the MariaDB Foundation	https://mariadb.com	Founding member, Platinum Sponsor of the MariaDB Foundation
Visma Foundation	https://visma.com	Gold Sponsor of the MariaDB Foundation
DBS Foundation	https://dbs.com	Gold Sponsor of the MariaDB Foundation
IBM Foundation	https://www.ibm.com	Gold Sponsor of the MariaDB Foundation
Tencent Games Foundation	http://game.qq.com/	Gold Sponsor of the MariaDB Foundation
Nexedi Foundation	https://www.nexedi.com	Silver Sponsor of the MariaDB Foundation
Acronis Foundation	https://www.acronis.com	Silver Sponsor of the MariaDB Foundation
Verkkokauppa.com Foundation	https://www.verkkokauppa.com	Bronze Sponsor of the MariaDB Foundation
Virtuozzo Foundation	https://virtuozzo.com	Bronze Sponsor of the MariaDB Foundation
Tencent Game DBA Foundation	http://tencentdba.com/about	Bronze Sponsor of the MariaDB Foundation
Tencent TDSQL Foundation	http://tdsql.org	Bronze Sponsor of the MariaDB Foundation
Percona Foundation	https://www.percona.com/	Bronze Sponsor of the MariaDB Foundation
Google replication and GTID	USA	Sponsoring encryption, parallel replication and GTID
Facebook ROWS EXAMINED etc	USA	Sponsoring non-blocking API, LIMIT ROWS EXAMINED etc
Ronald Bradford Auction	Brisbane, Australia	EFF contribution for UC2006
Sheeri Kritzer Auction	Boston, Mass. USA	EFF contribution for UC2006
Mark Shuttleworth Auction	London, UK.	EFF contribution for UC2006

See Also

- [Log of MariaDB contributors](#)
- [SHOW AUTHORS](#) list the authors of MariaDB (including documentation, QA etc).
- [MariaDB Foundation page on contributing financially](#)

H3Dr1.2.8.11 SHOW CREATE DATABASE

Syntax

```
SHOW CREATE {DATABASE | SCHEMA} db_name
```

Contents

1. [Syntax](#)
2. [Description](#)
3. [Examples](#)
4. [See Also](#)

Description

Shows the [CREATE DATABASE](#) statement that creates the given database. `SHOW CREATE SCHEMA` is a synonym for `SHOW CREATE DATABASE`. `SHOW CREATE DATABASE` quotes database names according to the value of the [sql_quote_show_create](#) server system variable.

Examples

```
SHOW CREATE DATABASE test;
+-----+
| Database | Create Database
+-----+
| test     | CREATE DATABASE `test` /*!40100 DEFAULT CHARACTER SET latin1 */ |
+-----+  
  
SHOW CREATE SCHEMA test;
+-----+
| Database | Create Database
+-----+
| test     | CREATE DATABASE `test` /*!40100 DEFAULT CHARACTER SET latin1 */ |
+-----+
```

With [sql_quote_show_create](#) off:

```
SHOW CREATE DATABASE test;
+-----+
| Database | Create Database
+-----+
| test     | CREATE DATABASE test /*!40100 DEFAULT CHARACTER SET latin1 */ |
+-----+
```

With a comment, from [MariaDB 10.5](#):

```
SHOW CREATE DATABASE p;
+-----+
| Database | Create Database
+-----+
| p       | CREATE DATABASE `p` /*!40100 DEFAULT CHARACTER SET latin1 */ COMMENT 'presentat
+-----+  
◀ | ▶
```

See Also

- [CREATE DATABASE](#)
- [ALTER DATABASE](#)
- [Character Sets and Collations](#)

H3Dr1.2.8.12 SHOW CREATE EVENT

Syntax

```
SHOW CREATE EVENT event_name
```

Description

This statement displays the [CREATE EVENT](#) statement needed to re-create a given event, as well as the [SQL_MODE](#) that was used when the trigger has been created and the character set used by the connection. To find out which events are present, use [SHOW EVENTS](#).

The output of this statement is unreliably affected by the [sql_quote_show_create](#) server system variable - see <http://bugs.mysql.com/bug.php?id=12719>

The [information_schema.EVENTS](#) table provides similar, but more complete, information.

Examples

```
SHOW CREATE EVENT test.e_daily\G
*****
*** 1. row ****
      Event: e_daily
      sql_mode:
      time_zone: SYSTEM
      Create Event: CREATE EVENT `e_daily`
                      ON SCHEDULE EVERY 1 DAY
                      STARTS CURRENT_TIMESTAMP + INTERVAL 6 HOUR
                      ON COMPLETION NOT PRESERVE
                      ENABLE
                      COMMENT 'Saves total number of sessions then
                                clears the table each day'
                      DO BEGIN
                          INSERT INTO site_activity.totals (time, total)
                          SELECT CURRENT_TIMESTAMP, COUNT(*)
                          FROM site_activity.sessions;
                          DELETE FROM site_activity.sessions;
                      END
      character_set_client: latin1
      collation_connection: latin1_swedish_ci
      Database Collation: latin1_swedish_ci
```

See also

- [Events Overview](#)
- [CREATE EVENT](#)
- [ALTER EVENT](#)
- [DROP EVENT](#)

H3Dr1.2.8.13 SHOW CREATE FUNCTION

Syntax

```
SHOW CREATE FUNCTION func_name
```

Description

This statement is similar to [SHOW CREATE PROCEDURE](#) but for stored functions.

The output of this statement is unreliably affected by the [sql_quote_show_create](#) server system variable - see <http://bugs.mysql.com/bug.php?id=12719>

Example

```
SHOW CREATE FUNCTION VatCents\G
*****
Function: VatCents
sql_mode:
Create Function: CREATE DEFINER='root'@`localhost` FUNCTION `VatCents`(price
DECIMAL(10,2)) RETURNS int(11)
DETERMINISTIC
BEGIN
DECLARE x INT;
SET x = price * 114;
RETURN x;
END
character_set_client: utf8
collation_connection: utf8_general_ci
Database Collation: latin1_swedish_ci
```

See Also

- [Stored Functions ↗](#)
- [CREATE FUNCTION ↗](#)

H3Dr1.2.8.14 SHOW CREATE PACKAGE

MariaDB starting with [10.3.5 ↗](#)

Oracle-style packages were introduced in [MariaDB 10.3.5 ↗](#).

Syntax

```
SHOW CREATE PACKAGE [ db_name . ] package_name
```

Contents

1. [Syntax](#)
2. [Description](#)
3. [Examples](#)
4. [See Also](#)

Description

The `SHOW CREATE PACKAGE` statement can be used when [Oracle SQL_MODE ↗](#) is set.

Shows the `CREATE` statement that creates the given package specification.

Examples

```
SHOW CREATE PACKAGE employee_tools\G
*****
Package: employee_tools
sql_mode:
PIPES_AS_CONCAT,ANSI_QUOTES,IGNORE_SPACE,ORACLE,NO_KEY_OPTIONS,NO_TABLE_OPTIONS,NO_FIELD_O
PTIONS,NO_AUTO_CREATE_USER
Create Package: CREATE DEFINER="root"@"localhost" PACKAGE "employee_tools" AS
FUNCTION getSalary(eid INT) RETURN DECIMAL(10,2);
PROCEDURE raiseSalary(eid INT, amount DECIMAL(10,2));
PROCEDURE raiseSalaryStd(eid INT);
PROCEDURE hire(ename TEXT, esalary DECIMAL(10,2));
END
character_set_client: utf8
collation_connection: utf8_general_ci
Database Collation: latin1_swedish_ci
```

See Also

- [CREATE PACKAGE](#)
- [DROP PACKAGE](#)
- [CREATE PACKAGE BODY](#)
- [SHOW CREATE PACKAGE BODY](#)
- [DROP PACKAGE BODY](#)
- [Oracle SQL_MODE](#)

H3Dr1.2.8.15 SHOW CREATE PACKAGE BODY

MariaDB starting with [10.3.5](#)

Oracle-style packages were introduced in [MariaDB 10.3.5](#).

Syntax

```
SHOW CREATE PACKAGE BODY [ db_name . ] package_name
```

Contents

1. [Syntax](#)
2. [Description](#)
3. [Examples](#)
4. [See Also](#)

Description

The `SHOW CREATE PACKAGE BODY` statement can be used when [Oracle SQL_MODE](#) is set.

Shows the `CREATE` statement that creates the given package body (i.e. the implementation).

Examples

```

SHOW CREATE PACKAGE BODY employee_tools\G
*****
***** 1. row *****
    Package body: employee_tools
      sql_mode:
PIPES_AS_CONCAT,ANSI_QUOTES,IGNORE_SPACE,ORACLE,NO_KEY_OPTIONS,NO_TABLE_OPTIONS,NO_FIELD_O
PTIONS,NO_AUTO_CREATE_USER
Create Package Body: CREATE DEFINER="root"@"localhost" PACKAGE BODY "employee_tools" AS

  stdRaiseAmount DECIMAL(10,2):=500;

  PROCEDURE log (eid INT, ecmnt TEXT) AS
  BEGIN
    INSERT INTO employee_log (id, cmnt) VALUES (eid, ecmnt);
  END;

  PROCEDURE hire(ename TEXT, esalary DECIMAL(10,2)) AS
    eid INT;
  BEGIN
    INSERT INTO employee (name, salary) VALUES (ename, esalary);
    eid:= last_insert_id();
    log(eid, 'hire ' || ename);
  END;

  FUNCTION getSalary(eid INT) RETURN DECIMAL(10,2) AS
    nSalary DECIMAL(10,2);
  BEGIN
    SELECT salary INTO nSalary FROM employee WHERE id=eid;
    log(eid, 'getSalary id=' || eid || ' salary=' || nSalary);
    RETURN nSalary;
  END;

  PROCEDURE raiseSalary(eid INT, amount DECIMAL(10,2)) AS
  BEGIN
    UPDATE employee SET salary=salary+amount WHERE id=eid;
    log(eid, 'raiseSalary id=' || eid || ' amount=' || amount);
  END;

  PROCEDURE raiseSalaryStd(eid INT) AS
  BEGIN
    raiseSalary(eid, stdRaiseAmount);
    log(eid, 'raiseSalaryStd id=' || eid);
  END;

  BEGIN
    log(0, 'Session ' || connection_id() || ' ' || current_user || ' started');
  END
character_set_client: utf8
collation_connection: utf8_general_ci
Database Collation: latin1_swedish_ci

```

See Also

- [CREATE PACKAGE](#)
- [SHOW CREATE PACKAGE](#)
- [DROP PACKAGE](#)
- [CREATE PACKAGE BODY](#)
- [DROP PACKAGE BODY](#)
- [Oracle SQL_MODE](#)

H3Dr1.2.8.16 SHOW CREATE PROCEDURE

Syntax

```
SHOW CREATE PROCEDURE proc_name
```

Contents

1. [Syntax](#)
2. [Description](#)
3. [Examples](#)
4. [See Also](#)

Description

This statement is a MariaDB extension. It returns the exact string that can be used to re-create the named stored procedure, as well as the [SQL_MODE](#) that was used when the trigger has been created and the character set used by the connection.. A similar statement, [SHOW CREATE FUNCTION](#), displays information about stored functions.

Both statements require that you are the owner of the routine or have the [SELECT](#) privilege on the [mysql.proc](#) table. When neither is true, the statements display NULL for the Create Procedure or Create Function field.

Warning Users with [SELECT](#) privileges on [mysql.proc](#) or [USAGE](#) privileges on [*.*](#) can view the text of routines, even when they do not have privileges for the function or procedure itself.

The output of these statements is unreliablely affected by the [sql_quote_show_create](#) server system variable - see <http://bugs.mysql.com/bug.php?id=12719>

Examples

Here's a comparison of the [SHOW CREATE PROCEDURE](#) and [SHOW CREATE FUNCTION](#) statements.

```
SHOW CREATE PROCEDURE test.simpleproc\G
*****
Procedure: simpleproc
sql_mode:
Create Procedure: CREATE PROCEDURE `simpleproc`(OUT param1 INT)
    BEGIN
        SELECT COUNT(*) INTO param1 FROM t;
    END
character_set_client: latin1
collation_connection: latin1_swedish_ci
Database Collation: latin1_swedish_ci

SHOW CREATE FUNCTION test.hello\G
*****
Function: hello
sql_mode:
Create Function: CREATE FUNCTION `hello`(s CHAR(20))
    RETURNS CHAR(50)
    RETURN CONCAT('Hello, ',s,'!')
character_set_client: latin1
collation_connection: latin1_swedish_ci
Database Collation: latin1_swedish_ci
```

When the user issuing the statement does not have privileges on the routine, attempting to [CALL](#) the procedure raises Error 1370.

```
CALL test.prc1();
Error 1370 (42000): execute command denied to user 'test_user'@'localhost' for routine 'tes
```

If the user neither has privilege to the routine nor the [SELECT](#) privilege on [mysql.proc](#) table, it raises Error 1305, informing them that the procedure does not exist.

```
SHOW CREATE TABLES test.prc1\G
Error 1305 (42000): PROCEDURE prc1 does not exist
```

See Also

- [Stored Procedure Overview](#)
- [CREATE PROCEDURE](#)
- [ALTER PROCEDURE](#)
- [DROP PROCEDURE](#)
- [SHOW PROCEDURE STATUS](#)
- [Stored Routine Privileges](#)
- [Information Schema ROUTINES Table](#)

H3Dr1.2.8.17 SHOW CREATE SEQUENCE

MariaDB starting with [10.3.1](#)

Sequences were introduced in [MariaDB 10.3](#).

Syntax

```
SHOW CREATE SEQUENCE sequence_name;
```

Contents

1. [Syntax](#)
2. [Description](#)
3. [Example](#)
4. [Notes](#)
5. [See Also](#)

Description

Shows the [CREATE SEQUENCE](#) statement that created the given sequence. The statement requires the `SELECT` privilege for the table.

Example

```
CREATE SEQUENCE s1 START WITH 50;
SHOW CREATE SEQUENCE s1\G;
*****
*** 1. row ****
Table: s1
Create Table: CREATE SEQUENCE `s1` start with 50 minvalue 1 maxvalue 9223372036854775806
increment by 1 cache 1000 nocycle ENGINE=InnoDB
```

Notes

If you want to see the underlying table structure used for the `SEQUENCE` you can use [SHOW CREATE TABLE](#) on the `SEQUENCE`. You can also use `SELECT` to read the current recorded state of the `SEQUENCE`:

```

SHOW CREATE TABLE s1\G
*****
      1. row *****
Table: s1
Create Table: CREATE TABLE `s1` (
  `next_not_cached_value` bigint(21) NOT NULL,
  `minimum_value` bigint(21) NOT NULL,
  `maximum_value` bigint(21) NOT NULL,
  `start_value` bigint(21) NOT NULL COMMENT 'start value when sequences is created
    or value if RESTART is used',
  `increment` bigint(21) NOT NULL COMMENT 'increment value',
  `cache_size` bigint(21) unsigned NOT NULL,
  `cycle_option` tinyint(1) unsigned NOT NULL COMMENT '0 if no cycles are allowed,
    1 if the sequence should begin a new cycle when maximum_value is passed',
  `cycle_count` bigint(21) NOT NULL COMMENT 'How many cycles have been done'
) ENGINE=InnoDB SEQUENCE=1

SELECT * FROM s1\G
*****
      1. row *****
next_not_cached_value: 50
  minimum_value: 1
  maximum_value: 9223372036854775806
    start_value: 50
    increment: 1
    cache_size: 1000
    cycle_option: 0
    cycle_count: 0

```

See Also

- [CREATE SEQUENCE](#)
- [ALTER SEQUENCE](#)

H3Dr1.2.8.18 SHOW CREATE TABLE

H3Dr1.2.8.19 SHOW CREATE TRIGGER

Syntax

```
SHOW CREATE TRIGGER trigger_name
```

Contents

1. [Syntax](#)
2. [Description](#)
3. [Examples](#)
4. [See also](#)

Description

This statement shows a [CREATE TRIGGER](#) statement that creates the given trigger, as well as the [SQL_MODE](#) that was used when the trigger has been created and the character set used by the connection.

The output of this statement is unreliablely affected by the [sql_quote_show_create](#) server system variable - see <http://bugs.mysql.com/bug.php?id=12719>

Examples

```
SHOW CREATE TRIGGER example\G
***** 1. row *****
Trigger: example
sql_mode: ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,STRICT_ALL_TABLES
,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_AUTO_CREATE_USER,NO_
ENGINE_SUBSTITUTION
SQL Original Statement: CREATE DEFINER=`root`@`localhost` TRIGGER example BEFORE
INSERT ON t FOR EACH ROW
BEGIN
    SET NEW.c = NEW.c * 2;
END
character_set_client: cp850
collation_connection: cp850_general_ci
Database Collation: utf8_general_ci
Created: 2016-09-29 13:53:34.35
```

MariaDB starting with [10.2.3](#)

The `Created` column was added in MySQL 5.7 and [MariaDB 10.2.3](#) as part of introducing multiple trigger events per action.

See also

- [Trigger Overview](#)
- [CREATE TRIGGER](#)
- [DROP TRIGGER](#)
- [information_schema.TRIGGERS Table](#)
- [SHOW TRIGGERS](#)
- [Trigger Limitations](#)

H3Dr1.2.8.20 SHOW CREATE USER

H3Dr1.2.8.21 SHOW CREATE VIEW

Syntax

```
SHOW CREATE VIEW view_name
```

Description

This statement shows a [CREATE VIEW](#) statement that creates the given [view](#), as well as the character set used by the connection when the view was created. This statement also works with views.

`SHOW CREATE VIEW` quotes table, column and stored function names according to the value of the [sql_quote_show_create](#) server system variable.

Examples

```
SHOW CREATE VIEW example\G
***** 1. row *****
View: example
Create View: CREATE ALGORITHM=UNDEFINED DEFINER=`root`@`localhost` SQL
SECURITY DEFINER VIEW `example` AS (select `t`.`id` AS `id`,`t`.`s` AS `s` from
`t`)
character_set_client: cp850
collation_connection: cp850_general_ci
```

With [sql_quote_show_create](#) off:

```
SHOW CREATE VIEW example\G
***** 1. row *****
View: example
Create View: CREATE ALGORITHM=UNDEFINED DEFINER=root@localhost SQL SECURITY DEFINER VIEW example AS (select t.id AS id,t.s AS s from t)
character_set_client: cp850
collation_connection: cp850_general_ci
```

Grants

To be able to see a view, you need to have the [SHOW VIEW](#) and the [SELECT](#) privilege on the view:

```
GRANT SHOW VIEW,SELECT ON test_database.test_view TO 'test'@'localhost';
```

See Also

- [Grant privileges to tables, views etc](#)

H3Dr1.2.8.22 SHOW DATABASES

Syntax

```
SHOW {DATABASES | SCHEMAS}
      [LIKE 'pattern' | WHERE expr]
```

Contents

1. [Syntax](#)
2. [Description](#)
3. [Examples](#)
4. [See Also](#)

Description

`SHOW DATABASES` lists the databases on the MariaDB server host. `SHOW SCHEMAS` is a synonym for `SHOW DATABASES`. The `LIKE` clause, if present on its own, indicates which database names to match. The `WHERE` and `LIKE` clauses can be given to select rows using more general conditions, as discussed in [Extended SHOW](#).

You see only those databases for which you have some kind of privilege, unless you have the global [SHOW DATABASES privilege](#). You can also get this list using the [mysqlshow](#) command.

If the server was started with the `--skip-show-database` option, you cannot use this statement at all unless you have the [SHOW DATABASES privilege](#).

The list of results returned by `SHOW DATABASES` is based on directories in the data directory, which is how MariaDB implements databases. It's possible that output includes directories that do not correspond to actual databases.

The [Information Schema SCHEMATA table](#) also contains database information.

Examples

```
SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| test |
+-----+
```

```
SHOW DATABASES LIKE 'm%';
+-----+
| Database (m%) |
+-----+
| mysql |
+-----+
```

See Also

- [CREATE DATABASE ↗](#)
- [ALTER DATABASE](#)
- [DROP DATABASE ↗](#)
- [SHOW CREATE DATABASE](#)
- [Character Sets and Collations ↗](#)
- [Information Schema SCHEMATA Table ↗](#)

H3Dr1.2.8.23 SHOW ENGINE

Contents

1. [Syntax](#)
2. [Description](#)
 1. [SHOW ENGINE INNODB STATUS](#)
 2. [SHOW ENGINE INNODB MUTEX](#)
 3. [SHOW ENGINE PERFORMANCE_SCHEMA STATUS](#)
 4. [SHOW ENGINE ROCKSDB STATUS](#)

Syntax

```
SHOW ENGINE engine_name {STATUS | MUTEX}
```

Description

`SHOW ENGINE` displays operational information about a storage engine. The following statements currently are supported:

```
SHOW ENGINE INNODB STATUS
SHOW ENGINE INNODB MUTEX
SHOW ENGINE PERFORMANCE_SCHEMA STATUS
SHOW ENGINE ROCKSDB STATUS
```

If the [Sphinx Storage Engine ↗](#) is installed, the following is also supported:

```
SHOW ENGINE SPHINX STATUS
```

See [SHOW ENGINE SPHINX STATUS ↗](#).

Older (and now removed) synonyms were `SHOW INNODB STATUS` for `SHOW ENGINE INNODB STATUS` and `SHOW MUTEX STATUS` for `SHOW ENGINE INNODB MUTEX`.

SHOW ENGINE INNODB STATUS

`SHOW ENGINE INNODB STATUS` displays extensive information from the standard InnoDB Monitor about the state of the InnoDB storage engine. See [SHOW ENGINE INNODB STATUS](#) for more.

SHOW ENGINE INNODB MUTEX

`SHOW ENGINE INNODB MUTEX` displays InnoDB mutex statistics.

The statement displays the following output fields:

- **Type:** Always InnoDB.
- **Name:** The source file where the mutex is implemented, and the line number in the file where the mutex is created. The line number is dependent on the MariaDB version.
- **Status:** This field displays the following values if `UNIV_DEBUG` was defined at compilation time (for example, in `include/univ.h` in the InnoDB part of the source tree). Only the `os_waits` value is displayed if `UNIV_DEBUG` was not defined. Without `UNIV_DEBUG`, the information on which the output is based is insufficient to distinguish regular mutexes and mutexes that protect rw-locks (which allow multiple readers or a single writer). Consequently, the output may appear to contain multiple rows for the same mutex.
 - `count` indicates how many times the mutex was requested.
 - `spin_waits` indicates how many times the spinlock had to run.
 - `spin_rounds` indicates the number of spinlock rounds. (`spin_rounds` divided by `spin_waits` provides the average round count.)
 - `os_waits` indicates the number of operating system waits. This occurs when the spinlock did not work (the mutex was not locked during the spinlock and it was necessary to yield to the operating system and wait).
 - `os_yields` indicates the number of times a the thread trying to lock a mutex gave up its timeslice and yielded to the operating system (on the presumption that allowing other threads to run will free the mutex so that it can be locked).
 - `os_wait_times` indicates the amount of time (in ms) spent in operating system waits, if the `timed_mutexes` system variable is 1 (ON). If `timed_mutexes` is 0 (OFF), timing is disabled, so `os_wait_times` is 0. `timed_mutexes` is off by default.

Information from this statement can be used to diagnose system problems. For example, large values of `spin_waits` and `spin_rounds` may indicate scalability problems.

The `information_schema.INNODB_MUTEXES` table provides similar information.

SHOW ENGINE PERFORMANCE_SCHEMA STATUS

This statement shows how much memory is used for [performance_schema](#) tables and internal buffers.

The output contains the following fields:

- **Type:** Always `performance_schema`.
- **Name:** The name of a table, the name of an internal buffer, or the `performance_schema` word, followed by a dot and an attribute. Internal buffers names are enclosed by parenthesis. `performance_schema` means that the attribute refers to the whole database (it is a total).
- **Status:** The value for the attribute.

The following attributes are shown, in this order, for all tables:

- **row_size:** The memory used for an individual record. This value will never change.
- **row_count:** The number of rows in the table or buffer. For some tables, this value depends on a server system variable.
- **memory:** For tables and `performance_schema`, this is the result of `row_size * row_count`.

For internal buffers, the attributes are:

- `count`
- `size`

SHOW ENGINE ROCKSDB STATUS

See also [MyRocks Performance Troubleshooting](#)

H3Dr1.2.8.24 SHOW ENGINE INNODB STATUS

SHOW ENGINE INNODB STATUS is a specific form of the [SHOW ENGINE](#) statement that displays the [InnoDB Monitor](#) output, which is extensive InnoDB information which can be useful in diagnosing problems.

The following sections are displayed

- **Status:** Shows the timestamp, monitor name and the number of seconds, or the elapsed time between the current time and the time the InnoDB Monitor output was last displayed. The per-second averages are based upon this time.
- **BACKGROUND THREAD:** srv_master_thread lines show work performed by the main background thread.
- **SEMAPHORES:** Threads waiting for a semaphore and stats on how the number of times threads have needed a spin or a wait on a mutex or rw-lock semaphore. If this number of threads is large, there may be I/O or contention issues. Reducing the size of the [innodb_thread_concurrency](#) system variable may help if contention is related to thread scheduling. Spin rounds per wait shows the number of spinlock rounds per OS wait for a mutex.
- **LATEST FOREIGN KEY ERROR:** Only shown if there has been a foreign key constraint error, it displays the failed statement and information about the constraint and the related tables.
- **LATEST DETECTED DEADLOCK:** Only shown if there has been a deadlock, it displays the transactions involved in the deadlock and the statements being executed, held and required locked and the transaction rolled back to.
- **TRANSACTIONS:** The output of this section can help identify lock contention, as well as reasons for the deadlocks.
- **FILE I/O:** InnoDB thread information as well as pending I/O operations and I/O performance statistics.
- **INSERT BUFFER AND ADAPTIVE HASH INDEX:** InnoDB insert buffer (old name for the [change buffer](#)) and adaptive hash index status information, including the number of each type of operation performed, and adaptive hash index performance.
- **LOG:** InnoDB log information, including current log sequence number, how far the log has been flushed to disk, the position at which InnoDB last took a checkpoint, pending writes and write performance statistics.
- **BUFFER POOL AND MEMORY:** Information on buffer pool pages read and written, which allows you to see the number of data file I/O operations performed by your queries. See [InnoDB Buffer Pool](#) for more. Similar information is also available from the [INFORMATION_SCHEMA.INNODB_BUFFER_POOL_STATS](#) table.
- **ROW OPERATIONS:** Information about the main thread, including the number and performance rate for each type of row operation.

If the [innodb_status_output_locks](#) system variable is set to 1, extended lock information will be displayed.

Example output:

```
=====
2019-09-06 12:44:13 0x7f93cc236700 INNODB MONITOR OUTPUT
=====
Per second averages calculated from the last 4 seconds
-----
BACKGROUND THREAD
-----
srv_master_thread loops: 2 srv_active, 0 srv_shutdown, 83698 srv_idle
srv_master_thread log flush and writes: 83682
-----
SEMAPHORES
-----
OS WAIT ARRAY INFO: reservation count 15
OS WAIT ARRAY INFO: signal count 8
RW-shared spins 0, rounds 20, OS waits 7
RW-excl spins 0, rounds 0, OS waits 0
RW-sx spins 0, rounds 0, OS waits 0
Spin rounds per wait: 20.00 RW-shared, 0.00 RW-excl, 0.00 RW-sx
-----
TRANSACTIONS
-----
Trx id counter 236
```

```

Purge done for trx's n:o < 236 undo n:o < 0 state: running
History list length 22
LIST OF TRANSACTIONS FOR EACH SESSION:
---TRANSACTION 421747401994584, not started
0 lock struct(s), heap size 1136, 0 row lock(s)
---TRANSACTION 421747401990328, not started
0 lock struct(s), heap size 1136, 0 row lock(s)
-----
FILE I/O
-----
I/O thread 0 state: waiting for completed aio requests (insert buffer thread)
I/O thread 1 state: waiting for completed aio requests (log thread)
I/O thread 2 state: waiting for completed aio requests (read thread)
I/O thread 3 state: waiting for completed aio requests (read thread)
I/O thread 4 state: waiting for completed aio requests (read thread)
I/O thread 5 state: waiting for completed aio requests (read thread)
I/O thread 6 state: waiting for completed aio requests (write thread)
I/O thread 7 state: waiting for completed aio requests (write thread)
I/O thread 8 state: waiting for completed aio requests (write thread)
I/O thread 9 state: waiting for completed aio requests (write thread)
Pending normal aio reads: [0, 0, 0, 0] , aio writes: [0, 0, 0, 0] ,
  ibuf aio reads:, log i/o's:, sync i/o's:
Pending flushes (fsync) log: 0; buffer pool: 0
286 OS file reads, 171 OS file writes, 22 OS fsyncs
0.00 reads/s, 0 avg bytes/read, 0.00 writes/s, 0.00 fsyncs/s
-----
INSERT BUFFER AND ADAPTIVE HASH INDEX
-----
Ibuf: size 1, free list len 0, seg size 2, 0 merges
merged operations:
  insert 0, delete mark 0, delete 0
discarded operations:
  insert 0, delete mark 0, delete 0
Hash table size 34679, node heap has 0 buffer(s)
Hash table size 34679, node heap has 0 buffer(s)
Hash table size 34679, node heap has 0 buffer(s)
Hash table size 34679, node heap has 0 buffer(s)
Hash table size 34679, node heap has 0 buffer(s)
Hash table size 34679, node heap has 0 buffer(s)
Hash table size 34679, node heap has 0 buffer(s)
Hash table size 34679, node heap has 0 buffer(s)
0.00 hash searches/s, 0.00 non-hash searches/s
---
LOG
---
Log sequence number 445926
Log flushed up to 445926
Pages flushed up to 445926
Last checkpoint at 445917
0 pending log flushes, 0 pending chkp writes
18 log i/o's done, 0.00 log i/o's/second
-----
BUFFER POOL AND MEMORY
-----
Total large memory allocated 167772160
Dictionary memory allocated 50768
Buffer pool size 8012
Free buffers 7611
Database pages 401
Old database pages 0
Modified db pages 0
Percent of dirty pages(LRU & free pages): 0.000
Max dirty pages percent: 75.000
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 264, created 137, written 156
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: 401, unzip_LRU len: 0
I/O sum[0]:cur[0], unzip sum[0]:cur[0]

```

```
-----  
ROW OPERATIONS  
-----  
0 queries inside InnoDB, 0 queries in queue  
0 read views open inside InnoDB  
Process ID=4267, Main thread ID=140272021272320, state: sleeping  
Number of rows inserted 1, updated 0, deleted 0, read 1  
0.00 inserts/s, 0.00 updates/s, 0.00 deletes/s, 0.00 reads/s  
Number of system rows inserted 0, updated 0, deleted 0, read 0  
0.00 inserts/s, 0.00 updates/s, 0.00 deletes/s, 0.00 reads/s  
-----  
END OF INNODB MONITOR OUTPUT  
=====
```

H3Dr1.2.8.25 SHOW ENGINES

Syntax

```
SHOW [STORAGE] ENGINES
```

Description

`SHOW ENGINES` displays status information about the server's storage engines. This is particularly useful for checking whether a storage engine is supported, or to see what the default engine is. `SHOW TABLE TYPES` is a deprecated synonym.

The [information_schema.ENGINES](#) table provides the same information.

Since storage engines are plugins, different information about them is also shown in the [information_schema.PLUGINS](#) table and by the `SHOW PLUGINS` statement.

Note that both MySQL's InnoDB and Percona's XtraDB replacement are labeled as `InnoDB`. However, if XtraDB is in use, it will be specified in the `COMMENT` field. See [XtraDB and InnoDB](#). The same applies to [FederatedX](#).

The output consists of the following columns:

- `Engine` indicates the engine's name.
- `Support` indicates whether the engine is installed, and whether it is the default engine for the current session.
- `Comment` is a brief description.
- `Transactions`, `XA` and `Savepoints` indicate whether [transactions](#), [XA transactions](#) and [transaction savepoints](#) are supported by the engine.

Examples

```
SHOW ENGINES\G  
***** 1. row *****  
    Engine: InnoDB  
    Support: DEFAULT  
    Comment: Supports transactions, row-level locking, and foreign keys  
    Transactions: YES  
    XA: YES  
    Savepoints: YES  
***** 2. row *****  
    Engine: CSV  
    Support: YES  
    Comment: CSV storage engine  
    Transactions: NO  
    XA: NO  
    Savepoints: NO  
***** 3. row *****  
    Engine: MyISAM  
    Support: YES
```

```

    Comment: MyISAM storage engine
Transactions: NO
    XA: NO
    Savepoints: NO
***** 4. row *****
    Engine: BLACKHOLE
    Support: YES
    Comment: /dev/null storage engine (anything you write to it disappears)
Transactions: NO
    XA: NO
    Savepoints: NO
***** 5. row *****
    Engine: FEDERATED
    Support: YES
    Comment: FederatedX pluggable storage engine
Transactions: YES
    XA: NO
    Savepoints: YES
***** 6. row *****
    Engine: MRG_MyISAM
    Support: YES
    Comment: Collection of identical MyISAM tables
Transactions: NO
    XA: NO
    Savepoints: NO
***** 7. row *****
    Engine: ARCHIVE
    Support: YES
    Comment: Archive storage engine
Transactions: NO
    XA: NO
    Savepoints: NO
***** 8. row *****
    Engine: MEMORY
    Support: YES
    Comment: Hash based, stored in memory, useful for temporary tables
Transactions: NO
    XA: NO
    Savepoints: NO
***** 9. row *****
    Engine: PERFORMANCE_SCHEMA
    Support: YES
    Comment: Performance Schema
Transactions: NO
    XA: NO
    Savepoints: NO
***** 10. row *****
    Engine: Aria
    Support: YES
    Comment: Crash-safe tables with MyISAM heritage
Transactions: NO
    XA: NO
    Savepoints: NO
10 rows in set (0.00 sec)

```

H3Dr1.2.8.26 SHOW ERRORS

Syntax

```

SHOW ERRORS [LIMIT [offset,] row_count]
SHOW ERRORS [LIMIT row_count OFFSET offset]
SHOW COUNT(*) ERRORS

```

Contents

1. [Syntax](#)
2. [Description](#)
3. [Examples](#)

Description

This statement is similar to [SHOW WARNINGS](#), except that instead of displaying errors, warnings, and notes, it displays only errors.

The `LIMIT` clause has the same syntax as for the [SELECT](#) statement.

The `SHOW COUNT(*) ERRORS` statement displays the number of errors. You can also retrieve this number from the `error_count` variable.

```
SHOW COUNT(*) ERRORS;
SELECT @@error_count;
```

The value of `error_count` might be greater than the number of messages displayed by [SHOW WARNINGS](#) if the `max_error_count` system variable is set so low that not all messages are stored.

For a list of MariaDB error codes, see [MariaDB Error Codes](#).

Examples

```
SELECT f();
ERROR 1305 (42000): FUNCTION f does not exist

SHOW COUNT(*) ERRORS;
+-----+
| @@session.error_count |
+-----+
|          1           |
+-----+

SHOW ERRORS;
+-----+
| Level | Code | Message           |
+-----+
| Error | 1305 | FUNCTION f does not exist |
+-----+
```

H3Dr1.2.8.27 SHOW EVENTS

Syntax

```
SHOW EVENTS [{FROM | IN} schema_name]
[LIKE 'pattern' | WHERE expr]
```

Description

Shows information about Event Manager events (created with [CREATE EVENT](#)). Requires the `EVENT` privilege. Without any arguments, `SHOW EVENTS` lists all of the events in the current schema:

```

SELECT CURRENT_USER(), SCHEMA();
+-----+-----+
| CURRENT_USER() | SCHEMA() |
+-----+-----+
| jon@ghidora | myschema |
+-----+-----+

SHOW EVENTS\G
***** 1. row *****
      Db: myschema
      Name: e_daily
      Definer: jon@ghidora
      Time zone: SYSTEM
      Type: RECURRING
      Execute at: NULL
      Interval value: 10
      Interval field: SECOND
      Starts: 2006-02-09 10:41:23
      Ends: NULL
      Status: ENABLED
      Originator: 0
character_set_client: latin1
collation_connection: latin1_swedish_ci
Database Collation: latin1_swedish_ci

```

To see the event action, use `SHOW CREATE EVENT` instead, or look at the [information_schema.EVENTS](#) table.

To see events for a specific schema, use the `FROM` clause. For example, to see events for the test schema, use the following statement:

```
SHOW EVENTS FROM test;
```

The `LIKE` clause, if present, indicates which event names to match. The `WHERE` clause can be given to select rows using more general conditions, as discussed in [Extended Show](#).

H3Dr1.2.8.28 SHOW FUNCTION CODE

H3Dr1.2.8.29 SHOW FUNCTION STATUS

Syntax

```
SHOW FUNCTION STATUS
[LIKE 'pattern' | WHERE expr]
```

Description

This statement is similar to `SHOW PROCEDURE STATUS` but for [stored functions](#).

The `LIKE` clause, if present on its own, indicates which function names to match.

The `WHERE` and `LIKE` clauses can be given to select rows using more general conditions, as discussed in [Extended SHOW](#).

The [information_schema.ROUTINES](#) table contains more detailed information.

Examples

Showing all stored functions:

```
SHOW FUNCTION STATUS\G
*****
1. row *****
Db: test
Name: VatCents
Type: FUNCTION
Definer: root@localhost
Modified: 2013-06-01 12:40:31
Created: 2013-06-01 12:40:31
Security_type: DEFINER
Comment:
character_set_client: utf8
collation_connection: utf8_general_ci
Database Collation: latin1_swedish_ci
```

Stored functions whose name starts with 'V':

```
SHOW FUNCTION STATUS LIKE 'V%' \G
*****
1. row *****
Db: test
Name: VatCents
Type: FUNCTION
Definer: root@localhost
Modified: 2013-06-01 12:40:31
Created: 2013-06-01 12:40:31
Security_type: DEFINER
Comment:
character_set_client: utf8
collation_connection: utf8_general_ci
Database Collation: latin1_swedish_ci
```

Stored functions with a security type of 'DEFINER':

```
SHOW FUNCTION STATUS WHERE Security_type LIKE 'DEFINER' \G
*****
1. row *****
Db: test
Name: VatCents
Type: FUNCTION
Definer: root@localhost
Modified: 2013-06-01 12:40:31
Created: 2013-06-01 12:40:31
Security_type: DEFINER
Comment:
character_set_client: utf8
collation_connection: utf8_general_ci
Database Collation: latin1_swedish_ci
```

H3Dr1.2.8.30 SHOW GRANTS

H3Dr1.2.8.31 SHOW INDEX

H3Dr1.2.8.32 SHOW INDEX_STATISTICS

Syntax

```
SHOW INDEX_STATISTICS
```

Description

The `SHOW INDEX_STATISTICS` statement was introduced in MariaDB 5.2 as part of the [User Statistics](#) feature. It was removed as a separate statement in MariaDB 10.1.1, but effectively replaced by the generic [SHOW information_schema_table](#) statement. The [information_schmea.INDEX_STATISTICS](#) table shows statistics on index usage and makes it possible to do such things as locating unused indexes and generating the commands to remove them.

The `userstat` system variable must be set to 1 to activate this feature. See the [User Statistics](#) and [information_schema.INDEX_STATISTICS](#) table for more information.

Example

```
SHOW INDEX_STATISTICS;
+-----+-----+-----+
| Table_schema | Table_name      | Index_name | Rows_read |
+-----+-----+-----+
| test        | employees_example | PRIMARY    |          1 |
+-----+-----+-----+
```

H3Dr1.2.8.33 SHOW LOCALES

`SHOW LOCALES` was introduced as part of the [Information Schema plugin extension](#).

`SHOW LOCALES` is used to return [locales](#) information as part of the [Locales](#) plugin. While the [information_schema.LOCALES](#) table has 8 columns, the `SHOW LOCALES` statement will only display 4 of them:

Example

```
SHOW LOCALES;
+-----+-----+-----+
| Id   | Name     | Description           | Error_Message_Language |
+-----+-----+-----+
| 0    | en_US    | English - United States | english                |
| 1    | en_GB    | English - United Kingdom | english                |
| 2    | ja_JP    | Japanese - Japan       | japanese               |
| 3    | sv_SE    | Swedish - Sweden       | swedish               |
...
...
```

H3Dr1.2.8.34 SHOW BINLOG STATUS

H3Dr1.2.8.35 SHOW OPEN TABLES

Syntax

```
SHOW OPEN TABLES [FROM db_name]
[LIKE 'pattern' | WHERE expr]
```

Description

`SHOW OPEN TABLES` lists the non- TEMPORARY tables that are currently open in the table cache. See <http://dev.mysql.com/doc/refman/5.1/en/table-cache.html>.

The `FROM` and `LIKE` clauses may be used.

The `FROM` clause, if present, restricts the tables shown to those present in the `db_name` database.

The `LIKE` clause, if present on its own, indicates which table names to match. The `WHERE` and `LIKE`

clauses can be given to select rows using more general conditions, as discussed in [Extended SHOW](#).

The following information is returned:

Column	Description
Database	Database name.
Name	Table name.
In_use	Number of table instances being used.
Name_locked	1 if the table is name-locked, e.g. if it is being dropped or renamed, otherwise 0 .

Before MariaDB 5.5, each use of, for example, `LOCK TABLE ... WRITE` would increment `In_use` for that table. With the implementation of the metadata locking improvements in MariaDB 5.5, `LOCK TABLE... WRITE` acquires a strong MDL lock, and concurrent connections will wait on this MDL lock, so any subsequent `LOCK TABLE... WRITE` will not increment `In_use`.

Example

```
SHOW OPEN TABLES;
+-----+-----+-----+
| Database | Table           | In_use | Name_locked |
+-----+-----+-----+
...
| test     | xjson            |      0 |      0 |
| test     | jaauthor          |      0 |      0 |
| test     | locks             |      1 |      0 |
...
+-----+-----+-----+
```

H3Dr1.2.8.36 SHOW PACKAGE BODY STATUS

MariaDB starting with [10.3.5](#)

Oracle-style packages were introduced in [MariaDB 10.3.5](#).

Syntax

```
SHOW PACKAGE BODY STATUS
[LIKE 'pattern' | WHERE expr]
```

Contents

1. [Syntax](#)
2. [Description](#)
3. [Examples](#)
4. [See Also](#)

Description

The `SHOW PACKAGE BODY STATUS` statement returns characteristics of stored package bodies (implementations), such as the database, name, type, creator, creation and modification dates, and character set information. A similar statement, `SHOW PACKAGE STATUS`, displays information about stored package specifications.

The `LIKE` clause, if present, indicates which package names to match. The `WHERE` and `LIKE` clauses can be given to select rows using more general conditions, as discussed in [Extended SHOW](#).

The [ROUTINES table](#) in the `INFORMATION_SCHEMA` database contains more detailed information.

Examples

```
SHOW PACKAGE BODY STATUS LIKE 'pkg1'\G
*****
Db: test
Name: pkg1
Type: PACKAGE BODY
Definer: root@localhost
Modified: 2018-02-27 14:44:14
Created: 2018-02-27 14:44:14
Security_type: DEFINER
Comment: This is my first package body
character_set_client: utf8
collation_connection: utf8_general_ci
Database Collation: latin1_swedish_ci
```

See Also

- [SHOW PACKAGE STATUS](#)
- [SHOW CREATE PACKAGE BODY](#)
- [CREATE PACKAGE BODY](#)
- [DROP PACKAGE BODY](#)
- [Oracle SQL_MODE](#)

H3Dr1.2.8.37 SHOW PACKAGE STATUS

MariaDB starting with [10.3.5](#)

Oracle-style packages were introduced in [MariaDB 10.3.5](#).

Syntax

```
SHOW PACKAGE STATUS
[LIKE 'pattern' | WHERE expr]
```

Contents

1. [Syntax](#)
2. [Description](#)
3. [Examples](#)
4. [See Also](#)

Description

The `SHOW PACKAGE STATUS` statement returns characteristics of stored package specifications, such as the database, name, type, creator, creation and modification dates, and character set information. A similar statement, `SHOW PACKAGE BODY STATUS`, displays information about stored package bodies (i.e. implementations).

The `LIKE` clause, if present, indicates which package names to match. The `WHERE` and `LIKE` clauses can be given to select rows using more general conditions, as discussed in [Extended SHOW](#).

The `ROUTINES` table in the `INFORMATION_SCHEMA` database contains more detailed information.

Examples

```
SHOW PACKAGE STATUS LIKE 'pkg1'\G
*****
Db: test
Name: pkg1
Type: PACKAGE
Definer: root@localhost
Modified: 2018-02-27 14:38:15
Created: 2018-02-27 14:38:15
Security_type: DEFINER
Comment: This is my first package
character_set_client: utf8
collation_connection: utf8_general_ci
Database Collation: latin1_swedish_ci
```

See Also

- [SHOW PACKAGE BODY](#)
- [SHOW CREATE PACKAGE](#)
- [CREATE PACKAGE ↗](#)
- [DROP PACKAGE ↗](#)
- [Oracle SQL_MODE ↗](#)

H3Dr1.2.8.38 SHOW PLUGINS

H3Dr1.2.8.39 SHOW PLUGINS SONAME

H3Dr1.2.8.40 SHOW PRIVILEGES

Syntax

```
SHOW PRIVILEGES
```

Description

`SHOW PRIVILEGES` shows the list of [system privileges](#) that the MariaDB server supports. The exact list of privileges depends on the version of your server.

Note that before [MariaDB 10.3.23 ↗](#), [MariaDB 10.4.13 ↗](#) and [MariaDB 10.5.2 ↗](#), the [Delete history](#) privilege displays as [Delete versioning rows](#) ([MDEV-20382 ↗](#)).

Example

From [MariaDB 10.5.9 ↗](#)

```
SHOW PRIVILEGES;
+-----+-----+-----+
| Privilege      | Context          | Comment
|               |
+-----+-----+-----+
| Alter          | Tables           | To alter the table
|               |
| Alter routine  | Functions,Procedures | To alter or drop
| stored functions/procedures |
| Create          | Databases,Tables,Indexes | To create new
| databases and tables |
| Create routine  | Databases        | To use CREATE
| FUNCTION/PROCEDURE |
```

Create temporary tables	Databases	To use CREATE
TEMPORARY TABLE		
Create view	Tables	To create new views
Create user	Server Admin	To create new users
Delete rows	Tables	To delete existing
Delete history table historical rows	Tables	To delete versioning
Drop tables, and views	Databases,Tables	To drop databases,
Event drop and execute events	Server Admin	To create, alter,
Execute routines	Functions,Procedures	To execute stored
File files on the server	File access on server	To read and write
Grant option users those privileges you possess	Databases,Tables,Functions,Procedures	To give to other
Index indexes	Tables	To create or drop
Insert tables	Tables	To insert data into
Lock tables (together with SELECT privilege)	Databases	To use LOCK TABLES
Process text of currently executing queries	Server Admin	To view the plain
Proxy possible	Server Admin	To make proxy user
References tables	Databases,Tables	To have references on
Reload tables, logs and privileges	Server Admin	To reload or refresh
Binlog admin	Server	To purge binary logs
Binlog monitor	Server	To use SHOW BINLOG
STATUS and SHOW BINARY LOG		
Binlog replay (generated by mariadb-binlog)	Server	To use BINLOG
Replication master admin slaves	Server	To monitor connected
Replication slave admin and apply binlog events	Server	To start/stop slave
Slave monitor	Server	To use SHOW SLAVE
STATUS and SHOW RELAYLOG EVENTS		
Replication slave events from the master	Server Admin	To read binary log
Select table	Tables	To retrieve rows from
Show databases with SHOW DATABASES	Server Admin	To see all databases
Show view SHOW CREATE VIEW	Tables	To see views with
Shutdown server	Server Admin	To shut down the
Super	Server Admin	To use KILL thread,
SET GLOBAL, CHANGE MASTER, etc.		
Trigger	Tables	To use triggers
Create tablespace tablespaces	Server Admin	To create/alter/drop
Update rows	Tables	To update existing
Set user stored routines with a different definer	Server	To create views and
Federated admin SERVER, ALTER SERVER, DROP SERVER statements	Server	To execute the CREATE
Connection admin limits and kill other users' connections	Server	To bypass connection
Read_only admin operations even if @@read_only=ON	Server	To perform write
Usage	Server Admin	No privileges - allow

```
connect only
+
41 rows in set (0.000 sec)
```

See Also

- [SHOW CREATE USER](#) shows how the user was created.
- [SHOW GRANTS](#) shows the GRANTS/PRIVILEGES for a user.

H3Dr1.2.8.41 SHOW PROCEDURE CODE

Syntax

```
SHOW PROCEDURE CODE proc_name
```

Contents

1. [Syntax](#)
2. [Description](#)
3. [Examples](#)
4. [See Also](#)

Description

This statement is a MariaDB extension that is available only for servers that have been built with debugging support. It displays a representation of the internal implementation of the named [stored procedure](#). A similar statement, [SHOW FUNCTION CODE](#), displays information about [stored functions](#).

Both statements require that you be the owner of the routine or have [SELECT](#) access to the [mysql.proc](#) table.

If the named routine is available, each statement produces a result set. Each row in the result set corresponds to one "instruction" in the routine. The first column is Pos, which is an ordinal number beginning with 0. The second column is Instruction, which contains an SQL statement (usually changed from the original source), or a directive which has meaning only to the stored-routine handler.

Examples

```
DELIMITER //

CREATE PROCEDURE p1 ()
BEGIN
    DECLARE fanta INT DEFAULT 55;
    DROP TABLE t2;
    LOOP
        INSERT INTO t3 VALUES (fanta);
        END LOOP;
    END//
```

Query OK, 0 rows affected (0.00 sec)

```
SHOW PROCEDURE CODE p1//
+-----+
| Pos | Instruction |
+-----+
| 0 | set fanta@0 55 |
| 1 | stmt 9 "DROP TABLE t2" |
| 2 | stmt 5 "INSERT INTO t3 VALUES (fanta)" |
| 3 | jump 2 |
+-----+
```

See Also

- [Stored Procedure Overview](#)
- [CREATE PROCEDURE](#)
- [ALTER PROCEDURE](#)
- [DROP PROCEDURE](#)
- [SHOW CREATE PROCEDURE](#)
- [SHOW PROCEDURE STATUS](#)
- [Stored Routine Privileges](#)
- [Information Schema ROUTINES Table](#)

H3Dr1.2.8.42 SHOW PROCEDURE STATUS

Syntax

```
SHOW PROCEDURE STATUS
[LIKE 'pattern' | WHERE expr]
```

Description

This statement is a MariaDB extension. It returns characteristics of a stored procedure, such as the database, name, type, creator, creation and modification dates, and character set information. A similar statement, [SHOW FUNCTION STATUS](#), displays information about stored functions.

The `LIKE` clause, if present, indicates which procedure or function names to match. The `WHERE` and `LIKE` clauses can be given to select rows using more general conditions, as discussed in [Extended SHOW](#).

The [ROUTINES table](#) in the INFORMATION_SCHEMA database contains more detailed information.

Examples

```
SHOW PROCEDURE STATUS LIKE 'p1'\G
*****
Db: test
Name: p1
Type: PROCEDURE
Definer: root@localhost
Modified: 2010-08-23 13:23:03
Created: 2010-08-23 13:23:03
Security_type: DEFINER
Comment:
character_set_client: latin1
collation_connection: latin1_swedish_ci
Database Collation: latin1_swedish_ci
```

See Also

- [Stored Procedure Overview](#)
- [CREATE PROCEDURE](#)
- [ALTER PROCEDURE](#)
- [DROP PROCEDURE](#)
- [SHOW CREATE PROCEDURE](#)
- [Stored Routine Privileges](#)
- [Information Schema ROUTINES Table](#)

H3Dr1.2.8.43 SHOW PROCESSLIST

Syntax

```
SHOW [FULL] PROCESSLIST
```

Description

`SHOW PROCESSLIST` shows you which threads are running. You can also get this information from the [information_schema.PROCESSLIST](#) table or the [mysqladmin processlist](#) command. If you have the `PROCESS privilege`, you can see all threads. Otherwise, you can see only your own threads (that is, threads associated with the MariaDB account that you are using). If you do not use the `FULL` keyword, only the first 100 characters of each statement are shown in the `Info` field.

The columns shown in `SHOW PROCESSLIST` are:

Name	Description
ID	The client's process ID.
USER	The username associated with the process.
HOST	The host the client is connected to.
DB	The default database of the process (NULL if no default).
COMMAND	The command type. See Thread Command Values .
TIME	The amount of time, in seconds, the process has been in its current state. For a replica SQL thread before MariaDB 10.1 , this is the time in seconds between the last replicated event's timestamp and the replica machine's real time.
STATE	See Thread States .
INFO	The statement being executed.
PROGRESS	The total progress of the process (0-100%) (see Progress Reporting).

See `TIME_MS` column in [information_schema.PROCESSLIST](#) for differences in the `TIME` column

between MariaDB and MySQL.

The [information_schema.PROCESSLIST](#) table contains the following additional columns:

Name	Description
TIME_MS	The amount of time, in milliseconds, the process has been in its current state.
STAGE	The stage the process is currently in.
MAX_STAGE	The maximum number of stages.
PROGRESS	The progress of the process within the current stage (0-100%).
MEMORY_USED	The amount of memory used by the process.
EXAMINED_ROWS	The number of rows the process has examined.
QUERY_ID	Query ID.

Note that the `PROGRESS` field from the information schema, and the `PROGRESS` field from `SHOW PROCESSLIST` display different results. `SHOW PROCESSLIST` shows the total progress, while the information schema shows the progress for the current stage only.

Threads can be killed using their `thread_id` or their `query_id`, with the [KILL](#) statement.

Since queries on this table are locking, if the [performance_schema](#) is enabled, you may want to query the [THREADS](#) table instead.

Examples

```
SHOW PROCESSLIST;
+-----+-----+-----+-----+-----+-----+-----+
| Id | User          | Host      | db     | Command | Time   | State           | Info
| Progress |
+-----+-----+-----+-----+-----+-----+-----+
| 2 | event_scheduler | localhost | NULL   | Daemon  | 2693   | Waiting on empty queue | NULL
| 0.000 |
| 4 | root           | localhost | NULL   | Query   | 0       | Table lock      | SHOW
PROCESSLIST | 0.000 |
+-----+-----+-----+-----+-----+-----+-----+
```

See Also

[CONNECTION_ID\(\)](#)

H3Dr1.2.8.44 SHOW PROFILE

Syntax

```

SHOW PROFILE [type [, type] ... ]
  [FOR QUERY n]
  [LIMIT row_count [OFFSET offset]]

type:
  ALL
  | BLOCK IO
  | CONTEXT SWITCHES
  | CPU
  | IPC
  | MEMORY
  | PAGE FAULTS
  | SOURCE
  | SWAPS

```

Description

The `SHOW PROFILE` and `SHOW PROFILES` statements display profiling information that indicates resource usage for statements executed during the course of the current session.

Profiling is controlled by the [profiling](#) session variable, which has a default value of `0` (OFF). Profiling is enabled by setting profiling to `1` or `ON`:

```
SET profiling = 1;
```

`SHOW PROFILES` displays a list of the most recent statements sent to the master. The size of the list is controlled by the [profiling_history_size](#) session variable, which has a default value of `15`. The maximum value is `100`. Setting the value to `0` has the practical effect of disabling profiling.

All statements are profiled except `SHOW PROFILES` and `SHOW PROFILE`, so you will find neither of those statements in the profile list. Malformed statements are profiled. For example, `SHOW PROFILING` is an illegal statement, and a syntax error occurs if you try to execute it, but it will show up in the profiling list.

`SHOW PROFILE` displays detailed information about a single statement. Without the `FOR QUERY n` clause, the output pertains to the most recently executed statement. If `FOR QUERY n` is included, `SHOW PROFILE` displays information for statement `n`. The values of `n` correspond to the `Query_ID` values displayed by `SHOW PROFILES`.

The `LIMIT row_count` clause may be given to limit the output to `row_count` rows. If `LIMIT` is given, `OFFSET offset` may be added to begin the output offset rows into the full set of rows.

By default, `SHOW PROFILE` displays Status and Duration columns. The Status values are like the State values displayed by `SHOW PROCESSLIST`, although there might be some minor differences in interpretation for the two statements for some status values (see <http://dev.mysql.com/doc/refman/5.6/en/thread-information.html>).

Optional type values may be specified to display specific additional types of information:

- `ALL` displays all information
- `BLOCK IO` displays counts for block input and output operations
- `CONTEXT SWITCHES` displays counts for voluntary and involuntary context switches
- `CPU` displays user and system CPU usage times
- `IPC` displays counts for messages sent and received
- `MEMORY` is not currently implemented
- `PAGE FAULTS` displays counts for major and minor page faults
- `SOURCE` displays the names of functions from the source code, together with the name and line number of the file in which the function occurs
- `SWAPS` displays swap counts

Profiling is enabled per session. When a session ends, its profiling information is lost.

The `information_schema.PROFILING` table contains similar information.

Examples

```

SELECT @@profiling;
+-----+
| @@profiling |
+-----+
|      0 |
+-----+


SET profiling = 1;

USE test;

DROP TABLE IF EXISTS t1;

CREATE TABLE T1 (id INT);

SHOW PROFILES;
+-----+-----+-----+
| Query_ID | Duration | Query          |
+-----+-----+-----+
|     1 | 0.00009200 | SELECT DATABASE() |
|     2 | 0.00023800 | show databases   |
|     3 | 0.00018900 | show tables     |
|     4 | 0.00014700 | DROP TABLE IF EXISTS t1 |
|     5 | 0.24476900 | CREATE TABLE T1 (id INT) |
+-----+-----+-----+


SHOW PROFILE;
+-----+-----+
| Status           | Duration |
+-----+-----+
| starting         | 0.000042 |
| checking permissions | 0.000044 |
| creating table   | 0.244645 |
| After create     | 0.000013 |
| query end        | 0.000003 |
| freeing items    | 0.000016 |
| logging slow query | 0.000003 |
| cleaning up      | 0.000003 |
+-----+-----+


SHOW PROFILE FOR QUERY 4;
+-----+-----+
| Status           | Duration |
+-----+-----+
| starting         | 0.000126 |
| query end        | 0.000004 |
| freeing items    | 0.000012 |
| logging slow query | 0.000003 |
| cleaning up      | 0.000002 |
+-----+-----+


SHOW PROFILE CPU FOR QUERY 5;
+-----+-----+-----+-----+
| Status           | Duration | CPU_user | CPU_system |
+-----+-----+-----+-----+
| starting         | 0.000042 | 0.000000 | 0.000000 |
| checking permissions | 0.000044 | 0.000000 | 0.000000 |
| creating table   | 0.244645 | 0.000000 | 0.000000 |
| After create     | 0.000013 | 0.000000 | 0.000000 |
| query end        | 0.000003 | 0.000000 | 0.000000 |
| freeing items    | 0.000016 | 0.000000 | 0.000000 |
| logging slow query | 0.000003 | 0.000000 | 0.000000 |
| cleaning up      | 0.000003 | 0.000000 | 0.000000 |
+-----+-----+-----+-----+

```

H3Dr1.2.8.45 SHOW PROFILES

Syntax

```
SHOW PROFILES
```

Description

The `SHOW PROFILES` statement displays profiling information that indicates resource usage for statements executed during the course of the current session. It is used together with [SHOW PROFILE](#).

H3Dr1.2.8.46 SHOW QUERY_RESPONSE_TIME

It is possible to use `SHOW QUERY_RESPONSE_TIME` as an alternative for retrieving information from the [QUERY_RESPONSE_TIME](#) plugin.

This was introduced as part of the [Information Schema plugin extension](#).

H3Dr1.2.8.47 SHOW RELAYLOG EVENTS

H3Dr1.2.8.48 SHOW REPLICA HOSTS

H3Dr1.2.8.49 SHOW REPLICA STATUS

H3Dr1.2.8.50 SHOW STATUS

Syntax

```
SHOW [GLOBAL | SESSION] STATUS  
[LIKE 'pattern' | WHERE expr]
```

Description

`SHOW STATUS` provides server status information. This information also can be obtained using the [mysqladmin extended-status](#) command, or by querying the [Information Schema GLOBAL_STATUS](#) and [SESSION_STATUS](#) tables. The `LIKE` clause, if present, indicates which variable names to match. The `WHERE` clause can be given to select rows using more general conditions.

With the `GLOBAL` modifier, `SHOW STATUS` displays the status values for all connections to MariaDB. With `SESSION`, it displays the status values for the current connection. If no modifier is present, the default is `SESSION`. `LOCAL` is a synonym for `SESSION`. If you see a lot of 0 values, the reason is probably that you have used `SHOW STATUS` with a new connection instead of `SHOW GLOBAL STATUS`.

Some status variables have only a global value. For these, you get the same value for both `GLOBAL` and `SESSION`.

See [Server Status Variables](#) for a full list, scope and description of the variables that can be viewed with `SHOW STATUS`.

The `LIKE` clause, if present on its own, indicates which variable name to match.

The `WHERE` and `LIKE` clauses can be given to select rows using more general conditions, as discussed in [Extended SHOW](#).

Examples

Full output from [MariaDB 10.1.17](#):

Variable_name	Value
Aborted_clients	0
Aborted_connects	0
Access_denied_errors	0
Acl_column_grants	0
Acl_database_grants	2
Acl_function_grants	0
Acl_procedure_grants	0
Acl_proxy_users	2
Acl_role_grants	0
Acl_roles	0
Acl_table_grants	0
Acl_users	6
Aria_pagecache_blocks_not_flushed	0
Aria_pagecache_blocks_unused	15706
Aria_pagecache_blocks_used	0
Aria_pagecache_read_requests	0
Aria_pagecache_reads	0
Aria_pagecache_write_requests	0
Aria_pagecache_writes	0
Aria_transaction_log_syncs	0
Binlog_commits	0
Binlog_group_commits	0
Binlog_group_commit_trigger_count	0
Binlog_group_commit_trigger_lock_wait	0
Binlog_group_commit_trigger_timeout	0
Binlog_snapshot_file	
Binlog_snapshot_position	0
Binlog_bytes_written	0
Binlog_cache_disk_use	0
Binlog_cache_use	0
Binlog_stmt_cache_disk_use	0

	0
Binlog_stmt_cache_use	0
Busy_time	0.000000
Bytes_received	432
Bytes_sent	15183
Com_admin_commands	1
Com_alter_db	0
Com_alter_db_upgrade	0
Com_alter_event	0
Com_alter_function	0
Com_alter_procedure	0
Com_alter_server	0
Com_alter_table	0
Com_alter_tablespace	0
Com_analyze	0
Com_assign_to_keycache	0
Com_begin	0
Com_binlog	0
Com_call_procedure	0
Com_change_db	0
Com_change_master	0
Com_check	0
Com_checksum	0
Com_commit	0
Com_compound_sql	0
Com_create_db	0
Com_create_event	0
Com_create_function	0
Com_create_index	0
Com_create_procedure	0
Com_create_role	0
Com_create_server	0
Com_create_table	0
Com_create_temporary_table	0
Com_create_trigger	0
Com_create_udf	0
Com_create_user	0

Com_create_view	0
Com_dealloc_sql	0
Com_delete	0
Com_delete_multi	0
Com_do	0
Com_drop_db	0
Com_drop_event	0
Com_drop_function	0
Com_drop_index	0
Com_drop_procedure	0
Com_drop_role	0
Com_drop_server	0
Com_drop_table	0
Com_drop_temporary_table	0
Com_drop_trigger	0
Com_drop_user	0
Com_drop_view	0
Com_empty_query	0
Com_execute_sql	0
Com_flush	0
Com_get_diagnostics	0
Com_grant	0
Com_grant_role	0
Com_ha_close	0
Com_ha_open	0
Com_ha_read	0
Com_help	0
Com_insert	0
Com_insert_select	0
Com_install_plugin	0
Com_kill	0
Com_load	0
Com_lock_tables	0
Com_optimize	0
Com_preload_keys	0
Com_prepare_sql	0

Com_purge	0
Com_purge_before_date	0
Com_release_savepoint	0
Com_rename_table	0
Com_rename_user	0
Com_repair	0
Com_replace	0
Com_replace_select	0
Com_reset	0
Com_resignal	0
Com_revoke	0
Com_revoke_all	0
Com_revoke_role	0
Com_rollback	0
Com_rollback_to_savepoint	0
Com_savepoint	0
Com_select	1
Com_set_option	0
Com_show_authors	0
Com_show_binlog_events	0
Com_show_binlogs	0
Com_showCharsets	0
Com_show_collations	0
Com_show_contributors	0
Com_show_create_db	0
Com_show_create_event	0
Com_show_create_func	0
Com_show_create_proc	0
Com_show_create_table	0
Com_show_create_trigger	0
Com_show_databases	0
Com_show_engine_logs	0
Com_show_engine_mutex	0
Com_show_engine_status	0
Com_show_errors	0
Com_show_events	0

Com_show_explain	0
Com_show_fields	0
Com_show_function_status	0
Com_show_generic	0
Com_show_grants	0
Com_show_keys	0
Com_show_master_status	0
Com_show_open_tables	0
Com_show_plugins	0
Com_show_privileges	0
Com_show_procedure_status	0
Com_show_processlist	0
Com_show_profile	0
Com_show_profiles	0
Com_show_relaylog_events	0
Com_show_slave_hosts	0
Com_show_slave_status	0
Com_show_status	2
Com_show_storage_engines	0
Com_show_table_status	0
Com_show_tables	0
Com_show_triggers	0
Com_show_variables	0
Com_show_warnings	0
Com_shutdown	0
Com_signal	0
Com_start_all_slaves	0
Com_start_slave	0
Com_stmt_close	0
Com_stmt_execute	0
Com_stmt_fetch	0
Com_stmt_prepare	0
Com_stmt_reprepare	0
Com_stmt_reset	0
Com_stmt_send_long_data	0
Com_stop_all_slaves	0
Com_stop_slave	0

Com_truncate	0
Com_uninstall_plugin	0
Com_unlock_tables	0
Com_update	0
Com_update_multi	0
Com_xa_commit	0
Com_xa_end	0
Com_xa_prepare	0
Com_xa_recover	0
Com_xa_rollback	0
Com_xa_start	0
Compression	OFF
Connection_errors_accept	0
Connection_errors_internal	0
Connection_errors_max_connections	0
Connection_errors_peer_address	0
Connection_errors_select	0
Connection_errors_tcpwrap	0
Connections	4
Cpu_time	0.000000
Created_tmp_disk_tables	0
Created_tmp_files	6
Created_tmp_tables	2
Delayed_errors	0
Delayed_insert_threads	0
Delayed_writes	0
Delete_scan	0
Empty_queries	0
Executed_events	0
Executed_triggers	0
Feature_delay_key_write	0
Feature_dynamic_columns	0
Feature_fulltext	0
Feature_gis	0
Feature_locale	0
Feature_subquery	0

Feature_timezone	0
Feature_trigger	0
Feature_xml	0
Flush_commands	1
Handler_commit	1
Handler_delete	0
Handler_discover	0
Handler_external_lock	0
Handler_icp_attempts	0
Handler_icp_match	0
Handler_mrr_init	0
Handler_mrr_key_refills	0
Handler_mrr_rowid_refills	0
Handler_prepare	0
Handler_read_first	3
Handler_read_key	0
Handler_read_last	0
Handler_read_next	0
Handler_read_prev	0
Handler_read_retry	0
Handler_read_rnd	0
Handler_read_rnd_deleted	0
Handler_read_rnd_next	537
Handler_rollback	0
Handler_savepoint	0
Handler_savepoint_rollback	0
Handler_tmp_update	0
Handler_tmp_write	516
Handler_update	0
Handler_write	0
Innodb_available_undo_logs	128
Innodb_background_log_sync	222
Innodb_buffer_pool_bytes_data	2523136
Innodb_buffer_pool_bytes_dirty	0
Innodb_buffer_pool_dump_status not yet started	Dumping buffer pool(s)
Innodb_buffer_pool_load_status not yet started	Loading buffer pool(s)

Innodb_buffer_pool_pages_data	154
Innodb_buffer_pool_pages_dirty	0
Innodb_buffer_pool_pages_flushed	1
Innodb_buffer_pool_pages_free	8037
Innodb_buffer_pool_pages_lru_flushed	0
Innodb_buffer_pool_pages_made_not_young	0
Innodb_buffer_pool_pages_made_young	0
Innodb_buffer_pool_pages_misc	0
Innodb_buffer_pool_pages_old	0
Innodb_buffer_pool_pages_total	8191
Innodb_buffer_pool_read_ahead	0
Innodb_buffer_pool_read_ahead_evicted	0
Innodb_buffer_pool_read_ahead_rnd	0
Innodb_buffer_pool_read_requests	558
Innodb_buffer_pool_reads	155
Innodb_buffer_pool_wait_free	0
Innodb_buffer_pool_write_requests	1
Innodb_checkpoint_age	0
Innodb_checkpoint_max_age	80826164
Innodb_data_fsyncs	5
Innodb_data_pending_fsyncs	0
Innodb_data_pending_reads	0
Innodb_data_pending_writes	0
Innodb_data_read	2609664
Innodb_data_reads	172
Innodb_data_writes	5
Innodb_data_written	34304
Innodb dblwr_pages_written	1
Innodb dblwr_writes	1
Innodb_deadlocks	0
Innodb_have_atomic_builtins	ON
Innodb_history_list_length	0
Innodb_ibuf_discarded_delete_marks	0
Innodb_ibuf_discarded_deletes	0
Innodb_ibuf_discarded_inserts	0
Innodb_ibuf_free_list	0
Innodb_ibuf_merged_delete_marks	0

Innodb_ibuf_merged_deletes	0
Innodb_ibuf_merged_inserts	0
Innodb_ibuf_merges	0
Innodb_ibuf_segment_size	2
Innodb_ibuf_size	1
Innodb_log_waits	0
Innodb_log_write_requests	0
Innodb_log_writes	1
Innodb_lsn_current	1616829
Innodb_lsn_flushed	1616829
Innodb_lsn_last_checkpoint	1616829
Innodb_master_thread_active_loops	0
Innodb_master_thread_idle_loops	222
Innodb_max_trx_id	2308
Innodb_mem_adaptive_hash	2217568
Innodb_mem_dictionary	630703
Innodb_mem_total	140771328
Innodb_mutex_os_waits	1
Innodb_mutex_spin_rounds	30
Innodb_mutex_spin_waits	1
Innodb_oldest_view_low_limit_trx_id	0
Innodb_os_log_fsyncs	3
Innodb_os_log_pending_fsyncs	0
Innodb_os_log_pending_writes	0
Innodb_os_log_written	512
Innodb_page_size	16384
Innodb_pages_created	0
Innodb_pages_read	154
Innodb_pages_written	1
Innodb_purge_trx_id	0
Innodb_purge_undo_no	0
Innodb_read_views_memory	88
Innodb_row_lock_current_waits	0
Innodb_row_lock_time	0
Innodb_row_lock_time_avg	0
Innodb_row_lock_time_max	0

Innodb_row_lock_waits	0
Innodb_rows_deleted	0
Innodb_rows_inserted	0
Innodb_rows_read	0
Innodb_rows_updated	0
Innodb_system_rows_deleted	0
Innodb_system_rows_inserted	0
Innodb_system_rows_read	0
Innodb_system_rows_updated	0
Innodb_s_lock_os_waits	2
Innodb_s_lock_spin_rounds	60
Innodb_s_lock_spin_waits	2
Innodb_truncated_status_writes	0
Innodb_x_lock_os_waits	0
Innodb_x_lock_spin_rounds	0
Innodb_x_lock_spin_waits	0
Innodb_page_compression_saved	0
Innodb_page_compression_trim_sect512	0
Innodb_page_compression_trim_sect1024	0
Innodb_page_compression_trim_sect2048	0
Innodb_page_compression_trim_sect4096	0
Innodb_page_compression_trim_sect8192	0
Innodb_page_compression_trim_sect16384	0
Innodb_page_compression_trim_sect32768	0
Innodb_num_index_pages_written	0
Innodb_num_non_index_pages_written	5
Innodb_num_pages_page_compressed	0
Innodb_num_page_compressed_trim_op	0
Innodb_num_page_compressed_trim_op_saved	0
Innodb_num_pages_page_decompressed	0
Innodb_num_pages_page_compression_error	0
Innodb_num_pages_encrypted	0
Innodb_num_pages_decrypted	0
Innodb_have_lz4	OFF
Innodb_have_lzo	OFF
Innodb_have_lzma	OFF

Innodb_have_bzip2	OFF
Innodb_have_snappy	OFF
Innodb_defragment_compression_failures	0
Innodb_defragment_failures	0
Innodb_defragment_count	0
Innodb_onlineddl_rowlog_rows	0
Innodb_onlineddl_rowlog_pct_used	0
Innodb_onlineddl_pct_progress	0
Innodb_secondary_index_triggered_cluster_reads	0
Innodb_secondary_index_triggered_cluster_reads_avoided	0
Innodb_encryption_rotation_pages_read_from_cache	0
Innodb_encryption_rotation_pages_read_from_disk	0
Innodb_encryption_rotation_pages_modified	0
Innodb_encryption_rotation_pages_flushed	0
Innodb_encryption_rotation_estimated_iops	0
Innodb_scrub_background_page_reorganizations	0
Innodb_scrub_background_page_splits	0
Innodb_scrub_background_page_split_failures_underflow	0
Innodb_scrub_background_page_split_failures_out_of_filespace	0
Innodb_scrub_background_page_split_failures_missing_index	0
Innodb_scrub_background_page_split_failures_unknown	0
Key_blocks_not_flushed	0
Key_blocks_unused	107163
Key_blocks_used	0
Key_blocks_warm	0
Key_read_requests	0
Key_reads	0
Key_write_requests	0
Key_writes	0
Last_query_cost	0.000000
Master_gtid_wait_count	0
Master_gtid_wait_time	0
Master_gtid_wait_timeouts	0
Max_statement_time_exceeded	0
Max_used_connections	1
Memory_used	273614696
Not_flushed_delayed_rows	0

NOT的信任延迟行	0
Open_files	25
Open_streams	0
Open_table_definitions	18
Open_tables	11
Opened_files	77
Opened_plugin_libraries	0
Opened_table_definitions	18
Opened_tables	18
Opened_views	0
Performance_schema_accounts_lost	0
Performance_schema_cond_classes_lost	0
Performance_schema_cond_instances_lost	0
Performance_schema_digest_lost	0
Performance_schema_file_classes_lost	0
Performance_schema_file_handles_lost	0
Performance_schema_file_instances_lost	0
Performance_schema_hosts_lost	0
Performance_schema_locker_lost	0
Performance_schema_mutex_classes_lost	0
Performance_schema_mutex_instances_lost	0
Performance_schema_rwlock_classes_lost	0
Performance_schema_rwlock_instances_lost	0
Performance_schema_session_connect_attrs_lost	0
Performance_schema_socket_classes_lost	0
Performance_schema_socket_instances_lost	0
Performance_schema_stage_classes_lost	0
Performance_schema_statement_classes_lost	0
Performance_schema_table_handles_lost	0
Performance_schema_table_instances_lost	0
Performance_schema_thread_classes_lost	0
Performance_schema_thread_instances_lost	0
Performance_schema_users_lost	0
Prepared_stmt_count	0
Qcache_free_blocks	1
Qcache_free_memory	1031336
Qcache_hits	0

Qcache_inserts	0
Qcache_lowmem_prunes	0
Qcache_not_cached	0
Qcache_queries_in_cache	0
Qcache_total_blocks	1
Queries	4
Questions	4
Rows_read	10
Rows_sent	517
Rows_tmp_read	516
Rpl_status	AUTH_MASTER
Select_full_join	0
Select_full_range_join	0
Select_range	0
Select_range_check	0
Select_scan	2
Slave_connections	0
Slave_heartbeat_period	0.000
Slave_open_temp_tables	0
Slave_received_heartbeats	0
Slave_retried_transactions	0
Slave_running	OFF
Slave_skipped_errors	0
Slaves_connected	0
Slaves_running	0
Slow_launch_threads	0
Slow_queries	0
Sort_merge_passes	0
Sort_priority_queue_sorts	0
Sort_range	0
Sort_rows	0
Sort_scan	0
Ssl_accept_renegotiates	0
Ssl_accepts	0
Ssl_callback_cache_hits	0
Ssl_cipher	

Ssl_cipher_list	
Ssl_client_connects	0
Ssl_connect_renegotiates	0
Ssl_ctx_verify_depth	0
Ssl_ctx_verify_mode	0
Ssl_default_timeout	0
Ssl_finished_accepts	0
Ssl_finished_connects	0
Ssl_server_not_after	
Ssl_server_not_before	
Ssl_session_cache_hits	0
Ssl_session_cache_misses	0
Ssl_session_cache_mode	NONE
Ssl_session_cache_overflows	0
Ssl_session_cache_size	0
Ssl_session_cache_timeouts	0
Ssl_sessions_reused	0
Ssl_used_session_cache_entries	0
Ssl_verify_depth	0
Ssl_verify_mode	0
Ssl_version	
Subquery_cache_hit	0
Subquery_cache_miss	0
Syncs	2
Table_locks_immediate	21
Table_locks_waited	0
Tc_log_max_pages_used	0
Tc_log_page_size	4096
Tc_log_page_waits	0
Threadpool_idle_threads	0
Threadpool_threads	0
Threads_cached	0
Threads_connected	1
Threads_created	2
Threads_running	1
Update_scan	0

Uptime	223
Uptime_since_flush_status	223
wsrep_cluster_conf_id	18446744073709551615
wsrep_cluster_size	0
wsrep_cluster_state_uuid	
wsrep_cluster_status	Disconnected
wsrep_connected	OFF
wsrep_local_bf_aborts	0
wsrep_local_index	18446744073709551615
wsrep_provider_name	
wsrep_provider_vendor	
wsrep_provider_version	
wsrep_ready	OFF
wsrep_thread_count	0

+-----+
-----+
516 rows in set (0.00 sec)

Example of filtered output:

SHOW STATUS LIKE 'Key%';		
Variable_name	Value	
Key_blocks_not_flushed	0	
Key_blocks_unused	107163	
Key_blocks_used	0	
Key_blocks_warm	0	
Key_read_requests	0	
Key_reads	0	
Key_write_requests	0	
Key_writes	0	

+-----+
-----+
8 rows in set (0.00 sec)

H3Dr1.2.8.51 SHOW TABLE STATUS

Syntax

```
SHOW TABLE STATUS [{FROM | IN} db_name]
[LIKE 'pattern' | WHERE expr]
```

Contents

1. [Syntax](#)
2. [Description](#)
3. [Views](#)
4. [Example](#)

Description

`SHOW TABLE STATUS` works like `SHOW TABLES`, but provides more extensive information about each non- TEMPORARY table.

The `LIKE` clause, if present on its own, indicates which table names to match. The `WHERE` and `LIKE` clauses can be given to select rows using more general conditions, as discussed in [Extended SHOW](#).

The following information is returned:

Column	Description
Name	Table name.
Engine	Table storage engine 🔗 .
Version	Version number from the table's .frm file.
Row_format	Row format (see InnoDB 🔗 , Aria 🔗 and MyISAM 🔗 row formats).
Rows	Number of rows in the table. Some engines, such as XtraDB and InnoDB 🔗 may store an estimate.
Avg_row_length	Average row length in the table.
Data_length	For InnoDB/XtraDB 🔗 , the index size, in pages, multiplied by the page size. For Aria 🔗 and MyISAM 🔗 , length of the data file, in bytes. For MEMORY 🔗 , the approximate allocated memory.
Max_data_length	Maximum length of the data file, ie the total number of bytes that could be stored in the table. Not used in XtraDB and InnoDB 🔗 .
Index_length	Length of the index file.
Data_free	Bytes allocated but unused. For InnoDB tables in a shared tablespace, the free space of the shared tablespace with small safety margin. An estimate in the case of partitioned tables - see the PARTITIONS 🔗 table.
Auto_increment	Next AUTO_INCREMENT 🔗 value.
Create_time	Time the table was created.
Update_time	Time the table was last updated. On Windows, the timestamp is not updated on update, so MyISAM values will be inaccurate. In InnoDB 🔗 , if shared tablespaces are used, will be <code>NULL</code> , while buffering can also delay the update, so the value will differ from the actual time of the last <code>UPDATE</code> , <code>INSERT</code> or <code>DELETE</code> .
Check_time	Time the table was last checked. Not kept by all storage engines, in which case will be <code>NULL</code> .
Collation	Character set and collation 🔗 .
Checksum	Live checksum value, if any.
Create_options	Extra <code>CREATE TABLE</code> options.
Comment	Table comment provided when MariaDB created the table.
Max_index_length	Maximum index length (supported by MyISAM and Aria tables). Added in MariaDB 10.3.5 🔗 .
Temporary	Placeholder to signal that a table is a temporary table. Currently always "N", except "Y" for generated information_schema tables and NULL for views 🔗 . Added in MariaDB 10.3.5 🔗 .

Similar information can be found in the `information_schema.TABLES` [🔗](#) table as well as by using `mysqlshow` [🔗](#):

```
mysqlshow --status db_name
```

Views

For views, all columns in `SHOW TABLE STATUS` are `NULL` except 'Name' and 'Comment'

Example

```
show table status\G
***** 1. row *****
  Name: bus_routes
  Engine: InnoDB
  Version: 10
Row_format: Dynamic
  Rows: 5
Avg_row_length: 3276
  Data_length: 16384
Max_data_length: 0
  Index_length: 0
  Data_free: 0
Auto_increment: NULL
  Create_time: 2017-05-24 11:17:46
  Update_time: NULL
  Check_time: NULL
  Collation: latin1_swedish_ci
  Checksum: NULL
Create_options:
  Comment:
```

H3Dr1.2.8.52 SHOW TABLES

Syntax

```
SHOW [FULL] TABLES [FROM db_name]
      [LIKE 'pattern' | WHERE expr]
```

Contents

1. [Syntax](#)
2. [Description](#)
3. [Examples](#)
4. [See Also](#)

Description

`SHOW TABLES` lists the non-TEMPORARY tables, [sequences](#) and [views](#) in a given database.

The `LIKE` clause, if present on its own, indicates which table names to match. The `WHERE` and `LIKE` clauses can be given to select rows using more general conditions, as discussed in [Extended SHOW](#). For example, when searching for tables in the `test` database, the column name for use in the `WHERE` and `LIKE` clauses will be `Tables_in_test`.

The `FULL` modifier is supported such that `SHOW FULL TABLES` displays a second output column. Values for the second column, `Table_type`, are `BASE TABLE` for a table, `VIEW` for a [view](#) and `SEQUENCE` for a [sequence](#).

You can also get this information using:

```
mysqlshow db_name
```

See [mysqlshow](#) for more details.

If you have no privileges for a base table or view, it does not show up in the output from `SHOW TABLES` or `mysqlshow db_name`.

The [information_schema.TABLES](#) table, as well as the [SHOW TABLE STATUS](#) statement, provide extended information about tables.

Examples

```
SHOW TABLES;
+-----+
| Tables_in_test      |
+-----+
| animal_count        |
| animals              |
| are_the_moooses_loose |
| aria_test2           |
| t1                   |
| view1                |
+-----+
```

Showing the tables beginning with a only.

```
SHOW TABLES WHERE Tables_in_test LIKE 'a%';
+-----+
| Tables_in_test      |
+-----+
| animal_count        |
| animals              |
| are_the_moooses_loose |
| aria_test2           |
+-----+
```

Showing tables and table types:

```
SHOW FULL TABLES;
+-----+-----+
| Tables_in_test | Table_type |
+-----+-----+
| s1            | SEQUENCE   |
| student       | BASE TABLE |
| v1            | VIEW        |
+-----+-----+
```

See Also

- [SHOW TABLE STATUS](#)
- The [information_schema.TABLES](#) table

H3Dr1.2.8.53 SHOW TABLE_STATISTICS

Syntax

```
SHOW TABLE_STATISTICS
```

Description

The `SHOW TABLE_STATISTICS` statement is part of the [User Statistics](#) feature. It was removed as a separate statement in [MariaDB 10.1.1](#), but effectively replaced by the generic `SHOW information_schema_table` statement. The [information_schema.TABLE_STATISTICS](#) table shows statistics on table usage.

The `userstat` system variable must be set to 1 to activate this feature. See the [User Statistics](#) and [information_schema.TABLE_STATISTICS](#) articles for more information.

Example

```
SHOW TABLE_STATISTICS\G
*****
1. row ****
Table_schema: mysql
Table_name: proxies_priv
Rows_read: 2
Rows_changed: 0
Rows_changed_x_indexes: 0
*****
2. row ****
Table_schema: test
Table_name: employees_example
Rows_read: 7
Rows_changed: 0
Rows_changed_x_indexes: 0
*****
3. row ****
Table_schema: mysql
Table_name: user
Rows_read: 16
Rows_changed: 0
Rows_changed_x_indexes: 0
*****
4. row ****
Table_schema: mysql
Table_name: db
Rows_read: 2
Rows_changed: 0
Rows_changed_x_indexes: 0
```

H3Dr1.2.8.54 SHOW TRIGGERS

Syntax

```
SHOW TRIGGERS [FROM db_name]
[LIKE 'pattern' | WHERE expr]
```

Contents

1. [Syntax](#)
2. [Description](#)
3. [Examples](#)
4. [See also](#)

Description

`SHOW TRIGGERS` lists the triggers currently defined for tables in a database (the default database unless a `FROM` clause is given). This statement requires the `TRIGGER` privilege (prior to MySQL 5.1.22, it required the `SUPER` privilege).

The `LIKE` clause, if present on its own, indicates which table names to match and causes the statement to display triggers for those tables. The `WHERE` and `LIKE` clauses can be given to select rows using more general conditions, as discussed in [Extended SHOW](#).

Similar information is stored in the `information_schema.TRIGGERS` table.

MariaDB starting with [10.2.3](#)

If there are multiple triggers for the same action, then the triggers are shown in action order.

Examples

For the trigger defined at [Trigger Overview](#):

```

SHOW triggers Like 'animals' \G
*****
1. row *****
Trigger: the_mooses_are_loose
Event: INSERT
Table: animals
Statement: BEGIN
IF NEW.name = 'Moose' THEN
UPDATE animal_count SET animal_count.animals = animal_count.animals+100;
ELSE
UPDATE animal_count SET animal_count.animals = animal_count.animals+1;
END IF;
END
Timing: AFTER
Created: 2016-09-29 13:53:34.35
sql_mode:
Definer: root@localhost
character_set_client: utf8
collation_connection: utf8_general_ci
Database Collation: latin1_swedish_ci

```

Listing all triggers associated with a certain table:

```

SHOW TRIGGERS FROM test WHERE `Table` = 'user' \G
*****
1. row *****
Trigger: user_ai
Event: INSERT
Table: user
Statement: BEGIN END
Timing: AFTER
Created: 2016-09-29 13:53:34.35
sql_mode:
Definer: root@%
character_set_client: utf8
collation_connection: utf8_general_ci
Database Collation: latin1_swedish_ci

```

```

SHOW triggers WHERE Event Like 'Insert' \G
*****
1. row *****
Trigger: the_mooses_are_loose
Event: INSERT
Table: animals
Statement: BEGIN
IF NEW.name = 'Moose' THEN
UPDATE animal_count SET animal_count.animals = animal_count.animals+100;
ELSE
UPDATE animal_count SET animal_count.animals = animal_count.animals+1;
END IF;
END
Timing: AFTER
Created: 2016-09-29 13:53:34.35
sql_mode:
Definer: root@localhost
character_set_client: utf8
collation_connection: utf8_general_ci
Database Collation: latin1_swedish_ci

```

- `character_set_client` is the session value of the [character_set_client](#) system variable when the trigger was created.
- `collation_connection` is the session value of the [collation_connection](#) system variable when the trigger was created.
- `Database Collation` is the collation of the database with which the trigger is associated.

These columns were added in MariaDB/MySQL 5.1.21.

Old triggers created before MySQL 5.7 and [MariaDB 10.2.3](#) has NULL in the `Created` column.

See also

- [Trigger Overview](#)
- [CREATE TRIGGER](#)
- [DROP TRIGGER](#)
- [information_schema.TRIGGERS](#) table
- [SHOW CREATE TRIGGER](#)
- [Trigger Limitations](#)

H3Dr1.2.8.55 SHOW USER_STATISTICS

Syntax

```
SHOW USER_STATISTICS
```

Description

The `SHOW USER_STATISTICS` statement is part of the [User Statistics](#) feature. It was removed as a separate statement in [MariaDB 10.1.1](#), but effectively replaced by the generic `SHOW information_schema_table` statement. The [information_schema.USER_STATISTICS](#) table holds statistics about user activity. You can use this table to find out such things as which user is causing the most load and which users are being abusive. You can also use this table to measure how close to capacity the server may be.

The `userstat` system variable must be set to 1 to activate this feature. See the [User Statistics](#) and [information_schema.USER_STATISTICS](#) table for more information.

Example

```
SHOW USER_STATISTICS\G
*****
1. row ****
    User: root
    Total_connections: 1
    Concurrent_connections: 0
        Connected_time: 3297
        Busy_time: 0.1411340000000006
        Cpu_time: 0.01763700000000003
        Bytes_received: 969
        Bytes_sent: 22355
    Binlog_bytes_written: 0
        Rows_read: 10
        Rows_sent: 67
        Rows_deleted: 0
        Rows_inserted: 0
        Rows_updated: 0
    Select_commands: 7
    Update_commands: 0
    Other_commands: 0
    Commit_transactions: 1
    Rollback_transactions: 0
        Denied_connections: 0
        Lost_connections: 0
        Access_denied: 0
        Empty_queries: 7
```

H3Dr1.2.8.56 SHOW VARIABLES

Syntax

```
SHOW [GLOBAL | SESSION] VARIABLES  
[LIKE 'pattern' | WHERE expr]
```

Contents

1. [Syntax](#)
2. [Description](#)
3. [Examples](#)

Description

`SHOW VARIABLES` shows the values of MariaDB [system variables](#). This information also can be obtained using the [mysqladmin](#) variables command. The `LIKE` clause, if present, indicates which variable names to match. The `WHERE` clause can be given to select rows using more general conditions.

With the `GLOBAL` modifier, `SHOW VARIABLES` displays the values that are used for new connections to MariaDB. With `SESSION`, it displays the values that are in effect for the current connection. If no modifier is present, the default is `SESSION`. `LOCAL` is a synonym for `SESSION`. With a `LIKE` clause, the statement displays only rows for those variables with names that match the pattern. To obtain the row for a specific variable, use a `LIKE` clause as shown:

```
SHOW VARIABLES LIKE 'maria_group_commit';  
SHOW SESSION VARIABLES LIKE 'maria_group_commit';
```

To get a list of variables whose name match a pattern, use the " % " wildcard character in a `LIKE` clause:

```
SHOW VARIABLES LIKE '%maria%';  
SHOW GLOBAL VARIABLES LIKE '%maria%';
```

Wildcard characters can be used in any position within the pattern to be matched. Strictly speaking, because " _ " is a wildcard that matches any single character, you should escape it as " _ " to match it literally. In practice, this is rarely necessary.

The `WHERE` and `LIKE` clauses can be given to select rows using more general conditions, as discussed in [Extended SHOW](#).

See [SET](#) for information on setting server system variables.

See [Server System Variables](#) for a list of all the variables that can be set.

You can also see the server variables by querying the [Information Schema GLOBAL_VARIABLES](#) and [SESSION_VARIABLES](#) tables.

Examples

```
SHOW VARIABLES LIKE 'aria%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| aria_block_size | 8192 |
| aria_checkpoint_interval | 30 |
| aria_checkpoint_log_activity | 1048576 |
| aria_force_start_after_recovery_failures | 0 |
| aria_group_commit | none |
| aria_group_commit_interval | 0 |
| aria_log_file_size | 1073741824 |
| aria_log_purge_type | immediate |
| aria_max_sort_file_size | 9223372036853727232 |
| aria_page_checksum | ON |
| aria_pagecache_age_threshold | 300 |
| aria_pagecache_buffer_size | 134217728 |
| aria_pagecache_division_limit | 100 |
| aria_recover | NORMAL |
| aria_repair_threads | 1 |
| aria_sort_buffer_size | 134217728 |
| aria_stats_method | nulls_unequal |
| aria_sync_log_dir | NEWFILE |
| aria_used_for_temp_tables | ON |
+-----+-----+
```

```
SELECT VARIABLE_NAME, SESSION_VALUE, GLOBAL_VALUE FROM
INFORMATION_SCHEMA.SYSTEM_VARIABLES WHERE
VARIABLE_NAME LIKE 'max_error_count' OR
VARIABLE_NAME LIKE 'innodb_sync_spin_loops';
```

```
+-----+-----+-----+
| VARIABLE_NAME | SESSION_VALUE | GLOBAL_VALUE |
+-----+-----+-----+
| MAX_ERROR_COUNT | 64 | 64 |
| INNODB_SYNC_SPIN_LOOPS | NULL | 30 |
+-----+-----+-----+
```

```
SET GLOBAL max_error_count=128;
```

```
SELECT VARIABLE_NAME, SESSION_VALUE, GLOBAL_VALUE FROM
INFORMATION_SCHEMA.SYSTEM_VARIABLES WHERE
VARIABLE_NAME LIKE 'max_error_count' OR
VARIABLE_NAME LIKE 'innodb_sync_spin_loops';
```

```
+-----+-----+-----+
| VARIABLE_NAME | SESSION_VALUE | GLOBAL_VALUE |
+-----+-----+-----+
| MAX_ERROR_COUNT | 64 | 128 |
| INNODB_SYNC_SPIN_LOOPS | NULL | 30 |
+-----+-----+-----+
```

```
SET GLOBAL max_error_count=128;
```

```
SHOW VARIABLES LIKE 'max_error_count';
```

```
+-----+
| Variable_name | Value |
+-----+
| max_error_count | 64 |
+-----+
```

```
SHOW GLOBAL VARIABLES LIKE 'max_error_count';
```

```
+-----+
| Variable_name | Value |
+-----+
| max_error_count | 128 |
+-----+
```

Because the following variable only has a global scope, the global value is returned even when specifying SESSION (in this case by default):

```
SHOW VARIABLES LIKE 'innodb_sync_spin_loops';
+-----+-----+
| Variable_name      | Value   |
+-----+-----+
| innodb_sync_spin_loops | 30     |
+-----+-----+
```

H3Dr1.2.8.57 SHOW WARNINGS

Syntax

```
SHOW WARNINGS [LIMIT [offset,] row_count]
SHOW ERRORS [LIMIT row_count OFFSET offset]
SHOW COUNT(*) WARNINGS
```

Contents

1. [Syntax](#)
2. [Description](#)
3. [Examples](#)
 1. [Stack Trace](#)
4. [See Also](#)

Description

`SHOW WARNINGS` shows the error, warning, and note messages that resulted from the last statement that generated messages in the current session. It shows nothing if the last statement used a table and generated no messages. (That is, a statement that uses a table but generates no messages clears the message list.) Statements that do not use tables and do not generate messages have no effect on the message list.

A note is different to a warning in that it only appears if the [sql_notes](#) variable is set to 1 (the default), and is not converted to an error if [strict mode](#) is enabled.

A related statement, `SHOW ERRORS`, shows only the errors.

The `SHOW COUNT(*) WARNINGS` statement displays the total number of errors, warnings, and notes. You can also retrieve this number from the [warning_count](#) variable:

```
SHOW COUNT(*) WARNINGS;
SELECT @@warning_count;
```

The value of [warning_count](#) might be greater than the number of messages displayed by `SHOW WARNINGS` if the [max_error_count](#) system variable is set so low that not all messages are stored.

The `LIMIT` clause has the same syntax as for the [SELECT statement](#).

`SHOW WARNINGS` can be used after `EXPLAIN EXTENDED` to see how a query is internally rewritten by MariaDB.

If the [sql_notes](#) server variable is set to 1, Notes are included in the output of `SHOW WARNINGS`; if it is set to 0, this statement will not show (or count) Notes.

The results of `SHOW WARNINGS` and `SHOW COUNT(*) WARNINGS` are directly sent to the client. If you need to access those information in a stored program, you can use the [GET DIAGNOSTICS](#) statement instead.

For a list of MariaDB error codes, see [MariaDB Error Codes](#).

The [mysql](#) client also has a number of options related to warnings. The `\w` command will show warnings after every statement, while `\W` will disable this. Starting the client with the `--show-warnings` option will show warnings after every statement.

[MariaDB 10.3.1](#) implements a stored routine error stack trace. `SHOW WARNINGS` can also be used to show more information. See the example below.

Examples

```
SELECT 1/0;
+-----+
| 1/0  |
+-----+
| NULL |
+-----+

SHOW COUNT(*) WARNINGS;
+-----+
| @@session.warning_count |
+-----+
|                      1 |
+-----+

SHOW WARNINGS;
+-----+-----+-----+
| Level   | Code  | Message        |
+-----+-----+-----+
| Warning | 1365  | Division by 0 |
+-----+-----+-----+
```

Stack Trace

From [MariaDB 10.3.1](#), displaying a stack trace:

```
DELIMITER $$  
CREATE OR REPLACE PROCEDURE p1()  
BEGIN  
    DECLARE c CURSOR FOR SELECT * FROM not_existing;  
    OPEN c;  
    CLOSE c;  
END;  
$$  
CREATE OR REPLACE PROCEDURE p2()  
BEGIN  
    CALL p1;  
END;  
$$  
DELIMITER ;  
CALL p2;  
ERROR 1146 (42S02): Table 'test.not_existing' doesn't exist  
  
SHOW WARNINGS;
+-----+-----+-----+
| Level | Code  | Message        |
+-----+-----+-----+
| Error | 1146  | Table 'test.not_existing' doesn't exist |
| Note  | 4091  | At line 6 in test.p1                    |
| Note  | 4091  | At line 4 in test.p2                    |
+-----+-----+-----+
```

SHOW WARNINGS displays a stack trace, showing where the error actually happened:

- Line 4 in test.p1 is the OPEN command which actually raised the error
- Line 3 in test.p2 is the CALL statement, calling p1 from p2.

See Also

- [SHOW ERRORS](#)

H3Dr1.2.8.58 SHOW WSREP_MEMBERSHIP

SHOW WSREP_MEMBERSHIP is part of the [WSREP_INFO](#) plugin.

Syntax

```
SHOW WSREP_MEMBERSHIP
```

Description

The `SHOW WSREP_MEMBERSHIP` statement returns Galera node cluster membership information. It returns the same information as found in the `information_schema.WSREP_MEMBERSHIP` table. Only users with the `SUPER` privilege can access this information.

Examples

```
SHOW WSREP_MEMBERSHIP;
+-----+-----+-----+
| Index | Uuid                | Name      | Address   |
+-----+-----+-----+
| 0     | 19058073-8940-11e4-8570-16af7bf8fcfd | my_node1  | 10.0.2.15:16001 |
| 1     | 19f2b0e0-8942-11e4-9cb8-b39e8ee0b5dd | my_node3  | 10.0.2.15:16003 |
| 2     | d85e62db-8941-11e4-b1ef-4bc9980e476d | my_node2  | 10.0.2.15:16002 |
+-----+-----+-----+
```

H3Dr1.2.8.59 SHOW WSREP_STATUS

SHOW WSREP_STATUS is part of the [WSREP_INFO](#) plugin.

Syntax

```
SHOW WSREP_STATUS
```

Description

The `SHOW WSREP_STATUS` statement returns Galera node and cluster status information. It returns the same information as found in the `information_schema.WSREP_STATUS` table. Only users with the `SUPER` privilege can access this information.

Examples

```
SHOW WSREP_STATUS;
+-----+-----+-----+-----+
| Node_Index | Node_Status | Cluster_Status | Cluster_Size |
+-----+-----+-----+-----+
|          0 | Synced      | Primary       |          3 |
+-----+-----+-----+-----+
```

H3Dr1.2.9 System Tables



Information Schema

[Articles about the Information Schema](#)



Performance Schema

[Monitoring server performance](#)



The mysql Database Tables

mysql database tables. ↗



Sys Schema

Collection of views, functions and procedures to help administrators get in... ↗



mariadb_schema

mariadb_schema is used to enforce MariaDB native types independent of SQL_MODE. ↗



Writing Logs Into Tables

The general query log and the slow query log can be written into system tables ↗

H3Dr1.2.9.1 Information Schema

Articles about the Information Schema



Information Schema Tables

Tables in the INFORMATION_SCHEMA database.



Extended Show

Extended SHOW with WHERE and LIKE.



TIME_MS column in INFORMATION_SCHEMA.PROCESSLIST

Microseconds in the INFORMATION_SCHEMA.PROCESSLIST table ↗

There are 1 related questions ↗.

H3Dr1.2.9.1.1 Information Schema Tables



Information Schema InnoDB Tables

All InnoDB-specific Information Schema tables.



Information Schema MyRocks Tables

List of Information Schema tables specifically related to MyRocks.



Information Schema XtraDB Tables

All XtraDB-specific Information Schema tables. ↗



ColumnStore Information Schema Tables

ColumnStore-related Information Schema tables ↗



Information Schema ALL_PLUGINS Table

Information about server plugins, whether installed or not. ↗



Information Schema APPLICABLE_ROLES Table

Roles available to be used. ↗



Information Schema CHARACTER_SETS Table

Supported character sets. ↗



Information Schema CHECK_CONSTRAINTS Table

Supported check constraints. ↗



Information Schema CLIENT_STATISTICS Table

Statistics about client connections. ↗

-  **Information Schema COLLATION_CHARACTER_SET_APPLICABILITY Table**
Collations and associated character sets. [🔗](#)
-  **Information Schema COLLATIONS Table**
Supported collations. [🔗](#)
-  **Information Schema COLUMN_PRIVILEGES Table**
Column privileges. [🔗](#)
-  **Information Schema COLUMNS Table**
Information about table fields. [🔗](#)
-  **Information Schema DISKS Table**
Plugin that allows the disk space situation to be monitored. [🔗](#)
-  **Information Schema ENABLED_ROLES Table**
Enabled roles for the current session. [🔗](#)
-  **Information Schema ENGINES Table**
Storage engine information. [🔗](#)
-  **Information Schema EVENTS Table**
Server event information. [🔗](#)
-  **Information Schema FEEDBACK Table**
Contents submitted by the Feedback Plugin. [🔗](#)
-  **Information Schema FILES Table**
The FILES tables is unused in MariaDB. [🔗](#)
-  **Information Schema GEOMETRY_COLUMNS Table**
Support for Spatial Reference systems for GIS data. [🔗](#)
-  **Information Schema GLOBAL_STATUS and SESSION_STATUS Tables**
Global and session status variables. [🔗](#)
-  **Information Schema GLOBAL_VARIABLES and SESSION_VARIABLES Tables**
Global and session system variables. [🔗](#)
-  **Information Schema INDEX_STATISTICS Table**
Statistics on index usage. [🔗](#)
-  **Information Schema KEY_CACHES Table**
Segmented key cache statistics. [🔗](#)
-  **Information Schema KEY_COLUMN_USAGE Table**
Key columns that have constraints. [🔗](#)
-  **Information Schema KEYWORDS Table**
MariaDB keywords. [🔗](#)
-  **Information Schema LOCALES Table**
Compiled-in server locales. [🔗](#)
-  **Information Schema METADATA_LOCK_INFO Table**
Active metadata locks. [🔗](#)
-  **Information Schema MROONGA_STATS Table**
Mroonga activities statistics. [🔗](#)

-  **Information Schema OPTIMIZER_TRACE Table**
Contains Optimizer Trace information. ↗
-  **Information Schema PARAMETERS Table**
Information about stored procedures and stored functions parameters. ↗
-  **Information Schema PARTITIONS Table**
Table partition information ↗
-  **Information Schema PLUGINS Table**
Information Schema table containing information on plugins installed on a server. ↗
-  **Information Schema PROCESSLIST Table**
Thread information. ↗
-  **Information Schema PROFILING Table**
Statement resource usage ↗
-  **Information Schema QUERY_CACHE_INFO Table**
View the contents of the query cache. ↗
-  **Information Schema QUERY_RESPONSE_TIME Table**
Query time information. ↗
-  **Information Schema REFERENTIAL_CONSTRAINTS Table**
Foreign key information ↗
-  **Information Schema ROUTINES Table**
Stored procedures and stored functions information ↗
-  **Information Schema SCHEMA_PRIVILEGES Table**
Database privilege information ↗
-  **Information Schema SCHEMATA Table**
Information about databases. ↗
-  **Information Schema SPATIAL_REF_SYS Table**
Information on each spatial reference system used in the database ↗
-  **Information Schema SPIDER_ALLOC_MEM Table**
Information about Spider's memory usage. ↗
-  **Information Schema SPIDER_WRAPPER_PROTOCOLS Table**
Installed along with the Spider storage engine. ↗
-  **Information Schema SQL_FUNCTIONS Table**
Functions in MariaDB. ↗
-  **Information Schema STATISTICS Table**
Table index information. ↗
-  **Information Schema SYSTEM_VARIABLES Table**
Current global and session values and various metadata of all system variables. ↗
-  **Information Schema TABLE_CONSTRAINTS Table**
Tables containing constraints. ↗
-  **Information Schema TABLE_PRIVILEGES Table**
Table privileges ↗

-  **Information Schema TABLE_STATISTICS Table**
Statistics on table usage. [🔗](#)
-  **Information Schema TABLES Table**
Database table information. [🔗](#)
-  **Information Schema TABLESPACES Table**
Information about active tablespaces. [🔗](#)
-  **Information Schema THREAD_POOL_GROUPS Table**
Information Schema THREAD_POOL_GROUPS Table. [🔗](#)
-  **Information Schema THREAD_POOL_QUEUES Table**
Information Schema THREAD_POOL_QUEUES Table. [🔗](#)
-  **Information Schema THREAD_POOL_STATS Table**
Information Schema THREAD_POOL_STATS Table. [🔗](#)
-  **Information Schema THREAD_POOL_WAITS Table**
Information Schema THREAD_POOL_WAITS Table. [🔗](#)
-  **Information Schema TRIGGERS Table**
Information about triggers [🔗](#)
-  **Information Schema USER_PRIVILEGES Table**
Global user privilege information derived from the mysql.user grant table [🔗](#)
-  **Information Schema USER_STATISTICS Table**
User activity [🔗](#)
-  **Information Schema USER_VARIABLES Table**
User-defined variable information. [🔗](#)
-  **Information Schema VIEWS Table**
Information about views. [🔗](#)
-  **Information Schema WSREP_MEMBERSHIP Table**
Galera node cluster membership information. [🔗](#)
-  **Information Schema WSREP_STATUS Table**
Galera node cluster status information. [🔗](#)

There are [2 related questions](#).

H3Dr1.2.9.1.1.1 Information Schema InnoDB Tables

List of Information Schema tables specifically related to [InnoDB](#). Tables that are specific to XtraDB shares with InnoDB are listed in [Information Schema XtraDB Tables](#).

-  **Information Schema INNODB_BUFFER_PAGE Table**
Buffer pool page information.
-  **Information Schema INNODB_BUFFER_PAGE_LRU Table**
Buffer pool pages and their eviction order.
-  **Information Schema INNODB_BUFFER_POOL_PAGES Table**
XtraDB buffer pool page information.

-  **Information Schema INNODB_BUFFER_POOL_PAGES_BLOB Table**
XtraDB buffer pool blob pages.
-  **Information Schema INNODB_BUFFER_POOL_PAGES_INDEX Table**
XtraDB buffer pool index pages.
-  **Information Schema INNODB_BUFFER_POOL_STATS Table**
InnoDB buffer pool information.
-  **Information Schema INNODB_CHANGED_PAGES Table**
Modified pages from the bitmap file data.
-  **Information Schema INNODB_CMP and INNODB_CMP_RESET Tables**
XtraDB/InnoDB compression performances with different page sizes.
-  **Information Schema INNODB_CMPMEM and INNODB_CMPMEM_RESET Tables**
Number of InnoDB compressed pages of different page sizes.
-  **Information Schema INNODB_CMP_PER_INDEX and INNODB_CMP_PER_INDEX_RESET Tables**
XtraDB/InnoDB compression performances for different indexes and tables.
-  **Information Schema INNODB_FT_BEING_DELETED Table**
Fulltext being deleted.
-  **Information Schema INNODB_FT_CONFIG Table**
InnoDB fulltext metadata.
-  **Information Schema INNODB_FT_DEFAULT_STOPWORD Table**
Default InnoDB stopwords.
-  **Information Schema INNODB_FT_DELETED Table**
Deleted InnoDB fulltext rows.
-  **Information Schema INNODB_FT_INDEX_CACHE Table**
Newly added fulltext row information.
-  **Information Schema INNODB_FT_INDEX_TABLE Table**
InnoDB fulltext information.
-  **Information Schema INNODB_LOCK_WAITS Table**
Blocked InnoDB transactions.
-  **Information Schema INNODB_LOCKS Table**
InnoDB lock information.
-  **Information Schema INNODB_METRICS Table**
InnoDB performance metrics.
-  **Information Schema INNODB_MUTEXES Table**
Monitor mutex waits.
-  **Information Schema INNODB_SYS_COLUMNS Table**
InnoDB column information.
-  **Information Schema INNODB_SYS_DATAFILES Table**
InnoDB tablespace paths.
-  **Information Schema INNODB_SYS_FIELDS Table**
Fields part of an InnoDB index.

-  **Information Schema INNODB_SYS_FOREIGN Table**
InnoDB foreign key information.
-  **Information Schema INNODB_SYS_FOREIGN_COLS Table**
Foreign key column information.
-  **Information Schema INNODB_SYS_INDEXES Table**
InnoDB index information.
-  **Information Schema INNODB_SYS_SEMAPHORE_WAITS Table**
Information about current semaphore waits.
-  **Information Schema INNODB_SYS_TABLES Table**
InnoDB table information.
-  **Information Schema INNODB_SYS_TABLESPACES Table**
InnoDB tablespace information.
-  **Information Schema INNODB_SYS_TABLESTATS Table**
InnoDB status for high-level performance monitoring.
-  **Information Schema INNODB_SYS_VIRTUAL Table**
Information about base columns of virtual columns.
-  **Information Schema INNODB_TABLESPACES_ENCRYPTION Table**
Encryption metadata for InnoDB tablespaces.
-  **Information Schema INNODB_TABLESPACES_SCRUBBING Table**
Data scrubbing information.
-  **Information Schema INNODB_TRX Table**
Currently-executing InnoDB locks.
-  **Information Schema TEMP_TABLES_INFO Table**
Information about active InnoDB temporary tables.

H3Dr1.2.9.1.1.1.1 Information Schema INNODB_BUFFER_PAGE Table

The [Information Schema](#) `INNODB_BUFFER_PAGE` table contains information about pages in the buffer pool [🔗](#)

The `PROCESS` privilege is required to view the table.

It has the following columns:

Column	Description
<code>POOL_ID</code>	Buffer Pool identifier. From MariaDB 10.5.1 🔗 returns a value of 0, since multiple InnoDB buffer pool instances has been removed.
<code>BLOCK_ID</code>	Buffer Pool Block identifier.
<code>SPACE</code>	Tablespace identifier. Matches the <code>SPACE</code> value in the <code>INNODB_SYS_TABLES</code> table.
<code>PAGE_NUMBER</code>	Buffer pool page number.

PAGE_TYPE	Page type; one of <code>allocated</code> (newly-allocated page), <code>index</code> (B-tree node), <code>undo_log</code> (undo log page), <code>inode</code> (index node), <code>ibuf_free_list</code> (insert buffer free list), <code>ibuf_bitmap</code> (insert buffer bitmap), <code>system</code> (system page), <code>trx_system</code> (transaction system data), <code>file_space_header</code> (file space header), <code>extent_descriptor</code> (extent descriptor page), <code>blob</code> (uncompressed blob page), <code>compressed_blob</code> (first compressed blob page), <code>compressed_blob2</code> (subsequent compressed blob page) or <code>unknown</code> .
FLUSH_TYPE	Flush type.
FIX_COUNT	Count of the threads using this block in the buffer pool. When it is zero, the block can be evicted from the buffer pool.
IS_HASHED	Whether or not a hash index has been built on this page.
NEWEST_MODIFICATION	Most recent modification's Log Sequence Number.
OLDEST_MODIFICATION	Oldest modification's Log Sequence Number.
ACCESS_TIME	Abstract number representing the time the page was first accessed.
TABLE_NAME	Table that the page belongs to.
INDEX_NAME	Index that the page belongs to, either a clustered index or a secondary index.
NUMBER_RECORDS	Number of records the page contains.
DATA_SIZE	Size in bytes of all the records contained in the page.
COMPRESSED_SIZE	Compressed size in bytes of the page, or <code>NULL</code> for pages that aren't compressed.
PAGE_STATE	Page state; one of <code>FILE_PAGE</code> (page from a file) or <code>MEMORY</code> (page from an in-memory object) for valid data, or one of <code>NULL</code> , <code>READY_FOR_USE</code> , <code>NOT_USED</code> , <code>REMOVE_HASH</code> .
IO_FIX	Whether there is I/O pending for the page; one of <code>IO_NONE</code> (no pending I/O), <code>IO_READ</code> (read pending), <code>IO_WRITE</code> (write pending).
IS_OLD	Whether the page is old or not.
FREE_PAGE_CLOCK	Freed_page_clock counter, which tracks the number of blocks removed from the end of the least recently used (LRU) list, at the time the block was last placed at the head of the list.

The related `INFORMATION_SCHEMA.INNODB_BUFFER_PAGE_LRU` table contains the same information, but with an LRU (least recently used) position rather than block id.

Examples

```
DESC information_schema.innodb_buffer_page;
```

Field	Type	Null	Key	Default	Extra
POOL_ID	bigint(21) unsigned	NO		0	
BLOCK_ID	bigint(21) unsigned	NO		0	
SPACE	bigint(21) unsigned	NO		0	
PAGE_NUMBER	bigint(21) unsigned	NO		0	
PAGE_TYPE	varchar(64)	YES		NULL	
FLUSH_TYPE	bigint(21) unsigned	NO		0	
FIX_COUNT	bigint(21) unsigned	NO		0	
IS_HASHED	varchar(3)	YES		NULL	
NEWEST_MODIFICATION	bigint(21) unsigned	NO		0	
OLDEST_MODIFICATION	bigint(21) unsigned	NO		0	
ACCESS_TIME	bigint(21) unsigned	NO		0	
TABLE_NAME	varchar(1024)	YES		NULL	
INDEX_NAME	varchar(1024)	YES		NULL	
NUMBER_RECORDS	bigint(21) unsigned	NO		0	
DATA_SIZE	bigint(21) unsigned	NO		0	
COMPRESSED_SIZE	bigint(21) unsigned	NO		0	
PAGE_STATE	varchar(64)	YES		NULL	
IO_FIX	varchar(64)	YES		NULL	
IS_OLD	varchar(3)	YES		NULL	
FREE_PAGE_CLOCK	bigint(21) unsigned	NO		0	

```
SELECT * FROM INFORMATION_SCHEMA.INNODB_BUFFER_PAGE\G
...
***** 6. row *****
    POOL_ID: 0
    BLOCK_ID: 5
    SPACE: 0
    PAGE_NUMBER: 11
    PAGE_TYPE: INDEX
    FLUSH_TYPE: 1
    FIX_COUNT: 0
    IS_HASHED: NO
    NEWEST_MODIFICATION: 2046835
    OLDEST_MODIFICATION: 0
        ACCESS_TIME: 2585566280
        TABLE_NAME: `SYS_INDEXES`
        INDEX_NAME: CLUST_IND
    NUMBER_RECORDS: 57
    DATA_SIZE: 4016
    COMPRESSED_SIZE: 0
    PAGE_STATE: FILE_PAGE
    IO_FIX: IO_NONE
    IS_OLD: NO
    FREE_PAGE_CLOCK: 0
...

```

H3Dr1.2.9.1.1.1.2 Information Schema INNODB_BUFFER_PAGE_LRU Table

The [Information Schema](#) INNODB_BUFFER_PAGE_LRU table contains information about pages in the [buffer pool](#) and how they are ordered for eviction purposes.

The `PROCESS` privilege is required to view the table.

It has the following columns:

Column	Description
POOL_ID	Buffer Pool identifier. From MariaDB 10.5.1 returns a value of 0, since multiple InnoDB buffer pool instances has been removed.

LRU_POSITION	LRU (Least recently-used), for determining eviction order from the buffer pool.
SPACE	Tablespace identifier. Matches the <code>SPACE</code> value on the INNODB_SYS_TABLES table.
PAGE_NUMBER	Buffer pool page number.
PAGE_TYPE	Page type; one of <code>allocated</code> (newly-allocated page), <code>index</code> (B-tree node), <code>undo_log</code> (undo log page), <code>inode</code> (index node), <code>ibuf_free_list</code> (insert buffer free list), <code>ibuf_bitmap</code> (insert buffer bitmap), <code>system</code> (system page), <code>trx_system</code> (transaction system data), <code>file_space_header</code> (file space header), <code>extent_descriptor</code> (extent descriptor page), <code>blob</code> (uncompressed blob page), <code>compressed_blob</code> (first compressed blob page), <code>compressed_blob2</code> (subsequent compressed blob page) or <code>unknown</code> .
FLUSH_TYPE	Flush type.
FIX_COUNT	Count of the threads using this block in the buffer pool. When it is zero, the block can be evicted from the buffer pool.
IS_HASHED	Whether or not a hash index has been built on this page.
NEWEST_MODIFICATION	Most recent modification's Log Sequence Number.
OLDEST_MODIFICATION	Oldest modification's Log Sequence Number.
ACCESS_TIME	Abstract number representing the time the page was first accessed.
TABLE_NAME	Table that the page belongs to.
INDEX_NAME	Index that the page belongs to, either a clustered index or a secondary index.
NUMBER_RECORDS	Number of records the page contains.
DATA_SIZE	Size in bytes of all the records contained in the page.
COMPRESSED_SIZE	Compressed size in bytes of the page, or <code>NULL</code> for pages that aren't compressed.
PAGE_STATE	Page state; one of <code>FILE_PAGE</code> (page from a file) or <code>MEMORY</code> (page from an in-memory object) for valid data, or one of <code>NULL</code> , <code>READY_FOR_USE</code> , <code>NOT_USED</code> , <code>REMOVE_HASH</code> .
IO_FIX	Whether there is I/O pending for the page; one of <code>IO_NONE</code> (no pending I/O), <code>IO_READ</code> (read pending), <code>IO_WRITE</code> (write pending).
IS_OLD	Whether the page is old or not.
FREE_PAGE_CLOCK	Freed_page_clock counter, which tracks the number of blocks removed from the end of the LRU list, at the time the block was last placed at the head of the list.

The related [INFORMATION_SCHEMA.INNODB_BUFFER_PAGE](#) table contains the same information, but with a block id rather than LRU position.

Example

Field	Type	Null	Key	Default	Extra
POOL_ID	bigint(21) unsigned	NO		0	
LRU_POSITION	bigint(21) unsigned	NO		0	
SPACE	bigint(21) unsigned	NO		0	
PAGE_NUMBER	bigint(21) unsigned	NO		0	
PAGE_TYPE	varchar(64)	YES		NULL	
FLUSH_TYPE	bigint(21) unsigned	NO		0	
FIX_COUNT	bigint(21) unsigned	NO		0	
IS_HASHED	varchar(3)	YES		NULL	
NEWEST_MODIFICATION	bigint(21) unsigned	NO		0	
OLDEST_MODIFICATION	bigint(21) unsigned	NO		0	
ACCESS_TIME	bigint(21) unsigned	NO		0	
TABLE_NAME	varchar(1024)	YES		NULL	
INDEX_NAME	varchar(1024)	YES		NULL	
NUMBER_RECORDS	bigint(21) unsigned	NO		0	
DATA_SIZE	bigint(21) unsigned	NO		0	
COMPRESSED_SIZE	bigint(21) unsigned	NO		0	
COMPRESSED	varchar(3)	YES		NULL	
IO_FIX	varchar(64)	YES		NULL	
IS_OLD	varchar(3)	YES		NULL	
FREE_PAGE_CLOCK	bigint(21) unsigned	NO		0	

```

SELECT * FROM INFORMATION_SCHEMA.INNODB_BUFFER_PAGE_LRU\G
...
*****
6. row *****
    POOL_ID: 0
    LRU_POSITION: 5
    SPACE: 0
    PAGE_NUMBER: 11
    PAGE_TYPE: INDEX
    FLUSH_TYPE: 1
    FIX_COUNT: 0
    IS_HASHED: NO
    NEWEST_MODIFICATION: 2046835
    OLDEST_MODIFICATION: 0
        ACCESS_TIME: 2585566280
        TABLE_NAME: `SYS_INDEXES`
        INDEX_NAME: CLUST_IND
    NUMBER_RECORDS: 57
    DATA_SIZE: 4016
    COMPRESSED_SIZE: 0
    COMPRESSED: NO
        IO_FIX: IO_NONE
        IS_OLD: NO
    FREE_PAGE_CLOCK: 0
...

```

H3Dr1.2.9.1.1.1.3 Information Schema INNODB_BUFFER_POOL_PAGES Table

The [Information Schema](#) `INNODB_BUFFER_POOL_PAGES` table is a Percona enhancement, and is only available for XtraDB, not InnoDB (see [XtraDB and InnoDB](#)). It contains a record for each page in the [buffer pool](#).

It has the following columns:

Column	Description
PAGE_TYPE	Type of page; one of <code>index</code> , <code>undo_log</code> , <code>inode</code> , <code>ibuf_free_list</code> , <code>allocated</code> , <code>bitmap</code> , <code>sys</code> , <code>trx_sys</code> , <code>fsp_hdr</code> , <code>xdes</code> , <code>blob</code> , <code>zblob</code> , <code>zblob2</code> and <code>unknown</code> .
SPACE_ID	Tablespace ID.

PAGE_NO	Page offset within tablespace.
LRU_POSITION	Page position in the LRU (least-recently-used) list.
FIX_COUNT	Page reference count, incremented each time the page is accessed. <code>0</code> if the page is not currently being accessed.
FLUSH_TYPE	Flush type of the most recent flush. <code>0</code> (LRU), <code>2</code> (flush_list)

H3Dr1.2.9.1.1.4 Information Schema INNODB_BUFFER_POOL_PAGES_BLOB Table

The [Information Schema](#) `INNODB_BUFFER_POOL_PAGES_BLOB` table is a Percona enhancement, and is only available for XtraDB, not InnoDB (see [XtraDB and InnoDB](#)). It contains information about [buffer pool](#) blob pages.

It has the following columns:

Column	Description
SPACE_ID	Tablespace ID.
PAGE_NO	Page offset within tablespace.
COMPRESSED	<code>1</code> if the blob contains compressed data, <code>0</code> if not.
PART_LEN	Page data length.
NEXT_PAGE_NO	Next page number.
LRU_POSITION	Page position in the LRU (least-recently-used) list.
FIX_COUNT	Page reference count, incremented each time the page is accessed. <code>0</code> if the page is not currently being accessed.
FLUSH_TYPE	Flush type of the most recent flush. <code>0</code> (LRU), <code>2</code> (flush_list)

H3Dr1.2.9.1.1.5 Information Schema INNODB_BUFFER_POOL_PAGES_INDEX Table

The [Information Schema](#) `INNODB_BUFFER_POOL_PAGES` table is a Percona enhancement, and is only available for XtraDB, not InnoDB (see [XtraDB and InnoDB](#)). It contains information about [buffer pool](#) index pages.

It has the following columns:

Column	Description
INDEX_ID	Index name
SPACE_ID	Tablespace ID
PAGE_NO	Page offset within tablespace.
N_RECS	Number of user records on the page.
DATA_SIZE	Total data size in bytes of records in the page.
HASHED	<code>1</code> if the block is in the adaptive hash index, <code>0</code> if not.
ACCESS_TIME	Page's last access time.
MODIFIED	<code>1</code> if the page has been modified since being loaded, <code>0</code> if not.

DIRTY	1 if the page has been modified since it was last flushed, 0 if not
OLD	1 if the page is in the <i>old</i> blocks of the LRU (least-recently-used) list, 0 if not.
LRU_POSITION	Position in the LRU (least-recently-used) list.
FIX_COUNT	Page reference count, incremented each time the page is accessed. 0 if the page is not currently being accessed.
FLUSH_TYPE	Flush type of the most recent flush. 0 (LRU), 2 (flush_list)

H3Dr1.2.9.1.1.1.6 Information Schema INNODB_BUFFER_POOL_STATS Table

The [Information Schema](#) `INNODB_BUFFER_POOL_STATS` table contains information about pages in the [buffer pool](#), similar to what is returned with the `SHOW ENGINE INNODB STATUS` statement.

The `PROCESS` privilege is required to view the table.

It has the following columns:

Column	Description
POOL_ID	Buffer Pool identifier. From MariaDB 10.5.1 returns a value of 0, since multiple InnoDB buffer pool instances have been removed.
POOL_SIZE	Size in pages of the buffer pool.
FREE_BUFFERS	Number of free pages in the buffer pool.
DATABASE_PAGES	Total number of pages in the buffer pool.
OLD_DATABASE_PAGES	Number of pages in the <i>old</i> sublist.
MODIFIED_DATABASE_PAGES	Number of dirty pages.
PENDING_DECOMPRESS	Number of pages pending decompression.
PENDING_READS	Pending buffer pool level reads.
PENDING_FLUSH_LRU	Number of pages in the LRU pending flush.
PENDING_FLUSH_LIST	Number of pages in the flush list pending flush.
PAGES_MADE_YOUNG	Pages moved from the <i>old</i> sublist to the <i>new</i> sublist.
PAGES_NOT_MADE_YOUNG	Pages that have remained in the <i>old</i> sublist without moving to the <i>new</i> sublist.
PAGES_MADE_YOUNG_RATE	Hits that cause blocks to move to the top of the <i>new</i> sublist.
PAGES_MADE_NOT_YOUNG_RATE	Hits that do not cause blocks to move to the top of the <i>new</i> sublist due to the innodb_old_blocks delay not being met.
NUMBER_PAGES_READ	Number of pages read.
NUMBER_PAGES_CREATED	Number of pages created.
NUMBER_PAGES_WRITTEN	Number of pages written.
PAGES_READ_RATE	Number of pages read since the last printout divided by the time elapsed, giving pages read per second.
PAGES_CREATE_RATE	Number of pages created since the last printout divided by the time elapsed, giving pages created per second.
PAGES_WRITTEN_RATE	Number of pages written since the last printout divided by the time elapsed, giving pages written per second.
NUMBER_PAGES_GET	Number of logical read requests.

HIT_RATE	Buffer pool hit rate.
YOUNG_MAKE_PER_THOUSAND_GETS	For every 1000 gets, the number of pages made young.
NOT_YOUNG_MAKE_PER_THOUSAND_GETS	For every 1000 gets, the number of pages not made young.
NUMBER_PAGES_READ_AHEAD	Number of pages read ahead.
NUMBER_READ_AHEAD_EVICTED	Number of pages read ahead by the read-ahead thread that were later evicted without being accessed by any queries.
READ_AHEAD_RATE	Pages read ahead since the last printout divided by the time elapsed, giving read-ahead rate per second.
READ_AHEAD_EVICTED_RATE	Read-ahead pages not accessed since the last printout divided by time elapsed, giving the number of read-ahead pages evicted without access per second.
LRU_IO_TOTAL	Total least-recently used I/O.
LRU_IO_CURRENT	Least-recently used I/O for the current interval.
UNCOMPRESS_TOTAL	Total number of pages decompressed.
UNCOMPRESS_CURRENT	Number of pages decompressed in the current interval

Examples

```
DESC information_schema.innodb_buffer_pool_stats;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| POOL_ID | bigint(21) unsigned | NO | | 0 | |
| POOL_SIZE | bigint(21) unsigned | NO | | 0 | |
| FREE_BUFFERS | bigint(21) unsigned | NO | | 0 | |
| DATABASE_PAGES | bigint(21) unsigned | NO | | 0 | |
| OLD_DATABASE_PAGES | bigint(21) unsigned | NO | | 0 | |
| MODIFIED_DATABASE_PAGES | bigint(21) unsigned | NO | | 0 | |
| PENDING_DECOMPRESS | bigint(21) unsigned | NO | | 0 | |
| PENDING_READS | bigint(21) unsigned | NO | | 0 | |
| PENDING_FLUSH_LRU | bigint(21) unsigned | NO | | 0 | |
| PENDING_FLUSH_LIST | bigint(21) unsigned | NO | | 0 | |
| PAGES_MADE_YOUNG | bigint(21) unsigned | NO | | 0 | |
| PAGES_NOT_MADE_YOUNG | bigint(21) unsigned | NO | | 0 | |
| PAGES_MADE_YOUNG_RATE | double | NO | | 0 | |
| PAGES_MADE_NOT_YOUNG_RATE | double | NO | | 0 | |
| NUMBER_PAGES_READ | bigint(21) unsigned | NO | | 0 | |
| NUMBER_PAGES_CREATED | bigint(21) unsigned | NO | | 0 | |
| NUMBER_PAGES_WRITTEN | bigint(21) unsigned | NO | | 0 | |
| PAGES_READ_RATE | double | NO | | 0 | |
| PAGES_CREATE_RATE | double | NO | | 0 | |
| PAGES_WRITTEN_RATE | double | NO | | 0 | |
| NUMBER_PAGES_GET | bigint(21) unsigned | NO | | 0 | |
| HIT_RATE | bigint(21) unsigned | NO | | 0 | |
| YOUNG_MAKE_PER_THOUSAND_GETS | bigint(21) unsigned | NO | | 0 | |
| NOT_YOUNG_MAKE_PER_THOUSAND_GETS | bigint(21) unsigned | NO | | 0 | |
| NUMBER_PAGES_READ_AHEAD | bigint(21) unsigned | NO | | 0 | |
| NUMBER_READ_AHEAD_EVICTED | bigint(21) unsigned | NO | | 0 | |
| READ_AHEAD_RATE | double | NO | | 0 | |
| READ_AHEAD_EVICTED_RATE | double | NO | | 0 | |
| LRU_IO_TOTAL | bigint(21) unsigned | NO | | 0 | |
| LRU_IO_CURRENT | bigint(21) unsigned | NO | | 0 | |
| UNCOMPRESS_TOTAL | bigint(21) unsigned | NO | | 0 | |
| UNCOMPRESS_CURRENT | bigint(21) unsigned | NO | | 0 | |
+-----+-----+-----+-----+-----+
```

H3Dr1.2.9.1.1.1.7 Information Schema

INNODB_CHANGED_PAGES Table

The [Information Schema](#) `INNODB_CHANGED_PAGES` Table contains data about modified pages from the bitmap file. It is updated at checkpoints by the log tracking thread parsing the log, so does not contain real-time data.

The number of records is limited by the value of the `innodb_max_changed_pages` system variable.

The `PROCESS` privilege is required to view the table.

It has the following columns:

Column	Description
SPACE_ID	Modified page space id
PAGE_ID	Modified page id
START_LSN	Interval start after which page was changed (equal to checkpoint LSN)
END_LSN	Interval end before which page was changed (equal to checkpoint LSN)

H3Dr1.2.9.1.1.8 Information Schema INNODB_CMP and INNODB_CMP_RESET Tables

The `INNODB_CMP` and `INNODB_CMP_RESET` tables contain status information on compression operations related to [compressed XtraDB/InnoDB tables](#).

The `PROCESS` privilege is required to query this table.

These tables contain the following columns:

Column Name	Description
<code>PAGE_SIZE</code>	Compressed page size, in bytes. This value is unique in the table; other values are totals which refer to pages of this size.
<code>COMPRESS_OPS</code>	How many times a page of the size <code>PAGE_SIZE</code> has been compressed. This happens when a new page is created because the compression log runs out of space. This value includes both successful operations and <i>compression failures</i> .
<code>COMPRESS_OPS_OK</code>	How many times a page of the size <code>PAGE_SIZE</code> has been successfully compressed. This value should be as close as possible to <code>COMPRESS_OPS</code> . If it is notably lower, either avoid compressing some tables, or increase the <code>KEY_BLOCK_SIZE</code> for some compressed tables.
<code>COMPRESS_TIME</code>	Time (in seconds) spent to compress pages of the size <code>PAGE_SIZE</code> . This value includes time spent in <i>compression failures</i> .
<code>UNCOMPRESS_OPS</code>	How many times a page of the size <code>PAGE_SIZE</code> has been uncompressed. This happens when an uncompressed version of a page is created in the buffer pool, or when a <i>compression failure</i> occurs.
<code>UNCOMPRESS_TIME</code>	Time (in seconds) spent to uncompress pages of the size <code>PAGE_SIZE</code> .

These tables can be used to measure the effectiveness of XtraDB/InnoDB table compression. When you have to decide a value for `KEY_BLOCK_SIZE`, you can create more than one version of the table (one for each candidate value) and run a realistic workload on them. Then, these tables can be used to see how the operations performed with different page sizes.

`INNODB_CMP` and `INNODB_CMP_RESET` have the same columns and always contain the same values, but when `INNODB_CMP_RESET` is queried, both the tables are cleared. `INNODB_CMP_RESET` can be used, for example, if a script periodically logs the performances of compression in the last period of time.

`INNODB_CMP` can be used to see the cumulated statistics.

Examples

```
SELECT * FROM information_schema.INNODB_CMP\G
*****
page_size: 1024
compress_ops: 0
compress_ops_ok: 0
compress_time: 0
uncompress_ops: 0
uncompress_time: 0
...
```

See Also

Other tables that can be used to monitor XtraDB/InnoDB compressed tables:

- [INNODB_CMP_PER_INDEX](#) and [INNODB_CMP_PER_INDEX_RESET](#)
- [INNODB_CMPPMEM](#) and [INNODB_CMPPMEM_RESET](#)

H3Dr1.2.9.1.1.1.9 Information Schema INNODB_CMPPMEM and INNODB_CMPPMEM_RESET Tables

The `INNODB_CMPPMEM` and `INNODB_CMPPMEM_RESET` tables contain status information on compressed pages in the [buffer pool](#) (see InnoDB [COMPRESSED](#) format).

The `PROCESS` privilege is required to query this table.

These tables contain the following columns:

Column Name	Description
PAGE_SIZE	Compressed page size, in bytes. This value is unique in the table; other values are totals which refer to pages of this size.
BUFFER_POOL_INSTANCE	Buffer Pool identifier. From MariaDB 10.5.1 returns a value of 0, since multiple InnoDB buffer pool instances has been removed.
PAGES_USED	Number of pages of the size <code>PAGE_SIZE</code> which are currently in the buffer pool.
PAGES_FREE	Number of pages of the size <code>PAGE_SIZE</code> which are currently free, and thus are available for allocation. This value represents the buffer pool's fragmentation. A totally unfragmented buffer pool has at most 1 free page.
RELOCATION_OPS	How many times a page of the size <code>PAGE_SIZE</code> has been relocated. This happens when data exceeds a page (because a row must be copied into a new page) and when two pages are merged (because their data shrunk and can now be contained in one page).
RELOCATION_TIME	Time (in seconds) spent in relocation operations for pages of the size <code>PAGE_SIZE</code> . This column is reset when the <code>INNODB_CMPPMEM_RESET</code> table is queried.

These tables can be used to measure the effectiveness of InnoDB table compression. When you have to decide a value for `KEY_BLOCK_SIZE`, you can create more than one version of the table (one for each candidate value) and run a realistic workload on them. Then, these tables can be used to see how the operations performed with different page sizes.

`INNODB_CMPPMEM` and `INNODB_CMPPMEM_RESET` have the same columns and always contain the same values, but when `INNODB_CMPPMEM_RESET` is queried, the `RELOCATION_TIME` column from both the tables are cleared. `INNODB_CMPPMEM_RESET` can be used, for example, if a script periodically logs the performances of compression in the last period of time. `INNODB_CMPPMEM` can be used to see the cumulated statistics.

Example

```
SELECT * FROM information_schema.INNODB_CMPMEM\G
*****
page_size: 1024
buffer_pool_instance: 0
pages_used: 0
pages_free: 0
relocation_ops: 0
relocation_time: 0
```

See Also

Other tables that can be used to monitor InnoDB compressed tables:

- [INNODB_CMP](#) and [INNODB_CMP_RESET](#)
- [INNODB_CMP_PER_INDEX](#) and [INNODB_CMP_PER_INDEX_RESET](#)

H3Dr1.2.9.1.1.10 Information Schema INNODB_CMP_PER_INDEX and INNODB_CMP_PER_INDEX_RESET Tables

The `INNODB_CMP_PER_INDEX` and `INNODB_CMP_PER_INDEX_RESET` tables contain status information on compression operations related to [compressed XtraDB/InnoDB tables](#), grouped by individual indexes. These tables are only populated if the [innodb_cmp_per_index_enabled](#) system variable is set to `ON`.

The `PROCESS` privilege is required to query this table.

These tables contains the following columns:

Column Name	Description
<code>DATABASE_NAME</code>	Database containing the index.
<code>TABLE_NAME</code>	Table containing the index.
<code>INDEX_NAME</code>	Other values are totals which refer to this index's compression.
<code>COMPRESS_OPS</code>	How many times a page of <code>INDEX_NAME</code> has been compressed. This happens when a new page is created because the compression log runs out of space. This value includes both successful operations and <i>compression failures</i> .
<code>COMPRESS_OPS_OK</code>	How many times a page of <code>INDEX_NAME</code> has been successfully compressed. This value should be as close as possible to <code>COMPRESS_OPS</code> . If it is notably lower, either avoid compressing some tables, or increase the <code>KEY_BLOCK_SIZE</code> for some compressed tables.
<code>COMPRESS_TIME</code>	Time (in seconds) spent to compress pages of the size <code>PAGE_SIZE</code> . This value includes time spent in <i>compression failures</i> .
<code>UNCOMPRESS_OPS</code>	How many times a page of <code>INDEX_NAME</code> has been uncompressed. This happens when an uncompressed version of a page is created in the buffer pool, or when a <i>compression failure</i> occurs.
<code>UNCOMPRESS_TIME</code>	Time (in seconds) spent to uncompress pages of <code>INDEX_NAME</code> .

These tables can be used to measure the effectiveness of XtraDB/InnoDB compression, per table or per index. The values in these tables show which tables perform better with index compression, and which tables cause too many *compression failures* or perform too many compression/uncompression operations. When compression performs badly for a table, this might mean that you should change its `KEY_BLOCK_SIZE`, or that the table should not be compressed.

`INNODB_CMP_PER_INDEX` and `INNODB_CMP_PER_INDEX_RESET` have the same columns and always contain

the same values, but when `INNODB_CMP_PER_INDEX_RESET` is queried, both the tables are cleared. `INNODB_CMP_PER_INDEX_RESET` can be used, for example, if a script periodically logs the performances of compression in the last period of time. `INNODB_CMP_PER_INDEX` can be used to see the cumulated statistics.

See Also

Other tables that can be used to monitor XtraDB/InnoDB compressed tables:

- [INNODB_CMP](#) and [INNODB_CMP_RESET](#)
- [INNODB_CMPMEM](#) and [INNODB_CMPMEM_RESET](#)

H3Dr1.2.9.1.1.11 Information Schema INNODB_FT_BEING_DELETED Table

The [Information Schema](#) `INNODB_FT_BEING_DELETED` table is only used while document ID's in the related `INNODB_FT_DELETED` are being removed from an InnoDB [fulltext index](#) while an [OPTIMIZE TABLE](#) is underway. At all other times the table will be empty.

The `SUPER` privilege is required to view the table, and it also requires the [innodb_ft_aux_table](#) system variable to be set.

It has the following column:

Column	Description
<code>DOC_ID</code>	Document ID of the row being deleted. Either an underlying ID value, or a sequence value generated by InnoDB if no usable option exists.

H3Dr1.2.9.1.1.12 Information Schema INNODB_FT_CONFIG Table

The [Information Schema](#) `INNODB_FT_CONFIG` table contains InnoDB [fulltext index](#) metadata.

The `SUPER` privilege is required to view the table, and it also requires the [innodb_ft_aux_table](#) system variable to be set.

It has the following columns:

Column	Description
<code>KEY</code>	Metadata item name.
<code>VALUE</code>	Associated value.

Example

```
SELECT * FROM INNODB_FT_CONFIG;
+-----+-----+
| KEY           | VALUE |
+-----+-----+
| optimize_checkpoint_limit | 180   |
| synced_doc_id    | 6     |
| last_optimized_word      |
| deleted_doc_count      | 0     |
| total_word_count        |
| optimize_start_time      |
| optimize_end_time        |
| stopword_table_name      |
| use_stopword          | 1     |
| table_state            | 0     |
+-----+-----+
```

H3Dr1.2.9.1.1.13 Information Schema INNODB_FT_DEFAULT_STOPWORD Table

The [Information Schema](#) `INNODB_FT_DEFAULT_STOPWORD` table contains a list of default [stopwords](#) used when creating an InnoDB [fulltext index](#).

The `PROCESS` [privilege](#) is required to view the table.

It has the following column:

Column	Description
VALUE	Default stopword for an InnoDB fulltext index . Setting either the <code>innodb_ft_server_stopword_table</code> or the <code>innodb_ft_user_stopword_table</code> system variable will override this.

Example

```
SELECT * FROM information_schema.INNODB_FT_DEFAULT_STOPWORD\G
*****
1. row *****
value: a
*****
2. row *****
value: about
*****
3. row *****
value: an
*****
4. row *****
value: are
...
*****
36. row *****
value: www
```

H3Dr1.2.9.1.1.14 Information Schema INNODB_FT_DELETED Table

The [Information Schema](#) `INNODB_FT_DELETED` table contains rows that have been deleted from an InnoDB [fulltext index](#). This information is then used to filter results on subsequent searches, removing the need to expensively reorganise the index each time a row is deleted.

The fulltext index is then only reorganized when an `OPTIMIZE TABLE` statement is underway. The related `INNODB_FT_BEING_DELETED` table contains rows being deleted while an `OPTIMIZE TABLE` is in the process of running.

The `SUPER` [privilege](#) is required to view the table, and it also requires the `innodb_ft_aux_table` system variable to be set.

It has the following column:

Column	Description
DOC_ID	Document ID of the deleted row deleted. Either an underlying ID value, or a sequence value generated by InnoDB if no usable option exists.

Example

```

SELECT * FROM INFORMATION_SCHEMA.INNODB_FT_DELETED;
+-----+
| DOC_ID |
+-----+
|      2 |
+-----+

DELETE FROM test.ft_innodb LIMIT 1;

SELECT * FROM INFORMATION_SCHEMA.INNODB_FT_DELETED;
+-----+
| DOC_ID |
+-----+
|      2 |
|      3 |
+-----+

```

H3Dr1.2.9.1.1.15 Information Schema INNODB_FT_INDEX_CACHE Table

The [Information Schema](#) `INNODB_FT_INDEX_CACHE` table contains information about rows that have recently been inserted into an InnoDB [fulltext index](#). To avoid re-organizing the fulltext index each time a change is made, which would be very expensive, new changes are stored separately and only integrated when an [OPTIMIZE TABLE](#) is run.

The `SUPER` privilege is required to view the table, and it also requires the [innodb_ft_aux_table](#) system variable to be set.

It has the following columns:

Column	Description
<code>WORD</code>	Word from the text of a newly added row. Words can appear multiple times in the table, once per <code>DOC_ID</code> and <code>POSITION</code> combination.
<code>FIRST_DOC_ID</code>	First document ID where this word appears in the index.
<code>LAST_DOC_ID</code>	Last document ID where this word appears in the index.
<code>DOC_COUNT</code>	Number of rows containing this word in the index.
<code>DOC_ID</code>	Document ID of the newly added row, either an appropriate ID column or an internal InnoDB value.
<code>POSITION</code>	Position of this word instance within the <code>DOC_ID</code> , as an offset added to the previous <code>POSITION</code> instance.

Note that for `OPTIMIZE TABLE` to process InnoDB fulltext index data, the [innodb_optimize_fulltext_only](#) system variable needs to be set to `1`. When this is done, and an `OPTIMIZE TABLE` statement run, the `INNODB_FT_INDEX_CACHE` table will be emptied, and the `INNODB_FT_INDEX_TABLE` table will be updated.

Examples

```

SELECT * FROM INNODB_FT_INDEX_CACHE;
+-----+-----+-----+-----+-----+-----+
| WORD      | FIRST_DOC_ID | LAST_DOC_ID | DOC_COUNT | DOC_ID | POSITION |
+-----+-----+-----+-----+-----+-----+
| and       |        4 |        4 |        1 |    4 |     0 |
| arrived   |        4 |        4 |        1 |    4 |    20 |
| ate       |        1 |        1 |        1 |    1 |     4 |
| everybody |        1 |        1 |        1 |    1 |     8 |
| goldilocks |        4 |        4 |        1 |    4 |     9 |
| hungry    |        3 |        3 |        1 |    3 |     8 |
| then      |        4 |        4 |        1 |    4 |     4 |
| wicked   |        2 |        2 |        1 |    2 |     4 |
| witch    |        2 |        2 |        1 |    2 |    11 |
+-----+-----+-----+-----+-----+-----+
9 rows in set (0.00 sec)

```

```
INSERT INTO test.ft_innodb VALUES(3,'And she ate a pear');
```

```

SELECT * FROM INNODB_FT_INDEX_CACHE;
+-----+-----+-----+-----+-----+-----+
| WORD      | FIRST_DOC_ID | LAST_DOC_ID | DOC_COUNT | DOC_ID | POSITION |
+-----+-----+-----+-----+-----+-----+
| and       |        4 |        5 |        2 |    4 |     0 |
| and       |        4 |        5 |        2 |    5 |     0 |
| arrived   |        4 |        4 |        1 |    4 |    20 |
| ate       |        1 |        5 |        2 |    1 |     4 |
| ate       |        1 |        5 |        2 |    5 |     8 |
| everybody |        1 |        1 |        1 |    1 |     8 |
| goldilocks |        4 |        4 |        1 |    4 |     9 |
| hungry    |        3 |        3 |        1 |    3 |     8 |
| pear      |        5 |        5 |        1 |    5 |    14 |
| she       |        5 |        5 |        1 |    5 |     4 |
| then      |        4 |        4 |        1 |    4 |     4 |
| wicked   |        2 |        2 |        1 |    2 |     4 |
| witch    |        2 |        2 |        1 |    2 |    11 |
+-----+-----+-----+-----+-----+-----+

```

```

OPTIMIZE TABLE test.ft_innodb\G
***** 1. row *****
Table: test.ft_innodb
Op: optimize
Msg_type: note
Msg_text: Table does not support optimize, doing recreate + analyze instead
***** 2. row *****
Table: test.ft_innodb
Op: optimize
Msg_type: status
Msg_text: OK
2 rows in set (2.24 sec)

```

```

SELECT * FROM INNODB_FT_INDEX_CACHE;
+-----+-----+-----+-----+-----+-----+
| WORD      | FIRST_DOC_ID | LAST_DOC_ID | DOC_COUNT | DOC_ID | POSITION |
+-----+-----+-----+-----+-----+-----+
| and       |        4 |        5 |        2 |    4 |     0 |
| and       |        4 |        5 |        2 |    5 |     0 |
| arrived   |        4 |        4 |        1 |    4 |    20 |
| ate       |        1 |        5 |        2 |    1 |     4 |
| ate       |        1 |        5 |        2 |    5 |     8 |
| everybody |        1 |        1 |        1 |    1 |     8 |
| goldilocks |        4 |        4 |        1 |    4 |     9 |
| hungry    |        3 |        3 |        1 |    3 |     8 |
| pear      |        5 |        5 |        1 |    5 |    14 |
| she       |        5 |        5 |        1 |    5 |     4 |
| then      |        4 |        4 |        1 |    4 |     4 |
| wicked   |        2 |        2 |        1 |    2 |     4 |
| witch    |        2 |        2 |        1 |    2 |    11 |
+-----+-----+-----+-----+-----+-----+
13 rows in set (0.00 sec)

```

The `OPTIMIZE TABLE` statement has no effect, because the `innodb_optimize_fulltext_only` variable wasn't set:

```
SHOW VARIABLES LIKE 'innodb_optimize_fulltext_only';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| innodb_optimize_fulltext_only | OFF   |
+-----+-----+

SET GLOBAL innodb_optimize_fulltext_only =1;

OPTIMIZE TABLE test.ft_innodb;
+-----+-----+-----+-----+
| Table      | Op       | Msg_type | Msg_text |
+-----+-----+-----+-----+
| test.ft_innodb | optimize | status    | OK      |
+-----+-----+-----+-----+

SELECT * FROM INNODB_FT_INDEX_CACHE;
Empty set (0.00 sec)
```

H3Dr1.2.9.1.1.1.16 Information Schema INNODB_FT_INDEX_TABLE Table

The [Information Schema](#) `INNODB_FT_INDEX_TABLE` table contains information about InnoDB fulltext indexes. To avoid re-organizing the fulltext index each time a change is made, which would be very expensive, new changes are stored separately and only integrated when an `OPTIMIZE TABLE` is run. See the [INNODB_FT_INDEX_CACHE](#) table.

The `SUPER` privilege is required to view the table, and it also requires the [innodb_ft_aux_table](#) system variable to be set.

It has the following columns:

Column	Description
WORD	Word from the text of a column with a fulltext index. Words can appear multiple times in the table, once per <code>DOC_ID</code> and <code>POSITION</code> combination.
FIRST_DOC_ID	First document ID where this word appears in the index.
LAST_DOC_ID	Last document ID where this word appears in the index.
DOC_COUNT	Number of rows containing this word in the index.
DOC_ID	Document ID of the newly added row, either an appropriate ID column or an internal InnoDB value.
POSITION	Position of this word instance within the <code>DOC_ID</code> , as an offset added to the previous <code>POSITION</code> instance.

Note that for `OPTIMIZE TABLE` to process InnoDB fulltext index data, the `innodb_optimize_fulltext_only` system variable needs to be set to `1`. When this is done, and an `OPTIMIZE TABLE` statement run, the [INNODB_FT_INDEX_CACHE](#) table will be emptied, and the [INNODB_FT_INDEX_TABLE](#) table will be updated.

Examples

```

SELECT * FROM INNODB_FT_INDEX_TABLE;
Empty set (0.00 sec)

SET GLOBAL innodb_optimize_fulltext_only =1;

OPTIMIZE TABLE test.ft_innodb;
+-----+-----+-----+
| Table | Op    | Msg_type | Msg_text |
+-----+-----+-----+
| test.ft_innodb | optimize | status   | OK      |
+-----+-----+-----+

SELECT * FROM INNODB_FT_INDEX_TABLE;
+-----+-----+-----+-----+-----+-----+
| WORD | FIRST_DOC_ID | LAST_DOC_ID | DOC_COUNT | DOC_ID | POSITION |
+-----+-----+-----+-----+-----+-----+
| and | 4 | 5 | 2 | 4 | 0 |
| and | 4 | 5 | 2 | 5 | 0 |
| arrived | 4 | 4 | 1 | 4 | 20 |
| ate | 1 | 5 | 2 | 1 | 4 |
| ate | 1 | 5 | 2 | 5 | 8 |
| everybody | 1 | 1 | 1 | 1 | 8 |
| goldilocks | 4 | 4 | 1 | 4 | 9 |
| hungry | 3 | 3 | 1 | 3 | 8 |
| pear | 5 | 5 | 1 | 5 | 14 |
| she | 5 | 5 | 1 | 5 | 4 |
| then | 4 | 4 | 1 | 4 | 4 |
| wicked | 2 | 2 | 1 | 2 | 4 |
| witch | 2 | 2 | 1 | 2 | 11 |
+-----+-----+-----+-----+-----+

```

H3Dr1.2.9.1.1.1.17 Information Schema INNODB_LOCK_WAITS Table

The [Information Schema](#) `INNODB_LOCK_WAITS` table contains information about blocked InnoDB transactions. The `PROCESS` privilege is required to view the table.

It contains the following columns:

Column	Description
<code>REQUESTING_TRX_ID</code>	Requesting transaction ID from the INNODB_TRX table.
<code>REQUESTED_LOCK_ID</code>	Lock ID from the INNODBLOCKS table for the waiting transaction.
<code>BLOCKING_TRX_ID</code>	Blocking transaction ID from the INNODB_TRX table.
<code>BLOCKING_LOCK_ID</code>	Lock ID from the INNODBLOCKS table of a lock held by a transaction that is blocking another transaction.

The table is often used in conjunction with the `INNODBLOCKS` and `INNODB_TRX` tables to diagnose problematic locks and transactions.

H3Dr1.2.9.1.1.1.18 Information Schema INNODBLOCKS Table

The [Information Schema](#) `INNODBLOCKS` table stores information about locks that InnoDB transactions have requested but not yet acquired, or that are blocking another transaction.

It has the following columns:

Column	Description
<code>LOCK_ID</code>	Lock ID number - the format is not fixed, so do not rely upon the number for information.

LOCK_TRX_ID	Lock's transaction ID. Matches the INNODB_TRX.TRX_ID column.
LOCK_MODE	Lock mode 🔗 . One of s (shared), x (exclusive), IS (intention shared), IX (intention exclusive row lock), S_GAP (shared gap lock), X_GAP (exclusive gap lock), IS_GAP (intention shared gap lock), IX_GAP (intention exclusive gap lock) or AUTO_INC (auto-increment table level lock 🔗).
LOCK_TYPE	Whether the lock is RECORD (row level) or TABLE level.
LOCK_TABLE	Name of the locked table, or table containing locked rows.
LOCK_INDEX	Index name if a RECORD LOCK_TYPE , or NULL if not.
LOCK_SPACE	Tablespace ID if a RECORD LOCK_TYPE , or NULL if not.
LOCK_PAGE	Locked record page number if a RECORD LOCK_TYPE , or NULL if not.
LOCK_REC	Locked record heap number if a RECORD LOCK_TYPE , or NULL if not.
LOCK_DATA	Locked record primary key as an SQL string if a RECORD LOCK_TYPE , or NULL if not. If no primary key exists, the internal InnoDB row_id number is instead used. To avoid unnecessary IO, also NULL if the locked record page is not in the buffer pool 🔗

The table is often used in conjunction with the INNODB_LOCK_WAITS and INNODB_TRX tables to diagnose problematic locks and transactions

Example

```

-- session 1
START TRANSACTION;
UPDATE t SET id = 15 WHERE id = 10;

-- session 2
DELETE FROM t WHERE id = 10;

-- session 1
USE information_schema;
SELECT 1.* , t.*
  FROM information_schema.INNODB_LOCKS l
  JOIN information_schema.INNODB_TRX t
    ON l.lock trx_id = t.trx_id
   WHERE trx_state = 'LOCK WAIT' \G
***** 1. row *****
lock_id: 840:40:3:2
lock_trx_id: 840
lock_mode: X
lock_type: RECORD
lock_table: `test`.`t`
lock_index: PRIMARY
lock_space: 40
lock_page: 3
lock_rec: 2
lock_data: 10
      trx_id: 840
      trx_state: LOCK WAIT
      trx_started: 2019-12-23 18:43:46
trx_requested_lock_id: 840:40:3:2
      trx_wait_started: 2019-12-23 18:43:46
      trx_weight: 2
      trx_mysql_thread_id: 46
      trx_query: DELETE FROM t WHERE id = 10
      trx_operation_state: starting index read
      trx_tables_in_use: 1
      trx_tables_locked: 1
      trx_lock_structs: 2
      trx_lock_memory_bytes: 1136
      trx_rows_locked: 1
      trx_rows_modified: 0
      trx_concurrency_tickets: 0
      trx_isolation_level: REPEATABLE READ
      trx_unique_checks: 1
      trx_foreign_key_checks: 1
      trx_last_foreign_key_error: NULL
      trx_is_read_only: 0
      trx_autocommit_non_locking: 0

```

H3Dr1.2.9.1.1.1.19 Information Schema INNODB_METRICS Table

Contents

1. [Enabling and Disabling Counters](#)
2. [Resetting Counters](#)
3. [Simplifying from MariaDB 10.6](#)
4. [Examples](#)

The [Information Schema](#) INNODB_METRICS table contains a list of useful InnoDB performance metrics. Each row in the table represents an instrumented counter that can be stopped, started and reset, and which can be grouped together by module.

The [PROCESS privilege](#) is required to view the table.

It has the following columns:

Column	Description
NAME	Unique counter name.
SUBSYSTEM	InnoDB subsystem. See below for the matching module to use to enable/disable monitoring this subsystem with the innodb_monitor_enable and innodb_monitor_disable system variables.
COUNT	Count since being enabled.
MAX_COUNT	Maximum value since being enabled.
MIN_COUNT	Minimum value since being enabled.
AVG_COUNT	Average value since being enabled.
COUNT_RESET	Count since last being reset.
MAX_COUNT_RESET	Maximum value since last being reset.
MIN_COUNT_RESET	Minimum value since last being reset.
AVG_COUNT_RESET	Average value since last being reset.
TIME_ENABLED	Time last enabled.
TIME_DISABLED	Time last disabled
TIME_ELAPSED	Time since enabled
TIME_RESET	Time last reset.
STATUS	Whether the counter is currently enabled to disabled.
TYPE	Item type; one of <code>counter</code> , <code>value</code> , <code>status_counter</code> , <code>set_owner</code> , <code>set_member</code> .
COMMENT	Counter description.

Enabling and Disabling Counters

Most of the counters are disabled by default. To enable them, use the [innodb_monitor_enable](#) system variable. You can either enable a variable by its name, for example:

```
SET GLOBAL innodb_monitor_enable = icp_match;
```

or enable a number of counters grouped by module. The `SUBSYSTEM` field indicates which counters are grouped together, but the following module names need to be used:

Module Name	Subsystem Field
module_metadata	metadata
module_lock	lock
module_buffer	buffer
module_buf_page	buffer_page_io
module_os	os
module_trx	transaction
module_purge	purge
module_compress	compression
module_file	file_system
module_index	index

module_adaptive_hash	adaptive_hash_index From MariaDB 10.6.2 ↗ , if innodb_adaptive_hash_index ↗ is disabled (the default), adaptive_hash_index will not be updated.
module_ibuf_system	change_buffer
module_srv	server
module_ddl	ddl
module_dml	dml
module_log	recovery
module_icp	icp

There are four counters in the `icp` subsystem:

```
SELECT NAME, SUBSYSTEM FROM INNODB_METRICS WHERE SUBSYSTEM='icp';
+-----+-----+
| NAME      | SUBSYSTEM |
+-----+-----+
| icp_attempts    | icp      |
| icp_no_match    | icp      |
| icp_out_of_range | icp      |
| icp_match       | icp      |
+-----+-----+
```

To enable them all, use the associated module name from the table above, `module_icp`.

```
SET GLOBAL innodb_monitor_enable = module_icp;
```

The `%` wildcard, used to represent any number of characters, can also be used when naming counters, for example:

```
SET GLOBAL innodb_monitor_enable = 'buffer%'
```

To disable counters, use the [innodb_monitor_disable](#) system variable, using the same naming rules as described above for enabling.

Counter status is not persistent, and will be reset when the server restarts. It is possible to use the options on the command line, or the `innodb_monitor_enable` option only in a configuration file.

Resetting Counters

Counters can also be reset. Resetting sets all the `*_COUNT_RESET` values to zero, while leaving the `*_COUNT` values, which perform counts since the counter was enabled, untouched. Resetting is performed with the [innodb_monitor_reset](#) (for individual counters) and [innodb_monitor_reset_all](#) (for all counters) system variables.

Simplifying from MariaDB 10.6 [↗](#)

MariaDB starting with [10.6](#)

From [MariaDB 10.6](#), the interface was simplified by removing the following:

- `buffer_LRU_batches_flush`
- `buffer_LRU_batch_flush_pages`
- `buffer_LRU_batches_evict`
- `buffer_LRU_batch_evict_pages`

and by making the following reflect the status variables:

- `buffer_LRU_batch_flush_total_pages`: [innodb_buffer_pool_pages_LRU_flushed](#)
- `buffer_LRU_batch_evict_total_pages`: [innodb_buffer_pool_pages_LRU_freed](#)

Examples

MariaDB 10.8:

```
SELECT name,subsystem,type,comment FROM INFORMATION_SCHEMA.INNODB_METRICS\G
*****
  name: metadata_table_handles_opened
subsystem: metadata
  type: counter
  comment: Number of table handles opened
***** 1. row *****
  name: lock_deadlocks
subsystem: lock
  type: value
  comment: Number of deadlocks
***** 2. row *****
  name: lock_timeouts
subsystem: lock
  type: value
  comment: Number of lock timeouts
***** 3. row *****
  name: lock_rec_lock_waits
subsystem: lock
  type: counter
  comment: Number of times enqueued into record lock wait queue
***** 4. row *****
  name: lock_table_lock_waits
subsystem: lock
  type: counter
  comment: Number of times enqueued into table lock wait queue
***** 5. row *****
  name: lock_rec_lock_requests
subsystem: lock
  type: counter
  comment: Number of record locks requested
***** 6. row *****
  name: lock_rec_lock_created
subsystem: lock
  type: counter
  comment: Number of record locks created
***** 7. row *****
  name: lock_rec_lock_removed
subsystem: lock
  type: counter
  comment: Number of record locks removed from the lock queue
***** 8. row *****
  name: lock_rec_locks
subsystem: lock
  type: counter
  comment: Current number of record locks on tables
***** 10. row *****
  name: lock_table_lock_created
subsystem: lock
  type: counter
  comment: Number of table locks created

...
***** 207. row *****
  name: icp_attempts
subsystem: icp
  type: counter
  comment: Number of attempts for index push-down condition checks
***** 208. row *****
  name: icp_no_match
subsystem: icp
  type: counter
  comment: Index push-down condition does not match
```

```
***** 209. row ****
    name: icp_out_of_range
  subsystem: icp
      type: counter
  comment: Index push-down condition out of range
***** 210. row ****
    name: icp_match
  subsystem: icp
      type: counter
  comment: Index push-down condition matches
```

H3Dr1.2.9.1.1.1.20 Information Schema INNODB_MUTEXES Table

The `INNODB_MUTEXES` table monitors mutex and rw locks waits. It has the following columns:

Column	Description
NAME	Name of the lock, as it appears in the source code.
CREATE_FILE	File name of the mutex implementation.
CREATE_LINE	Line number of the mutex implementation.
OS_WAITS	How many times the mutex occurred.

The `CREATE_FILE` and `CREATE_LINE` columns depend on the InnoDB/XtraDB version.

Note that since MariaDB 10.2.2 [↗](#), the table has only been providing information about `rw_lock_t`, not any mutexes. From MariaDB 10.2.2 [↗](#) until MariaDB 10.2.32 [↗](#), MariaDB 10.3.23 [↗](#), MariaDB 10.4.13 [↗](#) and MariaDB 10.5.1 [↗](#), the `NAME` column was not populated ([MDEV-21636 ↗](#)).

The `SHOW ENGINE INNODB STATUS` statement provides similar information.

Examples

```
SELECT * FROM INNODB_MUTEXES;
+-----+-----+-----+-----+
| NAME           | CREATE_FILE        | CREATE_LINE | OS_WAITS |
+-----+-----+-----+-----+
| &dict_sys->mutex          | dict0dict.cc      | 989         | 2         |
| &buf_pool->flush_state_mutex | buf0buf.cc       | 1388        | 1         |
| &log_sys->checkpoint_lock | log0log.cc       | 1014        | 2         |
| &block->lock             | combined buf0buf.cc | 1120        | 1         |
+-----+-----+-----+-----+
```

H3Dr1.2.9.1.1.1.21 Information Schema INNODB_SYS_COLUMNS Table

The `Information Schema` `INNODB_SYS_COLUMNS` table contains information about InnoDB fields.

The `PROCESS` privilege is required to view the table.

It has the following columns:

Column	Description
TABLE_ID	Table identifier, matching the value from <code>INNODB_SYS_TABLES.TABLE_ID</code> .
NAME	Column name.
POS	Ordinal position of the column in the table, starting from 0 . This value is adjusted when columns are added or removed.

MTYPE	Numeric column type identifier, (see the table below for an explanation of its values).
PRTYPE	Binary value of the InnoDB precise type, representing the data type, character set code and nullability.
LEN	Column length. For multi-byte character sets, represents the length in bytes.

The column `MTYPE` uses a numeric column type identifier, which has the following values:

Column Type Identifier	Description
1	VARCHAR 🔗
2	CHAR 🔗
3	FIXBINARY
4	BINARY 🔗
5	BLOB 🔗
6	INT 🔗
7	SYS_CHILD
8	SYS
9	FLOAT 🔗
10	DOUBLE 🔗
11	DECIMAL 🔗
12	VARMYSQL
13	MYSQL

Example

```

SELECT * FROM information_schema.INNODB_SYS_COLUMNS LIMIT 3\G
*****
1. row ****
TABLE_ID: 11
  NAME: ID
  POS: 0
 MTYPE: 1
PRTYPE: 524292
  LEN: 0
*****
2. row ****
TABLE_ID: 11
  NAME: FOR_NAME
  POS: 0
 MTYPE: 1
PRTYPE: 524292
  LEN: 0
*****
3. row ****
TABLE_ID: 11
  NAME: REF_NAME
  POS: 0
 MTYPE: 1
PRTYPE: 524292
  LEN: 0
3 rows in set (0.00 sec)

```

INNODB_SYS_DATAFILES Table

MariaDB until [10.5](#)

The `INNODB_SYS_DATAFILES` table was added in [MariaDB 10.0.4](#), and removed in [MariaDB 10.6.0](#).

The [Information Schema](#) `INNODB_SYS_DATAFILES` table contains information about InnoDB datafile paths. It was intended to provide metadata for tablespaces inside InnoDB tables, which was never implemented in MariaDB and was removed in [MariaDB 10.6](#). The `PROCESS` privilege is required to view the table.

It contains the following columns:

Column	Description
SPACE	Numeric tablespace. Matches the <code>INNODB_SYS_TABLES.SPACE</code> value.
PATH	Tablespace datafile path.

Example

```
SELECT * FROM INNODB_SYS_DATAFILES;
+-----+-----+
| SPACE | PATH      |
+-----+-----+
|   19  | ./test/t2.ibd |
|   20  | ./test/t3.ibd |
...
|   68  | ./test/animals.ibd |
|   69  | ./test/animal_count.ibd |
|   70  | ./test/t.ibd |
+-----+-----+
```

H3Dr1.2.9.1.1.1.23 Information Schema INNODB_SYS_FIELDS Table

The [Information Schema](#) `INNODB_SYS_FIELDS` table contains information about fields that are part of an InnoDB index.

The `PROCESS` privilege is required to view the table.

It has the following columns:

Column	Description
INDEX_ID	Index identifier, matching the value from <code>INNODB_SYS_INDEXES.INDEX_ID</code> .
NAME	Field name, matching the value from <code>INNODB_SYS_COLUMNS.NAME</code> .
POS	Ordinal position of the field within the index, starting from 0. This is adjusted as columns are removed.

Example

```

SELECT * FROM information_schema.INNODB_SYS_FIELDS LIMIT 3\G
*****
1. row ****
INDEX_ID: 11
  NAME: ID
  POS: 0
*****
2. row ****
INDEX_ID: 12
  NAME: FOR_NAME
  POS: 0
*****
3. row ****
INDEX_ID: 13
  NAME: REF_NAME
  POS: 0
3 rows in set (0.00 sec)

```

H3Dr1.2.9.1.1.1.24 Information Schema INNODB_SYS_FOREIGN Table

The [Information Schema](#) INNODB_SYS_FOREIGN table contains information about InnoDB [foreign keys](#).

The PROCESS privilege is required to view the table.

It has the following columns:

Column	Description
ID	Database name and foreign key name.
FOR_NAME	Database and table name of the foreign key child.
REF_NAME	Database and table name of the foreign key parent.
N_COLS	Number of foreign key index columns.
TYPE	Bit flag providing information about the foreign key.

The TYPE column provides a bit flag with information about the foreign key. This information is OR'ed together to read:

Bit Flag	Description
1	ON DELETE CASCADE
2	ON UPDATE SET NULL
4	ON UPDATE CASCADE
8	ON UPDATE SET NULL
16	ON DELETE NO ACTION
32	ON UPDATE NO ACTION

Example

```

SELECT * FROM INNODB_SYS_FOREIGN\G
*****
1. row ****
ID: mysql/innodb_index_stats_ibfk_1
FOR_NAME: mysql/innodb_index_stats
REF_NAME: mysql/innodb_table_stats
N_COLS: 2
TYPE: 0
...

```

H3Dr1.2.9.1.1.1.25 Information Schema

INNODB_SYS_FOREIGN_COLS Table

The [Information Schema](#) `INNODB_SYS_FOREIGN_COLS` table contains information about InnoDB foreign key columns.

The `PROCESS` privilege is required to view the table.

It has the following columns:

Column	Description
ID	Foreign key index associated with this column, matching the <code>INNODB_SYS_FOREIGN.ID</code> field.
FOR_COL_NAME	Child column name.
REF_COL_NAME	Parent column name.
POS	Ordinal position of the column in the table, starting from 0.

H3Dr1.2.9.1.1.26 Information Schema INNODB_SYS_INDEXES Table

The [Information Schema](#) `INNODB_SYS_INDEXES` table contains information about InnoDB indexes.

The `PROCESS` privilege is required to view the table.

It has the following columns:

Field	Type	Null	Key	Default	Description
INDEX_ID	bigint(21) unsigned	NO		0	A unique index identifier.
NAME	varchar(64)	NO			Index name, lowercase for all user-created indexes, or uppercase for implicitly-created indexes; <code>PRIMARY</code> (primary key), <code>GEN_CLUST_INDEX</code> (index representing primary key where there isn't one), <code>ID_IND</code> , <code>FOR_IND</code> (validating foreign key constraint), <code>REF_IND</code> .
TABLE_ID	bigint(21) unsigned	NO		0	Table identifier, matching the value from <code>INNODB_SYS_TABLES.TABLE_ID</code> .
TYPE	int(11)	NO		0	Numeric type identifier; one of 0 (secondary index), 1 (clustered index), 2 (unique index), 3 (primary index), 32 (full-text index).
N_FIELDS	int(11)	NO		0	Number of columns in the index. <code>GEN_CLUST_INDEX</code> 's have a value of 0 as the index is not based on an actual column in the table.
PAGE_NO	int(11)	NO		0	Index B-tree's root page number. -1 (unused) for full-text indexes, as they are laid out over several auxiliary tables.
SPACE	int(11)	NO		0	Tablespace identifier where the index resides. 0 represents the InnoDB system tablespace, while any other value represents a table created in file-per-table mode (see the <code>innodb_file_per_table</code> system variable). Remains unchanged after a <code>TRUNCATE TABLE</code> statement, and not necessarily unique.

MERGE_THRESHOLD	int(11)	NO		0	
-----------------	---------	----	--	---	--

Example

```

SELECT * FROM information_schema.INNODB_SYS_INDEXES LIMIT 3\G
*****
1. row ****
INDEX_ID: 11
      NAME: ID_IND
    TABLE_ID: 11
      TYPE: 3
  N_FIELDS: 1
  PAGE_NO: 302
    SPACE: 0
MERGE_THRESHOLD: 50
*****
2. row ****
INDEX_ID: 12
      NAME: FOR_IND
    TABLE_ID: 11
      TYPE: 0
  N_FIELDS: 1
  PAGE_NO: 303
    SPACE: 0
MERGE_THRESHOLD: 50
*****
3. row ****
INDEX_ID: 13
      NAME: REF_IND
    TABLE_ID: 11
      TYPE: 3
  N_FIELDS: 1
  PAGE_NO: 304
    SPACE: 0
MERGE_THRESHOLD: 50
3 rows in set (0.00 sec)

```

H3Dr1.2.9.1.1.1.27 Information Schema INNODB_SYS_SEMAPHORE_WAITS Table

The [Information Schema](#) INNODB_SYS_SEMAPHORE_WAITS table is meant to contain information about current semaphore waits. At present it is not correctly populated. See [MDEV-21330](#).

The [PROCESS privilege](#) is required to view the table.

It contains the following columns:

Column	Description
THREAD_ID	Thread id waiting for semaphore
OBJECT_NAME	Semaphore name
FILE	File name where semaphore was requested
LINE	Line number on above file

WAIT_TIME	Wait time
WAIT_OBJECT	
WAIT_TYPE	Object type (mutex, rw-lock)
HOLDER_THREAD_ID	Holder thread id
HOLDER_FILE	File name where semaphore was acquired
HOLDER_LINE	Line number for above
CREATED_FILE	Creation file name
CREATED_LINE	Line number for above
WRITER_THREAD	Last write request thread id
RESERVATION_MODE	Reservation mode (shared, exclusive)
READERS	Number of readers if only shared mode
WAITERS_FLAG	Flags
LOCK_WORD	Lock word (for developers)
LAST_READER_FILE	Removed
LAST_READER_LINE	Removed
LAST_WRITER_FILE	Last writer file name
LAST_WRITER_LINE	Above line number
OS_WAIT_COUNT	Wait count

H3Dr1.2.9.1.1.1.28 Information Schema INNODB_SYS_TABLES Table

The `Information Schema INNODB_SYS_TABLES` table contains information about InnoDB tables.

The `PROCESS` privilege is required to view the table.

It has the following columns:

Field	Type	Null	Key	Default	Description
TABLE_ID	bigint(21) unsigned	NO		0	Unique InnoDB table identifier.
NAME	varchar(655)	NO			Database and table name, or the uppercase InnoDB system table name.
FLAG	int(11)	NO		0	See Flag below
N_COLS	int(11) unsigned (>= MariaDB 10.5) int(11) (<= MariaDB 10.4)	NO		0	Number of columns in the table.
SPACE	int(11) unsigned (>= MariaDB 10.5) int(11) (<= MariaDB 10.4)	NO		0	Tablespace identifier where the index resides. 0 represents the InnoDB system tablespace, while any other value represents a table created in file-per-table mode (see the innodb_file_per_table system variable). Remains unchanged after a TRUNCATE TABLE statement.

FILE_FORMAT	varchar(10)	YES		NULL	InnoDB file format 🔗 (Antelope or Barracuda). Removed in MariaDB 10.3 🔗 .
ROW_FORMAT	enum('Redundant', 'Compact', 'Compressed', 'Dynamic') (\geq MariaDB 10.5 🔗) varchar(12) (\leq MariaDB 10.4 🔗)	YES		NULL	InnoDB storage format 🔗 (Compact, Redundant, Dynamic, or Compressed).
ZIP_PAGE_SIZE	int(11) unsigned	NO		0	For Compressed tables, the zipped page size.
SPACE_TYPE	enum('Single','System') (\geq MariaDB 10.5 🔗) varchar(10) (\leq MariaDB 10.4 🔗)	YES		NULL	

Flag

The flag field returns the dict_table_t::flags that correspond to the data dictionary record.

Bit	Description
0	Set if ROW_FORMAT is not REDUNDANT.
1 to 4	0 , except for ROW_FORMAT=COMPRESSED, where they will determine the KEY_BLOCK_SIZE (the compressed page size).
5	Set for ROW_FORMAT=DYNAMIC or ROW_FORMAT=COMPRESSED.
6	Set if the DATA DIRECTORY attribute was present when the table was originally created.
7	Set if the page_compressed attribute is present.
8 to 11	Determine the page_compression_level.
12	Normally 00 , but 11 for "no-rollback tables" (MariaDB 10.3 🔗 CREATE SEQUENCE). In MariaDB 10.1 🔗 , these bits could be 01 or 10 for ATOMIC_WRITES=ON or ATOMIC_WRITES=OFF.
13	

Note that the table flags returned here are not the same as tablespace flags (FSP_SPACE_FLAGS).

Example

```

SELECT * FROM information_schema.INNODB_SYS_TABLES LIMIT 2\G
***** 1. row *****
  TABLE_ID: 14
    NAME: SYS_DATAFILES
    FLAG: 0
   N_COLS: 5
   SPACE: 0
  ROW_FORMAT: Redundant
ZIP_PAGE_SIZE: 0
  SPACE_TYPE: System
***** 2. row *****
  TABLE_ID: 11
    NAME: SYS_FOREIGN
    FLAG: 0
   N_COLS: 7
   SPACE: 0
  ROW_FORMAT: Redundant
ZIP_PAGE_SIZE: 0
  SPACE_TYPE: System
2 rows in set (0.00 sec)

```

See Also

- [InnoDB Data Dictionary Troubleshooting](#)

H3Dr1.2.9.1.1.1.29 Information Schema INNODB_SYS_TABLESPACES Table

The `Information Schema INNODB_SYS_TABLESPACES` table contains information about InnoDB tablespaces. Until [MariaDB 10.5](#), it was based on the internal `SYS_TABLESPACES` table. This internal table was removed in [MariaDB 10.6.0](#), so this `Information Schema` table has been repurposed to directly reflect the filesystem (`fil_system.space_list`).

The `PROCESS privilege` is required to view the table.

It has the following columns:

Column	Description
SPACE	Unique InnoDB tablespace identifier.
NAME	Database and table name separated by a backslash, or the uppercase InnoDB system table name.
FLAG	1 if a <code>DATA DIRECTORY</code> option has been specified in <code>CREATE TABLE</code> , otherwise 0 .
FILE_FORMAT	InnoDB file format . Removed in MariaDB 10.3.1 .
ROW_FORMAT	InnoDB storage format used for this tablespace. If the Antelope file format is used, this value is always Compact or Redundant .
PAGE_SIZE	Page size in bytes for this tablespace. Until MariaDB 10.5.0 , this was the value of the <code>innodb_page_size</code> variable. From MariaDB 10.6.0 , contains the physical page size of a page (previously <code>ZIP_PAGE_SIZE</code>).
ZIP_PAGE_SIZE	Zip page size for this tablespace. Removed in MariaDB 10.6.0 .
SPACE_TYPE	Tablespace type. Can be General for general tablespaces or Single for file-per-table tablespaces. Introduced MariaDB 10.2.1 . Removed MariaDB 10.5.0 .
FS_BLOCK_SIZE	File system block size. Introduced MariaDB 10.2.1 .
FILE_SIZE	Maximum size of the file, uncompressed. Introduced MariaDB 10.2.1 .
ALLOCATED_SIZE	Actual size of the file as per space allocated on disk. Introduced MariaDB 10.2.1 .
FILENAME	Tablespace datafile path, previously part of the <code>INNODB_SYS_DATAFILES</code> table. Added in MariaDB 10.6.0 .

Examples

MariaDB 10.4 ↗:

Field	Type	Null	Key	Default	Extra
SPACE	int(11) unsigned	NO		0	
NAME	varchar(655)	NO			
FLAG	int(11) unsigned	NO		0	
ROW_FORMAT	varchar(22)	YES		NULL	
PAGE_SIZE	int(11) unsigned	NO		0	
ZIP_PAGE_SIZE	int(11) unsigned	NO		0	
SPACE_TYPE	varchar(10)	YES		NULL	
FS_BLOCK_SIZE	int(11) unsigned	NO		0	
FILE_SIZE	bigint(21) unsigned	NO		0	
ALLOCATED_SIZE	bigint(21) unsigned	NO		0	

From MariaDB 10.4 ↗:

```
SELECT * FROM information_schema.INNODB_SYS_TABLESPACES LIMIT 2\G
*****
1. row ****
    SPACE: 2
        NAME: mysql/innodb_table_stats
        FLAG: 33
    ROW_FORMAT: Dynamic
        PAGE_SIZE: 16384
    ZIP_PAGE_SIZE: 0
        SPACE_TYPE: Single
    FS_BLOCK_SIZE: 4096
        FILE_SIZE: 98304
    ALLOCATED_SIZE: 98304
*****
2. row ****
    SPACE: 3
        NAME: mysql/innodb_index_stats
        FLAG: 33
    ROW_FORMAT: Dynamic
        PAGE_SIZE: 16384
    ZIP_PAGE_SIZE: 0
        SPACE_TYPE: Single
    FS_BLOCK_SIZE: 4096
        FILE_SIZE: 98304
    ALLOCATED_SIZE: 98304
```

H3Dr1.2.9.1.1.1.30 Information Schema INNODB_SYS_TABLESTATS Table

The [Information Schema](#) `INNODB_SYS_TABLESTATS` table contains InnoDB status information. It can be used for developing new performance-related extensions, or high-level performance monitoring.

The `PROCESS` privilege is required to view the table.

Note that the MySQL InnoDB and Percona XtraDB versions of the tables differ (see [XtraDB](#) and [InnoDB](#) ↗).

It contains the following columns:

Column	Description
TABLE_ID	Table ID, matching the INNODB_SYS_TABLES.TABLE_ID value.
SCHEMA	Database name (XtraDB only).
NAME	Table name, matching the INNODB_SYS_TABLES.NAME value.

STATS_INITIALIZED	Initialized if statistics have already been collected, otherwise Uninitialized.
NUM_ROWS	Estimated number of rows currently in the table. Updated after each statement modifying the data, but uncommitted transactions mean it may not be accurate.
CLUST_INDEX_SIZE	Number of pages on disk storing the clustered index, holding InnoDB table data in primary key order, or <code>NULL</code> if not statistics yet collected.
OTHER_INDEX_SIZE	Number of pages on disk storing secondary indexes for the table, or <code>NULL</code> if not statistics yet collected.
MODIFIED_COUNTER	Number of rows modified by statements modifying data.
AUTOINC	Auto_increment value.
REF_COUNT	Countdown to zero, when table metadata can be removed from the table cache. (InnoDB only)
MYSQL_HANDLES_OPENED	(XtraDB only).

H3Dr1.2.9.1.1.1.31 Information Schema INNODB_SYS_VIRTUAL Table

MariaDB starting with [10.2](#)

The `INNODB_SYS_VIRTUAL` table was added in [MariaDB 10.2](#).

The [Information Schema](#) `INNODB_SYS_VIRTUAL` table contains information about base columns of [virtual columns](#). The `PROCESS` [privilege](#) is required to view the table.

It contains the following columns:

Field	Type	Null	Key	Default	Description
TABLE_ID	bigint(21) unsigned	NO		0	
POS	int(11) unsigned	NO		0	
BASE_POS	int(11) unsigned	NO		0	

H3Dr1.2.9.1.1.1.32 Information Schema INNODB_TABLESPACES_ENCRYPTION Table

The [Information Schema](#) `INNODB_TABLESPACES_ENCRYPTION` table contains metadata about [encrypted InnoDB tablespaces](#). When you [enable encryption for an InnoDB tablespace](#), an entry for the tablespace is added to this table. If you later [disable encryption for the InnoDB tablespace](#), then the row still remains in this table, but the `ENCRYPTION_SCHEME` and `CURRENT_KEY_VERSION` columns will be set to `0`.

Viewing this table requires the `PROCESS` privilege, although a bug in versions before [MariaDB 10.1.46](#), [10.2.33](#), [10.3.24](#), [10.4.14](#) and [10.5.5](#) mean the `SUPER` privilege was required ([MDEV-23003](#)).

It has the following columns:

Column	Description	Added
SPACE	InnoDB tablespace ID.	
NAME	Path to the InnoDB tablespace file, without the extension.	

ENCRYPTION_SCHEME	Key derivation algorithm. Only 1 is currently used to represent an algorithm. If this value is 0, then the tablespace is unencrypted.	
KEYSERVER_REQUESTS	Number of times InnoDB has had to request a key from the encryption key management plugin . The three most recent keys are cached internally.	
MIN_KEY_VERSION	Minimum key version used to encrypt a page in the tablespace. Different pages may be encrypted with different key versions.	
CURRENT_KEY_VERSION	Key version that will be used to encrypt pages. If this value is 0, then the tablespace is unencrypted.	
KEY_ROTATION_PAGE_NUMBER	Page that a background encryption thread is currently rotating. If key rotation is not enabled, then the value will be NULL.	
KEY_ROTATION_MAX_PAGE_NUMBER	When a background encryption thread starts rotating a tablespace, the field contains its current size. If key rotation is not enabled, then the value will be NULL.	
CURRENT_KEY_ID	Key ID for the encryption key currently in use.	MariaDB 10.1.13
ROTATING_OR_FLUSHING	Current key rotation status. If this value is 1, then the background encryption threads are working on the tablespace. See MDEV-11738 .	MariaDB 10.2.5 , MariaDB 10.1.23

When the [InnoDB system tablespace](#) is encrypted, it is represented in this table with the special name: `innodb_system`.

Example

```

SELECT * FROM information_schema.INNODB_TABLESPACES_ENCRYPTION
WHERE NAME LIKE 'db_encrypt%';
+-----+-----+-----+-----+
| SPACE | NAME          | ENCRYPTION_SCHEME |
| KEYSERVER_REQUESTS | MIN_KEY_VERSION | CURRENT_KEY_VERSION | KEY_ROTATION_PAGE_NUMBER |
| KEY_ROTATION_MAX_PAGE_NUMBER |
+-----+-----+-----+-----+
-----+
|   18 | db_encrypt/t_encrypted_existing_key |           1 |
| 1 |           1 |           1 |      NULL |
| NULL |
|   19 | db_encrypt/t_not_encrypted_existing_key |           1 |
| 0 |           1 |           1 |      NULL |
| NULL |
|   20 | db_encrypt/t_not_encrypted_non_existing_key |           1 |
| 0 | 4294967295 | 4294967295 |      NULL |
| NULL |
|   21 | db_encrypt/t_default_encryption_existing_key |           1 |
| 1 |           1 |           1 |      NULL |
| NULL |
|   22 | db_encrypt/t_encrypted_default_key |           1 |
| 1 |           1 |           1 |      NULL |
| NULL |
|   23 | db_encrypt/t_not_encrypted_default_key |           1 |
| 0 |           1 |           1 |      NULL |
| NULL |
|   24 | db_encrypt/t_defaults |           1 |
| 1 |           1 |           1 |      NULL |
+-----+
-----+
-----+
7 rows in set (0.00 sec)

```

See Also

- [Encrypting Data for InnoDB / XtraDB](#)
- [Data at Rest Encryption](#)
- [Why Encrypt MariaDB Data?](#)
- [Encryption Key Management](#)

H3Dr1.2.9.1.1.1.33 Information Schema INNODB_TABLESPACES_SCRUBBING Table

MariaDB 10.1.3 - 10.5.1

InnoDB and XtraDB data scrubbing was introduced in [MariaDB 10.1.3](#). The table was removed in [MariaDB 10.5.2](#) - see [MDEV-15528](#).

The [Information Schema](#) INNODB_TABLESPACES_SCRUBBING table contains [data scrubbing](#) information.

The [PROCESS privilege](#) is required to view the table.

It has the following columns:

Column	Description
SPACE	InnoDB table space id number.
NAME	Path to the table space file, without the extension.

COMPRESSED	The compressed page size, or zero if uncompressed.
LAST_SCRUB_COMPLETED	Date and time when the last scrub was completed, or <code>NULL</code> if never been performed.
CURRENT_SCRUB_STARTED	Date and time when the current scrub started, or <code>NULL</code> if never been performed.
CURRENT_SCRUB_ACTIVE_THREADS	Number of threads currently scrubbing the tablespace.
CURRENT_SCRUB_PAGE_NUMBER	Page that the scrubbing thread is currently scrubbing, or <code>NULL</code> if not enabled.
CURRENT_SCRUB_MAX_PAGE_NUMBER	When a scrubbing starts rotating a table space, the field contains its current size. <code>NULL</code> if not enabled.

Example

```
SELECT * FROM information_schema.INNODB_TABLESPACES_SCRUBBING LIMIT 1\G
*****
1. row ****
SPACE: 1
      NAME: mysql/innodb_table_stats
COMPRESSED: 0
      LAST_SCRUB_COMPLETED: NULL
      CURRENT_SCRUB_STARTED: NULL
      CURRENT_SCRUB_PAGE_NUMBER: NULL
CURRENT_SCRUB_MAX_PAGE_NUMBER: 0
      ROTATING_OR_FLUSHING: 0
1 rows in set (0.00 sec)
```

H3Dr1.2.9.1.1.1.34 Information Schema INNODB_TRX Table

The [Information Schema](#) `INNODB_TRX` table stores information about all currently executing InnoDB transactions.

It has the following columns:

Column	Description
TRX_ID	Unique transaction ID number.
TRX_STATE	Transaction execution state; one of <code>RUNNING</code> , <code>LOCK_WAIT</code> , <code>ROLLING BACK</code> or <code>COMMITTING</code> .
TRX_STARTED	Time that the transaction started.
TRX_REQUESTED_LOCK_ID	If <code>TRX_STATE</code> is <code>LOCK_WAIT</code> , the <code>INNODB_LOCKS.LOCK_ID</code> value of the lock being waited on. <code>NULL</code> if any other state.
TRX_WAIT_STARTED	If <code>TRX_STATE</code> is <code>LOCK_WAIT</code> , the time the transaction started waiting for the lock, otherwise <code>NULL</code> .
TRX_WEIGHT	Transaction weight, based on the number of locked rows and the number of altered rows. To resolve deadlocks, lower weighted transactions are rolled back first. Transactions that have affected non-transactional tables are always treated as having a heavier weight.
TRX_MYSQL_THREAD_ID	Thread ID from the PROCESSLIST table (note that the locking and transaction information schema tables use a different snapshot from the processlist, so records may appear in one but not the other).
TRX_QUERY	SQL that the transaction is currently running.
TRX_OPERATION_STATE	Transaction's current state, or <code>NULL</code> .

TRX_TABLES_IN_USE	Number of InnoDB tables currently being used for processing the current SQL statement.
TRX_TABLES_LOCKED	Number of InnoDB tables that have row locks held by the current SQL statement.
TRX_LOCK_STRUCTS	Number of locks reserved by the transaction.
TRX_LOCK_MEMORY_BYTES	Total size in bytes of the memory used to hold the lock structures for the current transaction in memory.
TRX_ROWS_LOCKED	Number of rows the current transaction has locked. Locked by this transaction. An approximation, and may include rows not visible to the current transaction that are delete-marked but physically present.
TRX_ROWS_MODIFIED	Number of rows added or changed in the current transaction.
TRX_CONCURRENCY_TICKETS	Indicates how much work the current transaction can do before being swapped out, see the innodb_concurrency_tickets system variable.
TRX_ISOLATION_LEVEL	Isolation level of the current transaction.
TRX_UNIQUE_CHECKS	Whether unique checks are <code>on</code> or <code>off</code> for the current transaction. Bulk data are a case where unique checks would be off.
TRX_FOREIGN_KEY_CHECKS	Whether foreign key checks are <code>on</code> or <code>off</code> for the current transaction. Bulk data are a case where foreign keys checks would be off.
TRX_LAST_FOREIGN_KEY_ERROR	Error message for the most recent foreign key error, or <code>NULL</code> if none.
TRX_ADAPTIVE_HASH_LATCHED	Whether the adaptive hash index is locked by the current transaction or not. One transaction at a time can change the adaptive hash index.
TRX_ADAPTIVE_HASH_TIMEOUT	Whether the adaptive hash index search latch should be relinquished immediately or reserved across all MariaDB calls. <code>0</code> if there is no contention on the adaptive hash index, in which case the latch is reserved until completion, otherwise counts down to zero and the latch is released after each row lookup.
TRX_IS_READ_ONLY	<code>1</code> if a read-only transaction, otherwise <code>0</code> .
TRX_AUTOCOMMIT_NON_LOCKING	<code>1</code> if the transaction only contains this one statement, that is, a SELECT statement not using <code>FOR UPDATE</code> or <code>LOCK IN SHARED MODE</code> , and with autocommit on. If this and <code>TRX_IS_READ_ONLY</code> are both <code>1</code> , the transaction can be optimized by the storage engine to reduce some overheads

The table is often used in conjunction with the [INNODB_LOCKS](#) and [INNODB_LOCK_WAIT](#)s tables to diagnose problematic locks and transactions.

[XA transactions](#) are not stored in this table. To see them, `XA RECOVER` can be used.

Example

```

-- session 1
START TRANSACTION;
UPDATE t SET id = 15 WHERE id = 10;

-- session 2
DELETE FROM t WHERE id = 10;

-- session 1
USE information_schema;
SELECT 1.* , t.* 
    FROM information_schema.INNODB_LOCKS l 
    JOIN information_schema.INNODB_TRX t 
        ON l.lock trx_id = t.trx_id 
    WHERE trx_state = 'LOCK WAIT' \G
*****
 1. row ****
lock_id: 840:40:3:2
lock_trx_id: 840
lock_mode: X
lock_type: RECORD
lock_table: `test`.`t`
lock_index: PRIMARY
lock_space: 40
lock_page: 3
lock_rec: 2
lock_data: 10
      trx_id: 840
      trx_state: LOCK WAIT
      trx_started: 2019-12-23 18:43:46
trx_requested_lock_id: 840:40:3:2
      trx_wait_started: 2019-12-23 18:43:46
      trx_weight: 2
      trx_mysql_thread_id: 46
      trx_query: DELETE FROM t WHERE id = 10
      trx_operation_state: starting index read
      trx_tables_in_use: 1
      trx_tables_locked: 1
      trx_lock_structs: 2
      trx_lock_memory_bytes: 1136
      trx_rows_locked: 1
      trx_rows_modified: 0
      trx_concurrency_tickets: 0
      trx_isolation_level: REPEATABLE READ
      trx_unique_checks: 1
      trx_foreign_key_checks: 1
      trx_last_foreign_key_error: NULL
      trx_is_read_only: 0
      trx_autocommit_non_locking: 0

```

H3Dr1.2.9.1.1.1.35 Information Schema TEMP_TABLES_INFO Table

MariaDB 10.2.2 [↗](#) - 10.2.3 [↗](#)

The `TEMP_TABLES_INFO` table was introduced in [MariaDB 10.2.2 ↗](#) and was removed in [MariaDB 10.2.4 ↗](#). See [MDEV-12459 ↗](#) progress on an alternative.

The `Information Schema TEMP_TABLES_INFO` table contains information about active InnoDB temporary tables. All user and system-created temporary tables are reported when querying this table, with the exception of optimized internal temporary tables. The data is stored in memory.

Previously, InnoDB temp table metadata was rather stored in InnoDB system tables.

It has the following columns:

Column	Description
--------	-------------

TABLE_ID	Table ID.
NAME	Table name.
N_COLS	Number of columns in the temporary table, including three hidden columns that InnoDB creates (DB_ROW_ID , DB_TRX_ID , and DB_ROLL_PTR).
SPACE	Numerical identifier for the tablespace identifier holding the temporary table. Compressed temporary tables are stored by default in separate per-table tablespaces in the temporary file directory. For non-compressed tables, the shared temporary table is named <code>ibtmp1</code> , found in the data directory. Always a non-zero value, and regenerated on server restart.
PER_TABLE_TABLESPACE	If TRUE , the temporary table resides in a separate per-table tablespace. If FALSE , it resides in the shared temporary tablespace.
IS_COMPRESSED	TRUE if the table is compressed.

The PROCESS privilege is required to view the table.

Examples

```
CREATE TEMPORARY TABLE t (i INT) ENGINE=INNODB;

SELECT * FROM INFORMATION_SCHEMA.INNODB_TEMP_TABLE_INFO;
+-----+-----+-----+-----+
| TABLE_ID | NAME      | N_COLS | SPACE | PER_TABLE_TABLESPACE | IS_COMPRESSED |
+-----+-----+-----+-----+
|     39 | #sql1c93_3_1 |      4 |    64 | FALSE                | FALSE          |
+-----+-----+-----+-----+
```

Adding a compressed table:

```
SET GLOBAL innodb_file_format="Barracuda";

CREATE TEMPORARY TABLE t2 (i INT) ROW_FORMAT=COMPRESSED ENGINE=INNODB;

SELECT * FROM INFORMATION_SCHEMA.INNODB_TEMP_TABLE_INFO;
+-----+-----+-----+-----+
| TABLE_ID | NAME      | N_COLS | SPACE | PER_TABLE_TABLESPACE | IS_COMPRESSED |
+-----+-----+-----+-----+
|     40 | #sql1c93_3_3 |      4 |    65 | TRUE                 | TRUE           |
|     39 | #sql1c93_3_1 |      4 |    64 | FALSE                | FALSE          |
+-----+-----+-----+-----+
```

H3Dr1.2.9.1.1.2 Information Schema MyRocks Tables

List of Information Schema tables specifically related to MyRocks [🔗](#).



Information Schema ROCKSDB_CFSTATS Table

The Information Schema ROCKSDB_CFSTATS table is included as part of the MyR... ↗



Information Schema ROCKSDB_CF_OPTIONS Table

Information about MyRocks Column Families. ↗



Information Schema ROCKSDB_COMPACTION_STATS Table

The Information Schema ROCKSDB_COMPACTION_STATS table is included as part o... ↗



Information Schema ROCKSDB_DBSTATS Table

The Information Schema ROCKSDB_DBSTATS table is included as part of the MyR... ↗



Information Schema ROCKSDB_DDL Table

The Information Schema ROCKSDB_DDL table is included as part of the MyRocks... ↗



Information Schema ROCKSDB_DEADLOCK Table

The Information Schema ROCKSDB_DEADLOCK table is included as part of the My... ↗



Information Schema ROCKSDB_GLOBAL_INFO Table

The Information Schema ROCKSDB_GLOBAL_INFO table is included as part of the... ↗



Information Schema ROCKSDB_INDEX_FILE_MAP Table

The Information Schema ROCKSDB_INDEX_FILE_MAP table is included as part of ... ↗



Information Schema ROCKSDB_LOCKS Table

The Information Schema ROCKSDB_LOCKS table is included as part of the MyRoc... ↗



Information Schema ROCKSDB_PERF_CONTEXT Table

Per-table/partition counters. ↗



Information Schema ROCKSDB_PERF_CONTEXT_GLOBAL Table

Global counters. ↗



Information Schema ROCKSDB_SST_PROPS Table

The Information Schema ROCKSDB_SST_PROPS table is included as part of the M... ↗



Information Schema ROCKSDB_TRX Table

The Information Schema ROCKSDB_TRX table is included as part of the MyRocks... ↗