

1. SQL Statements and Structure

1.1 SQL Statements

1.1.1 Account Management SQL Commands

1.1.1.1 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
}
```

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)
```

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](#)

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
<code>REQUIRE NONE</code>	TLS is not required for this account, but can still be used.
<code>REQUIRE SSL</code>	The account must use TLS, but no valid X509 certificate is required. This option cannot be combined with other TLS options.
<code>REQUIRE X509</code>	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.
<code>REQUIRE ISSUER 'issuer'</code>	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.
<code>REQUIRE SUBJECT 'subject'</code>	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 <code>REQUIRE X509</code> . This option can be combined with the <code>ISSUER</code> , and <code>CIPHER</code> options in any order.
<code>REQUIRE CIPHER 'cipher'</code>	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 <code>REQUIRE SSL</code> . This option can be combined with the <code>ISSUER</code> , and <code>SUBJECT</code> 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
<code>MAX_QUERIES_PER_HOUR</code>	Number of statements that the account can issue per hour (including updates)
<code>MAX_UPDATES_PER_HOUR</code>	Number of updates (not queries) that the account can issue per hour
<code>MAX_CONNECTIONS_PER_HOUR</code>	Number of connections that the account can start per hour
<code>MAX_USER_CONNECTIONS</code>	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.
<code>MAX_STATEMENT_TIME</code>	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.

1.1.1.2 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
}

```

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`:

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](#)

```
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'  
***** 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'  
***** 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') |  
+-----+  
| *54958E764CE10E50764C2EECB71D01F08549980 |  
+-----+
```

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

```
ALTER 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](#):

```
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 <code>REQUIRE X509</code> . This option can be combined with the <code>ISSUER</code> , and <code>CIPHER</code> 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 <code>REQUIRE SSL</code> . This option can be combined with the <code>ISSUER</code> , and <code>SUBJECT</code> 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/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
<code>MAX_QUERIES_PER_HOUR</code>	Number of statements that the account can issue per hour (including updates)
<code>MAX_UPDATES_PER_HOUR</code>	Number of updates (not queries) that the account can issue per hour
<code>MAX_CONNECTIONS_PER_HOUR</code>	Number of connections that the account can start per hour
<code>MAX_USER_CONNECTIONS</code>	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.
<code>MAX_STATEMENT_TIME</code>	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 `MAX_CONNECTIONS_PER_HOUR`), and `max_user_connections` (for `MAX_USER_CONNECTIONS`).

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.

1.1.1.3 DROP USER

Syntax

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

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'@'%'
```

Contents

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

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](#)

1.1.1.4 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
- 9. 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](#).
- [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 `binlog_annotation_row_events`, `binlog_cache_size`, `binlog_commit_wait_count`, `binlog_commit_wait_usec`, `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` system variables. 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`, `SHOW REPLICA STATUS` and `SHOW BINARY LOGS` statements.

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` and `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`, not executing the statements specified in `init_connect`, `killing connections` and `queries` owned by other users as well as setting the following connection-related system variables: `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_mode`, `thread_pool_oversubscribe`, `thread_pool_prio_pickup_timer`, `thread_pool_priority`, `thread_pool_size`, `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](#), [SHOW SLAVE 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).

REPLICATION MASTER ADMIN

Permits administration of primary servers, including the [SHOW REPLICAS 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 REPLICAS 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 REPLICAS STATUS](#), but after upgrading to a [MariaDB 10.5](#) minor release prior to [MariaDB 10.5.9](#), they could no longer run [SHOW REPLICAS 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 REPLICA 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 `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` system variables. 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 after 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 <code>CREATE DATABASE</code> 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 <code>CREATE PROCEDURE</code> and <code>CREATE FUNCTION</code> statements.
CREATE TEMPORARY TABLES	Create temporary tables with the <code>CREATE TEMPORARY TABLE</code> statement. This privilege enables writing and dropping those temporary tables.
DROP	Drop a database using the <code>DROP DATABASE</code> 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 <code>LOCK TABLES</code> 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 <code>ALTER PROCEDURE</code> statement.
EXECUTE	Execute a <code>stored procedure</code> using the <code>CALL</code> 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.

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';
1 |
```

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_dba'@'localhost' TO 'bob'@'localhost';
Query OK, 0 rows affected (0.004 sec)

GRANT PROXY ON 'app2_dba'@'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 to the `mysql.user` table.

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 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 |
+-----+
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 <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 <code>REQUIRE X509</code> . This option can be combined with the <code>ISSUER</code> , and <code>CIPHER</code> 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 <code>REQUIRE SSL</code> . This option can be combined with the <code>ISSUER</code> , and <code>SUBJECT</code> 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.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.

1.1.1.5 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';
```

1.1.1.6 REVOKE 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] ...
```

Contents

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

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
```

1.1.1.7 SET PASSWORD

Syntax

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

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 entry.

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

Contents

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

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](#)

1.1.1.8 CREATE ROLE

Syntax

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

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.

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

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:

Contents

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

```

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](#)

1.1.1.9 DROP ROLE

Syntax

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

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:

Contents

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

```
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.

Before MariaDB 10.1.13, the DROP ROLE statement was not permitted in prepared statements.

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](#)

1.1.1.10 SET ROLE

Syntax

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

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](#).

Contents

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

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          |
+-----+

```

1.1.1.11 SET DEFAULT ROLE

Syntax

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

Contents

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

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

```

Removing a default role for another user

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

1.1.1.12 SHOW GRANTS

Syntax

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

Contents

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

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](#)

1.1.1.13 SHOW CREATE USER

MariaDB starting with [10.2.0](#)

`SHOW CREATE USER` was introduced in [MariaDB 10.2.0](#)

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
***** 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.

1.1.2. Administrative SQL Statements

1.1.2.1 Table Statements

1.1.2.1.1 ALTER

1.1.2.1.1.1 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

```

```

| [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] ...

```

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. <>/product>>

This statement is equivalent to the following:

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

See the [ALGORITHM](#) alter specification for more information. <>/product>>

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). <>/product>>

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.

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](#)

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.

See also [Instant ADD COLUMN for InnoDB](#).

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`) 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 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](#)

1.1.2.1.1.2 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'
```

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](#)

1.1.2.1.1.3 ALTER EVENT

Modifies one or more characteristics of an existing event.

Contents

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

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]
```

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](#)

1.1.2.1.1.4 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'
```

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`.

Contents

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

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](#).

Contents

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

Example

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

See Also

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

1.1.2.1.1.5 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.

1.1.2.1.1.6 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](#)

1.1.2.1.1.7 ALTER SEQUENCE

MariaDB starting with [10.3.1](#)

`ALTER SEQUENCE` was introduced in MariaDB 10.3.

Syntax

```
ALTER SEQUENCE [IF EXISTS] sequence_name  
[ INCREMENT [ BY | = ] increment ]  
[ MINVALUE [=] minvalue | NO MINVALUE | NOMINVALUE ]  
[ MAXVALUE [=] maxvalue | NO MAXVALUE | NOMAXVALUE ]  
[ START [ WITH | = ] start ] [ CACHE [=] cache ] [ [ NO ] CYCLE ]  
[ RESTART [[WITH | =] restart]
```

`ALTER SEQUENCE` allows one to change any values for a `SEQUENCE` created with [CREATE SEQUENCE](#).

The options for `ALTER SEQUENCE` can be given in any order.

Description

`ALTER SEQUENCE` changes the parameters of an existing sequence generator. Any parameters not specifically set in the `ALTER SEQUENCE` command retain their prior settings.

`ALTER SEQUENCE` requires the [ALTER privilege](#).

Arguments to ALTER SEQUENCE

The following options may be used:

Option	Default value	Description
INCREMENT	1	Increment to use for values. May be negative.
MINVALUE	1 if <code>INCREMENT</code> > 0 and -9223372036854775807 if <code>INCREMENT</code> < 0	Minimum value for the sequence.
MAXVALUE	9223372036854775806 if <code>INCREMENT</code> > 0 and -1 if <code>INCREMENT</code> < 0	Max value for sequence.
START	<code>MINVALUE</code> if <code>INCREMENT</code> > 0 and <code>MAX_VALUE</code> if <code>INCREMENT</code> < 0	First value that the sequence will generate.
CACHE	1000	Number of values that should be cached. 0 if no <code>CACHE</code> . The underlying table will be updated first time a new sequence number is generated and each time the cache runs out.
CYCLE	0 (= <code>NO CYCLE</code>)	1 if the sequence should start again from <code>MINVALUE</code> # after it has run out of values.

Contents

1. [Syntax](#)
2. [Description](#)
 1. [Arguments to ALTER SEQUENCE](#)
 2. [INSERT](#)
 3. [Notes](#)
3. [See Also](#)

RESTART	START if restart value not is given	If RESTART option is used, NEXT VALUE will return the restart value.
---------	-------------------------------------	--

The optional clause `RESTART [WITH restart]` sets the next value for the sequence. This is equivalent to calling the `SETVAL()` function with the `is_used` argument as 0. The specified value will be returned by the next call of `nextval`. Using `RESTART` with no restart value is equivalent to supplying the start value that was recorded by `CREATE SEQUENCE` or last set by `ALTER SEQUENCE START WITH`.

`ALTER SEQUENCE` will not allow you to change the sequence so that it's inconsistent. For example:

```
CREATE SEQUENCE s1;
ALTER SEQUENCE s1 MINVALUE 10;
ERROR 4061 (HY000): Sequence 'test.t1' values are conflicting

ALTER SEQUENCE s1 MINVALUE 10 RESTART 10;
ERROR 4061 (HY000): Sequence 'test.t1' values are conflicting

ALTER SEQUENCE s1 MINVALUE 10 START 10 RESTART 10;
```

INSERT

To allow `SEQUENCE` objects to be backed up by old tools, like `mysqldump`, one can use `SELECT` to read the current state of a `SEQUENCE` object and use an `INSERT` to update the `SEQUENCE` object. `INSERT` is only allowed if all fields are specified:

```
CREATE SEQUENCE s1;
INSERT INTO s1 VALUES(1000,10,2000,1005,1,1000,0,0);
SELECT * FROM s1;

+-----+-----+-----+-----+-----+-----+
| next_value | min_value | max_value | start | increment | cache | cycle | round |
+-----+-----+-----+-----+-----+-----+
|      1000 |        10 |      2000 |    1005 |         1 |     1000 |       0 |       0 |
+-----+-----+-----+-----+-----+-----+

SHOW CREATE SEQUENCE s1;
+-----+-----+
| Table | Create Table
+-----+-----+
| s1    | CREATE SEQUENCE `s1` start with 1005 minvalue 10 maxvalue 2000 increment by 1 cache 1000 nocycle ENGINE=Aria |
+-----+-----+
```

Notes

`ALTER SEQUENCE` will instantly affect all future `SEQUENCE` operations. This is in contrast to some other databases where the changes requested by `ALTER SEQUENCE` will not be seen until the sequence cache has run out.

`ALTER SEQUENCE` will take a full table lock of the sequence object during its (brief) operation. This ensures that `ALTER SEQUENCE` is replicated correctly. If you only want to set the next sequence value to a higher value than current, then you should use `SETVAL()` instead, as this is not blocking.

If you want to change storage engine, sequence comment or rename the sequence, you can use `ALTER TABLE` for this.

See Also

- [Sequence Overview](#)
- [CREATE SEQUENCE](#)
- [DROP SEQUENCE](#)
- [NEXT VALUE FOR](#)
- [PREVIOUS VALUE FOR](#)
- [SETVAL\(\)](#). Set next value for the sequence.
- [AUTO INCREMENT](#)
- [ALTER TABLE](#)

1.1.2.1.1.8 ALTER SERVER

Syntax

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

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](#)

1.1.2.1.1.9 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.

1.1.2.1.1.10

1.1.2.1.2 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 [CASCADED | LOCAL] CHECK OPTION]
```

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

Contents

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

Contents

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

```
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](#)

1.1.2.1.3 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 ...]])]]
```

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.

Contents

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

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](#)

1.1.2.1.4 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 it 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.

1.1.2.1.5 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](#)

1.1.2.1.6 CHECKSUM TABLE

Syntax

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

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.

1.1.2.1.7 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)
```

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

Contents

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

Contents

1. [Syntax](#)
2. [Description](#)
3. [Privileges](#)
4. [CREATE OR REPLACE](#)
 1. [Things to be Aware of With CREATE](#)

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.

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`

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.

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](#)

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 `NULL` or `NOT NULL` options to specify that values in the column may or may not be `NULL`, respectively. By default, values may be `NULL`. See also [NULL 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 are 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 fails, 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/XtraDB, 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 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 PARTITIONs 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](#).

1.1.2.1.8 DELETE

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]
```

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](#)

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 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 \(video\)](#)

1.1.2.1.9 DROP TABLE

Syntax

```
DROP [TEMPORARY] TABLE [IF EXISTS] /*COMMENT TO SAVE*/
tbl_name [,tbl_name] ...
[WAIT n|NOWAIT]
[RESTRICT | CASCADE]
```

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.

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](#)

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`

Starting from [MariaDB 10.0.8](#), `DROP TABLE` on the master is treated on the slave 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`.

1.1.2.1.10 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](#)

1.1.2.1.11 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.

`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:

<code>mysqlrepair</code>	The default option will be <code>-r</code> (<code>--repair</code>)
<code>mysqlanalyze</code>	The default option will be <code>-a</code> (<code>--analyze</code>)
<code>mysqloptimize</code>	The default option will be <code>-o</code> (<code>--optimize</code>)

Options

`mysqlcheck` supports the following options:

Option	Description
--------	-------------

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](#)

-A , --all-databases	Check all the databases. This is the same as <code>--databases</code> with all databases selected.
-1 , --all-in-1	Instead of issuing one query for each table, use one query per database, naming all tables in the database in a comma-separated list.
-a , --analyze	Analyze given tables.
--auto-repair	If a checked table is corrupted, automatically fix it. Repairing will be done after all tables have been checked.
--character-sets-dir=name	Directory where character set files are installed.
-c , --check	Check table for errors.
-C , --check-only-changed	Check only tables that have changed since last check or haven't been closed properly.
-g , --check-upgrade	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
--compress	Compress all information sent between the client and server if both support compression.
-B , --databases	Check several databases. Note that normally <code>mysqlcheck</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.
-# , --debug[=name]	Output debug log. Often this is 'd:t:o,filename'.
--debug-check	Check memory and open file usage at exit.
--debug-info	Print some debug info at exit.
--default-auth=plugin	Default authentication client-side plugin to use.
--default-character-set=name	Set the default character set .
-e , --extended	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.
-F , --fast	Check only tables that haven't been closed properly.
--fix-db-names	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.
--fix-table-names	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.
--flush	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.
-f , --force	Continue even if we get an SQL error.
-? , --help	Display this help message and exit.
-h name , --host=name	Connect to the given host.
-m , --medium-check	Faster than extended-check, but only finds 99.99 percent of all errors. Should be good enough for most cases.
-o , --optimize	Optimize tables.
-p , --password[=name]	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, <code>mysqlcheck</code> 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.
-Z , --persistent	When using ANALYZE TABLE (<code>--analyze</code>), uses the PERSISTENT FOR ALL 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. Added in MariaDB 10.0.18 and MariaDB 5.5.43 .
<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. Added in MariaDB 10.0.18 and MariaDB 5.5.43 .
<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 CHECK TABLE , 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 REPAIR TABLE , 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.
<code>--ssl-ca=name</code>	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 <code>--ssl</code> option.
<code>--ssl-capath=name</code>	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 <code>openssl rehash</code> 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 <code>--ssl</code> option.
<code>--ssl-cert=name</code>	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 <code>--ssl</code> option.
<code>--ssl-cipher=name</code>	List of permitted ciphers or cipher suites to use for TLS . This option implies the <code>--ssl</code> option.
<code>--ssl-crl=name</code>	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.
<code>--ssl-crlpath=name</code>	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 <code>openssl rehash</code> 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.
<code>--ssl-key=name</code>	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 <code>--ssl</code> 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.
--usefrm	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
--print-defaults	Print the program argument list and exit.
--no-defaults	Don't read default options from any option file.
--defaults-file=#	Only read default options from the given file #.
--defaults-extra-file=#	Read this file after the global files are read.
--defaults-group-suffix=#	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
[mysqlcheck]	Options read by <code>mysqlcheck</code> , which includes both MariaDB Server and MySQL Server.
[mariadb-check]	Options read by <code>mysqlcheck</code> . Available starting with MariaDB 10.4.6 .
[client]	Options read by all MariaDB and MySQL client programs , which includes both MariaDB and MySQL clients. For example, <code>mysqldump</code> .
[client-server]	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.
[client-mariadb]	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.

MariaDB starting with 10.0.14

If you use three `--verbose` options you will also get, on stdout, all `ALTER`, `RENAME`, and `CHECK` commands that `mysqlcheck` executes.

1.1.2.1.12 OPTIMIZE TABLE

Syntax

```
OPTIMIZE [NO_WRITE_TO_BINLOG | LOCAL] TABLE
tbl_name [, tbl_name] ...
[WAIT n | NOWAIT]
```

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](#).

Contents

1. [Syntax](#)
2. [Description](#)
 1. [WAIT/NOWAIT](#)
 2. [Defragmenting](#)
 3. [Updating an InnoDB fulltext index](#)
 4. [Defragmenting InnoDB tablespaces](#)
3. [See Also](#)

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](#)

1.1.2.1.13 RENAME TABLE

Syntax

```
RENAME TABLE[S] [IF EXISTS] tbl_name  
[WAIT n | NOWAIT]  
TO new_tbl_name  
[, tbl_name2 TO new_tbl_name2] ...
```

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.

Contents

1. [Syntax](#)
2. [Description](#)
 1. [IF EXISTS](#)
 2. [WAIT/NOWAIT](#)
 3. [Privileges](#)
 4. [Atomic RENAME TABLE](#)

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.

1.1.2.1.14 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.

1.1.2.1.15 REPAIR VIEW

Syntax

```
REPAIR [NO_WRITE_TO_BINLOG | LOCAL] VIEW view_name[, view_name] ... [FROM MYSQL]
```

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](#)

1.1.2.1.16 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:

Contents

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

```

REPLACE [LOW_PRIORITY | DELAYED]
[INTO] tbl_name [PARTITION (partition_list)] [(col,...)]
SELECT ...
[RETURNING select_expr
[, select_expr ...]]

```

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

Contents

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

```
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

1.1.2.1.17 SHOW COLUMNS

Syntax

```

SHOW [FULL] {COLUMNS | FIELDS} FROM tbl_name [FROM db_name]
[LIKE 'pattern' | WHERE expr]

```

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.

Contents

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

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](#)

1.1.2.1.18 SHOW CREATE TABLE

Syntax

```
SHOW CREATE TABLE tbl_name
```

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.

Contents

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

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](#)

1.1.2.1.19 SHOW INDEX

Syntax

```

SHOW {INDEX | INDEXES | KEYS}
  FROM tbl_name [FROM db_name]
  [WHERE expr]

```

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;

```

Contents

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

`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 NULL's 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](#)

1.1.2.1.20 TRUNCATE TABLE

Syntax

```
TRUNCATE [TABLE] tbl_name  
[WAIT n | NOWAIT]
```

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.

Contents

1. [Syntax](#)
2. [Description](#)
 1. [WAIT/NOWAIT](#)
 2. [Oracle-mode](#)
 3. [Performance](#)
3. [See Also](#)

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

1.1.2.1.21 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]
```

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.

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

- 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](#)

1.1.2.1.22 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	<code>ER_DUP_KEY</code>	Can't write; duplicate key in table '%s'
1048	<code>ER_BAD_NULL_ERROR</code>	Column '%s' cannot be null
1062	<code>ER_DUP_ENTRY</code>	Duplicate entry '%s' for key %d
1242	<code>ER_SUBQUERY_NO_1_ROW</code>	Subquery returns more than 1 row
1264	<code>ER_WARN_DATA_OUT_OF_RANGE</code>	Out of range value for column '%s' at row %ld
1265	<code>WARN_DATA_TRUNCATED</code>	Data truncated for column '%s' at row %ld
1292	<code>ER_TRUNCATED_WRONG_VALUE</code>	Truncated incorrect %s value: '%s'
1366	<code>ER_TRUNCATED_WRONG_VALUE_FOR_FIELD</code>	Incorrect integer value
1369	<code>ER_VIEW_CHECK_FAILED</code>	CHECK OPTION failed '%s.%s'
1451	<code>ER_ROW_IS_REFERENCED_2</code>	Cannot delete or update a parent row
1452	<code>ER_NO_REFERENCED_ROW_2</code>	Cannot add or update a child row: a foreign key constraint fails (%s)
1526	<code>ER_NO_PARTITION_FOR_GIVEN_VALUE</code>	Table has no partition for value %s
1586	<code>ER_DUP_ENTRY_WITH_KEY_NAME</code>	Duplicate entry '%s' for key '%s'
1591	<code>ER_NO_PARTITION_FOR_GIVEN_VALUE_SILENT</code>	Table has no partition for some existing values
1748	<code>ER_ROW_DOES_NOT_MATCH_GIVEN_PARTITION_SET</code>	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.

1.1.2.2 ANALYZE and EXPLAIN Statements

1.1.2.2.1 ANALYZE FORMAT=JSON

`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.

Contents

1. Basic Execution Data
2. Advanced Execution Data
3. Data About Individual Query Plan Nodes
4. Use Cases

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](#).

1.1.2.2 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).

1.1.2.2.3 ANALYZE Statement

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

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](#)

```
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%'  
NG
```

```
***** 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_f_
	1 SIMPLE customer ALL PRIMARY,... NULL NULL NULL 149095 150000 18.08
	1 SIMPLE orders ref i_o_custkey i_o_custkey 5 customer.c_custkey 7 10 100.00

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_f
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 SIMPLE customer ALL PRIMARY,... NULL NULL NULL 149095 150000 18.08
1 SIMPLE orders ref i_o_custkey i_o_custkey 5 customer.c_custkey 7 NULL 100.00

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](#)

1.1.2.2.4 EXPLAIN

Syntax

```
EXPLAIN tbl_name
```

Or

```
EXPLAIN [EXTENDED | PARTITIONS]
 {SELECT select_options | UPDATE update_options | DELETE delete_options}
```

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
<code>id</code>	Sequence number that shows in which order tables are joined.
<code>select_type</code>	What kind of <code>SELECT</code> the table comes from.
<code>table</code>	Alias name of table. Materialized temporary tables for sub queries are named <subquery#>
<code>type</code>	How rows are found from the table (join type).
<code>possible_keys</code>	keys in table that could be used to find rows in the table
<code>key</code>	The name of the key that is used to retrieve rows. <code>NULL</code> is no key was used.
<code>key_len</code>	How many bytes of the key that was used (shows if we are using only parts of the multi-column key).
<code>ref</code>	The reference that is used as the key value.
<code>rows</code>	An estimate of how many rows we will find in the table for each key lookup.
<code>Extra</code>	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
<code>DEPENDENT SUBQUERY</code>	The <code>SUBQUERY</code> is <code>DEPENDENT</code> .	
<code>DEPENDENT UNION</code>	The <code>UNION</code> is <code>DEPENDENT</code> .	
<code>DERIVED</code>	The <code>SELECT</code> is <code>DERIVED</code> from the <code>PRIMARY</code> .	
<code>MATERIALIZED</code>	The <code>SUBQUERY</code> is <code>MATERIALIZED</code> .	Materialized tables will be populated at first access and will be accessed by the primary key (= one key lookup). Number of rows in <code>EXPLAIN</code> shows the cost of populating the table

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](#)

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 <code>SELECT</code> .
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 <code>NULL</code> , we will do a full table scan.
Impossible HAVING	The used <code>HAVING</code> clause is always false so the <code>SELECT</code> will return no rows.
Impossible WHERE noticed after reading const tables.	The used <code>WHERE</code> clause is always false so the <code>SELECT</code> will return no rows. This case was detected after we had read all 'const' tables and used the column values as constant in the <code>WHERE</code> clause. For example: <code>WHERE const_column=5 and const_column</code> had a value of 4.
Impossible WHERE	The used <code>WHERE</code> clause is always false so the <code>SELECT</code> will return no rows. For example: <code>WHERE 1=2</code>
No matching min/max row	During early optimization of <code>MIN()</code> / <code>MAX()</code> values it was detected that no row could match the <code>WHERE</code> clause. The <code>MIN()</code> / <code>MAX()</code> function will return <code>NULL</code> .
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 <code>SELECT</code> was a sub query that did not use any tables. For example a there was no <code>FROM</code> clause or a <code>FROM DUAL</code> clause.
Not exists	Stop searching after more row if we find one single matching row. This optimization is used with <code>LEFT JOIN</code> where one is explicitly searching for rows that doesn't exists in the <code>LEFT JOIN TABLE</code> . Example: <code>SELECT * FROM t1 LEFT JOIN t2 on (...) WHERE t2.not_null_column IS NULL</code> . As <code>t2.not_null_column</code> can only be <code>NULL</code> if there was no matching row for on condition, we can stop searching if we find a single matching row.
Open_frm_only	For <code>information_schema</code> tables. Only the <code>frm</code> (table definition file was opened) was opened for each matching row.
Open_full_table	For <code>information_schema</code> tables. A full table open for each matching row is done to retrieve the requested information. (Slow)
Open_trigger_only	For <code>information_schema</code> 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 GROUP BY or DISTINCT 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 index_merge 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 index_merge 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 GROUP BY, DISTINCT or ORDER BY.
Using where	A WHERE expression (in addition 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 ALL , 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 UPDATE 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
employees_example	0	PRIMARY	1	id	A	7		NULL	NULL
employees_example	0	employee_code	1	employee_code	A	7		NULL	NULL
employees_example	1	first_name	1	first_name	A	NULL		NULL	NULL
employees_example	1	first_name	2	last_name	A	NULL		NULL	NULL

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     | tbl1  | 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](#)

1.1.2.2.5 EXPLAIN ANALYZE

The syntax for the `EXPLAIN ANALYZE` feature was changed to `ANALYZE` statement , available since [MariaDB 10.1.0](#). See [ANALYZE statement](#).

1.1.2.2.6 EXPLAIN FORMAT=JSON

MariaDB starting with [10.1.2](#)

Starting from version 10.1.2, MariaDB supports the `EXPLAIN FORMAT=JSON` syntax.

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\N
```

Contents

1. [Synopsis](#)
2. [Output is different from MySQL](#)
3. [Output format](#)
4. [See also](#)

```
***** 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.

1.1.2.2.7 SHOW EXPLAIN

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   | Extra      |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1    | SIMPLE     | tbl   | index   | NULL        | a     | 5       | NULL   | 1000107 | Using index |
+-----+-----+-----+-----+-----+-----+-----+-----+
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 [lp:1013343](#). [lp: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 ...'`

Contents

1. [Syntax](#)
2. [Description](#)
 1. [Possible Errors](#)
 2. [Differences Between SHOW EXPLAIN and EXPLAIN Outputs](#)
 3. [Background](#)
 4. [List of Recorded Differences](#)
 5. [Required Permissions](#)
3. [See Also](#)

- 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](#).

1.1.2.2.8 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. The algorithm has always been in use, but has only been made visible in the EXPLAIN output since [MariaDB 10.0.5](#).

1.1.2.3 BACKUP Commands

1.1.2.3.1 BACKUP STAGE

MariaDB starting with [10.4.1](#)

The `BACKUP STAGE` commands were introduced in [MariaDB 10.4.1](#).

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 pages 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.

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](#)

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 BACKUP STAGE for safe external backups.

1.1.2.3.2 BACKUP LOCK

MariaDB starting with [10.4.2](#)

The `BUSCUP LOCK` command was introduced in [MariaDB 10.4.2](#).

`BUSCUP 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
```

Contents

1. [Syntax](#)
2. [Usage in a Backup Tool](#)
3. [Privileges](#)
4. [Notes](#)
5. [Implementation](#)
6. [See Also](#)

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

1.1.2.3.3 Mariabackup and BACKUP STAGE Commands

MariaDB starting with [10.4.1](#)

The [BACKUP STAGE](#) commands were introduced in [MariaDB 10.4.1](#).

The [BACKUP 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](#).

Mariabackup and BACKUP STAGE Commands in MariaDB Community Server

MariaDB starting with [10.4.1](#)

In MariaDB Community Server, Mariabackup first supported [BACKUP STAGE](#) commands in [MariaDB 10.4.1](#).

In [MariaDB 10.3](#) and before, the [BACKUP STAGE](#) commands are **not** supported, so Mariabackup executes the [FLUSH TABLES WITH READ LOCK](#) command to lock the database. When the backup is complete, it executes the [UNLOCK TABLES](#) command to unlock the database.

In [MariaDB 10.4](#) and later, the [BACKUP STAGE](#) commands are supported. However, the version of Mariabackup that is bundled with MariaDB Community Server does not yet use the [BACKUP STAGE](#) commands in the most efficient way. Mariabackup simply executes the following [BACKUP](#)

`STAGE` commands to lock the database:

```
BACKUP STAGE START;  
BACKUP STAGE BLOCK_COMMIT;
```

When the backup is complete, it executes the following `BACKUP STAGE` command to unlock the database:

```
BACKUP STAGE END;
```

If you would like to use a version of Mariabackup that uses the `BACKUP STAGE` commands in the most efficient way, then your best option is to use [MariaDB Enterprise Backup](#) that is bundled with [MariaDB Enterprise Server](#).

Tasks Performed Prior to `BACKUP STAGE` in MariaDB Community Server

- Copy some transactional tables.
 - [InnoDB](#) (i.e. `ibdataN` and file extensions `.ibd` and `.isl`)
- Copy the tail of some transaction logs.
 - The tail of the [InnoDB redo log](#) (i.e. `ib_logfileN` files) will be copied for [InnoDB](#) tables.

`BACKUP STAGE START` in MariaDB Community Server

Mariabackup from MariaDB Community Server does not currently perform any tasks in the `START` stage.

`BACKUP STAGE FLUSH` in MariaDB Community Server

Mariabackup from MariaDB Community Server does not currently perform any tasks in the `FLUSH` stage.

`BACKUP STAGE BLOCK_DDL` in MariaDB Community Server

Mariabackup from MariaDB Community Server does not currently perform any tasks in the `BLOCK_DDL` stage.

`BACKUP STAGE BLOCK_COMMIT` in MariaDB Community Server

Mariabackup from MariaDB Community Server performs the following tasks in the `BLOCK_COMMIT` stage:

- Copy other files.
 - i.e. file extensions `.frm`, `.isl`, `.TRG`, `.TRN`, `.opt`, `.par`
- Copy some transactional tables.
 - [Aria](#) (i.e. `aria_log_control` and file extensions `.MAD` and `.MAI`)
- Copy the non-transactional tables.
 - [MyISAM](#) (i.e. file extensions `.MYD` and `.MYI`)
 - [MERGE](#) (i.e. file extensions `.MRG`)
 - [ARCHIVE](#) (i.e. file extensions `.ARM` and `.ARZ`)
 - [CSV](#) (i.e. file extensions `.CSM` and `.CSV`)
- Create a [MyRocks](#) checkpoint using the `rocksdb_create_checkpoint` system variable.
- Copy the tail of some transaction logs.
 - The tail of the [InnoDB redo log](#) (i.e. `ib_logfileN` files) will be copied for [InnoDB](#) tables.
- Save the [binary log](#) position to `xtrabackup_binlog_info`.
- Save the [Galera Cluster](#) state information to `xtrabackup_galera_info`.

`BACKUP STAGE END` in MariaDB Community Server

Mariabackup from MariaDB Community Server performs the following tasks in the `END` stage:

- Copy the [MyRocks](#) checkpoint into the backup.

Contents

1. [Mariabackup and `BACKUP STAGE` Commands in MariaDB Community Server](#)
 1. [Tasks Performed Prior to `BACKUP STAGE` in MariaDB Community Server](#)
 2. [`BACKUP STAGE START` in MariaDB Community Server](#)
 3. [`BACKUP STAGE FLUSH` in MariaDB Community Server](#)
 4. [`BACKUP STAGE BLOCK_DDL` in MariaDB Community Server](#)
 5. [`BACKUP STAGE BLOCK_COMMIT` in MariaDB Community Server](#)
 6. [`BACKUP STAGE END` in MariaDB Community Server](#)
2. [Mariabackup and `BACKUP STAGE` Commands in MariaDB Enterprise Server](#)
 1. [`BACKUP STAGE START` in MariaDB Enterprise Server](#)
 2. [`BACKUP STAGE FLUSH` in MariaDB Enterprise Server](#)
 3. [`BACKUP STAGE BLOCK_DDL` in MariaDB Enterprise Server](#)
 4. [`BACKUP STAGE BLOCK_COMMIT` in MariaDB Enterprise Server](#)
 5. [`BACKUP STAGE END` in MariaDB Enterprise Server](#)

Mariabackup and BACKUP STAGE Commands in MariaDB Enterprise Server

MariaDB starting with 10.2.25

MariaDB Enterprise Backup first supported `BACKUP STAGE` commands in MariaDB Enterprise Server 10.4.6-1, MariaDB Enterprise Server 10.3.16-1, and MariaDB Enterprise Server 10.2.25-1.

The following sections describe how the [MariaDB Enterprise Backup](#) version of Mariabackup that is bundled with [MariaDB Enterprise Server](#) uses each `BACKUP STAGE` command in an efficient way.

BACKUP STAGE START in MariaDB Enterprise Server

Mariabackup from MariaDB Enterprise Server performs the following tasks in the `START` stage:

- Copy all transactional tables.
 - [InnoDB](#) (i.e. `ibdataN` and file extensions `.ibd` and `.isl`)
 - [Aria](#) (i.e. `aria_log_control` and file extensions `.MAD` and `.MAI`)
- Copy the tail of all transaction logs.
 - The tail of the [InnoDB redo log](#) (i.e. `ib_logfileN` files) will be copied for [InnoDB](#) tables.
 - The tail of the Aria redo log (i.e. `aria_log.N` files) will be copied for [Aria](#) tables.

BACKUP STAGE FLUSH in MariaDB Enterprise Server

Mariabackup from MariaDB Enterprise Server performs the following tasks in the `FLUSH` stage:

- Copy all non-transactional tables that are not in use. This list of used tables is found with `SHOW OPEN TABLES`.
 - [MyISAM](#) (i.e. file extensions `.MYD` and `.MYI`)
 - [MERGE](#) (i.e. file extensions `.MRG`)
 - [ARCHIVE](#) (i.e. file extensions `.ARM` and `.ARZ`)
 - [CSV](#) (i.e. file extensions `.CSM` and `.CSV`)
- Copy the tail of all transaction logs.
 - The tail of the [InnoDB redo log](#) (i.e. `ib_logfileN` files) will be copied for [InnoDB](#) tables.
 - The tail of the Aria redo log (i.e. `aria_log.N` files) will be copied for [Aria](#) tables.

BACKUP STAGE BLOCK_DDL in MariaDB Enterprise Server

Mariabackup from MariaDB Enterprise Server performs the following tasks in the `BLOCK_DDL` stage:

- Copy other files.
 - i.e. file extensions `.frm`, `.isl`, `.TRG`, `.TRN`, `.opt`, `.par`
- Copy the non-transactional tables that were in use during `BACKUP STAGE FLUSH`.
 - [MyISAM](#) (i.e. file extensions `.MYD` and `.MYI`)
 - [MERGE](#) (i.e. file extensions `.MRG`)
 - [ARCHIVE](#) (i.e. file extensions `.ARM` and `.ARZ`)
 - [CSV](#) (i.e. file extensions `.CSM` and `.CSV`)
- Check `ddl.log` for DDL executed before the `BLOCK_DDL` stage.
 - The file names of newly created tables can be read from `ddl.log`.
 - The file names of dropped tables can also be read from `ddl.log`.
 - The file names of renamed tables can also be read from `ddl.log`, so the files can be renamed instead of re-copying them.
- Copy changes to system log tables.
 - `mysql.general_log`
 - `mysql.slow_log`
 - This is easy as these are append only.
- Copy the tail of all transaction logs.
 - The tail of the [InnoDB redo log](#) (i.e. `ib_logfileN` files) will be copied for [InnoDB](#) tables.
 - The tail of the Aria redo log (i.e. `aria_log.N` files) will be copied for [Aria](#) tables.

BACKUP STAGE BLOCK_COMMIT in MariaDB Enterprise Server

Mariabackup from MariaDB Enterprise Server performs the following tasks in the `BLOCK_COMMIT` stage:

- Create a [MyRocks](#) checkpoint using the `rocksdb_create_checkpoint` system variable.
- Copy changes to system log tables.
 - `mysql.general_log`
 - `mysql.slow_log`
 - This is easy as these are append only.
- Copy changes to statistics tables.
 - `mysql.table_stats`
 - `mysql.column_stats`
 - `mysql.index_stats`
- Copy the tail of all transaction logs.
 - The tail of the [InnoDB redo log](#) (i.e. `ib_logfileN` files) will be copied for [InnoDB](#) tables.
 - The tail of the Aria redo log (i.e. `aria_log.N` files) will be copied for [Aria](#) tables.
- Save the [binary log](#) position to `xtrabackup_binlog_info`.
- Save the [Galera Cluster](#) state information to `xtrabackup_galera_info`.

BACKUP STAGE END in MariaDB Enterprise Server

Mariabackup from MariaDB Enterprise Server performs the following tasks in the `END` stage:

- Copy the [MyRocks](#) checkpoint into the backup.

1.1.2.3.4 Storage Snapshots and BACKUP STAGE Commands

MariaDB starting with [10.4.1](#)

The `BACKUP STAGE` commands were introduced in [MariaDB 10.4.1](#).

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.

Contents

- [Generic Backup Process with Storage Snapshots](#)

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 "#storage-snapshots-and-backup-stage-commandssql" prefix, as these are files used by concurrent running `ALTER TABLE`. Note that InnoDB will on server restart automatically delete any tables with the "#storage-snapshots-and-backup-stage-commandssql" prefix.

1.1.2.4 FLUSH Commands

1.1.2.4.1 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]
```

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
<code>CHANGED_PAGE_BITMAPS</code>	Internal command used for backup purposes. See the Information Schema CHANGED_PAGE_BITMAPS Table .
<code>CLIENT_STATISTICS</code>	Reset client statistics (see SHOW CLIENT_STATISTICS).
<code>DES_KEY_FILE</code>	Reloads the DES key file (Specified with the --des-key-file startup option).
<code>HOSTS</code>	Flush the hostname cache (used for converting ip to host names and for unblocking blocked hosts. See max_connect_errors)
<code>INDEX_STATISTICS</code>	Reset index statistics (see SHOW INDEX_STATISTICS).
<code>[ERROR ENGINE GENERAL SLOW BINARY RELAY] LOGS</code>	Close and reopen the specified log type, or all log types if none are specified. <code>FLUSH RELAY LOGS [connection-name]</code> can be used to flush the relay logs for a specific connection. Only one connection can be specified per <code>FLUSH</code> command. See Multi-source replication . <code>FLUSH ENGINE LOGS</code> will delete all unneeded Aria redo logs. Since MariaDB 10.1.30 and MariaDB 10.2.11 , <code>FLUSH BINARY LOGS DELETE_DOMAIN_ID= (list-of-domains)</code> 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 .
<code>MASTER</code>	Deprecated option, use RESET MASTER instead.
<code>PRIVILEGES</code>	Reload all privileges from the privilege tables in the <code>mysql</code> database. If the server is started with <code>--skip-grant-table</code> option, this will activate the privilege tables again.

Contents

- [Syntax](#)
- [Description](#)
- [FLUSH STATUS](#)
- [Compatibility with MySQL](#)
 - [Global Status Variables that Support FLUSH STATUS](#)
- [The different usage of FLUSH TABLES](#)
 - [The purpose of FLUSH TABLES](#)
 - [The purpose of FLUSH TABLES WITH READ LOCK](#)
 - [The purpose of FLUSH TABLES table_list](#)
 - [The purpose of FLUSH TABLES table_list WITH READ LOCK](#)
- [Implementation of FLUSH TABLES commands in MariaDB 10.4.8 and above](#)
 - [Implementation of FLUSH TABLES](#)
 - [Implementation of FLUSH TABLES WITH READ LOCK](#)
 - [Implementation of FLUSH TABLES table_list](#)
 - [Implementation of FLUSH TABLES table_list FOR EXPORT](#)
- [FLUSH SSL](#)
- [Reducing Memory Usage](#)

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. Introduced in MariaDB 10.0.8 . 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).

You can also use the [mysqladmin](#) client to flush things. Use `mysqladmin --help` to examine what flush commands it supports.

FLUSH STATUS

[Server status variables](#) can be reset by executing the following:

```
FLUSH STATUS;
```

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 'connection_name';
FLUSH RELAY LOGS FOR CHANNEL 'connection_name';
```

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 seems 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;
```

1.1.2.4.2 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.

1.1.2.4.3 FLUSH TABLES FOR EXPORT

Syntax

```
FLUSH TABLES table_name [, table_name] FOR EXPORT
```

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](#)
- [myisampack](#) - Compressing the MyISAM data file for easier distribution.
- [aria_pack](#) - Compressing the Aria data file for easier distribution

1.1.2.5 Replication Commands

1.1.2.5.1 CHANGE MASTER TO

Contents

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

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,...])
```

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 REPLICIA` on the primary).

For example:

```
STOP SLAVE;
CHANGE MASTER TO
    MASTER_USER='repl',
    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;
```

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_CERT
 - 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

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='rep1',
  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

MariaDB starting with 10.1.2

The `DO_DOMAIN_IDS` option for `CHANGE MASTER` was first added in [MariaDB 10.1.2](#).

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

MariaDB starting with 10.1.2

The `IGNORE_DOMAIN_IDS` option for `CHANGE MASTER` was first added in MariaDB 10.1.2.

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](#)

1.1.2.5.2 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](#) has begun the process of renaming. The documentation process is ongoing. See [MDEV-18777](#) to follow progress on this effort.

Syntax

```
START SLAVE ["start-replicaconnection_name"] [thread_type [, thread_type] ... ] [FOR CHANNEL "start-replicaconnection_name"]
START SLAVE ["start-replicaconnection_name"] [SQL_THREAD] UNTIL
    MASTER_LOG_FILE = 'log_name', MASTER_LOG_POS = log_pos [FOR CHANNEL "start-replicaconnection_name"]
START SLAVE ["start-replicaconnection_name"] [SQL_THREAD] UNTIL
    RELAY_LOG_FILE = 'log_name', RELAY_LOG_POS = log_pos [FOR CHANNEL "start-replicaconnection_name"]
START SLAVE ["start-replicaconnection_name"] [SQL_THREAD] UNTIL
    MASTER_GTID_POS = <GTID position> [FOR CHANNEL "start-replicaconnection_name"]
START ALL SLAVES [thread_type [, thread_type]]

START REPLICA ["start-replicaconnection_name"] [thread_type [, thread_type] ... ] -- from 10.5.1
START REPLICA ["start-replicaconnection_name"] [SQL_THREAD] UNTIL
    MASTER_LOG_FILE = 'log_name', MASTER_LOG_POS = log_pos -- from 10.5.1
START REPLICA ["start-replicaconnection_name"] [SQL_THREAD] UNTIL
    RELAY_LOG_FILE = 'log_name', RELAY_LOG_POS = log_pos -- from 10.5.1
START REPLICA ["start-replicaconnection_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
```

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 REPLICA 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.

Contents

1. [Syntax](#)
2. [Description](#)
 1. [START SLAVE UNTIL](#)
 2. [connection_name](#)
 3. [START ALL SLAVES](#)
 4. [START REPLICA](#)
3. [See Also](#)

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.

1.1.2.5.3 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 ["stop-replicaconnection_name"] [thread_type [, thread_type] ... ] [FOR CHANNEL "stop-replicaconnection_name"]  
STOP ALL SLAVES [thread_type [, thread_type]]  
STOP REPLICA ["stop-replicaconnection_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
```

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](#).

Contents

1. [Syntax](#)
2. [Description](#)
 1. [STOP ALL SLAVES](#)
 2. [connection_name](#)
 3. [STOP REPLICA](#)
3. [See Also](#)

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.

1.1.2.5.4 RESET 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

```
RESET SLAVE ["reset-replicaconnection_name"] [ALL] [FOR CHANNEL "reset-replicaconnection_name"]
```

Description

`RESET SLAVE` makes the slave 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 SLAVE`, the slave replication threads must be stopped (use `STOP SLAVE` if necessary).

Contents

1. [Syntax](#)
2. [Description](#)
 1. [connection_name](#)
 2. [RESET REPLICA](#)
3. [See Also](#)

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 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 slave SQL thread was in the middle of replicating temporary tables when it was stopped, and RESET 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 SLAVES STATUS`.

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 `RESET 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 `RESET SLAVE`.

RESET REPLICA

MariaDB starting with 10.5.1

`RESET REPLICA` is an alias for `RESET SLAVE` from [MariaDB 10.5.1](#).

See Also

- [STOP SLAVE](#) stops the slave, but it can be restarted with [START SLAVE](#) or after next MariaDB server restart.

1.1.2.5.5 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 master. 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` server 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 slave threads are not running. Otherwise, it produces an error.

The statement does not automatically restart the slave 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 slave 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](#)

1.1.2.5.6 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`.

1.1.2.5.7 SHOW SLAVE STATUS

Syntax

```
SHOW SLAVE ["connection_name"] STATUS [FOR CHANNEL "connection_name"]
SHOW REPLICAS ["connection_name"] STATUS -- From MariaDB 10.5.1
```

or

```
SHOW ALL SLAVES STATUS
SHOW ALL REPLICAS STATUS -- From MariaDB 10.5.1
```

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.

Contents

1. [Syntax](#)
2. [Description](#)
 1. [Multi-Source](#)
 2. [Column Descriptions](#)
 3. [SHOW REPLICAS STATUS](#)
3. [Examples](#)
4. [See Also](#)

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
<code>Connection_name</code>	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.	
<code>Slave_SQL_State</code>	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 .	
<code>Slave_IO_State</code>	State of I/O thread. See Slave I/O Thread States .	
<code>Master_host</code>	Master host that the replica is connected to.	
<code>Master_user</code>	Account user name being used to connect to the primary.	
<code>Master_port</code>	The port being used to connect to the primary.	
<code>Connect_Retry</code>	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 <code>replicate_do_db</code> option.
Replicate_Ignore_DB	Databases specified for ignoring with the <code>replicate_ignore_db</code> option.
Replicate_Do_Table	Tables specified for replicating with the <code>replicate_do_table</code> option.
Replicate_Ignore_Table	Tables specified for ignoring with the <code>replicate_ignore_table</code> option.
Replicate_Wild_Do_Table	Tables specified for replicating with the <code>replicate_wild_do_table</code> option.
Replicate_Wild_Ignore_Table	Tables specified for ignoring with the <code>replicate_wild_ignore_table</code> option.
Last_Error	Alias for <code>Last_SQL_Error</code> (see below)
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 <code>sql_slave_skip_counter</code> 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 (<code>Yes</code>), not permitted (<code>No</code>) or permitted but without the replica having SSL support enabled (<code>Ignored</code>)
Master_SSL_CA_File	The <code>MASTER_SSL_CA</code> option of the CHANGE MASTER TO statement.
Master_SSL_CA_Path	The <code>MASTER_SSL_CAPATH</code> option of the CHANGE MASTER TO statement.
Master_SSL_Cert	The <code>MASTER_SSL_CERT</code> option of the CHANGE MASTER TO statement.
Master_SSL_Cipher	The <code>MASTER_SSL_CIPHER</code> option of the CHANGE MASTER TO statement.
Master_SSL_Key	The <code>MASTER_SSL_KEY</code> option of the CHANGE MASTER TO 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. From MariaDB 10.0.23 and MariaDB 10.1.9 , with parallel replication , <code>seconds_behind_master</code> is updated only after transactions commit.
Master_SSL_Verify_Server_Cert	The <code>MASTER_SSL_VERIFY_SERVER_CERT</code> option of the CHANGE MASTER TO statement.
Last_IO_Error	Error code of the most recent error that caused the I/O thread to stop (also recorded in the replica's error log). <code>0</code> means no error. RESET SLAVE or RESET MASTER 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. RESET SLAVE or RESET MASTER will reset this value.
Last_SQL_Error	Error code of the most recent error that caused the SQL thread to stop (also recorded in the replica's error log). <code>0</code> means no error. RESET SLAVE or RESET MASTER 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. RESET SLAVE or RESET MASTER will reset this value.	
Replicate_Ignore_Server_Ids	List of server_ids that are currently being ignored for replication purposes, or an empty string for none, as specified in the IGNORE_SERVER_IDS option of the CHANGE MASTER TO statement.	
Master_Server_Id	The master's server_id value.	
Master_SSL_Crl	The MASTER_SSL_CRL option of the CHANGE MASTER TO statement.	MariaDB 10.0
Master_SSL_Crlpath	The MASTER_SSL_CRLPATH option of the CHANGE MASTER TO statement.	MariaDB 10.0
Using_Gtid	Whether or not global transaction ID's are being used for replication (can be <code>No</code> , <code>Slave_Pos</code> , or <code>Current_Pos</code>).	MariaDB 10.0.2
Gtid_IO_Pos	Current global transaction ID value.	MariaDB 10.0.2
Retried_transactions	Number of retried transactions for this connection. Returned with SHOW ALL SLAVES STATUS only.	
Max_relay_log_size	Max relay log size for this connection. Returned with SHOW ALL SLAVES STATUS only.	
Executed_log_entries	How many log entries the replica has executed. Returned with SHOW ALL SLAVES STATUS only.	
Slave_received_heartbeats	How many heartbeats we have got from the master. Returned with SHOW ALL SLAVES STATUS only.	
Slave_heartbeat_period	How often to request a heartbeat packet from the master (in seconds). Returned with SHOW ALL SLAVES STATUS 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 SHOW ALL SLAVES STATUS only.	
SQL_Delay	Value specified by MASTER_DELAY in CHANGE MASTER (or 0 if none).	MariaDB 10.2.3
SQL_Remaining_Delay	When the replica is delaying the execution of an event due to MASTER_DELAY , 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 SHOW PROCESSLIST . When the replica is delaying the execution of an event due to MASTER_DELAY , this field displays: "Waiting until MASTER_DELAY 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
    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_Error:
    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:
    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_Error:
Replicate_Ignore_Server_Ids:
    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](#)

1.1.2.5.8 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](#)

1.1.2.5.9 SHOW SLAVE HOSTS

Syntax

```
SHOW SLAVE HOSTS  
SHOW REPLICA HOSTS -- from MariaDB 10.5.1
```

Contents

1. [Syntax](#)
2. [Description](#)
 1. [SHOW REPLICA HOSTS](#)
3. [See Also](#)

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 (\geq [MariaDB 10.5.2](#)) or the [REPLICATION SLAVE](#) privilege (\leq [MariaDB 10.5.1](#)).

SHOW REPLICA HOSTS

MariaDB starting with 10.5.1

SHOW REPLICA 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

1.1.2.5.10 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.

1.1.2.6 Plugin SQL Statements

1.1.2.6.1 SHOW PLUGINS

Syntax

```
SHOW PLUGINS;
```

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.

Contents

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

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](#)

1.1.2.6.2 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.

1.1.2.6.3 INSTALL PLUGIN

Syntax

```
INSTALL PLUGIN [IF NOT EXISTS] plugin_name SONAME 'plugin_library'
```

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.

Contents

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

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.

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](#)

1.1.2.6.4 UNINSTALL PLUGIN

Syntax

```
UNINSTALL PLUGIN [IF EXISTS] plugin_name
```

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.

Contents

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

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](#)

1.1.2.6.5 INSTALL SONAME

Syntax

```
INSTALL SONAME 'plugin_library'
```

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:

Contents

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

```
[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 mysqld` 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](#)

1.1.2.6.6 UNINSTALL SONAME

Syntax

```
UNINSTALL SONAME [IF EXISTS] 'plugin_library'
```

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.

Contents

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

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](#)

1.1.2.6.7 mysql_plugin

MariaDB starting with [10.4.6](#)

From [MariaDB 10.4.6](#), `mariadb-plugin` is a symlink to `mysql_plugin`.

MariaDB starting with [10.5.2](#)

From [MariaDB 10.5.2](#), `mysql_plugin` is the symlink, and `mariadb-plugin` the binary name.

`mysql_plugin` is a tool for enabling or disabling [plugins](#).

It is a commandline alternative to the [INSTALL PLUGIN](#) and [UNINSTALL PLUGIN](#) statements, and the `--plugin-load` option to `mysqld`.

`mysql_plugin` must be run while the server is offline, and works by adding or removing rows from the `mysql.plugin` table.

Contents

1. [Usage](#)
2. [Options](#)
3. [See Also](#)

Usage

```
mysql_plugin [options] <plugin> ENABLE|DISABLE
```

`mysql_plugin` expects to find a configuration file that indicates how to configure the plugins. The configuration file is by default the same name as the plugin, with a `.ini` extension. For example:

```
mysql_plugin crazyplugins ENABLE
```

Here, `mysql_plugin` will look for a file called `crazyplugins.ini`

```
crazyplugins
crazyplugin1
crazyplugin2
crazyplugin3
```

The first line should contain the name of the library object file, with no extension. The other lines list the names of the components. Each value should be on a separate line, and the `#` character at the start of the line indicates a comment.

Options

The following options can be specified on the command line, while some can be specified in the `[mysqld]` group of any option file. For options specified in a `[mysqld]` group, only the `--basedir`, `--datadir`, and `--plugin-dir` options can be used - the rest are ignored.

Option	Description
<code>-b</code> , <code>--basedir=name</code>	The base directory for the server.
<code>-d</code> , <code>--datadir=name</code>	The data directory for the server.
<code>-?</code> , <code>--help</code>	Display help and exit.
<code>-f</code> , <code>--my-print-defaults=name</code>	Path to <code>my_print_defaults</code> executable. Example: <code>/source/temp11/extra</code>
<code>-m</code> , <code>--mysqld=name</code>	Path to <code>mysqld</code> executable. Example: <code>/sbin/temp1/mysql/bin</code>
<code>-n</code> , <code>--no-defaults</code>	Do not read values from configuration file.
<code>-p</code> , <code>--plugin-dir=name</code>	The plugin directory for the server.
<code>-i</code> , <code>--plugin-ini=name</code>	Read plugin information from configuration file specified instead of from <code><plugin-dir>/<plugin_name>.ini</code> .
<code>-P</code> , <code>--print-defaults</code>	Show default values from configuration file.
<code>-v</code> , <code>--verbose</code>	More verbose output; you can use this multiple times to get even more verbose output.
<code>-V</code> , <code>--version</code>	Output version information and exit.

See Also

- [List of Plugins](#)
- [Plugin Overview](#)
- [INFORMATION_SCHEMA.PLUGINS Table](#)
- [INSTALL PLUGIN](#)
- [INSTALL SONAME](#)
- [UNINSTALL PLUGIN](#)
- [UNINSTALL SONAME](#)

1.1.2.7 SET Commands

1.1.2.7.1 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
```

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 (from [MariaDB 10.1.2](#)), 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:

Contents

1. [Syntax](#)
2. [Description](#)
 1. [GLOBAL / SESSION](#)
 2. [DEFAULT](#)
3. [Examples](#)
4. [See Also](#)

```

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](#)

1.1.2.7.2 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](#)

1.1.2.7.3 SET GLOBAL SQL_SLAVE_SKIP_COUNTER

Syntax

```
SET GLOBAL sql_slave_skip_counter = N
```

Description

This statement skips the next *N* events from the master. 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` server 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 slave threads are not running. Otherwise, it produces an error.

Contents

1. [Syntax](#)
2. [Description](#)
3. [Example](#)
4. [Multiple Replication Domains](#)
5. [See Also](#)

The statement does not automatically restart the slave 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 slave 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](#)

1.1.2.7.4 SET NAMES

Syntax

```
SET NAMES {'charset_name'  
[COLLATE 'collation_name'] | DEFAULT}
```

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.

Contents

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

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](#)

1.1.2.7.5 SET PASSWORD

Syntax

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

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 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.

Contents

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

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](#)

1.1.2.7.6 SET ROLE

Syntax

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

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](#).

Contents

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

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          |
+-----+

```

1.1.2.7.7 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 has now been disabled as this was too dangerous as it could damage replication.

1.1.2.7.8 SET STATEMENT

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.

Contents

1. [Syntax](#)
2. [Description](#)
3. [Examples](#)
4. [Limitations](#)
5. [Source](#)

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](#).

1.1.2.7.9 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
```

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

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](#)

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

These clauses appeared in [MariaDB 10.0](#).

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 progress
```

1.1.2.7.10 SET Variable

Syntax

```
SET var_name = expr [, var_name = expr] ...
```

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;
```

Contents

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

See Also

- [SET](#)
- [SET STATEMENT](#)
- [DECLARE Variable](#)

1.1.2.8 SHOW

1.1.2.8.1 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](#).

1.1.2.8.2 Extended Show

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

Contents

1. Examples

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

1.1.2.8.3 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;		
Name	Location	Comment
Michael (Monty) Widenius	Tusby, Finland	Lead developer and main author
Sergei Golubchik	Kerpen, Germany	Architect, Full-text search, precision math, plugin frame
Igor Babaev	Bellevue, USA	Optimizer, keycache, core work
Sergey Petrunia	St. Petersburg, Russia	Optimizer
Oleksandr Byelkin	Lugansk, Ukraine	Query Cache (4.0), Subqueries (4.1), Views (5.0)
Timour Katchaounov	Sofia , Bulgaria	Optimizer
Kristian Nielsen	Copenhagen, Denmark	Replication, Async client protocol, General buildbot str
Alexander (Bar) Barkov	Izhevsk, Russia	Unicode and character sets
Alexey Botchkov (Holyfoot)	Izhevsk, Russia	GIS extensions, embedded server, precision math
Daniel Bartholomew	Raleigh, USA	MariaDB documentation
Colin Charles	Selangor, Malesia	MariaDB documentation, talks at a LOT of conferences
Sergey Vojtovich	Izhevsk, Russia	initial implementation of plugin architecture, maintained
Vladislav Vaintroub	Mannheim, Germany	MariaDB Java connector, new thread pool, Windows optimiza
Elena Stepanova	Sankt Petersburg, Russia	QA, test cases
Georg Richter	Heidelberg, Germany	New LGPL C connector, PHP connector
Jan Lindström	Ylämylly, Finland	Working on InnoDB
Lixun Peng	Hangzhou, China	Multi Source replication
Percona	CA, USA	XtraDB, microslow patches, extensions to slow log
...		

See Also

- [SHOW CONTRIBUTORS](#). This list all members and sponsors of the MariaDB Foundation and other sponsors.

1.1.2.8.4 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 |
+-----+-----+
```

1.1.2.8.5 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)
```

1.1.2.8.6 SHOW CHARACTER SET

Syntax

```
SHOW CHARACTER SET  
[LIKE 'pattern' | WHERE expr]
```

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](#)

1.1.2.8.7 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.

Contents

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

Example

```
SHOW CLIENT_STATISTICS\nG
*****
1. row ****
Client: localhost
Total_connections: 35
Concurrent_connections: 0
Connected_time: 708
Busy_time: 2.555797999999985
Cpu_time: 0.04123740000000002
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
```

1.1.2.8.8

1.1.2.8.9 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	https://www.alibabacloud.com/	Platinum Sponsor of the MariaDB Foundation
Tencent Cloud	https://cloud.tencent.com	Platinum Sponsor of the MariaDB Foundation
Microsoft	https://microsoft.com/	Platinum Sponsor of the MariaDB Foundation
MariaDB Corporation	https://mariadb.com	Founding member, Platinum Sponsor of the MariaDB Foundation
Visma	https://visma.com	Gold Sponsor of the MariaDB Foundation
DBS	https://dbs.com	Gold Sponsor of the MariaDB Foundation
IBM	https://www.ibm.com	Gold Sponsor of the MariaDB Foundation
Tencent Games	http://game.qq.com/	Gold Sponsor of the MariaDB Foundation
Nexedi	https://www.nexedi.com	Silver Sponsor of the MariaDB Foundation
Acronis	https://www.acronis.com	Silver Sponsor of the MariaDB Foundation
Verkkokauppa.com	https://www.verkkokauppa.com	Bronze Sponsor of the MariaDB Foundation
Virtuozzo	https://virtuozzo.com	Bronze Sponsor of the MariaDB Foundation
Tencent Game DBA	http://tencentdba.com/about	Bronze Sponsor of the MariaDB Foundation
Tencent TDSQL	http://tdsql.org	Bronze Sponsor of the MariaDB Foundation
Percona	https://www.percona.com/	Bronze Sponsor of the MariaDB Foundation
Google	USA	Sponsoring encryption, parallel replication and GTID
Facebook	USA	Sponsoring non-blocking API, LIMIT ROWS EXAMINED etc
Ronald Bradford	Brisbane, Australia	EFF contribution for UC2006 Auction
Sheeri Kritzer	Boston, Mass. USA	EFF contribution for UC2006 Auction
Mark Shuttleworth	London, UK.	EFF contribution for UC2006 Auction

See Also

- [Log of MariaDB contributors](#).
- [SHOW AUTHORS](#) list the authors of MariaDB (including documentation, QA etc).
- [MariaDB Foundation page on contributing financially](#)

1.1.2.8.10 SHOW CREATE DATABASE

Syntax

```
SHOW CREATE {DATABASE | SCHEMA} db_name
```

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.

Contents

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

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 'presentations' |
+-----+
```

See Also

- [CREATE DATABASE](#)
- [ALTER DATABASE](#)
- [Character Sets and Collations](#)

1.1.2.8.11 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 unreliablely 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](#)

1.1.2.8.12 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 unreliablely affected by the [sql_quote_show_create](#) server system variable - see <http://bugs.mysql.com/bug.php?id=12719>

Example

```
MariaDB [test]> SHOW CREATE FUNCTION VatCents\G
***** 1. row *****
    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](#)

1.1.2.8.13 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
```

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
*****
1. row ****
  Package: employee_tools
  sql_mode: PIPES_AS_CONCAT,ANSI_QUOTES,IGNORE_SPACE,ORACLE,NO_KEY_OPTIONS,NO_T
  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
```

Contents

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

See Also

- [CREATE PACKAGE](#)
- [DROP PACKAGE](#)
- [CREATE PACKAGE BODY](#)
- [SHOW CREATE PACKAGE BODY](#)
- [DROP PACKAGE BODY](#)
- [Oracle SQL_MODE](#)

1.1.2.8.14 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
```

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).

Contents

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

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_OPTIONS,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](#)

1.1.2.8.15 SHOW CREATE PROCEDURE

Syntax

```
SHOW CREATE PROCEDURE proc_name
```

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 unreliable 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 'test'.prc1'
```

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](#)

Contents

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

- Information Schema ROUTINES Table

1.1.2.8.16 SHOW CREATE SEQUENCE

MariaDB starting with 10.3.1

Sequences were introduced in MariaDB 10.3.

Syntax

```
SHOW CREATE SEQUENCE sequence_name;
```

Description

Shows the [CREATE SEQUENCE](#) statement that created the given sequence. The statement requires the `SELECT` privilege for the table.

Example

Contents

- [Syntax](#)
- [Description](#)
- [Example](#)
- [Notes](#)
- [See Also](#)

```
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](#)

1.1.2.8.17

1.1.2.8.18 SHOW CREATE TRIGGER

Syntax

```
SHOW CREATE TRIGGER trigger_name
```

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>

Contents

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

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](#)

1.1.2.8.19

1.1.2.8.20 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 SECU
RITY DEFINER VIEW example AS (select t.id AS id,t.s AS s from t)
character_set_client: cp850
collation_connection: cp850_general_ci
```

1.1.2.8.21 SHOW DATABASES

Syntax

```
SHOW {DATABASES | SCHEMAS}
      [LIKE 'pattern' | WHERE expr]
```

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.

Contents

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

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](#)

1.1.2.8.22 SHOW ENGINE

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.

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](#)

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 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](#)

1.1.2.8.23 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
=====
```

1.1.2.8.24 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)

```

1.1.2.8.25 SHOW ERRORS

Syntax

```

SHOW ERRORS [LIMIT [offset,] row_count]
SHOW ERRORS [LIMIT row_count OFFSET offset]
SHOW COUNT(*) ERRORS

```

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](#).

Contents

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

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 |
+-----+

```

1.1.2.8.26 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](#).

1.1.2.8.27 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
```

[1.1.2.8.28](#)

[1.1.2.8.29](#)

[**1.1.2.8.30 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 |
+-----+-----+-----+
```

1.1.2.8.32 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                |
...
```

1.1.2.8.33

1.1.2.8.34 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    | jauthor     |     0 |      0 |
| test    | locks        |     1 |      0 |
...
+-----+-----+-----+
```

1.1.2.8.35 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]
```

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.

Contents

- [Syntax](#)
- [Description](#)
- [Examples](#)
- [See Also](#)

Examples

```
SHOW PACKAGE BODY STATUS LIKE 'pkg1'\G
*****
1. row *****
    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](#)

1.1.2.8.36 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]
```

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.

Contents

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

Examples

```
SHOW PACKAGE STATUS LIKE 'pkg1'\G
*****
1. row *****
    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](#)

[1.1.2.8.37](#)

[1.1.2.8.38](#)

[1.1.2.8.39 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](#)

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	Tables	To delete existing rows
Delete history	Tables	To delete versioning table historical rows
Drop	Databases,Tables	To drop databases, tables, and views
Event	Server Admin	To create, alter, drop and execute events
Execute	Functions,Procedures	To execute stored routines
File	File access on server	To read and write files on the server
Grant option	Databases,Tables,Functions,Procedures	To give to other users those privileges you possess
Index	Tables	To create or drop indexes
Insert	Tables	To insert data into tables
Lock tables	Databases	To use LOCK TABLES (together with SELECT privilege)
Process	Server Admin	To view the plain text of currently executing queries
Proxy	Server Admin	To make proxy user possible
References	Databases,Tables	To have references on tables
Reload	Server Admin	To reload or refresh tables, logs and privileges
Binlog admin	Server	To purge binary logs
Binlog monitor	Server	To use SHOW BINLOG STATUS and SHOW BINARY LOG
Binlog replay	Server	To use BINLOG (generated by mariadb-binlog)
Replication master admin	Server	To monitor connected slaves
Replication slave admin	Server	To start/stop slave and apply binlog events
Slave monitor	Server	To use SHOW SLAVE STATUS and SHOW RELAYLOG EVENTS
Replication slave	Server Admin	To read binary log events from the master
Select	Tables	To retrieve rows from table
Show databases	Server Admin	To see all databases with SHOW DATABASES
Show view	Tables	To see views with SHOW CREATE VIEW
Shutdown	Server Admin	To shut down the server
Super	Server Admin	To use KILL thread, SET GLOBAL, CHANGE MASTER, etc.
Trigger	Tables	To use triggers
Create tablespace	Server Admin	To create/alter/drop tablespaces
Update	Tables	To update existing rows
Set user	Server	To create views and stored routines with a different definer
Federated admin	Server	To execute the CREATE SERVER, ALTER SERVER, DROP SERVER statement
Connection admin	Server	To bypass connection limits and kill other users' connections
Read_only admin	Server	To perform write operations even if @@read_only=ON
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.

1.1.2.8.40 SHOW PROCEDURE CODE

Syntax

```
SHOW PROCEDURE CODE proc_name
```

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.

Contents

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

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](#)

1.1.2.8.41 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
*****
1. row *****
    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](#)

1.1.2.8.42 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, since [MariaDB 10.0.5](#), 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\(\)](#)

1.1.2.8.43 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`.

1.1.2.8.44 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](#).

1.1.2.8.45

1.1.2.8.46

1.1.2.8.47

1.1.2.8.48 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_pagedelete_blocks_not_flushed	0
Aria_pagedelete_blocks_unused	15706
Aria_pagedelete_blocks_used	0
Aria_pagedelete_read_requests	0
Aria_pagedelete_reads	0
Aria_pagedelete_write_requests	0
Aria_pagedelete_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
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_purge_table	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	Dumping buffer pool(s) not yet started
Innodb_buffer_pool_load_status	Loading buffer pool(s) not yet started

Innodb_buffer_pool_status	Loading buffer pool(s) not yet started
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_written	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_statement_time_executed	0
Max_used_connections	1
Memory_used	273614696
Not_flushed_delayed_rows	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)

```

1.1.2.8.49 SHOW TABLE STATUS

Syntax

```

SHOW TABLE STATUS [{FROM | IN} db_name]
[LIKE 'pattern' | WHERE expr]

```

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:

Contents

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

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
```

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:
```

1.1.2.8.50 SHOW TABLES

Syntax

```
SHOW [FULL] TABLES [FROM db_name]
[LIKE 'pattern' | WHERE expr]
```

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.

Contents

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

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     |
+-----+
```

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

1.1.2.8.51 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
*****
***** 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

```

1.1.2.8.52 SHOW TRIGGERS

Syntax

```

SHOW TRIGGERS [FROM db_name]
[LIKE 'pattern' | WHERE expr]

```

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.

Contents

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

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

1.1.2.8.53 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
*****
***** 1. row *****
    User: root
    Total_connections: 1
    Concurrent_connections: 0
        Connected_time: 3297
        Busy_time: 0.1411340000000006
        Cpu_time: 0.0176370000000003
    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
```

1.1.2.8.54 SHOW VARIABLES

Syntax

```
SHOW [GLOBAL | SESSION] VARIABLES
[LIKE 'pattern' | WHERE expr]
```

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

Contents

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

```

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 |
+-----+

```

1.1.2.8.55 SHOW WARNINGS

Syntax

```

SHOW WARNINGS [LIMIT [offset,] row_count]
SHOW ERRORS [LIMIT row_count OFFSET offset]
SHOW COUNT(*) WARNINGS

```

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.

Contents

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

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](#)

1.1.2.8.56 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 |
+-----+-----+-----+

```

1.1.2.8.57 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 |
+-----+-----+-----+
```

1.1.2.9 System Tables

1.1.2.9.1 Information Schema

1.1.2.9.1.1 Information Schema Tables

1.1.2.9.1.1.1 Information Schema InnoDB Tables

1.1.2.9.1.1.1.1 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
POOL_ID	Buffer Pool identifier. From MariaDB 10.5.1 returns a value of 0, since multiple InnoDB buffer pool instances has been removed.
BLOCK_ID	Buffer Pool Block identifier.
SPACE	Tablespace identifier. Matches the <code>SPACE</code> value in 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 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
...

```

1.1.2.9.1.1.1.2 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

```
DESC information_schema.innodb_buffer_page_lru;
+-----+-----+-----+-----+-----+
| 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
...
```

1.1.2.9.1.1.1.3 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)

1.1.2.9.1.1.1.4 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)

1.1.2.9.1.1.1.5 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	1 if the page has been modified since being loaded, 0 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 old 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)

1.1.2.9.1.1.6 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 has 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 old 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 old sublist to the new sublist.
PAGES_NOT_MADE_YOUNG	Pages that have remained in the old sublist without moving to the new sublist.
PAGES_MADE_YOUNG_RATE	Hits that cause blocks to move to the top of the new sublist.
PAGES_MADE_NOT_YOUNG_RATE	Hits that do not cause blocks to move to the top of the new sublist due to the <code>innodb_old_blocks</code> 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	

1.1.2.9.1.1.1.7 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)

1.1.2.9.1.1.1.8 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
*****
 1. row ****
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](#)

1.1.2.9.1.1.1.9 Information Schema INNODB_CMPPMEM

and INNODB_CMPMEM_RESET Tables

The `INNODB_CMPMEM` and `INNODB_CMPMEM_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
<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>BUFFER_POOL_INSTANCE</code>	Buffer Pool identifier. From MariaDB 10.5.1 returns a value of 0, since multiple InnoDB buffer pool instances has been removed.
<code>PAGES_USED</code>	Number of pages of the size <code>PAGE_SIZE</code> which are currently in the buffer pool.
<code>PAGES_FREE</code>	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.
<code>RELOCATION_OPS</code>	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).
<code>RELOCATION_TIME</code>	Time (in seconds) spent in relocation operations for pages of the size <code>PAGE_SIZE</code> . This column is reset when the <code>INNODB_CMPMEM_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_CMPMEM` and `INNODB_CMPMEM_RESET` have the same columns and always contain the same values, but when `INNODB_CMPMEM_RESET` is queried, the `RELOCATION_TIME` column from both the tables are cleared. `INNODB_CMPMEM_RESET` can be used, for example, if a script periodically logs the performances of compression in the last period of time. `INNODB_CMPMEM` can be used to see the cumulated statistics.

Example

```
SELECT * FROM information_schema.INNODB_CMPMEM\G
*****
1. row *****
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`

1.1.2.9.1.1.10 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
DATABASE_NAME	Database containing the index.
TABLE_NAME	Table containing the index.
INDEX_NAME	Other values are totals which refer to this index's compression.
COMPRESS_OPS	How many times a page of INDEX_NAME 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> .
COMPRESS_OPS_OK	How many times a page of INDEX_NAME has been successfully compressed. This value should be as close as possible to COMPRESS_OPS . If it is notably lower, either avoid compressing some tables, or increase the KEY_BLOCK_SIZE for some compressed tables.
COMPRESS_TIME	Time (in seconds) spent to compress pages of the size PAGE_SIZE . This value includes time spent in <i>compression failures</i> .
UNCOMPRESS_OPS	How many times a page of INDEX_NAME 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.
UNCOMPRESS_TIME	Time (in seconds) spent to uncompress pages of INDEX_NAME .

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`

1.1.2.9.1.1.11 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
DOC_ID	Document ID of the row being deleted. Either an underlying ID value, or a sequence value generated by InnoDB if no usable option exists.

1.1.2.9.1.1.12 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
KEY	Metadata item name.
VALUE	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      |
+-----+-----+
```

1.1.2.9.1.1.13 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
```

1.1.2.9.1.1.14 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 |
+-----+
```

1.1.2.9.1.1.15 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
WORD	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.
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_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)
```

1.1.2.9.1.1.16 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 |
+-----+-----+-----+-----+-----+

```

1.1.2.9.1.1.1.17 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.

1.1.2.9.1.1.1.18 Information Schema INNODB_LOCKS Table

The [Information Schema](#) `INNODB_LOCKS` 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.
<code>LOCK_TRX_ID</code>	Lock's transaction ID. Matches the <code>INNODB_TRX.TRX_ID</code> 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 l.* , 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
```

1.1.2.9.1.1.19 Information Schema INNODB_METRICS Table

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:

Contents

1. [Enabling and Disabling Counters](#)
2. [Resetting Counters](#)
3. [Simplifying from MariaDB 10.6](#)
4. [Examples](#)

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 <code>innodb_monitor_enable</code> and <code>innodb_monitor_disable</code> 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 or 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 <code>innodb_adaptive_hash_index</code> is disabled (the default), <code>adaptive_hash_index</code> 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`

The intention is to eventually remove the interface entirely (see [MDEV-15706](#)).

Examples

Until MariaDB 10.5:

name	subsystem	type	comment
metadata_table_handles_opened	metadata	counter	Number of table handles opened
metadata_table_handles_closed	metadata	counter	Number of table handles closed
metadata_table_reference_count	metadata	counter	Table reference counter
lock_deadlocks	lock	counter	Number of deadlocks
lock_timeouts	lock	counter	Number of lock timeouts
lock_rec_lock_waits	lock	counter	Number of times enqueued into record lock wait
lock_table_lock_waits	lock	counter	Number of times enqueued into table lock wait
lock_rec_lock_requests	lock	counter	Number of record locks requested
lock_rec_lock_created	lock	counter	Number of record locks created
lock_rec_lock_removed	lock	counter	Number of record locks removed from the lock queue
lock_rec_locks	lock	counter	Current number of record locks on tables
lock_table_lock_created	lock	counter	Number of table locks created
lock_table_lock_removed	lock	counter	Number of table locks removed from the lock queue
lock_table_locks	lock	counter	Current number of table locks on tables
lock_row_lock_current_waits	lock	status_counter	Number of row locks currently being waited for
lock_row_lock_time	lock	status_counter	Time spent in acquiring row locks, in milliseconds
lock_row_lock_time_max	lock	value	The maximum time to acquire a row lock, in milliseconds
lock_row_lock_waits	lock	status_counter	Number of times a row lock had to be waited for
lock_row_lock_time_avg	lock	value	The average time to acquire a row lock, in milliseconds
buffer_pool_size	server	value	Server buffer pool size (all buffer pools) in pages
buffer_pool_reads	buffer	status_counter	Number of reads directly from disk (innodb_buffer_pool)
buffer_pool_read_requests	buffer	status_counter	Number of logical read requests (innodb_buffer_pool)
buffer_pool_write_requests	buffer	status_counter	Number of write requests (innodb_buffer_pool_writes)
buffer_pool_wait_free	buffer	status_counter	Number of times waited for free buffer (innodb_buffer_pool)
buffer_pool_read_ahead	buffer	status_counter	Number of pages read as read ahead (innodb_buffer_pool)
buffer_pool_read_ahead_evicted	buffer	status_counter	Read-ahead pages evicted without being accessed
buffer_pool_pages_total	buffer	value	Total buffer pool size in pages (innodb_buffer_pool)
buffer_pool_pages_misc	buffer	value	Buffer pages for misc use such as row locks or
buffer_pool_pages_data	buffer	value	Buffer pages containing data (innodb_buffer_pool)
buffer_pool_bytes_data	buffer	value	Buffer bytes containing data (innodb_buffer_pool)
buffer_pool_pages_dirty	buffer	value	Buffer pages currently dirty (innodb_buffer_pool)
buffer_pool_bytes_dirty	buffer	value	Buffer bytes currently dirty (innodb_buffer_pool)
buffer_pool_pages_free	buffer	value	Buffer pages currently free (innodb_buffer_pool)
buffer_pages_created	buffer	status_counter	Number of pages created (innodb_pages_created)
buffer_pages_written	buffer	status_counter	Number of pages written (innodb_pages_written)
buffer_index_pages_written	buffer	status_counter	Number of index pages written (innodb_index_page)
buffer_non_index_pages_written	buffer	status_counter	Number of non index pages written (innodb_non_index_page)
buffer_pages_read	buffer	status_counter	Number of pages read (innodb_pages_read)
buffer_index_sec_rec_cluster_reads	buffer	status_counter	Number of secondary record reads triggered by cluster
buffer_index_sec_rec_cluster_reads_avoided	buffer	status_counter	Number of secondary record reads avoided triggered by cluster
buffer_data_reads	buffer	status_counter	Amount of data read in bytes (innodb_data_reads)
buffer_data_written	buffer	status_counter	Amount of data written in bytes (innodb_data_written)
buffer_flush_batch_scanned	buffer	set_owner	Total pages scanned as part of flush batch
buffer_flush_batch_num_scan	buffer	set_member	Number of times buffer flush list flush is called
buffer_flush_batch_scanned_per_call	buffer	set_member	Pages scanned per flush batch scan
buffer_flush_batch_total_pages	buffer	set_owner	Total pages flushed as part of flush batch
buffer_flush_batches	buffer	set_member	Number of flush batches
buffer_flush_batch_pages	buffer	set_member	Pages queued as a flush batch
buffer_flush_neighbor_total_pages	buffer	set_owner	Total neighbors flushed as part of neighbor flushing
buffer_flush_neighbor	buffer	set_member	Number of times neighbors flushing is invoked
buffer_flush_neighbor_pages	buffer	set_member	Pages queued as a neighbor batch
buffer_flush_n_to_flush_requested	buffer	counter	Number of pages requested for flushing.
buffer_flush_n_to_flush_by_age	buffer	counter	Number of pages target by LSN Age for flushing
buffer_flush_adaptive_avg_time	buffer	counter	Avg time (ms) spent for adaptive flushing received
buffer_flush_adaptive_avg_pass	buffer	counter	Number of adaptive flushes passed during the round trip
buffer_LRU_get_free_loops	buffer	counter	Total loops in LRU get free.
buffer_LRU_get_free_waits	buffer	counter	Total sleep waits in LRU get free.
buffer_flush_avg_page_rate	buffer	counter	Average number of pages at which flushing is happening
buffer_flush_lsn_avg_rate	buffer	counter	Average redo generation rate

buffer_flush_pct_for_dirty	buffer	counter	Percent of IO capacity used to avoid max dirty
buffer_flush_pct_for_lsn	buffer	counter	Percent of IO capacity used to avoid reusable
buffer_flush_sync_waits	buffer	counter	Number of times a wait happens due to sync flu
buffer_flush_adaptive_total_pages	buffer	set_owner	Total pages flushed as part of adaptive flushi
buffer_flush_adaptive	buffer	set_member	Number of adaptive batches
buffer_flush_adaptive_pages	buffer	set_member	Pages queued as an adaptive batch
buffer_flush_sync_total_pages	buffer	set_member	Total pages flushed as part of sync batches
buffer_flush_sync	buffer	set_member	Number of sync batches
buffer_flush_sync_pages	buffer	set_member	Pages queued as a sync batch
buffer_flush_background_total_pages	buffer	set_member	Total pages flushed as part of background batcl
buffer_flush_background	buffer	set_member	Number of background batches
buffer_flush_background_pages	buffer	set_member	Pages queued as a background batch
buffer_LRU_batch_scanned	buffer	set_owner	Total pages scanned as part of LRU batch
buffer_LRU_batch_num_scan	buffer	set_member	Number of times LRU batch is called
buffer_LRU_batch_scanned_per_call	buffer	set_member	Pages scanned per LRU batch call
buffer_LRU_batch_flush_total_pages	buffer	set_owner	Total pages flushed as part of LRU batches
buffer_LRU_batches_flush	buffer	set_member	Number of LRU batches
buffer_LRU_batch_flush_pages	buffer	set_member	Pages queued as an LRU batch
buffer_LRU_batch_evict_total_pages	buffer	set_owner	Total pages evicted as part of LRU batches
buffer_LRU_batches_evict	buffer	set_member	Number of LRU batches
buffer_LRU_batch_evict_pages	buffer	set_member	Pages queued as an LRU batch
buffer_LRU_single_flush_failure_count	Buffer	counter	Number of times attempt to flush a single page
buffer_LRU_get_free_search	Buffer	counter	Number of searches performed for a clean page
buffer_LRU_search_scanned	buffer	set_owner	Total pages scanned as part of LRU search
buffer_LRU_search_num_scan	buffer	set_member	Number of times LRU search is performed
buffer_LRU_search_scanned_per_call	buffer	set_member	Page scanned per single LRU search
buffer_LRU_unzip_search_scanned	buffer	set_owner	Total pages scanned as part of LRU unzip search
buffer_LRU_unzip_search_num_scan	buffer	set_member	Number of times LRU unzip search is performed
buffer_LRU_unzip_search_scanned_per_call	buffer	set_member	Page scanned per single LRU unzip search
buffer_page_read_index_leaf	buffer_page_io	counter	Number of Index Leaf Pages read
buffer_page_read_index_non_leaf	buffer_page_io	counter	Number of Index Non-leaf Pages read
buffer_page_read_index_ibuf_leaf	buffer_page_io	counter	Number of Insert Buffer Index Leaf Pages read
buffer_page_read_index_ibuf_non_leaf	buffer_page_io	counter	Number of Insert Buffer Index Non-Leaf Pages re
buffer_page_read_undo_log	buffer_page_io	counter	Number of Undo Log Pages read
buffer_page_read_index_inode	buffer_page_io	counter	Number of Index Inode Pages read
buffer_page_read_ibuf_free_list	buffer_page_io	counter	Number of Insert Buffer Free List Pages read
buffer_page_read_ibuf_bitmap	buffer_page_io	counter	Number of Insert Buffer Bitmap Pages read
buffer_page_read_system_page	buffer_page_io	counter	Number of System Pages read
buffer_page_read_trx_system	buffer_page_io	counter	Number of Transaction System Pages read
buffer_page_read_fsp_hdr	buffer_page_io	counter	Number of File Space Header Pages read
buffer_page_read_xdes	buffer_page_io	counter	Number of Extent Descriptor Pages read
buffer_page_read_blob	buffer_page_io	counter	Number of Uncompressed BLOB Pages read
buffer_page_read_zblob	buffer_page_io	counter	Number of First Compressed BLOB Pages read
buffer_page_read_zblob2	buffer_page_io	counter	Number of Subsequent Compressed BLOB Pages read
buffer_page_read_other	buffer_page_io	counter	Number of other/unknown (old version of InnoDB)
buffer_page_written_index_leaf	buffer_page_io	counter	Number of Index Leaf Pages written
buffer_page_written_index_non_leaf	buffer_page_io	counter	Number of Index Non-leaf Pages written
buffer_page_written_index_ibuf_leaf	buffer_page_io	counter	Number of Insert Buffer Index Leaf Pages written
buffer_page_written_index_ibuf_non_leaf	buffer_page_io	counter	Number of Insert Buffer Index Non-Leaf Pages wri
buffer_page_written_undo_log	buffer_page_io	counter	Number of Undo Log Pages written
buffer_page_written_index_inode	buffer_page_io	counter	Number of Index Inode Pages written
buffer_page_written_ibuf_free_list	buffer_page_io	counter	Number of Insert Buffer Free List Pages written
buffer_page_written_ibuf_bitmap	buffer_page_io	counter	Number of Insert Buffer Bitmap Pages written
buffer_page_written_system_page	buffer_page_io	counter	Number of System Pages written
buffer_page_written_trx_system	buffer_page_io	counter	Number of Transaction System Pages written
buffer_page_written_fsp_hdr	buffer_page_io	counter	Number of File Space Header Pages written
buffer_page_written_xdes	buffer_page_io	counter	Number of Extent Descriptor Pages written
buffer_page_written_blob	buffer_page_io	counter	Number of Uncompressed BLOB Pages written
buffer_page_written_zblob	buffer_page_io	counter	Number of First Compressed BLOB Pages written
buffer_page_written_zblob2	buffer_page_io	counter	Number of Subsequent Compressed BLOB Pages wri
buffer_page_written_other	buffer_page_io	counter	Number of other/unknown (old version InnoDB) P
os_data_reads	os	status_counter	Number of reads initiated (innodb_data_reads)
os_data_writes	os	status_counter	Number of writes initiated (innodb_data_writes)
os_data_fsyncs	os	status_counter	Number of fsync() calls (innodb_data_fsyncs)
os_pending_reads	os	counter	Number of reads pending
os_pending_writes	os	counter	Number of writes pending
os_log_bytes_written	os	status_counter	Bytes of log written (innodb_os_log_written)
os_log_fsyncs	os	status_counter	Number of fsync log writes (innodb_os_log_fsyn
os_log_pending_fsyncs	os	status_counter	Number of pending fsync write (innodb_os_log_p
os_log_pending_writes	os	status_counter	Number of pending log file writes (innodb_os_lo
trx_rw_commits	transaction	counter	Number of read-write transactions committed
trx_ro_commits	transaction	counter	Number of read-only transactions committed

trx_nl_ro_commits	transaction	counter	Number of non-locking auto-commit read-only transactions
trx_commits_insert_update	transaction	counter	Number of transactions committed with inserts and updates
trx_rollbacks	transaction	counter	Number of transactions rolled back
trx_rollbacks_savepoint	transaction	counter	Number of transactions rolled back to savepoint
trx_active_transactions	transaction	counter	Number of active transactions
trx_rseg_history_len	transaction	value	Length of the TRX_RSEG_HISTORY list
trx_undo_slots_used	transaction	counter	Number of undo slots used
trx_undo_slots_cached	transaction	counter	Number of undo slots cached
trx_rseg_current_size	transaction	value	Current rollback segment size in pages
purge_del_mark_records	purge	counter	Number of delete-marked rows purged
purge_upd_exist_or_extern_records	purge	counter	Number of purges on updates of existing records
purge_invoked	purge	counter	Number of times purge was invoked
purge_undo_log_pages	purge	counter	Number of undo log pages handled by the purge
purge_dml_delay_usec	purge	value	Microseconds DML to be delayed due to purge latency
purge_stop_count	purge	value	Number of times purge was stopped
purge_resume_count	purge	value	Number of times purge was resumed
log_checkpoints	recovery	counter	Number of checkpoints
log_lsn_last_flush	recovery	value	LSN of Last flush
log_lsn_last_checkpoint	recovery	value	LSN at last checkpoint
log_lsn_current	recovery	value	Current LSN value
log_lsn_checkpoint_age	recovery	value	Current LSN value minus LSN at last checkpoint
log_lsn_buf_pool_oldest	recovery	value	The oldest modified block LSN in the buffer pool
log_max_modified_age_async	recovery	value	Maximum LSN difference; when exceeded, start a new log file
log_pending_log_flushes	recovery	value	Pending log flushes
log_pending_checkpoint_writes	recovery	value	Pending checkpoints
log_num_log_io	recovery	value	Number of log I/Os
log_waits	recovery	status_counter	Number of log waits due to small log buffer (innodb_log_waits)
log_write_requests	recovery	status_counter	Number of log write requests (innodb_log_write_requests)
log_writes	recovery	status_counter	Number of log writes (innodb_log_writes)
log_padded	recovery	status_counter	Bytes of log padded for log write ahead
compress_pages_compressed	compression	counter	Number of pages compressed
compress_pages_decompressed	compression	counter	Number of pages decompressed
compression_pad_increments	compression	counter	Number of times padding is incremented to avoid page fragmentation
compression_pad_decrements	compression	counter	Number of times padding is decremented due to page fragmentation
compress_saved	compression	counter	Number of bytes saved by page compression
compress_pages_page_compressed	compression	counter	Number of pages compressed by page compression
compress_page_compressed_trim_op	compression	counter	Number of TRIM operation performed by page compression
compress_pages_page_decompressed	compression	counter	Number of pages decompressed by page compression
compress_pages_page_compression_error	compression	counter	Number of page compression errors
compress_pages_encrypted	compression	counter	Number of pages encrypted
compress_pages_decrypted	compression	counter	Number of pages decrypted
index_page_splits	index	counter	Number of index page splits
index_page_merge_attempts	index	counter	Number of index page merge attempts
index_page_merge_successful	index	counter	Number of successful index page merges
index_page_reorg_attempts	index	counter	Number of index page reorganization attempts
index_page_reorg_successful	index	counter	Number of successful index page reorganization
index_page_discards	index	counter	Number of index pages discarded
adaptive_hash_searches	adaptive_hash_index	status_counter	Number of successful searches using Adaptive Hash Index
adaptive_hash_searches_btree	adaptive_hash_index	status_counter	Number of searches using B-tree on an index secondary
adaptive_hash_pages_added	adaptive_hash_index	counter	Number of index pages on which the Adaptive Hash Index is used
adaptive_hash_pages_removed	adaptive_hash_index	counter	Number of index pages whose corresponding Adaptive Hash Index is removed
adaptive_hash_rows_added	adaptive_hash_index	counter	Number of Adaptive Hash Index rows added
adaptive_hash_rows_removed	adaptive_hash_index	counter	Number of Adaptive Hash Index rows removed
adaptive_hash_rows_deleted_no_hash_entry	adaptive_hash_index	counter	Number of rows deleted that did not have corresponding Adaptive Hash Index entry
adaptive_hash_rows_updated	adaptive_hash_index	counter	Number of Adaptive Hash Index rows updated
file_num_open_files	file_system	value	Number of files currently open (innodb_num_open_files)
ibuf_merges_insert	change_buffer	status_counter	Number of inserted records merged by change buffer
ibuf_merges_delete_mark	change_buffer	status_counter	Number of deleted records merged by change buffer
ibuf_merges_delete	change_buffer	status_counter	Number of purge records merged by change buffer
ibuf_merges_discard_insert	change_buffer	status_counter	Number of insert merged operations discarded
ibuf_merges_discard_delete_mark	change_buffer	status_counter	Number of deleted merged operations discarded
ibuf_merges_discard_delete	change_buffer	status_counter	Number of purge merged operations discarded
ibuf_merges	change_buffer	status_counter	Number of change buffer merges
ibuf_size	change_buffer	status_counter	Change buffer size in pages
innodb_master_thread_sleeps	server	counter	Number of times (seconds) master thread sleeps
innodb_activity_count	server	status_counter	Current server activity count
innodb_master_active_loops	server	counter	Number of times master thread performs its tasks
innodb_master_idle_loops	server	counter	Number of times master thread performs its tasks
innodb_background_drop_table_usec	server	counter	Time (in microseconds) spent to process drop table
innodb_log_flush_usec	server	counter	Time (in microseconds) spent to flush log records
innodb_dict_lru_usec	server	counter	Time (in microseconds) spent to process DICT LRU

innodb_dict_lru_count_active	server	counter	Number of tables evicted from DICT LRU list in
innodb_dict_lru_count_idle	server	counter	Number of tables evicted from DICT LRU list in
innodb dblwr_writes	server	status_counter	Number of doublewrite operations that have been
innodb dblwr_pages_written	server	status_counter	Number of pages that have been written for dou
innodb_page_size	server	value	InnoDB page size in bytes (innodb_page_size)
innodb_rwlock_s_spin_waits	server	status_counter	Number of rwlock spin waits due to shared latch
innodb_rwlock_x_spin_waits	server	status_counter	Number of rwlock spin waits due to exclusive la
innodb_rwlock_sx_spin_waits	server	status_counter	Number of rwlock spin waits due to sx latch re
innodb_rwlock_s_spin_rounds	server	status_counter	Number of rwlock spin loop rounds due to share
innodb_rwlock_x_spin_rounds	server	status_counter	Number of rwlock spin loop rounds due to exclu
innodb_rwlock_sx_spin_rounds	server	status_counter	Number of rwlock spin loop rounds due to sx la
innodb_rwlock_s_os_waits	server	status_counter	Number of OS waits due to shared latch request
innodb_rwlock_x_os_waits	server	status_counter	Number of OS waits due to exclusive latch requ
innodb_rwlock_sx_os_waits	server	status_counter	Number of OS waits due to sx latch request
dml_reads	dml	status_counter	Number of rows read
dml_inserts	dml	status_counter	Number of rows inserted
dml_deletes	dml	status_counter	Number of rows deleted
dml_updates	dml	status_counter	Number of rows updated
dml_system_reads	dml	status_counter	Number of system rows read
dml_system_inserts	dml	status_counter	Number of system rows inserted
dml_system_deletes	dml	status_counter	Number of system rows deleted
dml_system_updates	dml	status_counter	Number of system rows updated
ddl_background_drop_indexes	ddl	counter	Number of indexes waiting to be dropped after
ddl_background_drop_tables	ddl	counter	Number of tables in background drop table list
ddl_online_create_index	ddl	counter	Number of indexes being created online
ddl_pending_alter_table	ddl	counter	Number of ALTER TABLE, CREATE INDEX, DROP INDE
ddl_sort_file_alter_table	ddl	counter	Number of sort files created during alter table
ddl_log_file_alter_table	ddl	counter	Number of log files created during alter table
icp_attempts	icp	counter	Number of attempts for index push-down conditio
icp_no_match	icp	counter	Index push-down condition does not match
icp_out_of_range	icp	counter	Index push-down condition out of range
icp_match	icp	counter	Index push-down condition matches

234 rows in set (0.001 sec)

From MariaDB 10.6

SELECT name, subsystem, type, comment FROM INFORMATION_SCHEMA.INNODB_METRICS;			
name	subsystem	type	comment
metadata_table_handles_opened	metadata	counter	Number of table handles opened
lock_deadlocks	lock	value	Number of deadlocks
lock_timeouts	lock	value	Number of lock timeouts
lock_rec_lock_waits	lock	counter	Number of times enqueued into record lock wait
lock_table_lock_waits	lock	counter	Number of times enqueued into table lock wait
lock_rec_lock_requests	lock	counter	Number of record locks requested
lock_rec_lock_created	lock	counter	Number of record locks created
lock_rec_lock_removed	lock	counter	Number of record locks removed from the lock qu
lock_rec_locks	lock	counter	Current number of record locks on tables
lock_table_lock_created	lock	counter	Number of table locks created
lock_table_lock_removed	lock	counter	Number of table locks removed from the lock qu
lock_table_locks	lock	counter	Current number of table locks on tables
lock_row_lock_current_waits	lock	status_counter	Number of row locks currently being waited for
lock_row_lock_time	lock	status_counter	Time spent in acquiring row locks, in millisec
lock_row_lock_time_max	lock	value	The maximum time to acquire a row lock, in mil
lock_row_lock_waits	lock	status_counter	Number of times a row lock had to be waited for
lock_row_lock_time_avg	lock	value	The average time to acquire a row lock, in mil
buffer_pool_size	server	value	Server buffer pool size (all buffer pools) in
buffer_pool_reads	buffer	status_counter	Number of reads directly from disk (innodb_buf
buffer_pool_read_requests	buffer	status_counter	Number of logical read requests (innodb_buffer
buffer_pool_write_requests	buffer	status_counter	Number of write requests (innodb_buffer_pool_w
buffer_pool_wait_free	buffer	status_counter	Number of times waited for free buffer (innodb
buffer_pool_read_ahead	buffer	status_counter	Number of pages read as read ahead (innodb_buf
buffer_pool_read_ahead_evicted	buffer	status_counter	Read-ahead pages evicted without being accessed
buffer_pool_pages_total	buffer	value	Total buffer pool size in pages (innodb_buffer
buffer_pool_pages_misc	buffer	value	Buffer pages for misc use such as row locks or
buffer_pool_pages_data	buffer	value	Buffer pages containing data (innodb_buffer_poi
buffer_pool_bytes_data	buffer	value	Buffer bytes containing data (innodb_buffer_poi
buffer_pool_pages_dirty	buffer	value	Buffer pages currently dirty (innodb_buffer poi

buffer_pool_bytes_dirty	buffer	value	Buffer bytes currently dirty (innodb_buffer_pool_bytes_dirty)
buffer_pool_pages_free	buffer	value	Buffer pages currently free (innodb_buffer_pool_pages_free)
buffer_pages_created	buffer	status_counter	Number of pages created (innodb_pages_created)
buffer_pages_written	buffer	status_counter	Number of pages written (innodb_pages_written)
buffer_index_pages_written	buffer	status_counter	Number of index pages written (innodb_index_pages_written)
buffer_non_index_pages_written	buffer	status_counter	Number of non index pages written (innodb_non_index_pages_written)
buffer_pages_read	buffer	status_counter	Number of pages read (innodb_pages_read)
buffer_index_sec_rec_cluster_reads	buffer	status_counter	Number of secondary record reads triggered cluster
buffer_index_sec_rec_cluster_reads_avoided	buffer	status_counter	Number of secondary record reads avoided triggered cluster
buffer_data_reads	buffer	status_counter	Amount of data read in bytes (innodb_data_reads)
buffer_data_written	buffer	status_counter	Amount of data written in bytes (innodb_data_written)
buffer_flush_batch_scanned	buffer	set_owner	Total pages scanned as part of flush batch
buffer_flush_batch_num_scan	buffer	set_member	Number of times buffer flush list flush is called
buffer_flush_batch_scanned_per_call	buffer	set_member	Pages scanned per flush batch scan
buffer_flush_batch_total_pages	buffer	set_owner	Total pages flushed as part of flush batch
buffer_flush_batches	buffer	set_member	Number of flush batches
buffer_flush_batch_pages	buffer	set_member	Pages queued as a flush batch
buffer_flush_neighbor_total_pages	buffer	set_owner	Total neighbors flushed as part of neighbor flush
buffer_flush_neighborhood	buffer	set_member	Number of times neighbors flushing is invoked
buffer_flush_neighbor_pages	buffer	set_member	Pages queued as a neighbor batch
buffer_flush_n_to_flush_requested	buffer	counter	Number of pages requested for flushing.
buffer_flush_n_to_flush_by_age	buffer	counter	Number of pages target by LSN Age for flushing
buffer_flush_adaptive_avg_time	buffer	counter	Avg time (ms) spent for adaptive flushing received
buffer_flush_adaptive_avg_pass	buffer	counter	Number of adaptive flushes passed during the run
buffer_LRU_get_free_loops	buffer	counter	Total loops in LRU get free.
buffer_LRU_get_free_waits	buffer	counter	Total sleep waits in LRU get free.
buffer_flush_avg_page_rate	buffer	counter	Average number of pages at which flushing is happening
buffer_flush_lsn_avg_rate	buffer	counter	Average redo generation rate
buffer_flush_pct_for_dirty	buffer	counter	Percent of IO capacity used to avoid max dirty
buffer_flush_pct_for_lsn	buffer	counter	Percent of IO capacity used to avoid reusable LSN
buffer_flush_sync_waits	buffer	counter	Number of times a wait happens due to sync flush
buffer_flush_adaptive_total_pages	buffer	set_owner	Total pages flushed as part of adaptive flushing
buffer_flush_adaptive	buffer	set_member	Number of adaptive batches
buffer_flush_adaptive_pages	buffer	set_member	Pages queued as an adaptive batch
buffer_flush_sync_total_pages	buffer	set_owner	Total pages flushed as part of sync batches
buffer_flush_sync	buffer	set_member	Number of sync batches
buffer_flush_sync_pages	buffer	set_member	Pages queued as a sync batch
buffer_flush_background_total_pages	buffer	set_owner	Total pages flushed as part of background batch
buffer_flush_background	buffer	set_member	Number of background batches
buffer_flush_background_pages	buffer	set_member	Pages queued as a background batch
buffer_LRU_batch_scanned	buffer	set_owner	Total pages scanned as part of LRU batch
buffer_LRU_batch_num_scan	buffer	set_member	Number of times LRU batch is called
buffer_LRU_batch_scanned_per_call	buffer	set_member	Pages scanned per LRU batch call
buffer_LRU_batch_flush_total_pages	buffer	status_counter	Total pages flushed as part of LRU batches
buffer_LRU_batch_evict_total_pages	buffer	status_counter	Total pages evicted as part of LRU batches
buffer_LRU_single_flush_failure_count	Buffer	counter	Number of times attempt to flush a single page
buffer_LRU_get_free_search	Buffer	counter	Number of searches performed for a clean page
buffer_LRU_search_scanned	buffer	set_owner	Total pages scanned as part of LRU search
buffer_LRU_search_num_scan	buffer	set_member	Number of times LRU search is performed
buffer_LRU_search_scanned_per_call	buffer	set_member	Page scanned per single LRU search
buffer_LRU_unzip_search_scanned	buffer	set_owner	Total pages scanned as part of LRU unzip search
buffer_LRU_unzip_search_num_scan	buffer	set_member	Number of times LRU unzip search is performed
buffer_LRU_unzip_search_scanned_per_call	buffer	set_member	Page scanned per single LRU unzip search
buffer_page_read_index_leaf	buffer_page_io	counter	Number of Index Leaf Pages read
buffer_page_read_index_non_leaf	buffer_page_io	counter	Number of Index Non-leaf Pages read
buffer_page_read_index_ibuf_leaf	buffer_page_io	counter	Number of Insert Buffer Index Leaf Pages read
buffer_page_read_index_ibuf_non_leaf	buffer_page_io	counter	Number of Insert Buffer Index Non-Leaf Pages read
buffer_page_read_undo_log	buffer_page_io	counter	Number of Undo Log Pages read
buffer_page_read_index_inode	buffer_page_io	counter	Number of Index Inode Pages read
buffer_page_read_ibuf_free_list	buffer_page_io	counter	Number of Insert Buffer Free List Pages read
buffer_page_read_ibuf_bitmap	buffer_page_io	counter	Number of Insert Buffer Bitmap Pages read
buffer_page_read_system_page	buffer_page_io	counter	Number of System Pages read
buffer_page_read_trx_system	buffer_page_io	counter	Number of Transaction System Pages read
buffer_page_read_fsp_hdr	buffer_page_io	counter	Number of File Space Header Pages read
buffer_page_read_xdes	buffer_page_io	counter	Number of Extent Descriptor Pages read
buffer_page_read_blob	buffer_page_io	counter	Number of Uncompressed BLOB Pages read
buffer_page_read_zblob	buffer_page_io	counter	Number of First Compressed BLOB Pages read
buffer_page_read_zblob2	buffer_page_io	counter	Number of Subsequent Compressed BLOB Pages read
buffer_page_read_other	buffer_page_io	counter	Number of other/unknown (old version of InnoDB)
buffer_page_written_index_leaf	buffer_page_io	counter	Number of Index Leaf Pages written
buffer_page_written_index_non_leaf	buffer_page_io	counter	Number of Index Non-leaf Pages written

buffer_page_written_index_ibut_leaf	buffer_page_io	counter	Number of Insert Buffer Index Leaf Pages written
buffer_page_written_index_ibuf_non_leaf	buffer_page_io	counter	Number of Insert Buffer Index Non-Leaf Pages written
buffer_page_written_undo_log	buffer_page_io	counter	Number of Undo Log Pages written
buffer_page_written_index_inode	buffer_page_io	counter	Number of Index Inode Pages written
buffer_page_written_ibuf_free_list	buffer_page_io	counter	Number of Insert Buffer Free List Pages written
buffer_page_written_ibuf_bitmap	buffer_page_io	counter	Number of Insert Buffer Bitmap Pages written
buffer_page_written_system_page	buffer_page_io	counter	Number of System Pages written
buffer_page_written_trx_system	buffer_page_io	counter	Number of Transaction System Pages written
buffer_page_written_fsp_hdr	buffer_page_io	counter	Number of File Space Header Pages written
buffer_page_written_xdes	buffer_page_io	counter	Number of Extent Descriptor Pages written
buffer_page_written_blob	buffer_page_io	counter	Number of Uncompressed BLOB Pages written
buffer_page_written_zblob	buffer_page_io	counter	Number of First Compressed BLOB Pages written
buffer_page_written_zblob2	buffer_page_io	counter	Number of Subsequent Compressed BLOB Pages written
buffer_page_written_other	buffer_page_io	counter	Number of other/unknown (old version InnoDB) Pages written
os_data_reads	os	status_counter	Number of reads initiated (innodb_data_reads)
os_data_writes	os	status_counter	Number of writes initiated (innodb_data_writes)
os_data_fsyncs	os	status_counter	Number of fsync() calls (innodb_data_fsyncs)
os_pending_reads	os	counter	Number of reads pending
os_pending_writes	os	counter	Number of writes pending
os_log_bytes_written	os	status_counter	Bytes of log written (innodb_os_log_written)
os_log_fsyncs	os	status_counter	Number of fsync log writes (innodb_os_log_fsyncs)
os_log_pending_fsyncs	os	status_counter	Number of pending fsync write (innodb_os_log_pending_fsyncs)
os_log_pending_writes	os	status_counter	Number of pending log file writes (innodb_os_log_pending_writes)
trx_rw_commits	transaction	counter	Number of read-write transactions committed
trx_ro_commits	transaction	counter	Number of read-only transactions committed
trx_nl_ro_commits	transaction	counter	Number of non-locking auto-commit read-only transactions
trx_commits_insert_update	transaction	counter	Number of transactions committed with inserts and updates
trx_rollback	transaction	counter	Number of transactions rolled back
trx_rollback_savepoint	transaction	counter	Number of transactions rolled back to savepoint
trx_rseg_history_len	transaction	value	Length of the TRX_RSEG_HISTORY list
trx_undo_slots_used	transaction	counter	Number of undo slots used
trx_undo_slots_cached	transaction	counter	Number of undo slots cached
trx_rseg_current_size	transaction	value	Current rollback segment size in pages
purge_del_mark_records	purge	counter	Number of delete-marked rows purged
purge_upd_exist_or_extern_records	purge	counter	Number of purges on updates of existing records
purge_invoked	purge	counter	Number of times purge was invoked
purge_undo_log_pages	purge	counter	Number of undo log pages handled by the purge
purge_dml_delay_usecs	purge	value	Microseconds DML to be delayed due to purge latency
purge_stop_count	purge	value	Number of times purge was stopped
purge_resume_count	purge	value	Number of times purge was resumed
log_checkpoints	recovery	counter	Number of checkpoints
log_lsn_last_flush	recovery	value	LSN of Last flush
log_lsn_last_checkpoint	recovery	value	LSN at last checkpoint
log_lsn_current	recovery	value	Current LSN value
log_lsn_checkpoint_age	recovery	value	Current LSN value minus LSN at last checkpoint
log_lsn_buf_pool_oldest	recovery	value	The oldest modified block LSN in the buffer pool
log_max_modified_age_async	recovery	value	Maximum LSN difference; when exceeded, start a new log
log_pending_log_flushes	recovery	value	Pending log flushes
log_pending_checkpoint_writes	recovery	value	Pending checkpoints
log_num_log_io	recovery	value	Number of log I/Os
log_waits	recovery	status_counter	Number of log waits due to small log buffer (in seconds)
log_write_requests	recovery	status_counter	Number of log write requests (innodb_log_write_requests)
log_writes	recovery	status_counter	Number of log writes (innodb_log_writes)
log_padded	recovery	status_counter	Bytes of log padded for log write ahead
compress_pages_compressed	compression	counter	Number of pages compressed
compress_pages_decompressed	compression	counter	Number of pages decompressed
compression_pad_increments	compression	counter	Number of times padding is incremented to avoid page compression errors
compression_pad_decrements	compression	counter	Number of times padding is decremented due to page compression errors
compress_saved	compression	counter	Number of bytes saved by page compression
compress_pages_page_compressed	compression	counter	Number of pages compressed by page compression
compress_page_compressed_trim_op	compression	counter	Number of TRIM operation performed by page compression
compress_pages_page_decompressed	compression	counter	Number of pages decompressed by page compression
compress_pages_page_compression_error	compression	counter	Number of page compression errors
compress_pages_encrypted	compression	counter	Number of pages encrypted
compress_pages_decrypted	compression	counter	Number of pages decrypted
index_page_splits	index	counter	Number of index page splits
index_page_merge_attempts	index	counter	Number of index page merge attempts
index_page_merge_successful	index	counter	Number of successful index page merges
index_page_reorg_attempts	index	counter	Number of index page reorganization attempts
index_page_reorg_successful	index	counter	Number of successful index page reorganization attempts
index_page_discards	index	counter	Number of index pages discarded
adaptive_hash_searches	adaptive hash index	status_counter	Number of successful searches using Adaptive Hash Index

adaptive_hash_searches_btree	adaptive_hash_index	status_counter	Number of searches using B-tree on an index set
adaptive_hash_pages_added	adaptive_hash_index	counter	Number of index pages on which the Adaptive Hash Index was added
adaptive_hash_pages_removed	adaptive_hash_index	counter	Number of index pages whose corresponding Adaptive Hash Index was removed
adaptive_hash_rows_added	adaptive_hash_index	counter	Number of Adaptive Hash Index rows added
adaptive_hash_rows_removed	adaptive_hash_index	counter	Number of Adaptive Hash Index rows removed
adaptive_hash_rows_deleted_no_hash_entry	adaptive_hash_index	counter	Number of rows deleted that did not have corresponding Adaptive Hash Index entry
adaptive_hash_rows_updated	adaptive_hash_index	counter	Number of Adaptive Hash Index rows updated
file_num_open_files	file_system	value	Number of files currently open (innodb_num_open_files)
ibuf_merges_insert	change_buffer	status_counter	Number of inserted records merged by change buffer
ibuf_merges_delete_mark	change_buffer	status_counter	Number of deleted records merged by change buffer
ibuf_merges_delete	change_buffer	status_counter	Number of purge records merged by change buffer
ibuf_merges_discard_insert	change_buffer	status_counter	Number of insert merged operations discarded
ibuf_merges_discard_delete_mark	change_buffer	status_counter	Number of deleted merged operations discarded
ibuf_merges_discard_delete	change_buffer	status_counter	Number of purge merged operations discarded
ibuf_merges	change_buffer	status_counter	Number of change buffer merges
ibuf_size	change_buffer	status_counter	Change buffer size in pages
innodb_master_thread_sleeps	server	counter	Number of times (seconds) master thread sleeps
innodb_activity_count	server	status_counter	Current server activity count
innodb_master_active_loops	server	counter	Number of times master thread performs its tasks
innodb_master_idle_loops	server	counter	Number of times master thread performs its tasks
innodb_log_flush_usec	server	counter	Time (in microseconds) spent to flush log records
innodb_dict_lru_usec	server	counter	Time (in microseconds) spent to process DICT LRU list
innodb_dict_lru_count_active	server	counter	Number of tables evicted from DICT LRU list in active state
innodb_dict_lru_count_idle	server	counter	Number of tables evicted from DICT LRU list in idle state
innodb dblwr writes	server	status_counter	Number of doublewrite operations that have been performed
innodb dblwr pages written	server	status_counter	Number of pages that have been written for doublewrite
innodb_page_size	server	value	InnoDB page size in bytes (innodb_page_size)
dml_reads	dml	status_counter	Number of rows read
dml_inserts	dml	status_counter	Number of rows inserted
dml_deletes	dml	status_counter	Number of rows deleted
dml_updates	dml	status_counter	Number of rows updated
dml_system_reads	dml	status_counter	Number of system rows read
dml_system_inserts	dml	status_counter	Number of system rows inserted
dml_system Deletes	dml	status_counter	Number of system rows deleted
dml_system_updates	dml	status_counter	Number of system rows updated
ddl_background_drop_indexes	ddl	counter	Number of indexes waiting to be dropped after background processing
ddl_online_create_index	ddl	counter	Number of indexes being created online
ddl_pending_alter_table	ddl	counter	Number of ALTER TABLE, CREATE INDEX, DROP INDEX operations
ddl_sort_file_alter_table	ddl	counter	Number of sort files created during alter table
ddl_log_file_alter_table	ddl	counter	Number of log files created during alter table
icp_attempts	icp	counter	Number of attempts for index push-down condition
icp_no_match	icp	counter	Index push-down condition does not match
icp_out_of_range	icp	counter	Index push-down condition out of range
icp_match	icp	counter	Index push-down condition matches

216 rows in set (0.000 sec)

1.1.2.9.1.1.1.20 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 |
+-----+-----+-----+-----+
```

1.1.2.9.1.1.1.21 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 INNODB_SYS_TABLES.TABLE_ID .
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)

```

1.1.2.9.1.1.1.22 Information Schema 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 INNODB_SYS_TABLES.SPACE 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    |
+-----+-----+

```

1.1.2.9.1.1.1.23 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 INNODB_SYS_INDEXES.INDEX_ID .
NAME	Field name, matching the value from INNODB_SYS_COLUMNS.NAME .
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)
```

1.1.2.9.1.1.1.24 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
...

```

1.1.2.9.1.1.1.25 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 INNODB_SYS_FOREIGN.ID 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.

1.1.2.9.1.1.1.26 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; PRIMARY (primary key), GEN_CLUST_INDEX (index representing primary key where there isn't one), ID_IND , FOR_IND (validating foreign key constraint) , REF_IND .
TABLE_ID	bigint(21) unsigned	NO		0	Table identifier, matching the value from INNODB_SYS_TABLES.TABLE_ID .
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. GEN_CLUST_INDEX'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 innodb_file_per_table system variable). Remains unchanged after a TRUNCATE TABLE 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)

```

1.1.2.9.1.1.1.27 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

1.1.2.9.1.1.1.28 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 <code>innodb_file_per_table</code> system variable). Remains unchanged after a <code>TRUNCATE TABLE</code> 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') (>= MariaDB 10.5) varchar(12) (<= 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') (>= MariaDB 10.5) varchar(10) (<= 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
FILE_FORMAT: Antelope
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
FILE_FORMAT: Antelope
ROW_FORMAT: Redundant
ZIP_PAGE_SIZE: 0
SPACE_TYPE: System
2 rows in set (0.00 sec)
```

See Also

- [InnoDB Data Dictionary Troubleshooting](#)

1.1.2.9.1.1.29 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 DATA DIRECTORY option has been specified in CREATE TABLE , otherwise 0 .
FILE_FORMAT	InnoDB file format .
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 innodb_page_size variable. From MariaDB 10.6.0 , contains the physical page size of a page (previously ZIP_PAGE_SIZE).
ZIP_PAGE_SIZE	Zip page size for this tablespace. Removed in MariaDB 10.6.0 .

SPACE_TYPE	Tablespace type. Can be <code>General</code> for general tablespaces or <code>Single</code> 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 INNODB_SYS_DATAFILES table . Added in MariaDB 10.6.0 .

Examples

MariaDB 10.4:

```
DESC information_schema.innodb_sys_tablespaces;
+-----+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| SPACE  | int(11) unsigned | NO   |   0  |          |          |
| NAME   | varchar(655)    | NO   |       |          |          |
| FLAG   | int(11) unsigned | NO   |   0  |          |          |
| FILE_FORMAT | varchar(10)    | YES  |     NULL |          |          |
| 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
    FILE_FORMAT: Barracuda
    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
    FILE_FORMAT: Barracuda
    ROW_FORMAT: Dynamic
    PAGE_SIZE: 16384
    ZIP_PAGE_SIZE: 0
    SPACE_TYPE: Single
    FS_BLOCK_SIZE: 4096
    FILE_SIZE: 98304
    ALLOCATED_SIZE: 98304
```

1.1.2.9.1.1.1.30 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 <code>INNODB_SYS_TABLES.TABLE_ID</code> value.
SCHEMA	Database name (XtraDB only).
NAME	Table name, matching the <code>INNODB_SYS_TABLES.NAME</code> value.
STATS_INITIALIZED	<code>Initialized</code> if statistics have already been collected, otherwise <code>Uninitialized</code> .
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	<code>Auto_increment</code> value.
REF_COUNT	Countdown to zero, when table metadata can be removed from the table cache. (InnoDB only)
MYSQL_HANDLES_OPENED	(XtraDB only).

1.1.2.9.1.1.1.31 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	

1.1.2.9.1.1.1.32 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_VERS:
+-----+-----+-----+-----+-----+
| 18   | db_encrypt/t_encrypted_existing_key | 1 | 1 | 1 |
| 19   | db_encrypt/t_not_encrypted_existing_key | 1 | 0 | 1 |
| 20   | db_encrypt/t_not_encrypted_non_existing_key | 1 | 0 | 4294967295 |
| 21   | db_encrypt/t_default_encryption_existing_key | 1 | 1 | 1 |
| 22   | db_encrypt/t_encrypted_default_key | 1 | 1 | 1 |
| 23   | db_encrypt/t_not_encrypted_default_key | 1 | 0 | 1 |
| 24   | db_encrypt/t_defaults | 1 | 1 | 1 |
+-----+-----+-----+-----+-----+
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](#)

1.1.2.9.1.1.1.33 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)
```

1.1.2.9.1.1.1.34 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 <code>PROCESSLIST</code> 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. 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 <code>SELECT</code> 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_WAITS` 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

```

1.1.2.9.1.1.1.35 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 ibtmp1 , 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          |
+-----+-----+-----+-----+
```

1.1.2.9.1.1.2 Information Schema MyRocks Tables

1.1.2.9.1.1.2. Information Schema ROCKSDB_CFSTATS Table

The Information Schema ROCKSDB_CFSTATS table is included as part of the MyRocks storage engine.

The PROCESS privilege is required to view the table.

It contains the following columns:

Column	Description
CF_NAME	
STAT_TYPE	
VALUE	

1.1.2.9.1.1.2.1 Information Schema

ROCKSDB_CF_OPTIONS Table

The [Information Schema](#) `ROCKSDB_CF_OPTIONS` table is included as part of the [MyRocks](#) storage engine, and contains information about MyRocks column families.

The `PROCESS` privilege is required to view the table.

It contains the following columns:

Column	Description
CF_NAME	Column family name.
OPTION_TYPE	
VALUE	

1.1.2.9.1.1.2.2 Information Schema ROCKSDB_COMPACTION_STATS Table

The [Information Schema](#) `ROCKSDB_COMPACTION_STATS` table is included as part of the [MyRocks](#) storage engine.

The `PROCESS` privilege is required to view the table.

It contains the following columns:

Column	Description
CF_NAME	
LEVEL	
TYPE	
VALUE	

1.1.2.9.1.1.2.3 Information Schema ROCKSDB_DBSTATS Table

The [Information Schema](#) `ROCKSDB_DBSTATS` table is included as part of the [MyRocks](#) storage engine.

The `PROCESS` privilege is required to view the table.

It contains the following columns:

Column	Description
STAT_TYPE	
VALUE	

1.1.2.9.1.1.2.4 Information Schema ROCKSDB_DDL Table

The [Information Schema](#) `ROCKSDB_DDL` table is included as part of the [MyRocks](#) storage engine.

The `PROCESS` privilege is required to view the table.

It contains the following columns:

Column	Description
TABLE_SCHEMA	
TABLE_NAME	
PARTITION_NAME	
INDEX_NAME	
COLUMN_FAMILY	
INDEX_NUMBER	

INDEX_TYPE	
KV_FORMAT_VERSION	
TTL_DURATION	
INDEX_FLAGS	
CF	
AUTO_INCREMENT	

1.1.2.9.1.1.2.5 Information Schema ROCKSDB_DEADLOCK Table

The `Information Schema ROCKSDB_DEADLOCK` table is included as part of the `MyRocks` storage engine.

The `PROCESS privilege` is required to view the table.

It contains the following columns:

Column	Description
DEADLOCK_ID	
TIMESTAMP	
TRANSACTION_ID	
CF_NAME	
WAITING_KEY	
LOCK_TYPE	
INDEX_NAME	
TABLE_NAME	
ROLLED_BACK	

1.1.2.9.1.1.2.6 Information Schema ROCKSDB_GLOBAL_INFO Table

The `Information Schema ROCKSDB_GLOBAL_INFO` table is included as part of the `MyRocks` storage engine.

The `PROCESS privilege` is required to view the table.

It contains the following columns:

Column	Description
TYPE	
NAME	
VALUE	

1.1.2.9.1.1.2.7 Information Schema ROCKSDB_INDEX_FILE_MAP Table

The `Information Schema ROCKSDB_INDEX_FILE_MAP` table is included as part of the `MyRocks` storage engine.

The `PROCESS privilege` is required to view the table.

It contains the following columns:

Column	Description
COLUMN_FAMILY	
INDEX_NUMBER	

SST_NAME	
NUM_ROWS	
DATA_SIZE	
ENTRY_DELETES	
ENTRY_SINGLEDELETES	
ENTRY_MERGES	
ENTRY_OTHERS	
DISTINCT_KEYS_PREFIX	

1.1.2.9.1.1.2.8 Information Schema ROCKSDB_LOCKS Table

The `Information Schema ROCKSDB_LOCKS` table is included as part of the `MyRocks` storage engine.

The `PROCESS privilege` is required to view the table.

It contains the following columns:

Column	Description
COLUMN_FAMILY_ID	
TRANSACTION_ID	
KEY	
MODE	

1.1.2.9.1.1.2.9 Information Schema ROCKSDB_PERF_CONTEXT Table

The `Information Schema ROCKSDB_PERF_CONTEXT` table is included as part of the `MyRocks` storage engine and includes per-table/partition counters .

The `PROCESS privilege` is required to view the table.

It contains the following columns:

Column	Description
TABLE_SCHEMA	
TABLE_NAME	
PARTITION_NAME	
STAT_TYPE	
VALUE	

Note: for multi-table queries, all counter increments are "billed" to the first table in the query: <https://github.com/facebook/mysql-5.6/issues/1018>

1.1.2.9.1.1.2.10 Information Schema ROCKSDB_PERF_CONTEXT_GLOBAL Table

The `Information Schema ROCKSDB_PERF_CONTEXT_GLOBAL` table is included as part of the `MyRocks` storage engine and includes global counter information.

The `PROCESS privilege` is required to view the table.

It contains the following columns:

Column	Description

STAT_TYPE	
VALUE	

1.1.2.9.1.1.2.11 Information Schema ROCKSDB_SST_PROPS Table

The [Information Schema](#) `ROCKSDB_SST_PROPS` table is included as part of the [My Rocks](#) storage engine.

The `PROCESS` privilege is required to view the table.

It contains the following columns:

Column	Description
<code>SST_NAME</code>	
<code>COLUMN_FAMILY</code>	
<code>DATA_BLOCKS</code>	
<code>ENTRIES</code>	
<code>RAW_KEY_SIZE</code>	
<code>RAW_VALUE_SIZE</code>	
<code>DATA_BLOCK_SIZE</code>	
<code>INDEX_BLOCK_SIZE</code>	
<code>INDEX_PARTITIONS</code>	
<code>TOP_LEVEL_INDEX_SIZE</code>	
<code>FILTER_BLOCK_SIZE</code>	
<code>COMPRESSION_ALGO</code>	
<code>CREATION_TIME</code>	

1.1.2.9.1.1.2.12 Information Schema ROCKSDB_TRX Table

The [Information Schema](#) `ROCKSDB_TRX` table is included as part of the [My Rocks](#) storage engine.

The `PROCESS` privilege is required to view the table.

It contains the following columns:

Column	Description
<code>TRANSACTION_ID</code>	
<code>STATE</code>	
<code>NAME</code>	
<code>WRITE_COUNT</code>	
<code>LOCK_COUNT</code>	
<code>TIMEOUT_SEC</code>	
<code>WAITING_KEY</code>	
<code>WAITING_COLUMN_FAMILY_ID</code>	
<code>IS_REPLICATION</code>	
<code>SKIP_TRX_API</code>	
<code>READ_ONLY</code>	
<code>HAS_DEADLOCK_DETECTION</code>	

NUM_ONGOING_BULKLOAD	
THREAD_ID	
QUERY	

1.1.2.9.1.1.3 ColumnStore Information Schema Tables

1. [COLUMNSTORE_TABLES](#)
2. [COLUMNSTORE_COLUMNS](#)
3. [COLUMNSTORE_EXTENTS](#)
4. [COLUMNSTORE_FILES](#)
5. [Stored Procedures](#)
 1. [total_usage\(\)](#)
 2. [table_usage\(\)](#)
 3. [compression_ratio\(\)](#)

MariaDB ColumnStore has four Information Schema tables that expose information about the table and column storage. These tables were added in version 1.0.5 of ColumnStore and were heavily modified for 1.0.6.

COLUMNSTORE_TABLES

The first table is the INFORMATION_SCHEMA.COLUMNSTORE_TABLES. This contains information about the tables inside ColumnStore. The table layout is as follows:

Column	Description
TABLE_SCHEMA	The database schema for the table
TABLE_NAME	The table name
OBJECT_ID	The ColumnStore object ID for the table
CREATION_DATE	The date the table was created
COLUMN_COUNT	The number of columns in the table
AUTOINCREMENT	The start autoincrement value for the table set during CREATE TABLE

Note: Tables created with ColumnStore 1.0.4 or lower will have the year field of the creation data set incorrectly by 1900 years.

COLUMNSTORE_COLUMNS

The INFORMATION_SCHEMA.COLUMNSTORE_COLUMNS table contains information about every single column inside ColumnStore. The table layout is as follows:

Column	Description
TABLE_SCHEMA	The database schema for the table
TABLE_NAME	The table name for the column
COLUMN_NAME	The column name
OBJECT_ID	The object ID for the column
DICTIONARY_OBJECT_ID	The dictionary object ID for the column (NULL if there is no dictionary object)
LIST_OBJECT_ID	Placeholder for future information
TREE_OBJECT_ID	Placeholder for future information
DATA_TYPE	The data type for the column
COLUMN_LENGTH	The data length for the column
COLUMN_POSITION	The position of the column in the table, starting at 0
COLUMN_DEFAULT	The default value for the column

IS_NULLABLE	Whether or not the column can be set to NULL
NUMERIC_PRECISION	The numeric precision for the column
NUMERIC_SCALE	The numeric scale for the column
IS_AUTOINCREMENT	Set to 1 if the column is an autoincrement column
COMPRESSION_TYPE	The type of compression (either "None" or "Snappy")

COLUMNSTORE_EXTENTS

This table displays the extent map in a user consumable form. An extent is a collection of details about a section of data related to a columnstore column. A majority of columns in ColumnStore will have multiple extents and the columns table above can be joined to this one to filter results by table or column. The table layout is as follows:

Column	Description
OBJECT_ID	The object ID for the extent
OBJECT_TYPE	Whether this is a "Column" or "Dictionary" extent
LOGICAL_BLOCK_START	ColumnStore's internal start LBID for this extent
LOGICAL_BLOCK_END	ColumnStore's internal end LBID for this extent
MIN_VALUE	This minimum value stored in this extent
MAX_VALUE	The maximum value stored in this extent
WIDTH	The data width for the extent
DBROOT	The DBRoot number for the extent
PARTITION_ID	The partition ID for the extent
SEGMENT_ID	The segment ID for the extent
BLOCK_OFFSET	The block offset for the data file, each data file can contain multiple extents for a column
MAX_BLOCKS	The maximum number of blocks for the extent
HIGH_WATER_MARK	The last block committed to the extent (starting at 0)
STATE	The state of the extent (see below)
STATUS	The availability status for the column which is either "Available", "Unavailable" or "Out of service"
DATA_SIZE	The uncompressed data size for the extent calculated as $(HWM + 1) * BLOCK_SIZE$

Notes:

1. The state is "Valid" for a normal state, "Invalid" if a cpimport has completed but the table has not yet been accessed (min/max values will be invalid) or "Updating" if there is a DML statement writing to the column
2. In ColumnStore the block size is 8192 bytes
3. By default ColumnStore will write create an extent file of $256 * 1024 * WIDTH$ bytes for the first partition, if this is too small then for uncompressed data it will create a file of the maximum size for the extent ($MAX_BLOCKS * BLOCK_SIZE$). Snappy always compression adds a header block.
4. Object IDs of less than 3000 are for internal tables and will not appear in any of the information schema tables
5. Prior to 1.0.12 / 1.1.2 DATA_SIZE was incorrectly calculated
6. HWM is set to zero for the lower segments when there are multiple segments in an extent file, these can be observed when $BLOCK_OFFSET > 0$
7. When HWM is 0 the DATA_SIZE will show 0 instead of 8192 to avoid confusion when there is multiple segments in an extent file

COLUMNSTORE_FILES

The columnstore_files table provides information about each file associated with extensions. Each extension can reuse a file at different block offsets so this is not a 1:1 relationship to the columnstore_extents table.

Column	Description
OBJECT_ID	The object ID for the extent
SEGMENT_ID	The segment ID for the extent
PARTITION_ID	The partition ID for the extent
FILENAME	The full path and filename for the extent file, multiple extents for the same column can point to this file with different BLOCK_OFFSETs
FILE_SIZE	The disk file size for the extent
COMPRESSED_DATA_SIZE	The amount of the compressed file used, NULL if this is an uncompressed file

Stored Procedures

A few stored procedures were added in 1.0.6 to provide summaries based on the information schema tables. These can be accessed from the COLUMNSTORE_INFO schema.

total_usage()

The total_usage() procedure gives a total disk usage summary for all the columns in ColumnStore with the exception of the columns used for internal maintenance. It is executed using the following query:

```
> call columnstore_info.total_usage();
```

table_usage()

The table_usage() procedure gives a the total data disk usage, dictionary disk usage and grand total disk usage per-table. It can be called in several ways, the first gives a total for each table:

```
> call columnstore_info.table_usage(NULL, NULL);
```

Or for a specific table, my_table in my_schema in this example:

```
> call columnstore_info.table_usage('my_schema', 'my_table');
```

You can also request all tables for a specified schema:

```
> call columnstore_info.table_usage('my_schema', NULL);
```

Note: The quotes around the table name are required, an error will occur without them.

compression_ratio()

The compression_ratio() procedure calculates the average compression ratio across all the compressed extents in ColumnStore. It is called using:

```
> call columnstore_info.compression_ratio();
```

Note: The compression ratio is incorrectly calculated before versions 1.0.12 / 1.1.2

1.1.2.9.1.1.4 Information Schema ALL_PLUGINS Table Description

The [Information Schema](#) `ALL_PLUGINS` table contains information about [server plugins](#), whether installed or not.

It contains the following columns:

Column	Description
<code>PLUGIN_NAME</code>	Name of the plugin.
<code>PLUGIN_VERSION</code>	Version from the plugin's general type descriptor.
<code>PLUGIN_STATUS</code>	Plugin status, one of <code>ACTIVE</code> , <code>INACTIVE</code> , <code>DISABLED</code> , <code>DELETED</code> or <code>NOT INSTALLED</code> .
<code>PLUGIN_TYPE</code>	Plugin type; <code>STORAGE ENGINE</code> , <code>INFORMATION_SCHEMA</code> , <code>AUTHENTICATION</code> , <code>REPLICATION</code> , <code>DAEMON</code> or <code>AUDIT</code> .
<code>PLUGIN_TYPE_VERSION</code>	Version from the plugin's type-specific descriptor.
<code>PLUGIN_LIBRARY</code>	Plugin's shared object file name, located in the directory specified by the <code>plugin_dir</code> system variable, and used by the <code>INSTALL PLUGIN</code> and <code>UNINSTALL PLUGIN</code> statements. <code>NULL</code> if the plugin is compiled in and cannot be uninstalled.
<code>PLUGIN_LIBRARY_VERSION</code>	Version from the plugin's API interface.
<code>PLUGIN_AUTHOR</code>	Author of the plugin.
<code>PLUGIN_DESCRIPTION</code>	Description.
<code>PLUGIN_LICENSE</code>	Plugin's licence.
<code>LOAD_OPTION</code>	How the plugin was loaded; one of <code>OFF</code> , <code>ON</code> , <code>FORCE</code> or <code>FORCE_PLUS_PERMANENT</code> . See Installing Plugins .
<code>PLUGIN_Maturity</code>	Plugin's maturity level; one of <code>Unknown</code> , <code>Experimental</code> , <code>Alpha</code> , <code>Beta</code> , <code>'Gamma</code> , and <code>Stable</code> .
<code>PLUGIN_AUTH_VERSION</code>	Plugin's version as determined by the plugin author. An example would be '0.99 beta 1'.

It provides a superset of the information shown by the `SHOW PLUGINS SONAME` statement, as well as the `information_schema.PLUGINS` table. For specific information about storage engines (a particular type of plugin), see the [Information Schema ENGINES table](#) and the `SHOW ENGINES` statement.

The table is not a standard Information Schema table, and is a MariaDB extension.

Example

```
SELECT * FROM information_schema.all_plugins\G
*****
 1. row *****
    PLUGIN_NAME: binlog
    PLUGIN_VERSION: 1.0
    PLUGIN_STATUS: ACTIVE
    PLUGIN_TYPE: STORAGE ENGINE
    PLUGIN_TYPE_VERSION: 100314.0
    PLUGIN_LIBRARY: NULL
    PLUGIN_LIBRARY_VERSION: NULL
    PLUGIN_AUTHOR: MySQL AB
    PLUGIN_DESCRIPTION: This is a pseudo storage engine to represent the binlog in a transaction
    PLUGIN_LICENSE: GPL
    LOAD_OPTION: FORCE
    PLUGIN_Maturity: Stable
    PLUGIN_AUTH_VERSION: 1.0
*****
 2. row *****
    PLUGIN_NAME: mysql_native_password
    PLUGIN_VERSION: 1.0
    PLUGIN_STATUS: ACTIVE
    PLUGIN_TYPE: AUTHENTICATION
    PLUGIN_TYPE_VERSION: 2.1
    PLUGIN_LIBRARY: NULL
    PLUGIN_LIBRARY_VERSION: NULL
    PLUGIN_AUTHOR: R.J.Silk, Sergei Golubchik
    PLUGIN_DESCRIPTION: Native MySQL authentication
    PLUGIN_LICENSE: GPL
    LOAD_OPTION: FORCE
    PLUGIN_Maturity: Stable
    PLUGIN_AUTH_VERSION: 1.0
*****
 3. row *****
    PLUGIN_NAME: mysql_old_password
    PLUGIN_VERSION: 1.0
    PLUGIN_STATUS: ACTIVE
```

```

PLUGIN_STATUS: ACTIVE
PLUGIN_TYPE: AUTHENTICATION
PLUGIN_TYPE_VERSION: 2.1
PLUGIN_LIBRARY: NULL
PLUGIN_LIBRARY_VERSION: NULL
PLUGIN_AUTHOR: R.J.Silk, Sergei Golubchik
PLUGIN_DESCRIPTION: Old MySQL-4.0 authentication
PLUGIN_LICENSE: GPL
LOAD_OPTION: FORCE
PLUGIN_MATURITY: Stable
PLUGIN_AUTH_VERSION: 1.0
...
*****104. row *****
PLUGIN_NAME: WSREP_MEMBERSHIP
PLUGIN_VERSION: 1.0
PLUGIN_STATUS: NOT INSTALLED
PLUGIN_TYPE: INFORMATION SCHEMA
PLUGIN_TYPE_VERSION: 100314.0
PLUGIN_LIBRARY: wsrep_info.so
PLUGIN_LIBRARY_VERSION: 1.13
PLUGIN_AUTHOR: Nirbhay Choubey
PLUGIN_DESCRIPTION: Information about group members
PLUGIN_LICENSE: GPL
LOAD_OPTION: OFF
PLUGIN_MATURITY: Stable
PLUGIN_AUTH_VERSION: 1.0
*****
*****105. row *****
PLUGIN_NAME: WSREP_STATUS
PLUGIN_VERSION: 1.0
PLUGIN_STATUS: NOT INSTALLED
PLUGIN_TYPE: INFORMATION SCHEMA
PLUGIN_TYPE_VERSION: 100314.0
PLUGIN_LIBRARY: wsrep_info.so
PLUGIN_LIBRARY_VERSION: 1.13
PLUGIN_AUTHOR: Nirbhay Choubey
PLUGIN_DESCRIPTION: Group view information
PLUGIN_LICENSE: GPL
LOAD_OPTION: OFF
PLUGIN_MATURITY: Stable

```

1.1.2.9.1.1.5 Information Schema APPLICABLE_ROLES Table

The `Information Schema APPLICABLE_ROLES` table shows the [role authorizations](#) that the current user may use.

It contains the following columns:

Column	Description	Added
GRANTEE	Account that the role was granted to.	
ROLE_NAME	Name of the role.	
IS_GRANTABLE	Whether the role can be granted or not.	
IS_DEFAULT	Whether the role is the user's default role or not	MariaDB 10.1.3

The current role is in the `ENABLED_ROLES` Information Schema table.

Example

```

SELECT * FROM information_schema.APPLICABLE_ROLES;
+-----+-----+-----+-----+
| GRANTEE      | ROLE_NAME | IS_GRANTABLE | IS_DEFAULT |
+-----+-----+-----+-----+
| root@localhost | journalist | YES          | NO          |
| root@localhost | staff     | YES          | NO          |
| root@localhost | dd        | YES          | NO          |
| root@localhost | dog       | YES          | NO          |
+-----+-----+-----+-----+

```

1.1.2.9.1.1.6 Information Schema CHARACTER_SETS Table

The `Information Schema CHARACTER_SETS` table contains a list of supported [character sets](#), their default collations and maximum lengths.

It contains the following columns:

Column	Description
CHARACTER_SET_NAME	Name of the character set.
DEFAULT_COLLATE_NAME	Default collation used.
DESCRIPTION	Character set description.
MAXLEN	Maximum length.

The `SHOW CHARACTER SET` statement returns the same results (although in a different order), and both can be refined in the same way. For example, the following two statements return the same results:

```
SHOW CHARACTER SET WHERE Maxlen LIKE '2';
```

and

```

SELECT * FROM information_schema.CHARACTER_SETS
WHERE MAXLEN LIKE '2';

```

See [Setting Character Sets and Collations](#) for details on specifying the character set at the server, database, table and column levels, and [Supported Character Sets and Collations](#) for a full list of supported characters sets and collations.

Example

```

SELECT CHARACTER_SET_NAME FROM information_schema.CHARACTER_SETS
WHERE DEFAULT_COLLATE_NAME LIKE '%chinese%';
+-----+
| CHARACTER_SET_NAME |
+-----+
| big5              |
| gb2312            |
| gbk               |
+-----+

```

1.1.2.9.1.1.7 Information Schema CHECK_CONSTRAINTS Table

MariaDB starting with 10.2.22

The Information Schema `CHECK_CONSTRAINTS` Table was introduced in [MariaDB 10.3.10](#) and [MariaDB 10.2.22](#).

The `Information Schema CHECK_CONSTRAINTS` table stores metadata about the [constraints](#) defined for tables in all databases.

It contains the following columns:

Column	Description
CONSTRAINT_CATALOG	Always contains the string 'def'.
CONSTRAINT_SCHEMA	Database name.
TABLE_NAME	Table name.
CONSTRAINT_NAME	Constraint name.

MariaDB starting with 10.5.10

LEVEL	Type of the constraint ('Column' or 'Table').
-------	---

CHECK_CLAUSE	Constraint clause.
--------------	--------------------

Example

A table with a numeric table check constraint and with a default check constraint name:

```
CREATE TABLE t ( a int, CHECK (a>10));
```

To see check constraint call `check_constraints` table from [information schema](#).

```
SELECT * from INFORMATION_SCHEMA.CHECK_CONSTRAINTS\G
```

```
***** 1. row *****
CONSTRAINT_CATALOG: def
CONSTRAINT_SCHEMA: test
CONSTRAINT_NAME: CONSTRAINT_1
TABLE_NAME: t
CHECK_CLAUSE: `a` > 10
```

A new table check constraint called `a_upper`:

```
ALTER TABLE t ADD CONSTRAINT a_upper CHECK (a<100);
```

```
SELECT * from INFORMATION_SCHEMA.CHECK_CONSTRAINTS\G
```

```
***** 1. row *****
CONSTRAINT_CATALOG: def
CONSTRAINT_SCHEMA: test
CONSTRAINT_NAME: CONSTRAINT_1
TABLE_NAME: t
CHECK_CLAUSE: `a` > 10
***** 2. row *****
CONSTRAINT_CATALOG: def
CONSTRAINT_SCHEMA: test
CONSTRAINT_NAME: a_upper
TABLE_NAME: t
CHECK_CLAUSE: `a` < 100
```

A new table `tt` with a field check constraint called `b`, as well as a table check constraint called `b_upper`:

```

CREATE TABLE tt(b int CHECK(b>0),CONSTRAINT b_upper CHECK(b<50));

SELECT * from INFORMATION_SCHEMA.CHECK_CONSTRAINTS;
+-----+-----+-----+-----+
| CONSTRAINT_CATALOG | CONSTRAINT_SCHEMA | CONSTRAINT_NAME | TABLE_NAME | CHECK_CLAUSE |
+-----+-----+-----+-----+
| def                | test            | b                 | tt          | `b` > 0      |
| def                | test            | b_upper           | tt          | `b` < 50     |
| def                | test            | CONSTRAINT_1      | t           | `a` > 10    |
| def                | test            | a_upper           | t           | `a` < 100   |
+-----+-----+-----+-----+

```

Note: The name of the field constraint is the same as the field name.

After dropping the default table constraint called CONSTRAINT_1 :

```

ALTER TABLE t DROP CONSTRAINT CONSTRAINT_1;

SELECT * from INFORMATION_SCHEMA.CHECK_CONSTRAINTS;
+-----+-----+-----+-----+
| CONSTRAINT_CATALOG | CONSTRAINT_SCHEMA | CONSTRAINT_NAME | TABLE_NAME | CHECK_CLAUSE |
+-----+-----+-----+-----+
| def                | test            | b                 | tt          | `b` > 0      |
| def                | test            | b_upper           | tt          | `b` < 50     |
| def                | test            | a_upper           | t           | `a` < 100   |
+-----+-----+-----+-----+

```

Trying to insert invalid arguments into table t and tt generates an error.

```

INSERT INTO t VALUES (10),(20),(100);
ERROR 4025 (23000): CONSTRAINT `a_upper` failed for `test`.`t`

INSERT INTO tt VALUES (10),(-10),(100);
ERROR 4025 (23000): CONSTRAINT `b` failed for `test`.`tt`

INSERT INTO tt VALUES (10),(20),(100);
ERROR 4025 (23000): CONSTRAINT `b_upper` failed for `test`.`tt`

```

From MariaDB 10.5.10:

```

create table majra(check(x>0), x int, y int check(y < 0), z int,
                   constraint z check(z>0), constraint xyz check(x<10 and y<10 and z<10));
Query OK, 0 rows affected (0.036 sec)

show create table majra;
+-----+
| Table | Create Table
+-----+
| majra | CREATE TABLE `majra` (
  `x` int(11) DEFAULT NULL,
  `y` int(11) DEFAULT NULL CHECK (`y` < 0),
  `z` int(11) DEFAULT NULL,
  CONSTRAINT `CONSTRAINT_1` CHECK (`x` > 0),
  CONSTRAINT `z` CHECK (`z` > 0),
  CONSTRAINT `xyz` CHECK (`x` < 10 and `y` < 10 and `z` < 10)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 |
+-----+
1 row in set (0.000 sec)

```

```
select * from information_schema.check_constraints where table_name='majra';
```

CONSTRAINT_CATALOG	CONSTRAINT_SCHEMA	TABLE_NAME	CONSTRAINT_NAME	LEVEL	CHECK_CLAUSE
def	test	majra	y	Column	`y` < 0
def	test	majra	CONSTRAINT_1	Table	`x` > 0
def	test	majra	z	Table	`z` > 0
def	test	majra	xyz	Table	`x` < 10 and `y` < 10 and `z` < 10

```
4 rows in set (0.001 sec)
```

1.1.2.9.1.1.8 Information Schema CLIENT_STATISTICS Table

The `Information Schema` `CLIENT_STATISTICS` table holds statistics about client connections. This is part of the [User Statistics](#) feature, which is not enabled by default.

It contains the following columns:

Field	Type	Notes
CLIENT	VARCHAR(64)	The IP address or hostname the connection originated from.
TOTAL_CONNECTIONS	INT(21)	The number of connections created for this client.
CONCURRENT_CONNECTIONS	INT(21)	The number of concurrent connections for this client.
CONNECTED_TIME	INT(21)	The cumulative number of seconds elapsed while there were connections from this client.
BUSY_TIME	DOUBLE	The cumulative number of seconds there was activity on connections from this client.
CPU_TIME	DOUBLE	The cumulative CPU time elapsed while servicing this client's connections. Note that this number may be wrong on SMP system if there was a CPU migration for the thread during the execution of the query.
BYTES_RECEIVED	INT(21)	The number of bytes received from this client's connections.
BYTES_SENT	INT(21)	The number of bytes sent to this client's connections.
BINLOG_BYTES_WRITTEN	INT(21)	The number of bytes written to the binary log from this client's connections.
ROWS_READ	INT(21)	The number of rows read by this client's connections.
ROWS_SENT	INT(21)	The number of rows sent by this client's connections.
ROWS_DELETED	INT(21)	The number of rows deleted by this client's connections.
ROWS_INSERTED	INT(21)	The number of rows inserted by this client's connections.

ROWS_UPDATED	INT(21)	The number of rows updated by this client's connections.
SELECT_COMMANDS	INT(21)	The number of <code>SELECT</code> commands executed from this client's connections.
UPDATE_COMMANDS	INT(21)	The number of <code>UPDATE</code> commands executed from this client's connections.
OTHER_COMMANDS	INT(21)	The number of other commands executed from this client's connections.
COMMIT_TRANSACTIONS	INT(21)	The number of <code>COMMIT</code> commands issued by this client's connections.
ROLLBACK_TRANSACTIONS	INT(21)	The number of <code>ROLLBACK</code> commands issued by this client's connections.
DENIED_CONNECTIONS	INT(21)	The number of connections denied to this client.
LOST_CONNECTIONS	INT(21)	The number of this client's connections that were terminated uncleanly.
ACCESS_DENIED	INT(21)	The number of times this client's connections issued commands that were denied.
EMPTY_QUERIES	INT(21)	The number of times this client's connections sent queries that returned no results to the server.
TOTAL_SSL_CONNECTIONS	INT(21)	The number of <code>TLS connections</code> created for this client. (>= MariaDB 10.1.1)
MAX_STATEMENT_TIME_EXCEEDED	INT(21)	The number of times a statement was aborted, because it was executed longer than its <code>MAX_STATEMENT_TIME</code> threshold. (>= MariaDB 10.1.1)

Example

```
SELECT * FROM information_schema.CLIENT_STATISTICS\G
*****
1. row *****
CLIENT: localhost
TOTAL_CONNECTIONS: 3
CONCURRENT_CONNECTIONS: 0
CONNECTED_TIME: 4883
BUSY_TIME: 0.009722
CPU_TIME: 0.0102131
BYTES_RECEIVED: 841
BYTES_SENT: 13897
BINLOG_BYTES_WRITTEN: 0
ROWS_READ: 0
ROWS_SENT: 214
ROWS_DELETED: 0
ROWS_INSERTED: 207
ROWS_UPDATED: 0
SELECT_COMMANDS: 10
UPDATE_COMMANDS: 0
OTHER_COMMANDS: 13
COMMIT_TRANSACTIONS: 0
ROLLBACK_TRANSACTIONS: 0
DENIED_CONNECTIONS: 0
LOST_CONNECTIONS: 0
ACCESS_DENIED: 0
EMPTY_QUERIES: 1
```

1.1.2.9.1.1.9 Information Schema COLLATION_CHARACTER_SET_APPLICABILITY Table

The `Information Schema COLLATION_CHARACTER_SET_APPLICABILITY` table shows which `character sets` are associated with which collations.

It contains the following columns:

Column	Description
COLLATION_NAME	Collation name.
CHARACTER_SET_NAME	Name of the associated character set.

`COLLATION_CHARACTER_SET_APPLICABILITY` is essentially a subset of the `COLLATIONS` table.

```
SELECT COLLATION_NAME,CHARACTER_SET_NAME FROM information_schema.COLLATIONS;
```

and

```
SELECT * FROM information_schema.COLLATION_CHARACTER_SET_APPLICABILITY;
```

will return identical results.

See [Setting Character Sets and Collations](#) for details on specifying the character set at the server, database, table and column levels.

Example

```
SELECT * FROM information_schema.COLLATION_CHARACTER_SET_APPLICABILITY
WHERE CHARACTER_SET_NAME='utf32';
+-----+-----+
| COLLATION_NAME | CHARACTER_SET_NAME |
+-----+-----+
| utf32_general_ci | utf32 |
| utf32_bin | utf32 |
| utf32_unicode_ci | utf32 |
| utf32_icelandic_ci | utf32 |
| utf32_latvian_ci | utf32 |
| utf32_romanian_ci | utf32 |
| utf32_slovenian_ci | utf32 |
| utf32_polish_ci | utf32 |
| utf32_estonian_ci | utf32 |
| utf32_spanish_ci | utf32 |
| utf32_swedish_ci | utf32 |
| utf32_turkish_ci | utf32 |
| utf32_czech_ci | utf32 |
| utf32_danish_ci | utf32 |
| utf32_lithuanian_ci | utf32 |
| utf32_slovak_ci | utf32 |
| utf32_spanish2_ci | utf32 |
| utf32_roman_ci | utf32 |
| utf32_persian_ci | utf32 |
| utf32_esperanto_ci | utf32 |
| utf32_hungarian_ci | utf32 |
| utf32_sinhala_ci | utf32 |
| utf32_german2_ci | utf32 |
| utf32_croatian_ci | utf32 |
+-----+-----+
```

1.1.2.9.1.1.10 Information Schema COLLATIONS Table

The [Information Schema](#) `COLLATIONS` table contains a list of supported collations.

It contains the following columns:

Column	Description
COLLATION_NAME	Name of the collation.
CHARACTER_SET_NAME	Associated character set.
ID	Collation id.
IS_DEFAULT	Whether the collation is the character set's default.
IS_COMPILED	Whether the collation is compiled into the server.
SORTLEN	Sort length, used for determining the memory used to sort strings in this collation.

Contents

1. [NO PAD collations](#)
2. [Example](#)
3. [See Also](#)

The `SHOW COLLATION` statement returns the same results and both can be reduced in a similar way.

For example, in MariaDB Server 10.6, the following two statements return the same results:

```
SHOW COLLATION WHERE Charset LIKE 'utf8mb3';
```

and

```
SELECT * FROM information_schema.COLLATIONS  
WHERE CHARACTER_SET_NAME LIKE 'utf8mb3';
```

In MariaDB Server 10.5 and before, `utf8` should be specified instead of `utf8mb3`.

NO PAD collations

MariaDB starting with 10.2

NO PAD collations regard trailing spaces as normal characters. You can get a list of all NO PAD collations as follows:

```
SELECT collation_name FROM information_schema.COLLATIONS  
WHERE collation_name LIKE "%nopad%";  
+-----+  
| collation_name |  
+-----+  
| big5_chinese_nopad_ci |  
| big5_nopad_bin |  
... |
```

Example

```
SELECT * FROM information_schema.COLLATIONS;
```

COLLATION_NAME	CHARACTER_SET_NAME	ID	IS_DEFAULT	IS_COMPILED	SORTLEN
big5_chinese_ci	big5	1	Yes	Yes	1
big5_bin	big5	84		Yes	1
big5_chinese_nopad_ci	big5	1025		Yes	1
big5_nopad_bin	big5	1108		Yes	1
dec8_swedish_ci	dec8	3	Yes	Yes	1
dec8_bin	dec8	69		Yes	1
dec8_swedish_nopad_ci	dec8	1027		Yes	1
dec8_nopad_bin	dec8	1093		Yes	1
cp850_general_ci	cp850	4	Yes	Yes	1
cp850_bin	cp850	80		Yes	1
...					

See Also

- [Setting Character Sets and Collations](#) - specifying the character set at the server, database, table and column levels
- [Supported Character Sets and Collations](#) - full list of supported characters sets and collations.

1.1.2.9.1.1.11 Information Schema COLUMN_PRIVILEGES Table

The `Information Schema COLUMN_PRIVILEGES` table contains column privilege information derived from the `mysql.columns_priv` grant table.

It has the following columns:

Column	Description
GRANTEE	In the format <code>user_name@host_name</code> .
TABLE_CATALOG	Always def .
TABLE_SCHEMA	Database name.
TABLE_NAME	Table name.
COLUMN_NAME	Column name.

PRIVILEGE_TYPE	One of <code>SELECT</code> , <code>INSERT</code> , <code>UPDATE</code> or <code>REFERENCES</code> .
IS_GRANTABLE	Whether the user has the <code>GRANT OPTION</code> for this privilege.

Similar information can be accessed with the `SHOW FULL COLUMNS` and `SHOW GRANTS` statements. See the [GRANT](#) article for more about privileges.

This information is also stored in the `columns_priv` table, in the `mysql` system database.

For a description of the privileges that are shown in this table, see [column privileges](#).

Example

In the following example, no column-level privilege has been explicitly assigned:

```
SELECT * FROM information_schema.COLUMN_PRIVILEGES;
Empty set
```

1.1.2.9.1.1.12 Information Schema COLUMNS Table

The `Information Schema COLUMNS` table provides information about columns in each table on the server.

It contains the following columns:

Column	Description	Introduced
TABLE_CATALOG	Always contains the string 'def'.	
TABLE_SCHEMA	Database name.	
TABLE_NAME	Table name.	
COLUMN_NAME	Column name.	
ORDINAL_POSITION	Column position in the table. Can be used for ordering.	
COLUMN_DEFAULT	Default value for the column. From MariaDB 10.2.7 , literals are quoted to distinguish them from expressions. <code>NULL</code> means that the column has no default. In MariaDB 10.2.6 and earlier, no quotes were used for any type of default and <code>NULL</code> can either mean that there is no default, or that the default column value is <code>NULL</code> .	
IS_NULLABLE	Whether the column can contain <code>NULL</code> s.	
DATA_TYPE	The column's data type .	
CHARACTER_MAXIMUM_LENGTH	Maximum length.	
CHARACTER_OCTET_LENGTH	Same as the <code>CHARACTER_MAXIMUM_LENGTH</code> except for multi-byte character sets .	
NUMERIC_PRECISION	For numeric types, the precision (number of significant digits) for the column. <code>NULL</code> if not a numeric field.	
NUMERIC_SCALE	For numeric types, the scale (significant digits to the right of the decimal point). <code>NULL</code> if not a numeric field.	
DATETIME_PRECISION	Fractional-seconds precision, or <code>NULL</code> if not a time data type .	
CHARACTER_SET_NAME	Character set if a non-binary string data type , otherwise <code>NULL</code> .	
COLLATION_NAME	Collation if a non-binary string data type , otherwise <code>NULL</code> .	
COLUMN_TYPE	Column definition, a MySQL and MariaDB extension.	
COLUMN_KEY	Index type. <code>PRI</code> for primary key, <code>UNI</code> for unique index, <code>MUL</code> for multiple index. A MySQL and MariaDB extension.	
EXTRA	Additional information about a column, for example whether the column is an invisible column , or, from MariaDB 10.3.6 , <code>WITHOUT SYSTEM VERSIONING</code> if the table is not a system-versioned table . A MySQL and MariaDB extension.	
PRIVILEGES	Which privileges you have for the column. A MySQL and MariaDB extension.	
COLUMN_COMMENT	Column comments.	

IS_GENERATED	Indicates whether the column value is generated (virtual, or computed). Can be ALWAYS or NEVER .	MariaDB 10.2.5
GENERATION_EXPRESSION	The expression used for computing the column value in a generated (virtual, or computed) column.	MariaDB 10.2.5

It provides information similar to, but more complete, than `SHOW COLUMNS` and `mysqlshow`.

Examples

```
SELECT * FROM information_schema.COLUMNS\G
...
***** 9. row *****
    TABLE_CATALOG: def
    TABLE_SCHEMA: test
    TABLE_NAME: t2
    COLUMN_NAME: j
    ORDINAL_POSITION: 1
    COLUMN_DEFAULT: NULL
    IS_NULLABLE: YES
    DATA_TYPE: longtext
CHARACTER_MAXIMUM_LENGTH: 4294967295
CHARACTER_OCTET_LENGTH: 4294967295
    NUMERIC_PRECISION: NULL
    NUMERIC_SCALE: NULL
    DATETIME_PRECISION: NULL
CHARACTER_SET_NAME: utf8mb4
    COLLATION_NAME: utf8mb4_bin
    COLUMN_TYPE: longtext
    COLUMN_KEY:
        EXTRA:
    PRIVILEGES: select,insert,update,references
    COLUMN_COMMENT:
    IS_GENERATED: NEVER
    GENERATION_EXPRESSION: NULL
...
...
```

```
CREATE TABLE t (
    s1 VARCHAR(20) DEFAULT 'ABC',
    s2 VARCHAR(20) DEFAULT (concat('A','B')),
    s3 VARCHAR(20) DEFAULT ("concat('A','B')"),
    s4 VARCHAR(20),
    s5 VARCHAR(20) DEFAULT NULL,
    s6 VARCHAR(20) NOT NULL,
    s7 VARCHAR(20) DEFAULT 'NULL' NULL,
    s8 VARCHAR(20) DEFAULT 'NULL' NOT NULL
);
```

```
SELECT
    table_name,
    column_name,
    ordinal_position,
    column_default,
    column_default IS NULL
FROM information_schema.COLUMNS
WHERE table_schema=DATABASE()
AND TABLE_NAME='t';
```

From MariaDB 10.2.7:

table_name	column_name	ordinal_position	column_default	column_default IS NULL
t	s1	1	'ABC'	0
t	s2	2	concat('A','B')	0
t	s3	3	'concat(''A'',''B'')'	0
t	s4	4	NULL	0
t	s5	5	NULL	0
t	s6	6	NULL	1
t	s7	7	'NULL'	0
t	s8	8	'NULL'	0

In the results above, the two single quotes in `concat(''A'',''B'')` indicate an escaped single quote - see [string-literals](#). Note that while [mysql-command-line-client](#) appears to show the same default value for columns `s5` and `s6`, the first is a 4-character string "NULL", while the second is the SQL NULL value.

MariaDB 10.2.6 and before:

table_name	column_name	ordinal_position	column_default	column_default IS NULL
t	s1	1	ABC	0
t	s2	2	concat('A','B')	0
t	s3	3	concat('A','B')	0
t	s4	4	NULL	1
t	s5	5	NULL	1
t	s6	6	NULL	1
t	s7	7	NULL	0
t	s8	8	NULL	0

1.1.2.9.1.1.13 Information Schema DISKS Table

MariaDB 10.1.32

The `DISKS` table was introduced in [MariaDB 10.1.32](#), [MariaDB 10.2.14](#), and [MariaDB 10.3.6](#) as part of the `DISKS` plugin.

Description

The `DISKS` table is created when the `DISKS` plugin is enabled, and shows metadata about disks on the system.

Before [MariaDB 10.4.7](#), [MariaDB 10.3.17](#), [MariaDB 10.2.26](#) and [MariaDB 10.1.41](#), this plugin did **not** check [user privileges](#). When it is enabled, **any** user can query the `INFORMATION_SCHEMA.DISKS` table and see all the information it provides.

Since [MariaDB 10.4.7](#), [MariaDB 10.3.17](#), [MariaDB 10.2.26](#) and [MariaDB 10.1.41](#), it requires the `FILE` privilege.

The plugin only works on Linux.

The table contains the following columns:

Contents

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

Column	Description
DISK	Name of the disk itself.
PATH	Mount point of the disk.
TOTAL	Total space in KiB.
USED	Used amount of space in KiB.
AVAILABLE	Amount of space in KiB available to non-root users.

Note that as the amount of space available to root (OS user) may be more than what is available to non-root users, 'available' + 'used' may be less than 'total'.

All paths to which a particular disk has been mounted are reported. The rationale is that someone might want to take different action e.g. depending on which disk is relevant for a particular path. This leads to the same disk being reported multiple times.

Example

```
SELECT * FROM information_schema.DISKS;

+-----+-----+-----+-----+
| Disk | Path | Total | Used | Available |
+-----+-----+-----+-----+
| /dev/vda1 | / | 26203116 | 2178424 | 24024692 |
| /dev/vda1 | /boot | 26203116 | 2178424 | 24024692 |
| /dev/vda1 | /etc | 26203116 | 2178424 | 24024692 |
+-----+-----+-----+-----+
```

See Also

- [Disks Plugin](#) for details on installing, options
- [Plugin Overview](#) for details on managing plugins.

1.1.2.9.1.1.14 Information Schema ENABLED_ROLES Table

The `Information Schema` `ENABLED_ROLES` table shows the enabled `roles` for the current session.

It contains the following column:

Column	Description
<code>ROLE_NAME</code>	The enabled role name, or <code>NULL</code> .

This table lists all roles that are currently enabled, one role per row — the current role, roles granted to the current role, roles granted to these roles and so on. If no role is set, the row contains a `NULL` value.

The roles that the current user can enable are listed in the `APPLICABLE_ROLES` Information Schema table.

See also [CURRENT_ROLE\(\)](#).

Examples

```
SELECT * FROM information_schema.ENABLED_ROLES;
+-----+
| ROLE_NAME |
+-----+
| NULL      |
+-----+

SET ROLE staff;

SELECT * FROM information_schema.ENABLED_ROLES;
+-----+
| ROLE_NAME |
+-----+
| staff     |
+-----+
```

1.1.2.9.1.1.15 Information Schema ENGINES Table

The `Information Schema` `ENGINES` table displays status information about the server's `storage engines`.

It contains the following columns:

Column	Description
<code>ENGINE</code>	Name of the storage engine.
<code>SUPPORT</code>	Whether the engine is the default, or is supported or not.

COMMENT	Storage engine comments.
TRANSACTIONS	Whether or not the engine supports transactions .
XA	Whether or not the engine supports XA transactions .
SAVEPOINTS	Whether or not savepoints are supported.

It provides identical information to the `SHOW ENGINES` statement. Since storage engines are plugins, different information about them is also shown in the `information_schema.PLUGINS` table and by the `SHOW PLUGINS` statement.

The table is not a standard Information Schema table, and is a MySQL and MariaDB extension.

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](#).

Example

```
SELECT * FROM information_schema.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
```

```

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)

```

Check if a given storage engine is available:

```

SELECT SUPPORT FROM information_schema.ENGINES WHERE ENGINE LIKE 'tokudb';
Empty set

```

Check which storage engine supports XA transactions:

```

SELECT ENGINE FROM information_schema.ENGINES WHERE XA = 'YES';
+-----+
| ENGINE |
+-----+
| InnoDB |
+-----+

```

1.1.2.9.1.1.16 Information Schema EVENTS Table

The [Information Schema](#) `EVENTS` table stores information about [Events](#) on the server.

It contains the following columns:

Column	Description
<code>EVENT_CATALOG</code>	Always <code>def</code> .
<code>EVENT_SCHEMA</code>	Database where the event was defined.
<code>EVENT_NAME</code>	Event name.
<code>DEFINER</code>	Event definer.
<code>TIME_ZONE</code>	Time zone used for the event's scheduling and execution, by default <code>SYSTEM</code> .
<code>EVENT_BODY</code>	SQL .
<code>EVENT_DEFINITION</code>	The SQL defining the event.
<code>EVENT_TYPE</code>	Either <code>ONE TIME</code> or <code>RECURRING</code> .
<code>EXECUTE_AT</code>	<code>DATETIME</code> when the event is set to execute, or <code>NULL</code> if recurring.
<code>INTERVAL_VALUE</code>	Numeric interval between event executions for a recurring event, or <code>NULL</code> if not recurring.
<code>INTERVAL_FIELD</code>	Interval unit (e.g., <code>HOUR</code>)
<code>SQL_MODE</code>	The <code>SQL_MODE</code> at the time the event was created.
<code>STARTS</code>	Start <code>DATETIME</code> for a recurring event, <code>NULL</code> if not defined or not recurring.
<code>ENDS</code>	End <code>DATETIME</code> for a recurring event, <code>NULL</code> if not defined or not recurring.
<code>STATUS</code>	One of <code>ENABLED</code> , <code>DISABLED</code> or <code>/SLAVESIDE_DISABLED</code> .

ON_COMPLETION	The ON COMPLETION clause, either PRESERVE or NOT PRESERVE .
CREATED	When the event was created.
LAST_ALTERED	When the event was last changed.
LAST_EXECUTED	When the event was last run.
EVENT_COMMENT	The comment provided in the CREATE EVENT statement, or an empty string if none.
ORIGINATOR	MariaDB server ID on which the event was created.
CHARACTER_SET_CLIENT	character_set_client system variable session value at the time the event was created.
COLLATION_CONNECTION	collation_connection system variable session value at the time the event was created.
DATABASE_COLLATION	Database collation with which the event is linked.

The SHOW EVENTS and SHOW CREATE EVENT statements provide similar information.

1.1.2.9.1.1.17 Information Schema FEEDBACK Table

The Information Schema FEEDBACK table is created when the Feedback Plugin is enabled, and contains the complete contents submitted by the plugin.

It contains two columns:

Column	Description
VARIABLE_NAME	Name of the item of information being collected.
VARIABLE_VALUE	Contents of the item of information being collected.

It is possible to disable automatic collection, by setting the feedback_url variable to an empty string, and to submit the contents manually, as follows:

```
$ mysql -e 'SELECT * FROM information_schema.FEEDBACK' > report.txt
```

Then you can send it by opening https://mariadb.org/feedback_plugin/post in your browser, and uploading your generated report.txt . Or you can do it from the command line with (for example):

```
$ curl -F data=@report.txt https://mariadb.org/feedback_plugin/post
```

Manual uploading allows you to be absolutely sure that we receive only the data shown in the information_schema.FEEDBACK table and that no private or sensitive information is being sent.

Example

```

SELECT * FROM information_schema.FEEDBACK\G
...
***** 906. row *****
VARIABLE_NAME: Uname_sysname
VARIABLE_VALUE: Linux
***** 907. row *****
VARIABLE_NAME: Uname_release
VARIABLE_VALUE: 3.13.0-53-generic
***** 908. row *****
VARIABLE_NAME: Uname_version
VARIABLE_VALUE: #89-Ubuntu SMP Wed May 20 10:34:39 UTC 2015
***** 909. row *****
VARIABLE_NAME: Uname_machine
VARIABLE_VALUE: x86_64
***** 910. row *****
VARIABLE_NAME: Uname_distribution
VARIABLE_VALUE: lsb: Ubuntu 14.04.2 LTS
***** 911. row *****
VARIABLE_NAME: Collation used latin1_german1_ci
VARIABLE_VALUE: 1
***** 912. row *****
VARIABLE_NAME: Collation used latin1_swedish_ci
VARIABLE_VALUE: 18
***** 913. row *****
VARIABLE_NAME: Collation used utf8_general_ci
VARIABLE_VALUE: 567
***** 914. row *****
VARIABLE_NAME: Collation used latin1_bin
VARIABLE_VALUE: 1
***** 915. row *****
VARIABLE_NAME: Collation used binary
VARIABLE_VALUE: 16
***** 916. row *****
VARIABLE_NAME: Collation used utf8_bin
VARIABLE_VALUE: 4044

```

1.1.2.9.1.1.18 Information Schema FILES Table

The `FILES` table is unused in MariaDB. See [MDEV-11426](#).

1.1.2.9.1.1.19 Information Schema GEOMETRY_COLUMNS Table

Description

The `Information Schema GEOMETRY_COLUMNS` table provides support for Spatial Reference systems for GIS data.

It contains the following columns:

Column	Type	Null	Description
<code>F_TABLE_CATALOG</code>	<code>VARCHAR(512)</code>	NO	Together with <code>F_TABLE_SCHEMA</code> and <code>F_TABLE_NAME</code> , the fully qualified name of the featured table containing the geometry column.
<code>F_TABLE_SCHEMA</code>	<code>VARCHAR(64)</code>	NO	Together with <code>F_TABLE_CATALOG</code> and <code>F_TABLE_NAME</code> , the fully qualified name of the featured table containing the geometry column.
<code>F_TABLE_NAME</code>	<code>VARCHAR(64)</code>	NO	Together with <code>F_TABLE_CATALOG</code> and <code>F_TABLE_SCHEMA</code> , the fully qualified name of the featured table containing the geometry column.
<code>F_GEOMETRY_COLUMN</code>	<code>VARCHAR(64)</code>	NO	Name of the column in the featured table that is the geometry column.
<code>G_TABLE_CATALOG</code>	<code>VARCHAR(512)</code>	NO	
<code>G_TABLE_SCHEMA</code>	<code>VARCHAR(64)</code>	NO	Database name of the table implementing the geometry column.
<code>G_TABLE_NAME</code>	<code>VARCHAR(64)</code>	NO	Table name that is implementing the geometry column.

G_GEOMETRY_COLUMN	VARCHAR(64)	NO	
STORAGE_TYPE	TINYINT(2)	NO	Binary geometry implementation. Always 1 in MariaDB.
GEOMETRY_TYPE	INT(7)	NO	Integer reflecting the type of geometry stored in this column (see table below).
COORD_DIMENSION	TINYINT(2)	NO	Number of dimensions in the spatial reference system. Always 2 in MariaDB.
MAX_PPR	TINYINT(2)	NO	Always 0 in MariaDB.
SRID	SMALLINT(5)	NO	ID of the Spatial Reference System used for the coordinate geometry in this table. It is a foreign key reference to the SPATIAL_REF_SYS table .

Storage_type

The integers in the `storage_type` field match the geometry types as follows:

Integer	Type
0	GEOMETRY
1	POINT
3	LINESTRING
5	POLYGON
7	MULTIPOINT
9	MULTILINESTRING
11	MULTIPOLYGON

Example

```
CREATE TABLE g1(g GEOMETRY(9,4) REF_SYSTEM_ID=101);

SELECT * FROM information_schema.GEOMETRY_COLUMNS\G
***** 1. row *****
F_TABLE_CATALOG: def
F_TABLE_SCHEMA: test
F_TABLE_NAME: g1
F_GEOMETRY_COLUMN:
  G_TABLE_CATALOG: def
  G_TABLE_SCHEMA: test
  G_TABLE_NAME: g1
G_GEOMETRY_COLUMN: g
  STORAGE_TYPE: 1
  GEOMETRY_TYPE: 0
COORD_DIMENSION: 2
  MAX_PPR: 0
  SRID: 101
```

See also

- The [SPATIAL_REF_SYS table](#).

1.1.2.9.1.1.20 Information Schema GLOBAL_STATUS and SESSION_STATUS Tables

The `Information Schema GLOBAL_STATUS` and `SESSION_STATUS` tables store a record of all `status variables` and their global and session values respectively. This is the same information as displayed by the `SHOW STATUS` commands `SHOW GLOBAL STATUS` and `SHOW SESSION STATUS`.

They contain the following columns:

Column	Description
--------	-------------

VARIABLE_NAME	Status variable name.
VARIABLE_VALUE	Global or session value.

Example

```
SELECT * FROM information_schema.GLOBAL_STATUS;
+-----+-----+
| VARIABLE_NAME          | VARIABLE_VALUE |
+-----+-----+
...
| BINLOG_SNAPSHOT_FILE   | mariadb-bin.000208 |
| BINLOG_SNAPSHOT_POSITION | 369             |
...
| THREADS_CONNECTED       | 1                |
| THREADS_CREATED         | 1                |
| THREADS_RUNNING         | 1                |
| UPTIME                  | 57358           |
| UPTIME_SINCE_FLUSH_STATUS | 57358           |
+-----+-----+
```

1.1.2.9.1.1.21 Information Schema GLOBAL_VARIABLES and SESSION_VARIABLES Tables

The `Information Schema GLOBAL_VARIABLES` and `SESSION_VARIABLES` tables stores a record of all `system variables` and their global and session values respectively. This is the same information as displayed by the `SHOW VARIABLES` commands `SHOW GLOBAL VARIABLES` and `SHOW SESSION VARIABLES`.

It contains the following columns:

Column	Description
VARIABLE_NAME	System variable name.
VARIABLE_VALUE	Global or session value.

Example

```
SELECT * FROM information_schema.GLOBAL_VARIABLES ORDER BY VARIABLE_NAME\G
***** 1. row *****
VARIABLE_NAME: ARIA_BLOCK_SIZE
VARIABLE_VALUE: 8192
***** 2. row *****
VARIABLE_NAME: ARIA_CHECKPOINT_LOG_ACTIVITY
VARIABLE_VALUE: 1048576
***** 3. row *****
VARIABLE_NAME: ARIA_CHECKPOINT_INTERVAL
VARIABLE_VALUE: 30
...
***** 455. row *****
VARIABLE_NAME: VERSION_COMPILE_MACHINE
VARIABLE_VALUE: x86_64
***** 456. row *****
VARIABLE_NAME: VERSION_COMPILE_OS
VARIABLE_VALUE: debian-linux-gnu
***** 457. row *****
VARIABLE_NAME: WAIT_TIMEOUT
VARIABLE_VALUE: 600
```

1.1.2.9.1.1.22 Information Schema INDEX_STATISTICS Table

The [Information Schema 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.

This is part of the [User Statistics](#) feature, which is not enabled by default.

It contains the following columns:

Field	Type	Notes
TABLE_SCHEMA	VARCHAR(192)	The schema (database) name.
TABLE_NAME	VARCHAR(192)	The table name.
INDEX_NAME	VARCHAR(192)	The index name (as visible in SHOW CREATE TABLE).
ROWS_READ	INT(21)	The number of rows read from this index.

Example

```
SELECT * FROM information_schema.INDEX_STATISTICS
WHERE TABLE_NAME = "author";
+-----+-----+-----+-----+
| TABLE_SCHEMA | TABLE_NAME | INDEX_NAME | ROWS_READ |
+-----+-----+-----+-----+
| books        | author     | by_name    |      15   |
+-----+-----+-----+-----+
```

1.1.2.9.1.1.23 Information Schema KEY_CACHES Table

The [Information Schema KEY_CACHES](#) table shows statistics about the [segmented key cache](#).

It contains the following columns:

Column Name	Description
KEY_CACHE_NAME	The name of the key cache
SEGMENTS	total number of segments (set to <code>NULL</code> for regular key caches)
SEGMENT_NUMBER	segment number (set to <code>NULL</code> for any regular key caches and for rows containing aggregation statistics for segmented key caches)
FULL_SIZE	memory for cache buffers/auxiliary structures
BLOCK_SIZE	size of the blocks
USED_BLOCKS	number of currently used blocks
UNUSED_BLOCKS	number of currently unused blocks
DIRTY_BLOCKS	number of currently dirty blocks
READ_REQUESTS	number of read requests
READS	number of actual reads from files into buffers
WRITE_REQUESTS	number of write requests
WRITES	number of actual writes from buffers into files

Example

```

SELECT * FROM information_schema.KEY_CACHES \G
*****
1. row *****
KEY_CACHE_NAME: default
SEGMENTS: NULL
SEGMENT_NUMBER: NULL
  FULL_SIZE: 134217728
  BLOCK_SIZE: 1024
  USED_BLOCKS: 36
  UNUSED_BLOCKS: 107146
  DIRTY_BLOCKS: 0
  READ_REQUESTS: 40305
    READS: 21
  WRITE_REQUESTS: 19239
    WRITES: 358

```

1.1.2.9.1.1.24 Information Schema KEY_COLUMN_USAGE Table

The `Information Schema KEY_COLUMN_USAGE` table shows which key columns have constraints.

It contains the following columns:

Column	Description
CONSTRAINT_CATALOG	Always def .
CONSTRAINT_SCHEMA	Database name of the constraint.
CONSTRAINT_NAME	Name of the constraint (PRIMARY for the primary key).
TABLE_CATALOG	Always #def .
TABLE_SCHEMA	Database name of the column constraint.
TABLE_NAME	Table name of the column constraint.
COLUMN_NAME	Column name of the constraint.
ORDINAL_POSITION	Position of the column within the constraint.
POSITION_IN_UNIQUE_CONSTRAINT	For <code>foreign keys</code> , the position in the unique constraint.
REFERENCED_TABLE_SCHEMA	For foreign keys, the referenced database name.
REFERENCED_TABLE_NAME	For foreign keys, the referenced table name.
REFERENCED_COLUMN_NAME	For foreign keys, the referenced column name.

Example

```

SELECT * FROM information_schema.KEY_COLUMN_USAGE LIMIT 1 \G
*****
1. row *****
CONSTRAINT_CATALOG: def
CONSTRAINT_SCHEMA: my_website
CONSTRAINT_NAME: PRIMARY
  TABLE_CATALOG: def
  TABLE_SCHEMA: users
  COLUMN_NAME: user_id
  ORDINAL_POSITION: 1
POSITION_IN_UNIQUE_CONSTRAINT: NULL
  REFERENCED_TABLE_SCHEMA: NULL
  REFERENCED_TABLE_NAME: NULL
  REFERENCED_COLUMN_NAME: NULL

```

See Also

- [Finding Tables Without Primary Keys](#)

1.1.2.9.1.1.25 Information Schema KEYWORDS Table

MariaDB starting with 10.6.3

The KEYWORDS table was added in MariaDB 10.6.3.

Description

The `Information Schema KEYWORDS` table contains the list of MariaDB keywords.

It contains a single column:

Column	Description
WORD	Keyword

The table is not a standard Information Schema table, and is a MariaDB extension.

Example

```
SELECT * FROM INFORMATION_SCHEMA.KEYWORDS;
+-----+
| WORD |
+-----+
| &&
| <=
| <>
| !=
| >=
| <<
| >>
| <=>
| ACCESSIBLE
| ACCOUNT
| ACTION
| ADD
| ADMIN
| AFTER
| AGAINST
| AGGREGATE
| ALL
| ALGORITHM
| ALTER
| ALWAYS
| ANALYZE
| AND
| ANY
| AS
| ASC
| ASCII
| ASENSITIVE
| AT
| ATOMIC
| AUTHORS
| AUTO_INCREMENT
| AUTOEXTEND_SIZE
| AUTO
| AVG
| AVG_ROW_LENGTH
| BACKUP
| BEFORE
| BEGIN
| BETWEEN
| BIGINT
| BINARY
| BINLOG
| BIT
```

| BLOB
| BLOCK
| BODY
| BOOL
| BOOLEAN
| BOTH
| BTREE
| BY
| BYTE
| CACHE
| CALL
| CASCADE
| CASCADED
| CASE
| CATALOG_NAME
| CHAIN
| CHANGE
| CHANGED
| CHAR
| CHARACTER
| CHARSET
| CHECK
| CHECKPOINT
| CHECKSUM
| CIPHER
| CLASS_ORIGIN
| CLIENT
| CLOB
| CLOSE
| COALESCE
| CODE
| COLLATE
| COLLATION
| COLUMN
| COLUMN_NAME
| COLUMNS
| COLUMN_ADD
| COLUMN_CHECK
| COLUMN_CREATE
| COLUMN_DELETE
| COLUMN_GET
| COMMENT
| COMMIT
| COMMITTED
| COMPACT
| COMPLETION
| COMPRESSED
| CONCURRENT
| CONDITION
| CONNECTION
| CONSISTENT
| CONSTRAINT
| CONSTRAINT_CATALOG
| CONSTRAINT_NAME
| CONSTRAINT_SCHEMA
| CONTAINS
| CONTEXT
| CONTINUE
| CONTRIBUTORS
| CONVERT
| CPU
| CREATE
| CROSS
| CUBE
| CURRENT
| CURRENT_DATE
| CURRENT_POS
| CURRENT_ROLE
| CURRENT_TIME
| CURRENT_TIMESTAMP
| CURRENT_USER
| cursor

CURSOR	
CURSOR_NAME	
CYCLE	
DATA	
DATABASE	
DATABASES	
DATAFILE	
DATE	
DATETIME	
DAY	
DAY_HOUR	
DAY_MICROSECOND	
DAY_MINUTE	
DAY_SECOND	
DEALLOCATE	
DEC	
DECIMAL	
DECLARE	
DEFAULT	
DEFINER	
DELAYED	
DELAY_KEY_WRITE	
DELETE	
DELETE_DOMAIN_ID	
DESC	
DESCRIBE	
DES_KEY_FILE	
DETERMINISTIC	
DIAGNOSTICS	
DIRECTORY	
DISABLE	
DISCARD	
DISK	
DISTINCT	
DISTINCTROW	
DIV	
DO	
DOUBLE	
DO_DOMAIN_IDS	
DROP	
DUAL	
DUMPFILE	
DUPLICATE	
DYNAMIC	
EACH	
ELSE	
ELSEIF	
ELSIF	
EMPTY	
ENABLE	
ENCLOSED	
END	
ENDS	
ENGINE	
ENGINES	
ENUM	
ERROR	
ERRORS	
ESCAPE	
ESCAPED	
EVENT	
EVENTS	
EVERY	
EXAMINED	
EXCEPT	
EXCHANGE	
EXCLUDE	
EXECUTE	
EXCEPTION	
EXISTS	
EXIT	
EXPANSION	

```
| EXPIRE
| EXPORT
| EXPLAIN
| EXTENDED
| EXTENT_SIZE
| FALSE
| FAST
| FAULTS
| FEDERATED
| FETCH
| FIELDS
| FILE
| FIRST
| FIXED
| FLOAT
| FLOAT4
| FLOAT8
| FLUSH
| FOLLOWING
| FOLLOWS
| FOR
| FORCE
| FOREIGN
| FORMAT
| FOUND
| FROM
| FULL
| FULLTEXT
| FUNCTION
| GENERAL
| GENERATED
| GET_FORMAT
| GET
| GLOBAL
| GOTO
| GRANT
| GRANTS
| GROUP
| HANDLER
| HARD
| HASH
| HAVING
| HELP
| HIGH_PRIORITY
| HISTORY
| HOST
| HOSTS
| HOUR
| HOUR_MICROSECOND
| HOUR_MINUTE
| HOUR_SECOND
| ID
| IDENTIFIED
| IF
| IGNORE
| IGNORED
| IGNORE_DOMAIN_IDS
| IGNORE_SERVER_IDS
| IMMEDIATE
| IMPORT
| INTERSECT
| IN
| INCREMENT
| INDEX
| INDEXES
| INFILE
| INITIAL_SIZE
| INNER
| INOUT
| INSENSITIVE
| INSERT
```

INSERT_METHOD	
INSTALL	
INT	
INT1	
INT2	
INT3	
INT4	
INT8	
INTEGER	
INTERVAL	
INVISIBLE	
INTO	
IO	
IO_THREAD	
IPC	
IS	
ISOLATION	
ISOPEN	
ISSUER	
ITERATE	
INVOKER	
JOIN	
JSON	
JSON_TABLE	
KEY	
KEYS	
KEY_BLOCK_SIZE	
KILL	
LANGUAGE	
LAST	
LAST_VALUE	
LASTVAL	
LEADING	
LEAVE	
LEAVES	
LEFT	
LESS	
LEVEL	
LIKE	
LIMIT	
LINEAR	
LINES	
LIST	
LOAD	
LOCAL	
LOCALTIME	
LOCALTIMESTAMP	
LOCK	
LOCKED	
LOCKS	
LOGFILE	
LOGS	
LONG	
LONGBLOB	
LONGTEXT	
LOOP	
LOW_PRIORITY	
MASTER	
MASTER_CONNECT_RETRY	
MASTER_DELAY	
MASTER_GTID_POS	
MASTER_HOST	
MASTER_LOG_FILE	
MASTER_LOG_POS	
MASTER_PASSWORD	
MASTER_PORT	
MASTER_SERVER_ID	
MASTER_SSL	
MASTER_SSL_CA	
MASTER_SSL_CAPATH	
MASTER_SSL_CERT	
MASTER_SSL_CIPHER	

```
| MASTER_SSL_CRL
| MASTER_SSL_CRLPATH
| MASTER_SSL_KEY
| MASTER_SSL_VERIFY_SERVER_CERT
| MASTER_USER
| MASTER_USE_GTID
| MASTER_HEARTBEAT_PERIOD
| MATCH
| MAX_CONNECTIONS_PER_HOUR
| MAX_QUERIES_PER_HOUR
| MAX_ROWS
| MAX_SIZE
| MAX_STATEMENT_TIME
| MAX_UPDATES_PER_HOUR
| MAX_USER_CONNECTIONS
| MAXVALUE
| MEDIUM
| MEDIUMBLOB
| MEDIUMINT
| MEDIUMTEXT
| MEMORY
| MERGE
| MESSAGE_TEXT
| MICROSECOND
| MIDDLEINT
| MIGRATE
| MINUS
| MINUTE
| MINUTE_MICROSECOND
| MINUTE_SECOND
| MINVALUE
| MIN_ROWS
| MOD
| MODE
| MODIFIES
| MODIFY
| MONITOR
| MONTH
| MUTEX
| MYSQL
| MYSQL_ERRNO
| NAME
| NAMES
| NATIONAL
| NATURAL
| NCHAR
| NESTED
| NEVER
| NEW
| NEXT
| NEXTVAL
| NO
| NOMAXVALUE
| NOMINVALUE
| NOCACHE
| NOCYCLE
| NO_WAIT
| NOWAIT
| NODEGROUP
| NONE
| NOT
| NOTFOUND
| NO_WRITE_TO_BINLOG
| NULL
| NUMBER
| NUMERIC
| NVARCHAR
| OF
| OFFSET
| OLD_PASSWORD
| ON
```

ONE	
ONLINE	
ONLY	
OPEN	
OPTIMIZE	
OPTIONS	
OPTION	
OPTIONALLY	
OR	
ORDER	
ORDINALITY	
OTHERS	
OUT	
OUTER	
OUTFILE	
OVER	
OVERLAPS	
OWNER	
PACKAGE	
PACK_KEYS	
PAGE	
PAGE_CHECKSUM	
PARSER	
PARSE_VCOL_EXPR	
PATH	
PERIOD	
PARTIAL	
PARTITION	
PARTITIONING	
PARTITIONS	
PASSWORD	
PERSISTENT	
PHASE	
PLUGIN	
PLUGINS	
PORT	
PORTION	
PRECEDES	
PRECEDING	
PRECISION	
PREPARE	
PRESERVE	
PREV	
PREVIOUS	
PRIMARY	
PRIVILEGES	
PROCEDURE	
PROCESS	
PROCESSLIST	
PROFILE	
PROFILES	
PROXY	
PURGE	
QUARTER	
QUERY	
QUICK	
RAISE	
RANGE	
RAW	
READ	
READ_ONLY	
READ_WRITE	
READS	
REAL	
REBUILD	
RECOVER	
RECURSIVE	
REDO_BUFFER_SIZE	
REDOFILE	
REDUNDANT	
REFERENCES	
REGEXP	

```
| RELAY
| RELAYLOG
| RELAY_LOG_FILE
| RELAY_LOG_POS
| RELAY_THREAD
| RELEASE
| RELOAD
| REMOVE
| RENAME
| REORGANIZE
| REPAIR
| REPEATABLE
| REPLACE
| REPLAY
| REPLICA
| REPLICAS
| REPLICA_POS
| REPLICATION
| REPEAT
| REQUIRE
| RESET
| RESIGNAL
| RESTART
| RESTORE
| RESTRICT
| RESUME
| RETURNED_SQLSTATE
| RETURN
| RETURNING
| RETURNS
| REUSE
| REVERSE
| REVOKE
| RIGHT
| RLIKE
| ROLE
| ROLLBACK
| ROLLUP
| ROUTINE
| ROW
| ROWCOUNT
| ROWNUM
| ROWS
| ROWTYPE
| ROW_COUNT
| ROW_FORMAT
| RTREE
| SAVEPOINT
| SCHEDULE
| SCHEMA
| SCHEMA_NAME
| SCHEMAS
| SECOND
| SECOND_MICROSECOND
| SECURITY
| SELECT
| SENSITIVE
| SEPARATOR
| SEQUENCE
| SERIAL
| SERIALIZABLE
| SESSION
| SERVER
| SET
| SETVAL
| SHARE
| SHOW
| SHUTDOWN
| SIGNAL
| SIGNED
| SIMPLE
```

SKIP	
SLAVE	
SLAVES	
SLAVE_POS	
SLOW	
SNAPSHOT	
SMALLINT	
SOCKET	
SOFT	
SOME	
SONAME	
SOUNDS	
SOURCE	
STAGE	
STORED	
SPATIAL	
SPECIFIC	
REF_SYSTEM_ID	
SQL	
SQLEXCEPTION	
SQLSTATE	
SQLWARNING	
SQL_BIG_RESULT	
SQL_BUFFER_RESULT	
SQL_CACHE	
SQL_CALC_FOUND_ROWS	
SQL_NO_CACHE	
SQL_SMALL_RESULT	
SQL_THREAD	
SQL_TSI_SECOND	
SQL_TSI_MINUTE	
SQL_TSI_HOUR	
SQL_TSI_DAY	
SQL_TSI_WEEK	
SQL_TSI_MONTH	
SQL_TSI_QUARTER	
SQL_TSI_YEAR	
SSL	
START	
STARTING	
STARTS	
STATEMENT	
STATS_AUTO_RECALC	
STATS_PERSISTENT	
STATS_SAMPLE_PAGES	
STATUS	
STOP	
STORAGE	
STRAIGHT_JOIN	
STRING	
SUBCLASS_ORIGIN	
SUBJECT	
SUBPARTITION	
SUBPARTITIONS	
SUPER	
SUSPEND	
SWAPS	
SWITCHES	
SYSDATE	
SYSTEM	
SYSTEM_TIME	
TABLE	
TABLE_NAME	
TABLES	
TABLESPACE	
TABLE_CHECKSUM	
TEMPORARY	
TEMPTABLE	
TERMINATED	
TEXT	
THAN	
THEN	

```
| TIES
| TIME
| TIMESTAMP
| TIMESTAMPADD
| TIMESTAMPDIFF
| TINYBLOB
| TINYINT
| TINYTEXT
| TO
| TRAILING
| TRANSACTION
| TRANSACTIONAL
| THREADS
| TRIGGER
| TRIGGERS
| TRUE
| TRUNCATE
| TYPE
| TYPES
| UNBOUNDED
| UNCOMMITTED
| UNDEFINED
| UNDO_BUFFER_SIZE
| UNDOFILE
| UNDO
| UNICODE
| UNION
| UNIQUE
| UNKNOWN
| UNLOCK
| UNINSTALL
| UNSIGNED
| UNTIL
| UPDATE
| UPGRADE
| USAGE
| USE
| USER
| USER_RESOURCES
| USE_FRM
| USING
| UTC_DATE
| UTC_TIME
| UTC_TIMESTAMP
| VALUE
| VALUES
| VARBINARY
| VARCHAR
| VARCHARACTER
| VARCHAR2
| VARIABLES
| VARYING
| VIA
| VIEW
| VIRTUAL
| VISIBLE
| VERSIONING
| WAIT
| WARNINGS
| WEEK
| WEIGHT_STRING
| WHEN
| WHERE
| WHILE
| WINDOW
| WITH
| WITHIN
| WITHOUT
| WORK
| WRAPPER
| WRITE
```

```

| X509          |
| XOR          |
| XA           |
| XML          |
| YEAR         |
| YEAR_MONTH   |
| ZEROFILL     |
| ||           |
+-----+
694 rows in set (0.000 sec)

```

See Also

- [Reserved Words](#)

1.1.2.9.1.1.26 Information Schema LOCALES Table

Description

The `Information Schema LOCALES` table contains a list of all compiled-in locales. It is only available if the [LOCALES plugin](#) has been installed.

It contains the following columns:

Column	Description
ID	Row ID.
NAME	Locale name, for example <code>en_GB</code> .
DESCRIPTION	Locale description, for example <code>English - United Kingdom</code> .
MAX_MONTH_NAME_LENGTH	Numeric length of the longest month in the locale
MAX_DAY_NAME_LENGTH	Numeric length of the longest day name in the locale.
DECIMAL_POINT	Decimal point character (some locales use a comma).
THOUSAND_SEP	Thousand's character separator,
ERROR_MESSAGE_LANGUAGE	Error message language.

The table is not a standard Information Schema table, and is a MariaDB extension.

The `SHOW LOCALES` statement returns a subset of the information.

Example

```

SELECT * FROM information_schema.LOCALES;
+----+----+----+----+----+----+----+----+
| ID | NAME | DESCRIPTION          | MAX_MONTH_NAME_LENGTH | MAX_DAY_NAME_LENGTH | DECIMAL_POINT | THOUSAND_SEP | ERROR_MESSAGE_LANGUAGE |
+----+----+----+----+----+----+----+----+
|  0 | en_US | English - United States | 9 | 9 | . | , | en |
|  1 | en_GB | English - United Kingdom | 9 | 9 | . | , | en |
|  2 | ja_JP | Japanese - Japan | 3 | 3 | . | , | ja |
|  3 | sv_SE | Swedish - Sweden | 9 | 7 | , | . | si |
|  4 | de_DE | German - Germany | 9 | 10 | , | . | ge |
|  5 | fr_FR | French - France | 9 | 8 | , | . | fi |
|  6 | ar_AE | Arabic - United Arab Emirates | 6 | 8 | . | , | ei |
|  7 | ar_BH | Arabic - Bahrain | 6 | 8 | . | , | ei |
|  8 | ar_JO | Arabic - Jordan | 12 | 8 | . | , | ei |
...
| 106 | no_NO | Norwegian - Norway | 9 | 7 | , | . | no |
| 107 | sv_FI | Swedish - Finland | 9 | 7 | , | . | si |
| 108 | zh_HK | Chinese - Hong Kong SAR | 3 | 3 | . | , | ei |
| 109 | el_GR | Greek - Greece | 11 | 9 | . | . | gr |
+----+----+----+----+----+----+----+----+

```

1.1.2.9.1.1.27 Information Schema METADATA_LOCK_INFO Table

The `Information Schema METADATA_LOCK_INFO` table is created by the `metadata_lock_info` plugin. It shows active `metadata locks` and user locks (the locks acquired with `GET_LOCK`).

It has the following columns:

Column	Description
THREAD_ID	
LOCK_MODE	One of <code>MDL_INTENTION_EXCLUSIVE</code> , <code>MDL_SHARED</code> , <code>MDL_SHARED_HIGH_PRIO</code> , <code>MDL_SHARED_READ</code> , <code>MDL_SHARED_READ_ONLY</code> , <code>MDL_SHARED_WRITE</code> , <code>MDL_SHARED_NO_WRITE</code> , <code>MDL_SHARED_NO_READ_WRITE</code> , <code>MDL_SHARED_UPGRADABLE</code> or <code>MDL_EXCLUSIVE</code> .
LOCK_DURATION	One of <code>MDL_STATEMENT</code> , <code>MDL_TRANSACTION</code> or <code>MDL_EXPLICIT</code>
LOCK_TYPE	One of Global read lock , Schema metadata lock , Table metadata lock , Stored function metadata lock , Stored procedure metadata lock , Trigger metadata lock , Event metadata lock , Commit lock or User lock .
TABLE_SCHEMA	
TABLE_NAME	

"LOCK_MODE" Descriptions

The `LOCK_MODE` column can have the following values:

Value	Description
<code>MDL_INTENTION_EXCLUSIVE</code>	An intention exclusive metadata lock (IX). Used only for scoped locks. Owner of this type of lock can acquire upgradable exclusive locks on individual objects. Compatible with other IX locks, but is incompatible with scoped S and X locks. IX lock is taken in SCHEMA namespace when we intend to modify object metadata. Object may refer table, stored procedure, trigger, view/etc.
<code>MDL_SHARED</code>	A shared metadata lock (S). To be used in cases when we are interested in object metadata only and there is no intention to access object data (e.g. for stored routines or during preparing prepared statements). We also mis-use this type of lock for open HANDLERs, since lock acquired by this statement has to be compatible with lock acquired by LOCK TABLES ... WRITE statement, i.e. SNRW (We can't get by by acquiring S lock at HANDLER ... OPEN time and upgrading it to SR lock for HANDLER ... READ as it doesn't solve problem with need to abort DML statements which wait on table level lock while having open HANDLER in the same connection). To avoid deadlock which may occur when SNRW lock is being upgraded to X lock for table on which there is an active S lock which is owned by thread which waits in its turn for table-level lock owned by thread performing upgrade we have to use <code>thr_abort_locks_for_thread()</code> facility in such situation. This problem does not arise for locks on stored routines as we don't use SNRW locks for them. It also does not arise when S locks are used during PREPARE calls as table-level locks are not acquired in this case. This lock is taken for global read lock, when caching a stored procedure in memory for the duration of the transaction and for tables used by prepared statements.
<code>MDL_SHARED_HIGH_PRIO</code>	A high priority shared metadata lock. Used for cases when there is no intention to access object data (i.e. data in the table). "High priority" means that, unlike other shared locks, it is granted ignoring pending requests for exclusive locks. Intended for use in cases when we only need to access metadata and not data, e.g. when filling an INFORMATION_SCHEMA table. Since SH lock is compatible with SNRW lock, the connection that holds SH lock lock should not try to acquire any kind of table-level or row-level lock, as this can lead to a deadlock. Moreover, after acquiring SH lock, the connection should not wait for any other resource, as it might cause starvation for X locks and a potential deadlock during upgrade of SNW or SNRW to X lock (e.g. if the upgrading connection holds the resource that is being waited for).
<code>MDL_SHARED_READ</code>	A shared metadata lock (SR) for cases when there is an intention to read data from table. A connection holding this kind of lock can read table metadata and read table data (after acquiring appropriate table and row-level locks). This means that one can only acquire TL_READ, TL_READ_NO_INSERT, and similar table-level locks on table if one holds SR MDL lock on it. To be used for tables in SELECTs, subqueries, and LOCK TABLE ... READ statements.
<code>MDL_SHARED_WRITE</code>	A shared metadata lock (SW) for cases when there is an intention to modify (and not just read) data in the table. A connection holding SW lock can read table metadata and modify or read table data (after acquiring appropriate table and row-level locks). To be used for tables to be modified by INSERT, UPDATE, DELETE statements, but not LOCK TABLE ... WRITE or DDL). Also taken by SELECT ... FOR UPDATE.

MDL_SHARED_UPGRADABLE	An upgradable shared metadata lock for cases when there is an intention to modify (and not just read) data in the table. Can be upgraded to MDL_SHARED_NO_WRITE and MDL_EXCLUSIVE. A connection holding SU lock can read table metadata and modify or read table data (after acquiring appropriate table and row-level locks). To be used for the first phase of ALTER TABLE.
MDL_SHARED_READ_ONLY	A shared metadata lock for cases when we need to read data from table and block all concurrent modifications to it (for both data and metadata). Used by LOCK TABLES READ statement.
MDL_SHARED_NO_WRITE	An upgradable shared metadata lock which blocks all attempts to update table data, allowing reads. A connection holding this kind of lock can read table metadata and read table data. Can be upgraded to X metadata lock. Note, that since this type of lock is not compatible with SNRW or SW lock types, acquiring appropriate engine-level locks for reading (TL_READ* for MyISAM, shared row locks in InnoDB) should be contention-free. To be used for the first phase of ALTER TABLE, when copying data between tables, to allow concurrent SELECTs from the table, but not UPDATEs.
MDL_SHARED_NO_READ_WRITE	An upgradable shared metadata lock which allows other connections to access table metadata, but not data. It blocks all attempts to read or update table data, while allowing INFORMATION_SCHEMA and SHOW queries. A connection holding this kind of lock can read table metadata modify and read table data. Can be upgraded to X metadata lock. To be used for LOCK TABLES WRITE statement. Not compatible with any other lock type except S and SH.
MDL_EXCLUSIVE	An exclusive metadata lock (X). A connection holding this lock can modify both table's metadata and data. No other type of metadata lock can be granted while this lock is held. To be used for CREATE/DROP/RENAME TABLE statements and for execution of certain phases of other DDL statements.

Examples

User lock:

```
SELECT GET_LOCK('abc',1000);
+-----+
| GET_LOCK('abc',1000) |
+-----+
|          1 |
+-----+

SELECT * FROM information_schema.METADATA_LOCK_INFO;
+-----+-----+-----+-----+-----+
| THREAD_ID | LOCK_MODE      | LOCK_DURATION | LOCK_TYPE | TABLE_SCHEMA | TABLE_NAME |
+-----+-----+-----+-----+-----+
|       61 | MDL_SHARED_NO_READ_WRITE | MDL_EXPLICIT | User lock | abc           |          |
+-----+-----+-----+-----+-----+
```

Table metadata lock:

```
START TRANSACTION;

INSERT INTO t VALUES (1,2);

SELECT * FROM information_schema.METADATA_LOCK_INFO \G
***** 1. row *****
THREAD_ID: 4
LOCK_MODE: MDL_SHARED_WRITE
LOCK_DURATION: MDL_TRANSACTION
LOCK_TYPE: Table metadata lock
TABLE_SCHEMA: test
TABLE_NAME: t
```

```
SELECT * FROM information_schema.METADATA_LOCK_INFO;
+-----+-----+-----+-----+-----+
| THREAD_ID | LOCK_MODE | LOCK_DURATION | LOCK_TYPE | TABLE_SCHEMA | TABLE_NAME |
+-----+-----+-----+-----+-----+
| 31 | MDL_INTENTION_EXCLUSIVE | MDL_EXPLICIT | Global read lock |    | |
| 31 | MDL_INTENTION_EXCLUSIVE | MDL_EXPLICIT | Commit lock |    |
| 31 | MDL_INTENTION_EXCLUSIVE | MDL_EXPLICIT | Schema metadata lock | dbname |
| 31 | MDL_SHARED_NO_READ_WRITE | MDL_EXPLICIT | Table metadata lock | dbname | exotics |
+-----+-----+-----+-----+-----+
```

See also

- [metadata locks](#)
- [Performance Schema metadata_locks table](#)
- [GET_LOCK\(\)](#).

1.1.2.9.1.1.28 Information Schema MROONGA_STATS Table

The `Information Schema MROONGA_STATS` table only exists if the [Mroonga](#) storage engine is installed, and contains information about its activities.

Column	Description
VERSION	Mroonga version.
rows_written	Number of rows written into Mroonga tables.
rows_read	Number of rows read from all Mroonga tables.

This table always contains 1 row.

1.1.2.9.1.1.29 Information Schema OPTIMIZER_TRACE Table

MariaDB starting with [10.4.3](#)

[Optimizer Trace](#) was introduced in [MariaDB 10.4.3](#).

Description

The `Information Schema OPTIMIZER_TRACE` table contains [Optimizer Trace](#) information.

It contains the following columns:

Column	Description
QUERY	Displays the query that was asked to be traced.
TRACE	A JSON document displaying the stats we collected when the query was run.
MISSING_BYTES_BEYOND_MAX_MEM_SIZE	For huge trace, where the trace is truncated due to the optimizer_trace_max_mem_size limit being reached, displays the bytes that are missing in the trace
INSUFFICIENT_PRIVILEGES	Set to 1 if the user running the trace does not have the privileges to see the trace.

Structure:

```
SHOW CREATE TABLE INFORMATION_SCHEMA.OPTIMIZER_TRACE \G
*****
1. row *****
  Table: OPTIMIZER_TRACE
Create Table: CREATE TEMPORARY TABLE `OPTIMIZER_TRACE` (
  `QUERY` longtext NOT NULL DEFAULT '',
  `TRACE` longtext NOT NULL DEFAULT '',
  `MISSING_BYTES_BEYOND_MAX_MEM_SIZE` int(20) NOT NULL DEFAULT 0,
  `INSUFFICIENT_PRIVILEGES` tinyint(1) NOT NULL DEFAULT 0
) ENGINE=Aria DEFAULT CHARSET=utf8 PAGE_CHECKSUM=0
```

1.1.2.9.1.1.30 Information Schema PARAMETERS Table

The `Information Schema PARAMETERS` table stores information about [stored procedures](#) and [stored functions](#) parameters.

It contains the following columns:

Column	Description
--------	-------------

SPECIFIC_CATALOG	Always def.
SPECIFIC_SCHEMA	Database name containing the stored routine parameter.
SPECIFIC_NAME	Stored routine name.
ORDINAL_POSITION	Ordinal position of the parameter, starting at 1. 0 for a function RETURNS clause.
PARAMETER_MODE	One of IN , OUT , INOUT or NULL for RETURNS.
PARAMETER_NAME	Name of the parameter, or NULL for RETURNS.
DATA_TYPE	The column's data type .
CHARACTER_MAXIMUM_LENGTH	Maximum length.
CHARACTER_OCTET_LENGTH	Same as the CHARACTER_MAXIMUM_LENGTH except for multi-byte character sets .
NUMERIC_PRECISION	For numeric types, the precision (number of significant digits) for the column. NULL if not a numeric field.
NUMERIC_SCALE	For numeric types, the scale (significant digits to the right of the decimal point). NULL if not a numeric field.
DATETIME_PRECISION	Fractional-seconds precision, or NULL if not a time data type .
CHARACTER_SET_NAME	Character set if a non-binary string data type , otherwise NULL .
COLLATION_NAME	Collation if a non-binary string data type , otherwise NULL .
DTD_IDENTIFIER	Description of the data type.
ROUTINE_TYPE	PROCEDURE OR FUNCTION .

Information from this table is similar to that found in the `param_list` column in the `mysql.proc` table, and the output of the `SHOW CREATE PROCEDURE` and `SHOW CREATE FUNCTION` statements.

To obtain information about the routine itself, you can query the [Information Schema ROUTINES table](#).

Example

```
SELECT * FROM information_schema.PARAMETERS
LIMIT 1 \G
*****
1. row *****
SPECIFIC_CATALOG: def
SPECIFIC_SCHEMA: accounts
SPECIFIC_NAME: user_counts
ORDINAL_POSITION: 1
PARAMETER_MODE: IN
PARAMETER_NAME: user_order
DATA_TYPE: varchar
CHARACTER_MAXIMUM_LENGTH: 255
CHARACTER_OCTET_LENGTH: 765
NUMERIC_PRECISION: NULL
NUMERIC_SCALE: NULL
DATETIME_PRECISION: NULL
CHARACTER_SET_NAME: utf8
COLLATION_NAME: utf8_general_ci
DTD_IDENTIFIER: varchar(255)
ROUTINE_TYPE: PROCEDURE
```

1.1.2.9.1.1.31 Information Schema PARTITIONS Table

The [Information Schema PARTITIONS](#) contains information about table partitions, with each record corresponding to a single partition or subpartition of a partitioned table. Each non-partitioned table also has a record in the PARTITIONS table, but most of the values are NULL .

It contains the following columns:

Column	Description
TABLE_CATALOG	Always def.
TABLE_SCHEMA	Database name.

TABLE_NAME	Table name containing the partition.
PARTITION_NAME	Partition name.
SUBPARTITION_NAME	Subpartition name, or <code>NULL</code> if not a subpartition.
PARTITION_ORDINAL_POSITION	Order of the partition starting from 1.
SUBPARTITION_ORDINAL_POSITION	Order of the subpartition starting from 1.
PARTITION_METHOD	The partitioning type; one of <code>RANGE</code> , <code>LIST</code> , <code>HASH</code> , <code>LINEAR HASH</code> , <code>KEY</code> or <code>LINEAR KEY</code> .
SUBPARTITION_METHOD	Subpartition type; one of <code>HASH</code> , <code>LINEAR HASH</code> , <code>KEY</code> or <code>LINEAR KEY</code> , or <code>NULL</code> if not a subpartition.
PARTITION_EXPRESSION	Expression used to create the partition by the <code>CREATE TABLE</code> or <code>ALTER TABLE</code> statement.
SUBPARTITION_EXPRESSION	Expression used to create the subpartition by the <code>CREATE TABLE</code> or <code>ALTER TABLE</code> statement, or <code>NULL</code> if not a subpartition.
PARTITION_DESCRIPTION	For a <code>RANGE</code> partition, contains either <code>MAXINTEGER</code> or an integer, as set in the <code>VALUES LESS THAN</code> clause. For a <code>LIST</code> partition, contains a comma-separated list of integers, as set in the <code>VALUES IN</code> . <code>NULL</code> if another type of partition.
TABLE_ROWS	Number of rows in the table (may be an estimate for some storage engines).
AVG_ROW_LENGTH	Average row length, that is <code>DATA_LENGTH</code> divided by <code>TABLE_ROWS</code>
DATA_LENGTH	Total number of bytes stored in all rows of the partition.
MAX_DATA_LENGTH	Maximum bytes that could be stored in the partition.
INDEX_LENGTH	Size in bytes of the partition index file.
DATA_FREE	Unused bytes allocated to the partition.
CREATE_TIME	Time the partition was created
UPDATE_TIME	Time the partition was last modified.
CHECK_TIME	Time the partition was last checked, or <code>NULL</code> for storage engines that don't record this information.
CHECKSUM	Checksum value, or <code>NULL</code> if none.
PARTITION_COMMENT	Partition comment, truncated to 80 characters, or an empty string if no comment.
NODEGROUP	Node group, only used for MySQL Cluster, defaults to <code>0</code> .
TABLESPACE_NAME	Always <code>default</code> .

1.1.2.9.1.1.32 Information Schema PLUGINS Table

The `Information Schema` `PLUGINS` table contains information about `server plugins`.

It contains the following columns:

Column	Description
PLUGIN_NAME	Name of the plugin.
PLUGIN_VERSION	Version from the plugin's general type descriptor.
PLUGIN_STATUS	Plugin status, one of <code>ACTIVE</code> , <code>INACTIVE</code> , <code>DISABLED</code> or <code>DELETED</code> .
PLUGIN_TYPE	Plugin type; <code>STORAGE ENGINE</code> , <code>INFORMATION_SCHEMA</code> , <code>AUTHENTICATION</code> , <code>REPLICATION</code> , <code>DAEMON</code> or <code>AUDIT</code> .
PLUGIN_TYPE_VERSION	Version from the plugin's type-specific descriptor.
PLUGIN_LIBRARY	Plugin's shared object file name, located in the directory specified by the <code>plugin_dir</code> system variable, and used by the <code>INSTALL PLUGIN</code> and <code>UNINSTALL PLUGIN</code> statements. <code>NULL</code> if the plugin is compiled in and cannot be uninstalled.
PLUGIN_LIBRARY_VERSION	Version from the plugin's API interface.
PLUGIN_AUTHOR	Author of the plugin.
PLUGIN_DESCRIPTION	Description.
PLUGIN_LICENSE	Plugin's licence.

LOAD_OPTION	How the plugin was loaded; one of OFF , ON , FORCE or FORCE_PLUS_PERMANENT . See Installing Plugins .
PLUGIN_Maturity	Plugin's maturity level; one of Unknown , Experimental , Alpha , Beta , 'Gamma , and Stable .
PLUGIN_AUTH_VERSION	Plugin's version as determined by the plugin author. An example would be '0.99 beta 1'.

It provides a superset of the information shown by the `SHOW PLUGINS` statement. For specific information about storage engines (a particular type of plugins), see the `information_schema.ENGINES` table and the `SHOW ENGINES` statement.

This table provides a subset of the Information Schema `information_schema.ALL_PLUGINS` table, which contains all available plugins, installed or not.

The table is not a standard Information Schema table, and is a MariaDB extension.

Examples

The easiest way to get basic information on plugins is with `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_LOCKS	ACTIVE	INFORMATION SCHEMA	NULL	GPL
INNODB_LOCK_WAITS	ACTIVE	INFORMATION SCHEMA	NULL	GPL
INNODB_CMP	ACTIVE	INFORMATION SCHEMA	NULL	GPL
INNODB_CMP_RESET	ACTIVE	INFORMATION SCHEMA	NULL	GPL
INNODB_CMPMEM	ACTIVE	INFORMATION SCHEMA	NULL	GPL
INNODB_CMPMEM_RESET	ACTIVE	INFORMATION SCHEMA	NULL	GPL
INNODB_BUFFER_PAGE	ACTIVE	INFORMATION SCHEMA	NULL	GPL
INNODB_BUFFER_PAGE_LRU	ACTIVE	INFORMATION SCHEMA	NULL	GPL
INNODB_BUFFER_POOL_STATS	ACTIVE	INFORMATION SCHEMA	NULL	GPL
INNODB_METRICS	ACTIVE	INFORMATION SCHEMA	NULL	GPL
INNODB_FT_DEFAULT_STOPWORD	ACTIVE	INFORMATION SCHEMA	NULL	GPL
INNODB_FT_INSERTED	ACTIVE	INFORMATION SCHEMA	NULL	GPL
INNODB_FT_DELETED	ACTIVE	INFORMATION SCHEMA	NULL	GPL
INNODB_FT_BEING_DELETED	ACTIVE	INFORMATION SCHEMA	NULL	GPL
INNODB_FT_CONFIG	ACTIVE	INFORMATION SCHEMA	NULL	GPL
INNODB_FT_INDEX_CACHE	ACTIVE	INFORMATION SCHEMA	NULL	GPL
INNODB_FT_INDEX_TABLE	ACTIVE	INFORMATION SCHEMA	NULL	GPL
INNODB_SYS_TABLES	ACTIVE	INFORMATION SCHEMA	NULL	GPL
INNODB_SYS_TABLESTATS	ACTIVE	INFORMATION SCHEMA	NULL	GPL
INNODB_SYS_INDEXES	ACTIVE	INFORMATION SCHEMA	NULL	GPL
INNODB_SYS_COLUMNS	ACTIVE	INFORMATION SCHEMA	NULL	GPL
INNODB_SYS_FIELDS	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

```
SELECT LOAD_OPTION
FROM INFORMATION_SCHEMA.PLUGINS
WHERE PLUGIN_NAME LIKE 'tokudb';
Empty set
```

The equivalent `SELECT` query would be:

```
SELECT PLUGIN_NAME, PLUGIN_STATUS,
PLUGIN_TYPE, PLUGIN_LIBRARY, PLUGIN_LICENSE
FROM INFORMATION_SCHEMA.PLUGINS;
```

Other `SELECT` queries can be used to see additional information. For example:

```

SELECT PLUGIN_NAME, PLUGIN_DESCRIPTION,
PLUGIN_MATURITY, PLUGIN_AUTH_VERSION
FROM INFORMATION_SCHEMA.PLUGINS
WHERE PLUGIN_TYPE='STORAGE ENGINE'
ORDER BY PLUGIN_MATURITY [G]

***** 1. row *****
PLUGIN_NAME: FEDERATED
PLUGIN_DESCRIPTION: FederatedX pluggable storage engine
PLUGIN_MATURITY: Beta
PLUGIN_AUTH_VERSION: 2.1
***** 2. row *****
PLUGIN_NAME: Aria
PLUGIN_DESCRIPTION: Crash-safe tables with MyISAM heritage
PLUGIN_MATURITY: Gamma
PLUGIN_AUTH_VERSION: 1.5
***** 3. row *****
PLUGIN_NAME: PERFORMANCE_SCHEMA
PLUGIN_DESCRIPTION: Performance Schema
PLUGIN_MATURITY: Gamma
PLUGIN_AUTH_VERSION: 0.1
***** 4. row *****
PLUGIN_NAME: binlog
PLUGIN_DESCRIPTION: This is a pseudo storage engine to represent the binlog in a transaction
PLUGIN_MATURITY: Stable
PLUGIN_AUTH_VERSION: 1.0
***** 5. row *****
PLUGIN_NAME: MEMORY
PLUGIN_DESCRIPTION: Hash based, stored in memory, useful for temporary tables
PLUGIN_MATURITY: Stable
PLUGIN_AUTH_VERSION: 1.0
***** 6. row *****
PLUGIN_NAME: MyISAM
PLUGIN_DESCRIPTION: MyISAM storage engine
PLUGIN_MATURITY: Stable
PLUGIN_AUTH_VERSION: 1.0
***** 7. row *****
PLUGIN_NAME: MRG_MyISAM
PLUGIN_DESCRIPTION: Collection of identical MyISAM tables
PLUGIN_MATURITY: Stable
PLUGIN_AUTH_VERSION: 1.0
***** 8. row *****
PLUGIN_NAME: CSV
PLUGIN_DESCRIPTION: CSV storage engine
PLUGIN_MATURITY: Stable
PLUGIN_AUTH_VERSION: 1.0
***** 9. row *****
PLUGIN_NAME: InnoDB
PLUGIN_DESCRIPTION: Supports transactions, row-level locking, and foreign keys
PLUGIN_MATURITY: Stable
PLUGIN_AUTH_VERSION: 1.2.5
***** 10. row *****
PLUGIN_NAME: BLACKHOLE
PLUGIN_DESCRIPTION: /dev/null storage engine (anything you write to it disappears)
PLUGIN_MATURITY: Stable
PLUGIN_AUTH_VERSION: 1.0
***** 11. row *****
PLUGIN_NAME: ARCHIVE
PLUGIN_DESCRIPTION: Archive storage engine
PLUGIN_MATURITY: Stable
PLUGIN_AUTH_VERSION: 1.0
***** 12. row *****
PLUGIN_NAME: partition
PLUGIN_DESCRIPTION: Partition Storage Engine Helper
PLUGIN_MATURITY: Stable
PLUGIN_AUTH_VERSION: 1.0

```

Check if a given plugin is available:

```
SELECT LOAD_OPTION
FROM INFORMATION_SCHEMA.PLUGINS
WHERE PLUGIN_NAME LIKE 'tokudb';
Empty set
```

Show authentication plugins:

```
SELECT PLUGIN_NAME, LOAD_OPTION
FROM INFORMATION_SCHEMA.PLUGINS
WHERE PLUGIN_TYPE LIKE 'authentication' 🔗

***** 1. row *****
PLUGIN_NAME: mysql_native_password
LOAD_OPTION: FORCE
***** 2. row *****
PLUGIN_NAME: mysql_old_password
LOAD_OPTION: FORCE
```

See Also

- [List of Plugins](#)
- [Plugin Overview](#)
- [SHOW PLUGINS](#)
- [INSTALL PLUGIN](#)
- [INSTALL SONAME](#)
- [UNINSTALL PLUGIN](#)
- [UNINSTALL SONAME](#)

1.1.2.9.1.1.33 Information Schema PROCESSLIST Table

The `Information Schema` `PROCESSLIST` table contains information about running threads.

Similar information can also be returned with the `SHOW [FULL] PROCESSLIST` statement, or the `mysqladmin processlist` command.

It contains the following columns:

Contents

1. [Example](#)
2. [See Also](#)

Column	Description	Added
ID	Connection identifier.	
USER	MariaDB User.	
HOST	Connecting host.	
DB	Default database, or <code>NULL</code> if none.	
COMMAND	Type of command running, corresponding to the <code>Com_ status variables</code> . See Thread Command Values .	
TIME	Seconds that the thread has spent on the current <code>COMMAND</code> so far.	
STATE	Current state of the thread. See Thread States .	
INFO	Statement the thread is executing, or <code>NULL</code> if none.	
TIME_MS	Time in milliseconds with microsecond precision that the thread has spent on the current <code>COMMAND</code> so far (see more).	MariaDB 5.1
STAGE	The stage the process is currently in.	MariaDB 5.3
MAX_STAGE	The maximum number of stages.	MariaDB 5.3
PROGRESS	The progress of the process within the current stage (0-100%).	MariaDB 5.3
MEMORY_USED	Memory in bytes used by the thread.	MariaDB 10.0.1

EXAMINED_ROWS	Rows examined by the thread. Only updated by UPDATE, DELETE, and similar statements. For SELECT and other statements, the value remains zero.	MariaDB 10.0.1
QUERY_ID	Query ID.	MariaDB 10.0.5
INFO_BINARY	Binary data information	MariaDB 10.1.5
TID	Thread ID (MDEV-6756)	MariaDB 10.1.8

Note that as a difference to MySQL, in MariaDB the `TIME` column (and also the `TIME_MS` column) are not affected by any setting of `@TIMESTAMP`. This means that it can be reliably used also for threads that change `@TIMESTAMP` (such as the `replication` SQL thread). See also [MySQL Bug #22047](#).

As a consequence of this, the `TIME` column of `SHOW FULL PROCESSLIST` and `INFORMATION_SCHEMA.PROCESSLIST` can not be used to determine if a slave is lagging behind. For this, use instead the `Seconds_Behind_Master` column in the output of `SHOW SLAVE STATUS`.

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.. To retrieve a similar "total" Progress value from `information_schema.PROCESSLIST` as the one from `SHOW PROCESSLIST`, use

```
SELECT CASE WHEN Max_Stage < 2 THEN Progress ELSE (Stage-1)/Max_Stage*100+Progress/Max_Stage END
AS Progress FROM INFORMATION_SCHEMA.PROCESSLIST;
```

Example

```
SELECT * FROM INFORMATION_SCHEMA.PROCESSLIST\G
*****
1. row *****
    ID: 9
    USER: msandbox
    HOST: localhost
    DB: NULL
    COMMAND: Query
    TIME: 0
    STATE: Filling schema table
    INFO: SELECT * FROM INFORMATION_SCHEMA.PROCESSLIST
    TIME_MS: 0.351
    STAGE: 0
    MAX_STAGE: 0
    PROGRESS: 0.000
    MEMORY_USED: 85392
    EXAMINED_ROWS: 0
    QUERY_ID: 15
    INFO_BINARY: SELECT * FROM INFORMATION_SCHEMA.PROCESSLIST
    TID: 11838
*****
2. row *****
    ID: 5
    USER: system user
    HOST:
    DB: NULL
    COMMAND: Daemon
    TIME: 0
    STATE: InnoDB shutdown handler
    INFO: NULL
    TIME_MS: 0.000
    STAGE: 0
    MAX_STAGE: 0
    PROGRESS: 0.000
    MEMORY_USED: 24160
    EXAMINED_ROWS: 0
    QUERY_ID: 0
    INFO_BINARY: NULL
    TID: 3856
...
...
```

See Also

- [TIME_MS column in Information Schema SHOW PROCESSLIST](#)

1.1.2.9.1.1.34 Information Schema PROFILING Table

The [Information Schema PROFILING](#) table contains information about statement resource usage. Profiling information is only recorded if the [profiling](#) session variable is set to 1.

It contains the following columns:

Column Name	Description
QUERY_ID	Query_ID.
SEQ	Sequence number showing the display order for rows with the same QUERY_ID .
STATE	Profiling state.
DURATION	Time in seconds that the statement has been in the current state.
CPU_USER	User CPU usage in seconds.
CPU_SYSTEM	System CPU usage in seconds.
CONTEXT_VOLUNTARY	Number of voluntary context switches.
CONTEXT_INVOLUNTARY	Number of involuntary context switches.
BLOCK_OPS_IN	Number of block input operations.
BLOCK_OPS_OUT	Number of block output operations.
MESSAGES_SENT	Number of communications sent.
MESSAGES RECEIVED	Number of communications received.
PAGE_FAULTS_MAJOR	Number of major page faults.
PAGE_FAULTS_MINOR	Number of minor page faults.
SWAPS	Number of swaps.
SOURCE_FUNCTION	Function in the source code executed by the profiled state.
SOURCE_FILE	File in the source code executed by the profiled state.
SOURCE_LINE	Line in the source code executed by the profiled state.

It contains similar information to the [SHOW PROFILE](#) and [SHOW PROFILES](#) statements.

Profiling is enabled per session. When a session ends, its profiling information is lost.

1.1.2.9.1.1.35 Information Schema QUERY_CACHE_INFO Table

Description

The table is not a standard Information Schema table, and is a MariaDB extension.

The [QUERY_CACHE_INFO](#) table is created by the [QUERY_CACHE_INFO](#) plugin, and allows you to see the contents of the [query cache](#). It creates a table in the [information_schema](#) database that shows all queries that are in the cache. You must have the [PROCESS](#) privilege (see [GRANT](#)) to use this table.

It contains the following columns:

Column	Description
STATEMENT_SCHEMA	Database used when query was included
STATEMENT_TEXT	Query text
RESULT_BLOCKS_COUNT	Number of result blocks

RESULT_BLOCKS_SIZE	Size in bytes of result blocks
RESULT_BLOCKS_SIZE_USED	Size in bytes of used blocks
LIMIT	Added MariaDB 10.1.8.
MAX_SORT_LENGTH	Added MariaDB 10.1.8.
GROUP_CONCAT_MAX_LENGTH	Added MariaDB 10.1.8.
CHARACTER_SET_CLIENT	Added MariaDB 10.1.8.
CHARACTER_SET_RESULT	Added MariaDB 10.1.8.
COLLATION	Added MariaDB 10.1.8.
TIMEZONE	Added MariaDB 10.1.8.
DEFAULT_WEEK_FORMAT	Added MariaDB 10.1.8.
DIV_PRECISION_INCREMENT	Added MariaDB 10.1.8.
SQL_MODE	Added MariaDB 10.1.8.
LC_TIME_NAMES	Added MariaDB 10.1.8.
CLIENT_LONG_FLAG	Added MariaDB 10.1.8.
CLIENT_PROTOCOL_41	Added MariaDB 10.1.8.
PROTOCOL_TYPE	Added MariaDB 10.1.8.
MORE_RESULTS_EXISTS	Added MariaDB 10.1.8.
IN_TRANS	Added MariaDB 10.1.8.
AUTOCOMMIT	Added MariaDB 10.1.8.
PACKET_NUMBER	Added MariaDB 10.1.8.
HITS	Incremented each time the query cache is hit. Added MariaDB 10.3.2.

For example:

```
SELECT * FROM information_schema.QUERY_CACHE_INFO;
+-----+-----+-----+-----+
| STATEMENT_SCHEMA | STATEMENT_TEXT | RESULT_BLOCKS_COUNT | RESULT_BLOCKS_SIZE | RESULT_BLOCKS_SIZE_USED |
+-----+-----+-----+-----+
...
| test           | SELECT * FROM a |          1 |        512 |         143 |
| test           | select * FROM a |          1 |        512 |         143 |
...
+-----+-----+-----+-----+
```

1.1.2.9.1.1.36 Information Schema QUERY_RESPONSE_TIME Table

Description

The `Information Schema QUERY_RESPONSE_TIME` table contains information about queries that take a long time to execute . It is only available if the `QUERY_RESPONSE_TIME` plugin has been installed.

It contains the following columns:

Column	Description
TIME	Time interval
COUNT	Count of queries falling into the time interval
TOTAL	Total execution time of all queries for this interval

See [QUERY_RESPONSE_TIME](#) plugin for a full description.

The table is not a standard Information Schema table, and is a MariaDB extension.

`SHOW QUERY_RESPONSE_TIME` is available from [MariaDB 10.1.1](#) as an alternative for retrieving the data.

Example

```
SELECT * FROM information_schema.QUERY_RESPONSE_TIME;
+-----+-----+
| TIME      | COUNT | TOTAL      |
+-----+-----+
| 0.000001 |    0 | 0.000000 |
| 0.000010 |   17 | 0.000094 |
| 0.000100 |  4301 | 0.236555 |
| 0.001000 | 1499 | 0.824450 |
| 0.010000 | 14851 | 81.680502 |
| 0.100000 | 8066 | 443.635693 |
| 1.000000 |    0 | 0.000000 |
| 10.000000 |    0 | 0.000000 |
| 100.000000 |    1 | 55.937094 |
| 1000.000000 |    0 | 0.000000 |
| 10000.000000 |    0 | 0.000000 |
| 100000.000000 |    0 | 0.000000 |
| 1000000.000000 |    0 | 0.000000 |
| TOO LONG   |    0 | TOO LONG   |
+-----+-----+
```

1.1.2.9.1.1.37 Information Schema REFERENTIAL_CONSTRAINTS Table

The [Information Schema REFERENTIAL_CONSTRAINTS](#) table contains information about [foreign keys](#). The single columns are listed in the [KEY_COLUMN_USAGE](#) table.

It has the following columns:

Column	Description
CONSTRAINT_CATALOG	Always def .
CONSTRAINT_SCHEMA	Database name, together with CONSTRAINT_NAME identifies the foreign key.
CONSTRAINT_NAME	Foreign key name, together with CONSTRAINT_SCHEMA identifies the foreign key.
UNIQUE_CONSTRAINT_CATALOG	Always def .
UNIQUE_CONSTRAINT_SCHEMA	Database name, together with UNIQUE_CONSTRAINT_NAME and REFERENCED_TABLE_NAME identifies the referenced key.
UNIQUE_CONSTRAINT_NAME	Referenced key name, together with UNIQUE_CONSTRAINT_SCHEMA and REFERENCED_TABLE_NAME identifies the referenced key.
MATCH_OPTION	Always NONE .
UPDATE_RULE	The Update Rule; one of CASCADE , SET NULL , SET DEFAULT , RESTRICT , NO ACTION .
DELETE_RULE	The Delete Rule; one of CASCADE , SET NULL , SET DEFAULT , RESTRICT , NO ACTION .
TABLE_NAME	Table name from the TABLE_CONSTRAINTS table.
REFERENCED_TABLE_NAME	Referenced key table name, together with UNIQUE_CONSTRAINT_SCHEMA and UNIQUE_CONSTRAINT_NAME identifies the referenced key.

1.1.2.9.1.1.38 Information Schema ROUTINES Table

The [Information Schema ROUTINES](#) table stores information about [stored procedures](#) and [stored functions](#).

It contains the following columns:

Column	Description
--------	-------------

SPECIFIC_NAME	
ROUTINE_CATALOG	Always def .
ROUTINE_SCHEMA	Database name associated with the routine.
ROUTINE_NAME	Name of the routine.
ROUTINE_TYPE	Whether the routine is a PROCEDURE or a FUNCTION .
DATA_TYPE	The return value's data type (for stored functions).
CHARACTER_MAXIMUM_LENGTH	Maximum length.
CHARACTER_OCTET_LENGTH	Same as the CHARACTER_MAXIMUM_LENGTH except for multi-byte character sets .
NUMERIC_PRECISION	For numeric types, the precision (number of significant digits) for the column. NULL if not a numeric field.
NUMERIC_SCALE	For numeric types, the scale (significant digits to the right of the decimal point). NULL if not a numeric field.
DATETIME_PRECISION	Fractional-seconds precision, or NULL if not a time data type .
CHARACTER_SET_NAME	Character set if a non-binary string data type , otherwise NULL .
COLLATION_NAME	Collation if a non-binary string data type , otherwise NULL .
DATA_TYPE	The column's data type .
ROUTINE_BODY	Always SQL .
ROUTINE_DEFINITION	Definition of the routine.
EXTERNAL_NAME	Always NULL .
EXTERNAL_LANGUAGE	Always SQL .
PARAMETER_STYLE	Always SQL .
IS_DETERMINISTIC	Whether the routine is deterministic (can produce only one result for a given list of parameters) or not.
SQL_DATA_ACCESS	One of READS SQL DATA , MODIFIES SQL DATA , CONTAINS SQL , or NO SQL .
SQL_PATH	Always NULL .
SECURITY_TYPE	INVOKER or DEFINER . Indicates which user's privileges apply to this routine.
CREATED	Date and time the routine was created.
LAST_ALTERED	Date and time the routine was last changed.
SQL_MODE	The SQL_MODE at the time the routine was created.
ROUTINE_COMMENT	Comment associated with the routine.
DEFINER	If the SECURITY_TYPE is DEFINER , this value indicates which user defined this routine.
CHARACTER_SET_CLIENT	The character set used by the client that created the routine.
COLLATION_CONNECTION	The collation (and character set) used by the connection that created the routine.
DATABASE_COLLATION	The default collation (and character set) for the database, at the time the routine was created.

It provides information similar to, but more complete, than the `SHOW PROCEDURE STATUS` and `SHOW FUNCTION STATUS` statements.

For information about the parameters accepted by the routine, you can query the `information_schema.PARAMETERS` table.

See also

- [Stored Function Overview](#)
- [Stored Procedure Overview](#)

1.1.2.9.1.1.39 Information Schema SCHEMA_PRIVILEGES Table

The `Information Schema SCHEMA_PRIVILEGES` table contains information about [database privileges](#).

It contains the following columns:

Column	Description
GRANTEE	Account granted the privilege in the format <code>user_name@host_name</code> .
TABLE_CATALOG	Always <code>def</code>
TABLE_SCHEMA	Database name.
PRIVILEGE_TYPE	The granted privilege.
IS_GRANTABLE	Whether the privilege can be granted.

The same information in a different format can be found in the `mysql.db` table.

1.1.2.9.1.1.40 Information Schema SCHEMATA Table

The `Information Schema SCHEMATA` table stores information about databases on the server.

It contains the following columns:

Column	Description
CATALOG_NAME	Always <code>def</code> .
SCHEMA_NAME	Database name.
DEFAULT_CHARACTER_SET_NAME	Default <code>character set</code> for the database.
DEFAULT_COLLATION_NAME	Default <code>collation</code> .
SQL_PATH	Always <code>NULL</code> .
SCHEMA_COMMENT	Database comment. From MariaDB 10.5.0 .

Example

```
SELECT * FROM INFORMATION_SCHEMA.SCHEMATA\G
*****
1. row *****
    CATALOG_NAME: def
        SCHEMA_NAME: information_schema
    DEFAULT_CHARACTER_SET_NAME: utf8
        DEFAULT_COLLATION_NAME: utf8_general_ci
            SQL_PATH: NULL
*****
2. row *****
    CATALOG_NAME: def
        SCHEMA_NAME: mysql
    DEFAULT_CHARACTER_SET_NAME: latin1
        DEFAULT_COLLATION_NAME: latin1_swedish_ci
            SQL_PATH: NULL
*****
3. row *****
    CATALOG_NAME: def
        SCHEMA_NAME: performance_schema
    DEFAULT_CHARACTER_SET_NAME: utf8
        DEFAULT_COLLATION_NAME: utf8_general_ci
            SQL_PATH: NULL
*****
4. row *****
    CATALOG_NAME: def
        SCHEMA_NAME: test
    DEFAULT_CHARACTER_SET_NAME: latin1
        DEFAULT_COLLATION_NAME: latin1_swedish_ci
            SQL_PATH: NULL
...
...
```

From [MariaDB 10.5.0](#):

```

SELECT * FROM INFORMATION_SCHEMA.SCHEMATA\G
...
***** 2. row *****
CATALOG_NAME: def
SCHEMA_NAME: presentations
DEFAULT_CHARACTER_SET_NAME: latin1
DEFAULT_COLLATION_NAME: latin1_swedish_ci
SQL_PATH: NULL
SCHEMA_COMMENT: Presentations for conferences
...

```

See Also

- [CREATE DATABASE](#)
- [ALTER DATABASE](#)
- [DROP DATABASE](#)
- [SHOW CREATE DATABASE](#)
- [SHOW DATABASES](#)
- [Character Sets and Collations](#)

1.1.2.9.1.1.41 Information Schema SPATIAL_REF_SYS Table

MariaDB starting with [10.1.2](#)

The `SPATIAL_REF_SYS` table was introduced in MariaDB 10.1.2

Description

The `Information Schema SPATIAL_REF_SYS` table stores information on each spatial reference system used in the database.

It contains the following columns:

Column	Type	Null	Description
SRID	smallint(5)	NO	An integer value that uniquely identifies each Spatial Reference System within a database.
AUTH_NAME	varchar(512)	NO	The name of the standard or standards body that is being cited for this reference system.
AUTH_SRID	smallint(5)	NO	The numeric ID of the coordinate system in the above authority's catalog.
SRTEXT	varchar(2048)	NO	The Well-known Text Representation of the Spatial Reference System.

Note: See [MDEV-7540](#).

See also

- [information_schema.GEOMETRY_COLUMNS](#) table.

1.1.2.9.1.1.42 Information Schema SPIDER_ALLOC_MEM Table

The `Information Schema SPIDER_ALLOC_MEM` table is installed along with the `Spider` storage engine. It shows information about Spider's memory usage.

It contains the following columns:

Column	Description
ID	
FUNC_NAME	

FILE_NAME	
LINE_NO	
TOTAL_ALLOC_MEM	
CURRENT_ALLOC_MEM	
ALLOC_MEM_COUNT	
FREE_MEM_COUNT	

1.1.2.9.1.1.43 Information Schema SPIDER_WRAPPER_PROTOCOLS Table

MariaDB starting with 10.5.4

The `Information Schema SPIDER_WRAPPER_PROTOCOLS` table is installed along with the `Spider` storage engine from MariaDB 10.5.4.

It contains the following columns:

Column	Type	Description
WRAPPER_NAME	varchar(64)	
WRAPPER_VERSION	varchar(20)	
WRAPPER_DESCRIPTION	longtext	
WRAPPER_MATURITY	varchar(12)	

1.1.2.9.1.1.44 Information Schema SQL_FUNCTIONS Table

MariaDB starting with 10.6.3

The `SQL_FUNCTIONS` table was added in MariaDB 10.6.3.

Description

The `Information Schema SQL_FUNCTIONS` table contains the list of MariaDB functions.

It contains a single column:

Column	Description
FUNCTION	Function name

The table is not a standard Information Schema table, and is a MariaDB extension.

Example

```
SELECT * FROM INFORMATION_SCHEMA.SQL_FUNCTIONS;
+-----+
| FUNCTION          |
+-----+
| ADDDATE           |
| ADD_MONTHS        |
| BIT_AND           |
| BIT_OR            |
| BIT_XOR           |
| CAST              |
| COUNT             |
| CUME_DIST         |
| CURDATE           |
+-----+
```

```
| CURTIME  
| DATE_ADD  
| DATE_SUB  
| DATE_FORMAT  
| DECODE  
| DENSE_RANK  
| EXTRACT  
| FIRST_VALUE  
| GROUP_CONCAT  
| JSON_ARRAYAGG  
| JSON_OBJECTAGG  
| LAG  
| LEAD  
| MAX  
| MEDIAN  
| MID  
| MIN  
| NOW  
| NTH_VALUE  
| NTILE  
| POSITION  
| PERCENT_RANK  
| PERCENTILE_CONT  
| PERCENTILE_DISC  
| RANK  
| ROW_NUMBER  
| SESSION_USER  
| STD  
| STDDEV  
| STDDEV_POP  
| STDDEV_SAMP  
| SUBDATE  
| SUBSTR  
| SUBSTRING  
| SUM  
| SYSTEM_USER  
| TRIM  
| TRIM_ORACLE  
| VARIANCE  
| VAR_POP  
| VAR_SAMP  
| ABS  
| ACOS  
| ADDTIME  
| AES_DECRYPT  
| AES_ENCRYPT  
| ASIN  
| ATAN  
| ATAN2  
| BENCHMARK  
| BIN  
| BINLOG_GTID_POS  
| BIT_COUNT  
| BIT_LENGTH  
| CEIL  
| CEILING  
| CHARACTER_LENGTH  
| CHAR_LENGTH  
| CHR  
| COERCIBILITY  
| COLUMN_CHECK  
| COLUMN_EXISTS  
| COLUMN_LIST  
| COLUMN_JSON  
| COMPRESS  
| CONCAT  
| CONCAT_OPERATOR_ORACLE  
| CONCAT_WS  
| CONNECTION_ID  
| CONV  
| CONVERT_TZ
```

COS	
COT	
CRC32	
DATEDIFF	
DAYNAME	
DAYOFMONTH	
DAYOFWEEK	
DAYOFYEAR	
DEGREES	
DECODE_HISTOGRAM	
DECODE_ORACLE	
DES_DECRYPT	
DES_ENCRYPT	
ELT	
ENCODE	
ENCRYPT	
EXP	
EXPORT_SET	
EXTRACTVALUE	
FIELD	
FIND_IN_SET	
FLOOR	
FORMAT	
FOUND_ROWS	
FROM_BASE64	
FROM_DAYS	
FROM_UNIXTIME	
GET_LOCK	
GREATEST	
HEX	
IFNULL	
INSTR	
ISNULL	
IS_FREE_LOCK	
IS_USED_LOCK	
JSON_ARRAY	
JSON_ARRAY_APPEND	
JSON_ARRAY_INSERT	
JSON_COMPACT	
JSON_CONTAINS	
JSON_CONTAINS_PATH	
JSON_DEPTH	
JSON_DETAILED	
JSON_EXISTS	
JSON_EXTRACT	
JSON_INSERT	
JSON_KEYS	
JSON_LENGTH	
JSON_LOOSE	
JSON_MERGE	
JSON_MERGE_PATCH	
JSON_MERGE_PRESERVE	
JSON_QUERY	
JSON_QUOTE	
JSON_OBJECT	
JSON_REMOVE	
JSON_REPLACE	
JSON_SET	
JSON_SEARCH	
JSON_TYPE	
JSON_UNQUOTE	
JSON_VALID	
JSON_VALUE	
LAST_DAY	
LAST_INSERT_ID	
LCASE	
LEAST	
LENGTH	
LENGTHB	
LN	
LOAD_FILE	
LOCATE	

```
| LOG  
| LOG10  
| LOG2  
| LOWER  
| LPAD  
| LPAD_ORACLE  
| LTRIM  
| LTRIM_ORACLE  
| MAKEDATE  
| MAKETIME  
| MAKE_SET  
| MASTER_GTID_WAIT  
| MASTER_POS_WAIT  
| MD5  
| MONTHNAME  
| NAME_CONST  
| NVL  
| NVL2  
| NULLIF  
| OCT  
| OCTET_LENGTH  
| ORD  
| PERIOD_ADD  
| PERIOD_DIFF  
| PI  
| POW  
| POWER  
| QUOTE  
| REGEXP_INSTR  
| REGEXP_REPLACE  
| REGEXP_SUBSTR  
| RADIANS  
| RAND  
| RELEASE_ALL_LOCKS  
| RELEASE_LOCK  
| REPLACE_ORACLE  
| REVERSE  
| ROUND  
| RPAD  
| RPAD_ORACLE  
| RTRIM  
| RTRIM_ORACLE  
| SEC_TO_TIME  
| SHA  
| SHA1  
| SHA2  
| SIGN  
| SIN  
| SLEEP  
| SOUNDEX  
| SPACE  
| SQRT  
| STRCMP  
| STR_TO_DATE  
| SUBSTR_ORACLE  
| SUBSTRING_INDEX  
| SUBTIME  
| SYS_GUID  
| TAN  
| TIMEDIFF  
| TIME_FORMAT  
| TIME_TO_SEC  
| TO_BASE64  
| TO_CHAR  
| TO_DAYS  
| TO_SECONDS  
| UCASE  
| UNCOMPRESS  
| UNCOMPRESSED_LENGTH  
| UNHEX  
| UNIX_TIMESTAMP
```

```

| UPDATEXML          |
| UPPER              |
| UUID               |
| UUID_SHORT         |
| VERSION            |
| WEEKDAY            |
| WEEKOFYEAR          |
| WSREP_LAST_WRITTEN_GTID |
| WSREP_LAST_SEEN_GTID |
| WSREP_SYNC_WAIT_UPTO_GTID |
| YEARWEEK           |
+-----+
234 rows in set (0.001 sec)

```

See Also

- [Reserved Words](#)

1.1.2.9.1.1.45 Information Schema STATISTICS Table

The `Information Schema` `STATISTICS` table provides information about table indexes.

It contains the following columns:

Column	Description
TABLE_CATALOG	Always def .
TABLE_SCHEMA	Database name.
TABLE_NAME	Table name.
NON_UNIQUE	1 if the index can have duplicates, 0 if not.
INDEX_SCHEMA	Database name.
INDEX_NAME	Index name. The primary key is always named PRIMARY .
SEQ_IN_INDEX	The column sequence number, starting at 1.
COLUMN_NAME	Column name.
COLLATION	A for sorted in ascending order, or NULL for unsorted.
CARDINALITY	Estimate of the number of unique values stored in the index based on statistics stored as integers. Higher cardinalities usually mean a greater chance of the index being used in a join. Updated by the <code>ANALYZE TABLE</code> statement or <code>myisamchk -a</code> .
SUB_PART	NULL if the whole column is indexed, or the number of indexed characters if partly indexed.
PACKED	NULL if not packed, otherwise how the index is packed.
NULLABLE	YES if the column may contain NULLs, empty string if not.
INDEX_TYPE	Index type, one of BTREE , RTREE , HASH or FULLTEXT . See Storage Engine Index Types .
COMMENT	Index comments from the <code>CREATE INDEX</code> statement.
IGNORED	Whether or not an index will be ignored by the optimizer. See Ignored Indexes . From MariaDB 10.6.0.

The `SHOW INDEX` statement produces similar output.

Example

```

SELECT * FROM INFORMATION_SCHEMA.STATISTICS\G
...
***** 85. row *****
TABLE_CATALOG: def
TABLE_SCHEMA: test
TABLE_NAME: table1
NON_UNIQUE: 1
INDEX_SCHEMA: test
INDEX_NAME: col2
SEQ_IN_INDEX: 1
COLUMN_NAME: col2
COLLATION: A
CARDINALITY: 6
SUB_PART: NULL
PACKED: NULL
NULLABLE:
INDEX_TYPE: BTREE
COMMENT:
INDEX_COMMENT:
...

```

1.1.2.9.1.1.46 Information Schema SYSTEM_VARIABLES Table

MariaDB starting with [10.1.1](#)

The `information_schema.SYSTEM_VARIABLES` table was introduced in [MariaDB 10.1.1](#)

The `Information Schema SYSTEM_VARIABLES` table shows current values and various metadata of all [system variables](#).

It contains the following columns:

Column	Description
VARIABLE_NAME	System variable name.
SESSION_VALUE	Session value of the variable or NULL if the variable only has a global scope.
GLOBAL_VALUE	Global value of the variable or NULL if the variable only has a session scope.
GLOBAL_VALUE_ORIGIN	How the global value was set — a compile-time default, auto-configured by the server, configuration file (or a command line), with the SQL statement.
DEFAULT_VALUE	Compile-time default value of the variable.
VARIABLE_SCOPE	Global, session, or session-only.
VARIABLE_TYPE	Data type of the variable value.
VARIABLE_COMMENT	Help text, usually shown in <code>mysqld --help --verbose</code> .
NUMERIC_MIN_VALUE	For numeric variables — minimal allowed value.
NUMERIC_MAX_VALUE	For numeric variables — maximal allowed value.
NUMERIC_BLOCK_SIZE	For numeric variables — a valid value must be a multiple of the "block size".
ENUM_VALUE_LIST	For <code>ENUM</code> , <code>SET</code> , and <code>FLAGSET</code> variables — the list of recognized values.
READ_ONLY	Whether a variable can be set with the SQL statement. Note that many "read only" variables can still be set on the command line.
COMMAND_LINE_ARGUMENT	Whether an argument is required when setting the variable on the command line. <code>NULL</code> when a variable can not be set on the command line.
GLOBAL_VALUE_PATH	Which config file the variable got its value from. <code>NULL</code> if not set in any config file. Added in MariaDB 10.5.0 .

Example

```

SELECT * FROM information_schema.SYSTEM_VARIABLES
WHERE VARIABLE_NAME='JOIN_BUFFER_SIZE'\G
*****
***** 1. row *****
VARIABLE_NAME: JOIN_BUFFER_SIZE
SESSION_VALUE: 131072
GLOBAL_VALUE: 131072
GLOBAL_VALUE_ORIGIN: COMPILE-TIME
DEFAULT_VALUE: 131072
VARIABLE_SCOPE: SESSION
VARIABLE_TYPE: BIGINT UNSIGNED
VARIABLE_COMMENT: The size of the buffer that is used for joins
NUMERIC_MIN_VALUE: 128
NUMERIC_MAX_VALUE: 18446744073709551615
NUMERIC_BLOCK_SIZE: 128
ENUM_VALUE_LIST: NULL
READ_ONLY: NO
COMMAND_LINE_ARGUMENT: REQUIRED

```

1.1.2.9.1.1.47 Information Schema TABLE_CONSTRAINTS Table

The [Information Schema TABLE_CONSTRAINTS](#) table contains information about tables that have [constraints](#).

It has the following columns:

Column	Description
CONSTRAINT_CATALOG	Always def.
CONSTRAINT_SCHEMA	Database name containing the constraint.
CONSTRAINT_NAME	Constraint name.
TABLE_SCHEMA	Database name.
TABLE_NAME	Table name.
CONSTRAINT_TYPE	Type of constraint; one of UNIQUE , PRIMARY KEY , FOREIGN KEY or CHECK .

The [REFERENTIAL_CONSTRAINTS](#) table has more information about foreign keys.

1.1.2.9.1.1.48 Information Schema TABLE_PRIVILEGES Table

The [Information Schema TABLE_PRIVILEGES](#) table contains table privilege information derived from the [mysql.tables_priv](#) grant table.

It has the following columns:

Column	Description
GRANTEE	In the format user_name@host_name .
TABLE_CATALOG	Always def.
TABLE_SCHEMA	Database name.
TABLE_NAME	Table name.
PRIVILEGE_TYPE	One of SELECT , INSERT , UPDATE , REFERENCES , ALTER , INDEX , DROP or CREATE VIEW .
IS_GRANTABLE	Whether the user has the GRANT OPTION for this privilege.

Similar information can be accessed with the [SHOW GRANTS](#) statement. See the [GRANT](#) article for more about privileges.

For a description of the privileges that are shown in this table, see [table privileges](#).

1.1.2.9.1.1.49 Information Schema TABLE_STATISTICS Table

The [Information Schema](#) `TABLE_STATISTICS` table shows statistics on table usage.

This is part of the [User Statistics](#) feature, which is not enabled by default.

It contains the following columns:

Field	Type	Notes
TABLE_SCHEMA	varchar(192)	The schema (database) name.
TABLE_NAME	varchar(192)	The table name.
ROWS_READ	int(21)	The number of rows read from the table.
ROWS_CHANGED	int(21)	The number of rows changed in the table.
ROWS_CHANGED_X_INDEXES	int(21)	The number of rows changed in the table, multiplied by the number of indexes changed.

Example

```
SELECT * FROM INFORMATION_SCHEMA.TABLE_STATISTICS WHERE TABLE_NAME='user';
+-----+-----+-----+-----+
| TABLE_SCHEMA | TABLE_NAME | ROWS_READ | ROWS_CHANGED | ROWS_CHANGED_X_INDEXES |
+-----+-----+-----+-----+
| mysql        | user       |      5 |          2 |                  2 |
+-----+-----+-----+-----+
```

1.1.2.9.1.1.50 Information Schema TABLES Table

The [Information Schema](#) table shows information about the various non- TEMPORARY tables (except tables from the [Information Schema](#) database) and [views](#) on the server.

It contains the following columns:

Column	Description
TABLE_CATALOG	Always def .
TABLE_SCHEMA	Database name.
TABLE_NAME	Table name.
TABLE_TYPE	One of <code>BASE TABLE</code> for a regular table, <code>VIEW</code> for a view , <code>SYSTEM VIEW</code> for Information Schema tables, <code>SYSTEM VERSIONED</code> for system-versioned tables or <code>SEQUENCE</code> for sequences .
ENGINE	Storage Engine .
VERSION	Version number from the table's .frm file
ROW_FORMAT	Row format (see InnoDB , Aria and MyISAM row formats).
TABLE_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 <code>AUTO_INCREMENT</code> value.
CREATE_TIME	Time the table was created.

Contents

1. [Examples](#)
 1. [View Tables in Order of Size](#)
 2. [See Also](#)

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 NULL, while buffering can also delay the update, so the value will differ from the actual time of the last UPDATE , INSERT OR DELETE .
CHECK_TIME	Time the table was last checked. Not kept by all storage engines, in which case will be NULL .
TABLE_COLLATION	Character set and collation.
CHECKSUM	Live checksum value, if any.
CREATE_OPTIONS	Extra CREATE TABLE options.
TABLE_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.

Although the table is standard in the Information Schema, all but TABLE_CATALOG , TABLE_SCHEMA , TABLE_NAME , TABLE_TYPE , ENGINE and VERSION are MySQL and MariaDB extensions.

[SHOW TABLES](#) lists all tables in a database.

Examples

From [MariaDB 10.3.5](#):

```

SELECT * FROM information_schema.tables WHERE table_schema='test'\G
***** 1. row *****
TABLE_CATALOG: def
TABLE_SCHEMA: test
TABLE_NAME: xx5
TABLE_TYPE: BASE TABLE
ENGINE: InnoDB
VERSION: 10
ROW_FORMAT: Dynamic
TABLE_ROWS: 0
AVG_ROW_LENGTH: 0
DATA_LENGTH: 16384
MAX_DATA_LENGTH: 0
INDEX_LENGTH: 0
DATA_FREE: 0
AUTO_INCREMENT: NULL
CREATE_TIME: 2020-11-18 15:57:10
UPDATE_TIME: NULL
CHECK_TIME: NULL
TABLE_COLLATION: latin1_swedish_ci
CHECKSUM: NULL
CREATE_OPTIONS:
TABLE_COMMENT:
MAX_INDEX_LENGTH: 0
TEMPORARY: N
***** 2. row *****
TABLE_CATALOG: def
TABLE_SCHEMA: test
TABLE_NAME: xx4
TABLE_TYPE: BASE TABLE
ENGINE: MyISAM
VERSION: 10
ROW_FORMAT: Fixed
TABLE_ROWS: 0
AVG_ROW_LENGTH: 0
DATA_LENGTH: 0
MAX_DATA_LENGTH: 1970324836974591
INDEX_LENGTH: 1024
DATA_FREE: 0
AUTO_INCREMENT: NULL
CREATE_TIME: 2020-11-18 15:56:57
UPDATE_TIME: 2020-11-18 15:56:57
CHECK_TIME: NULL
TABLE_COLLATION: latin1_swedish_ci
CHECKSUM: NULL
CREATE_OPTIONS:
TABLE_COMMENT:
MAX_INDEX_LENGTH: 17179868160
TEMPORARY: N
...

```

Example with temporary = 'y', from [MariaDB 10.3.5](#):

```

SELECT * FROM information_schema.tables WHERE temporary='y'\G
***** 1. row *****
TABLE_CATALOG: def
TABLE_SCHEMA: information_schema
TABLE_NAME: INNODB_FT_DELETED
TABLE_TYPE: SYSTEM VIEW
ENGINE: MEMORY
VERSION: 11
ROW_FORMAT: Fixed
TABLE_ROWS: NULL
AVG_ROW_LENGTH: 9
DATA_LENGTH: 0
MAX_DATA_LENGTH: 9437184
INDEX_LENGTH: 0
DATA_FREE: 0
AUTO_INCREMENT: NULL
CREATE_TIME: 2020-11-17 21:54:02
UPDATE_TIME: NULL
CHECK_TIME: NULL
TABLE_COLLATION: utf8_general_ci
CHECKSUM: NULL
CREATE_OPTIONS: max_rows=1864135
TABLE_COMMENT:
MAX_INDEX_LENGTH: 0
TEMPORARY: Y
...

```

View Tables in Order of Size

Returns a list of all tables in the database, ordered by size:

```

SELECT table_schema AS `DB`, table_name AS `Table`,
ROUND(((data_length + index_length) / 1024 / 1024), 2) `Size (MB)`
FROM information_schema.TABLES
ORDER BY (data_length + index_length) DESC;

```

DB	Table	Size (MB)
wordpress	wp_simple_history_contexts	7.05
wordpress	wp_posts	6.59
wordpress	wp_simple_history	3.05
wordpress	wp_comments	2.73
wordpress	wp_commentmeta	2.47
wordpress	wp_simple_login_log	2.03

See Also

- [mysqlshow](#)
- [SHOW TABLE STATUS](#)
- [Finding Tables Without Primary Keys](#)

1.1.2.9.1.1.51 Information Schema TABLESPACES Table

The [Information Schema TABLESPACES](#) table contains information about active tablespaces..

The table is a MariaDB and MySQL extension, and does not include information about InnoDB tablespaces.

Column	Description
TABLESPACE_NAME	
ENGINE	
TABLESPACE_TYPE	

LOGFILE_GROUP_NAME	
EXTENT_SIZE	
AUTOEXTEND_SIZE	
MAXIMUM_SIZE	
NODEGROUP_ID	
TABLESPACE_COMMENT	

1.1.2.9.1.1.52 Information Schema THREAD_POOL_GROUPS Table

MariaDB starting with 10.5

The `Information Schema THREAD_POOL_GROUPS` table was introduced in MariaDB 10.5.0.

The table provides information about `thread pool` groups, and contains the following columns:

Column	Description
GROUP_ID	
CONNECTIONS	
THREADS	
ACTIVE_THREADS	
STANDBY_THREADS	
QUEUE_LENGTH	
HAS_LISTENER	
IS_STALLED	

Setting `thread_pool_dedicated_listener` will give each group its own dedicated listener, and the listener thread will not pick up work items. As a result, the actual queue size in the table will be more exact, since IO requests are immediately dequeued from poll, without delay.

1.1.2.9.1.1.53 Information Schema THREAD_POOL_STATS Table

MariaDB starting with 10.5

The `Information Schema THREAD_POOL_STATS` table was introduced in MariaDB 10.5.0.

The table provides performance counter information for the `thread pool`, and contains the following columns:

Column	Description
GROUP_ID	
THREAD_CREATATIONS	
THREAD_CREATATIONS_DUE_TO_STALL	
WAKES	
WAKES_DUE_TO_STALL	
THROTTLES	
STALLS	
POLLS_BY_LISTENER	
POLLS_BY_WORKER	

DEQUEUES_BY_LISTENER	
DEQUEUES_BY_WORKER	

1.1.2.9.1.1.54 Information Schema THREAD_POOL_WAITS Table

MariaDB starting with 10.5

The [Information Schema](#) `THREAD_POOL_WAITS` table was introduced in [MariaDB 10.5.0](#).

The table provides wait counters for the [thread pool](#), and contains the following columns:

Column	Description
REASON	
COUNT	

1.1.2.9.1.1.55 Information Schema TRIGGERS Table

The [Information Schema](#) `TRIGGERS` table contains information about [triggers](#).

It has the following columns:

Column	Description
<code>TRIGGER_CATALOG</code>	Always def.
<code>TRIGGER_SCHEMA</code>	Database name in which the trigger occurs.
<code>TRIGGER_NAME</code>	Name of the trigger.
<code>EVENT_MANIPULATION</code>	The event that activates the trigger. One of <code>INSERT</code> , <code>UPDATE</code> or <code>'DELETE'</code> .
<code>EVENT_OBJECT_CATALOG</code>	Always def.
<code>EVENT_OBJECT_SCHEMA</code>	Database name on which the trigger acts.
<code>EVENT_OBJECT_TABLE</code>	Table name on which the trigger acts.
<code>ACTION_ORDER</code>	Indicates the order that the action will be performed in (of the list of a table's triggers with identical <code>EVENT_MANIPULATION</code> and <code>ACTION_TIMING</code> values). Before MariaDB 10.2.3 introduced the FOLLOWS and PRECEDES clauses, always 0
<code>ACTION_CONDITION</code>	NULL
<code>ACTION_STATEMENT</code>	Trigger body, UTF-8 encoded.
<code>ACTION_ORIENTATION</code>	Always <code>ROW</code> .
<code>ACTION_TIMING</code>	Whether the trigger acts <code>BEFORE</code> or <code>AFTER</code> the event that triggers it.
<code>ACTION_REFERENCE_OLD_TABLE</code>	Always <code>NULL</code> .
<code>ACTION_REFERENCE_NEW_TABLE</code>	Always <code>NULL</code> .
<code>ACTION_REFERENCE_OLD_ROW</code>	Always <code>OLD</code> .
<code>ACTION_REFERENCE_NEW_ROW</code>	Always <code>NEW</code> .
<code>CREATED</code>	Always <code>NULL</code> .
<code>SQL_MODE</code>	The SQL_MODE when the trigger was created, and which it uses for execution.
<code>DEFINER</code>	The account that created the trigger, in the form <code>user_name@host_name</code>

Contents

1. [See also](#)

CHARACTER_SET_CLIENT	The client <code>character set</code> when the trigger was created, from the session value of the <code>character_set_client</code> system variable.
COLLATION_CONNECTION	The client <code>collation</code> when the trigger was created, from the session value of the <code>collation_connection</code> system variable.
DATABASE_COLLATION	<code>Collation</code> of the associated database.

Queries to the `TRIGGERS` table will return information only for databases and tables for which you have the `TRIGGER` privilege. Similar information is returned by the `SHOW TRIGGERS` statement.

See also

- [Trigger Overview](#)
- [CREATE TRIGGER](#)
- [DROP TRIGGER](#)
- [SHOW TRIGGERS](#)
- [SHOW CREATE TRIGGER](#)
- [Trigger Limitations](#)

1.1.2.9.1.1.56 Information Schema USER_PRIVILEGES Table

The `Information Schema` `USER_PRIVILEGES` table contains global user privilege information derived from the `mysql.user` grant table.

It contains the following columns:

Column	Description
GRANTEE	In the format <code>user_name@host_name</code> .
TABLE_CATALOG	Always <code>def</code> .
PRIVILEGE_TYPE	The specific privilege, for example <code>SELECT</code> , <code>INSERT</code> , <code>UPDATE</code> or <code>REFERENCES</code> .
IS_GRANTABLE	Whether the user has the <code>GRANT OPTION</code> for this privilege.

Similar information can be accessed with the `SHOW GRANTS` statement. See the `GRANT` article for more about privileges.

This information is also stored in the `user` table, in the `mysql` system database.

1.1.2.9.1.1.57 Information Schema USER_STATISTICS Table

The `Information Schema` `USER_STATISTICS` table holds statistics about user activity. This is part of the `User Statistics` feature, which is not enabled by default.

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.

It contains the following columns:

Field	Type	Notes
USER	varchar(48)	The username. The value ' <code>#mysql_system_user#</code> ' appears when there is no username (such as for the slave SQL thread).
TOTAL_CONNECTIONS	int(21)	The number of connections created for this user.
CONCURRENT_CONNECTIONS	int(21)	The number of concurrent connections for this user.
CONNECTED_TIME	int(21)	The cumulative number of seconds elapsed while there were connections from this user.
BUSY_TIME	double	The cumulative number of seconds there was activity on connections from this user.
CPU_TIME	double	The cumulative CPU time elapsed while servicing this user's connections.
BYTES_RECEIVED	int(21)	The number of bytes received from this user's connections.
BYTES_SENT	int(21)	The number of bytes sent to this user's connections.

BINLOG_BYTES_WRITTEN	int(21)	The number of bytes written to the binary log from this user's connections.
ROWS_READ	int(21)	The number of rows read by this user's connections.
ROWS_SENT	int(21)	The number of rows sent by this user's connections.
ROWS_DELETED	int(21)	The number of rows deleted by this user's connections.
ROWS_INSERTED	int(21)	The number of rows inserted by this user's connections.
ROWS_UPDATED	int(21)	The number of rows updated by this user's connections.
SELECT_COMMANDS	int(21)	The number of SELECT commands executed from this user's connections.
UPDATE_COMMANDS	int(21)	The number of UPDATE commands executed from this user's connections.
OTHER_COMMANDS	int(21)	The number of other commands executed from this user's connections.
COMMIT_TRANSACTIONS	int(21)	The number of COMMIT commands issued by this user's connections.
ROLLBACK_TRANSACTIONS	int(21)	The number of ROLLBACK commands issued by this user's connections.
DENIED_CONNECTIONS	int(21)	The number of connections denied to this user.
LOST_CONNECTIONS	int(21)	The number of this user's connections that were terminated uncleanly.
ACCESS_DENIED	int(21)	The number of times this user's connections issued commands that were denied.
EMPTY_QUERIES	int(21)	The number of times this user's connections sent empty queries to the server.
TOTAL_SSL_CONNECTIONS	int(21)	The number of TLS connections created for this user. (>= MariaDB 10.1.1)
MAX_STATEMENT_TIME_EXCEEDED	int(21)	The number of times a statement was aborted, because it was executed longer than its MAX_STATEMENT_TIME threshold. (>= MariaDB 10.1.1)

Example

```
SELECT * FROM information_schema.USER_STATISTICS\G
*****
1. row *****
    USER: root
    TOTAL_CONNECTIONS: 1
    CONCURRENT_CONNECTIONS: 0
    CONNECTED_TIME: 297
        BUSY_TIME: 0.001725
        CPU_TIME: 0.001982
    BYTES_RECEIVED: 388
    BYTES_SENT: 2327
    BINLOG_BYTES_WRITTEN: 0
        ROWS_READ: 0
        ROWS_SENT: 12
        ROWS_DELETED: 0
        ROWS_INSERTED: 13
        ROWS_UPDATED: 0
    SELECT_COMMANDS: 4
    UPDATE_COMMANDS: 0
    OTHER_COMMANDS: 3
    COMMIT_TRANSACTIONS: 0
    ROLLBACK_TRANSACTIONS: 0
    DENIED_CONNECTIONS: 0
    LOST_CONNECTIONS: 0
    ACCESS_DENIED: 0
    EMPTY_QUERIES: 1
```

1.1.2.9.1.1.58 Information Schema USER_VARIABLES Table

MariaDB 10.2.0

The `USER_VARIABLES` table was introduced in [MariaDB 10.2.0](#) as part of the `user_variables` plugin.

Description

The `USER_VARIABLES` table is created when the `user_variables` plugin is enabled, and contains information about [user-defined variables](#).

The table contains the following columns:

Column	Description
VARIABLE_NAME	Variable name.
VARIABLE_VALUE	Variable value.
VARIABLE_TYPE	Variable type .
CHARACTER_SET_NAME	Character set .

Example

```
SELECT * FROM information_schema.USER_VARIABLES ORDER BY VARIABLE_NAME;
+-----+-----+-----+
| VARIABLE_NAME | VARIABLE_VALUE | VARIABLE_TYPE | CHARACTER_SET_NAME |
+-----+-----+-----+
| var          | 0             | INT           | utf8            |
| var2         | abc           | VARCHAR       | utf8            |
+-----+-----+-----+
```

1.1.2.9.1.1.59 Information Schema VIEWS Table

The `Information Schema VIEWS` table contains information about [views](#). The `SHOW VIEW` privilege is required to view the table.

It has the following columns:

Column	Description	Added
TABLE_CATALOG	Always def.	
TABLE_SCHEMA	Database name containing the view.	
TABLE_NAME	View table name.	
VIEW_DEFINITION	Definition of the view.	
CHECK_OPTION	YES if the WITH CHECK_OPTION clause has been specified, NO otherwise.	
IS_UPDATABLE	Whether the view is updatable or not.	
DEFINER	Account specified in the DEFINER clause (or the default when created).	
SECURITY_TYPE	SQL SECURITY characteristic, either DEFINER or INVOKER.	
CHARACTER_SET_CLIENT	The client character set when the view was created, from the session value of the <code>character_set_client</code> system variable.	
COLLATION_CONNECTION	The client collation when the view was created, from the session value of the <code>collation_connection</code> system variable.	
ALGORITHM	The algorithm used in the view. See View Algorithms .	MariaDB 10.1.3

Example

```

SELECT * FROM information_schema.VIEWS\G
*****
1. row *****
TABLE_CATALOG: def
TABLE_SCHEMA: test
TABLE_NAME: v
VIEW_DEFINITION: select `test`.`t`.`qty` AS `qty`, `test`.`t`.`price` AS `price`,(`test`.`t`.`qty` * `test`.`t`.`price`) AS `val`
CHECK_OPTION: NONE
IS_UPDATABLE: YES
DEFINER: root@localhost
SECURITY_TYPE: DEFINER
CHARACTER_SET_CLIENT: utf8
COLLATION_CONNECTION: utf8_general_ci
ALGORITHM: UNDEFINED

```

See also

- [CREATE VIEW](#)
- [ALTER VIEW](#)
- [DROP VIEW](#)
- [SHOW CREATE VIEWS](#)

1.1.2.9.1.1.60 Information Schema WSREP_MEMBERSHIP Table

The `WSREP_STATUS` table makes [Galera](#) node cluster membership information available through the [Information Schema](#). The same information can be returned using the `SHOW WSREP_MEMBERSHIP` statement. Only users with the `SUPER` privilege can access information from this table.

The `WSREP_MEMBERSHIP` table is part of the [WSREP_INFO plugin](#).

Example

```

SELECT * FROM information_schema.WSREP_MEMBERSHIP;
+-----+-----+-----+
| INDEX | UUID           | NAME   | ADDRESS      |
+-----+-----+-----+
|     0 | 46da96e3-6e9e-11e4-95a2-f609aa5444b3 | node1 | 10.0.2.15:16000 |
|     1 | 5f6bc72a-6e9e-11e4-84ed-57ec6780a3d3 | node2 | 10.0.2.15:16001 |
|     2 | 7473fd75-6e9e-11e4-91de-0b541ad91bd0 | node3 | 10.0.2.15:16002 |
+-----+-----+-----+

```

1.1.2.9.1.1.61 Information Schema WSREP_STATUS Table

The `WSREP_STATUS` table makes [Galera](#) node cluster status information available through the [Information Schema](#). The same information can be returned using the `SHOW WSREP_STATUS` statement. Only users with the `SUPER` privilege can access information from this table.

The `WSREP_STATUS` table is part of the [WSREP_INFO plugin](#).

Example

```

SELECT * FROM information_schema.WSREP_STATUS\G
*****
1. row *****
  NODE_INDEX: 0
  NODE_STATUS: Synced
  CLUSTER_STATUS: Primary
  CLUSTER_SIZE: 3
  CLUSTER_STATE_UUID: 00b0fbad-6e84-11e4-8a8b-376f19ce8ee7
  CLUSTER_STATE_SEQNO: 2
  CLUSTER_CONF_ID: 3
  GAP: NO
  PROTOCOL_VERSION: 3

```

1.1.2.9.1.2

1.1.2.9.1.3 TIME_MS column in INFORMATION_SCHEMA.PROCESSLIST

In MariaDB, an extra column `TIME_MS` has been added to the `INFORMATION_SCHEMA.PROCESSLIST` table. This column shows the same information as the column '`TIME`', but in units of milliseconds with microsecond precision (the unit and precision of the `TIME` column is one second).

For details about microseconds support in MariaDB, see [microseconds in MariaDB](#).

The value displayed in the `TIME` and `TIME_MS` columns is the period of time that the given thread has been in its current state. Thus it can be used to check for example how long a thread has been executing the current query, or for how long it has been idle.

```

select id, time, time_ms, command, state from
  information_schema.processlist, (select sleep(2)) t;
+----+-----+-----+-----+
| id | time | time_ms | command | state   |
+----+-----+-----+-----+
| 37 |    2 | 2000.493 | Query   | executing |
+----+-----+-----+-----+

```

Note that as a difference to MySQL, in MariaDB the `TIME` column (and also the `TIME_MS` column) are not affected by any setting of `@TIMESTAMP`. This means that it can be reliably used also for threads that change `@TIMESTAMP` (such as the `replication` SQL thread). See also [MySQL Bug #22047](#).

As a consequence of this, the `TIME` column of `SHOW FULL PROCESSLIST` and `INFORMATION_SCHEMA.PROCESSLIST` can not be used to determine if a slave is lagging behind. For this, use instead the `Seconds_Behind_Master` column in the output of `SHOW SLAVE STATUS`.

The addition of the `TIME_MS` column is based on the `microsec_process` patch, developed by [Percona](#).

1.1.2.9.2 Performance Schema

1.1.2.9.2.1 Performance Schema Tables

1.1.2.9.2.1.1 List of Performance Schema Tables

Below is a list of all [Performance Schema](#) tables as well as a brief description of each of them.

Table	Description
<code>accounts</code>	Client account connection statistics.
<code>cond_instances</code>	Synchronization object instances.
<code>events_stages_current</code>	Current stage events.
<code>events_stages_history</code>	Ten most recent stage events per thread.
<code>events_stages_history_long</code>	Ten thousand most recent stage events.
<code>events_stages_summary_by_account_by_event_name</code>	Summarized stage events per account and event name.

events_stages_summary_by_host_by_event_name	Summarized stage events per host and event name.
events_stages_summary_by_thread_by_event_name	Summarized stage events per thread and event name.
events_stages_summary_by_user_by_event_name	Summarized stage events per user name and event name.
events_stages_summary_global_by_event_name	Summarized stage events per event name.
events_statements_current	Current statement events.
events_statements_history	Ten most recent events per thread.
events_statements_history_long	Ten thousand most recent stage events.
events_statements_summary_by_account_by_event_name	Summarized statement events per account and event name.
events_statements_summary_by_digest	Summarized statement events by scheme and digest.
events_statements_summary_by_host_by_event_name	Summarized statement events by host and event name.
events_statements_summary_by_program	
events_statements_summary_by_thread_by_event_name	Summarized statement events by thread and event name.
events_statements_summary_by_user_by_event_name	Summarized statement events by user and event name.
events_statements_summary_global_by_event_name	Summarized statement events by event name.
events_transactions_current	Current transaction events for each thread.
events_transactions_history	Most recent completed transaction events for each thread.
events_transactions_history_long	Most recent completed transaction events that have ended globally.
events_transactions_summary_by_account_by_event_name	Transaction events aggregated by account and event.
events_transactions_summary_by_host_by_event_name	Transaction events aggregated by host and event..
events_transactions_summary_by_thread_by_event_name	Transaction events aggregated by thread and event..
events_transactions_summary_by_user_by_event_name	Transaction events aggregated by user and event..
events_transactions_summary_global_by_event_name	Transaction events aggregated by event name.
events_waits_current	Current wait events.
events_waits_history	Ten most recent wait events per thread.
events_waits_history_long	Ten thousand most recent wait events per thread.
events_waits_summary_by_account_by_event_name	Summarized wait events by account and event name.
events_waits_summary_by_host_by_event_name	Summarized wait events by host and event name.
events_waits_summary_by_instance	Summarized wait events by instance.
events_waits_summary_by_thread_by_event_name	Summarized wait events by thread and event name.
events_waits_summary_by_user_by_event_name	Summarized wait events by user and event name.
events_waits_summary_global_by_event_name	Summarized wait events by event name.
file_instances	Seen files.
file_summary_by_event_name	File events summarized by event name.
file_summary_by_instance	File events summarized by instance.
global_status	Global status variables and values.
host_cache	Host and IP information.
hosts	Connections by host.
memory_summary_by_account_by_event_name	Memory usage statistics aggregated by account and event.
memory_summary_by_host_by_event_name	Memory usage statistics aggregated by host. and event.
memory_summary_by_thread_by_event_name	Memory usage statistics aggregated by thread and event..
memory_summary_by_user_by_event_name	Memory usage statistics aggregated by user and event..

memory_summary_global_by_event_name	Memory usage statistics aggregated by event.
metadata_locks	Metadata locks.
mutex_instances	Seen mutexes.
objects_summary_global_by_type	Object wait events.
performance_timers	Available event timers.
prepared_statements_instances	Aggregate statistics of prepared statements.
replication_applier_configuration	Configuration settings affecting replica transactions.
replication_applier_status	General transaction execution status on the replica.
replication_applier_status_by_coordinator	Coordinator thread specific information.
replication_applier_status_by_worker	Replica worker thread specific information.
replication_connection_configuration	Replica's configuration settings used for connecting to the primary.
rwlock_instances	Seen read-write locks.
session_account_connect_attrs	Current session connection attributes.
session_connect_attrs	All session connection attributes.
session_status	Session status variables and values.
setup_actors	Details on foreground thread monitoring.
setup_consumers	Consumers for which event information is stored.
setup_instruments	Instrumented objects for which events are collected.
setup_objects	Objects to be monitored.
setup_timers	Currently selected event timers.
socket_instances	Active connections.
socket_summary_by_event_name	Timer and byte count statistics by socket instrument.
socket_summary_by_instance	Timer and byte count statistics by socket instance.
status_by_thread	Status variable info about active foreground threads.
table_io_waits_summary_by_index_usage	Aggregate table I/O wait events by index.
table_io_waits_summary_by_table	Aggregate table I/O wait events by table.
table_lock_waits_summary_by_table	Aggregate table lock wait events by table.
threads	Server thread information.
users	Connection statistics by user.

1.1.2.9.2.1.2 Performance Schema accounts Table

Description

Each account that connects to the server is stored as a row in the accounts table, along with current and total connections.

The table size is determined at startup by the value of the `performance_schema_accounts_size` system variable. If this is set to 0, account statistics will be disabled.

Column	Description
USER	The connection's client user name for the connection, or NULL if an internal thread.
HOST	The connection client's host name, or NULL if an internal thread.
CURRENT_CONNECTIONS	Current connections for the account.
TOTAL_CONNECTIONS	Total connections for the account.

The `USER` and `HOST` values shown here are the username and host used for user connections, not the patterns used to check permissions.

Example

```
SELECT * FROM performance_schema.accounts;
+-----+-----+-----+-----+
| USER | HOST | CURRENT_CONNECTIONS | TOTAL_CONNECTIONS |
+-----+-----+-----+-----+
| root | localhost | 1 | 2 |
| NULL | NULL | 20 | 23 |
| debian-sys-maint | localhost | 0 | 35 |
+-----+-----+-----+-----+
```

1.1.2.9.2.1.3 Performance Schema cond_instances Table

Description

The `cond_instances` table lists all conditions while the server is executing. A condition, or instrumented condition object, is an internal code mechanism used for signalling that a specific event has occurred so that any threads waiting for this condition can continue.

The maximum number of conditions stored in the performance schema is determined by the `performance_schema_max_cond_instances` system variable.

Column	Description
NAME	Client user name for the connection, or <code>NULL</code> if an internal thread.
OBJECT_INSTANCE_BEGIN	Address in memory of the instrumented condition.

1.1.2.9.2.1.4 Performance Schema events_stages_current Table

The `events_stages_current` table contains current stage events, with each row being a record of a thread and its most recent stage event.

The table contains the following columns:

Column	Description
THREAD_ID	Thread associated with the event. Together with <code>EVENT_ID</code> uniquely identifies the row.
EVENT_ID	Thread's current event number at the start of the event. Together with <code>THREAD_ID</code> uniquely identifies the row.
END_EVENT_ID	<code>NULL</code> when the event starts, set to the thread's current event number at the end of the event.
EVENT_NAME	Event instrument name and a <code>NAME</code> from the <code>setup_instruments</code> table
SOURCE	Name and line number of the source file containing the instrumented code that produced the event.
TIMER_START	Value in picoseconds when the event timing started or <code>NULL</code> if timing is not collected.
TIMER_END	Value in picoseconds when the event timing ended, or <code>NULL</code> if the event has not ended or timing is not collected.
TIMER_WAIT	Value in picoseconds of the event's duration or <code>NULL</code> if the event has not ended or timing is not collected.
NESTING_EVENT_ID	<code>EVENT_ID</code> of event within which this event nests.
NESTING_EVENT_TYPE	Nesting event type. One of <code>transaction</code> , <code>statement</code> , <code>stage</code> or <code>wait</code> .

It is possible to empty this table with a `TRUNCATE TABLE` statement.

The related tables, `events_stages_history` and `events_stages_history_long` derive their values from the current events.

1.1.2.9.2.1.5 Performance Schema events_stages_history Table

The `events_stages_history` table by default contains the ten most recent completed stage events per thread. This number can be adjusted by

setting the [performance_schema_events_stages_history_size](#) system variable when the server starts up.

The table structure is identical to the [events_stage_current](#) table structure, and contains the following columns:

Column	Description
THREAD_ID	Thread associated with the event. Together with EVENT_ID uniquely identifies the row.
EVENT_ID	Thread's current event number at the start of the event. Together with THREAD_ID uniquely identifies the row.
END_EVENT_ID	NULL when the event starts, set to the thread's current event number at the end of the event.
EVENT_NAME	Event instrument name and a NAME from the setup_instruments table
SOURCE	Name and line number of the source file containing the instrumented code that produced the event.
TIMER_START	Value in picoseconds when the event timing started or NULL if timing is not collected.
TIMER_END	Value in picoseconds when the event timing ended, or NULL if timing is not collected.
TIMER_WAIT	Value in picoseconds of the event's duration or NULL if timing is not collected.
NESTING_EVENT_ID	EVENT_ID of event within which this event nests.
NESTING_EVENT_TYPE	Nesting event type. One of transaction , statement , stage or wait .

It is possible to empty this table with a TRUNCATE TABLE statement.

[events_stages_current](#) and [events_stages_history_long](#) are related tables.

1.1.2.9.2.1.6 Performance Schema events_stages_history_long Table

The `events_stages_history_long` table by default contains the ten thousand most recent completed stage events. This number can be adjusted by setting the [performance_schema_events_stages_history_long_size](#) system variable when the server starts up.

The table structure is identical to the `events_stage_current` table structure, and contains the following columns:

Column	Description
THREAD_ID	Thread associated with the event. Together with EVENT_ID uniquely identifies the row.
EVENT_ID	Thread's current event number at the start of the event. Together with THREAD_ID uniquely identifies the row.
END_EVENT_ID	NULL when the event starts, set to the thread's current event number at the end of the event.
EVENT_NAME	Event instrument name and a NAME from the setup_instruments table
SOURCE	Name and line number of the source file containing the instrumented code that produced the event.
TIMER_START	Value in picoseconds when the event timing started or NULL if timing is not collected.
TIMER_END	Value in picoseconds when the event timing ended, or NULL if timing is not collected.
TIMER_WAIT	Value in picoseconds of the event's duration or NULL if timing is not collected.
NESTING_EVENT_ID	EVENT_ID of event within which this event nests.
NESTING_EVENT_TYPE	Nesting event type. One of transaction , statement , stage or wait .

It is possible to empty this table with a TRUNCATE TABLE statement.

[events_stages_current](#) and [events_stages_history](#) are related tables.

1.1.2.9.2.1.7 Performance Schema events_stages_summary_by_account_by_event_name Table

The table lists stage events, summarized by account and event name.

It contains the following columns:

Column	Description
--------	-------------

USER	User. Used together with HOST and EVENT_NAME for grouping events.
HOST	Host. Used together with USER and EVENT_NAME for grouping events.
EVENT_NAME	Event name. Used together with USER and HOST for grouping events.
COUNT_STAR	Number of summarized events, which includes all timed and untimed events.
SUM_TIMER_WAIT	Total wait time of the timed summarized events.
MIN_TIMER_WAIT	Minimum wait time of the timed summarized events.
AVG_TIMER_WAIT	Average wait time of the timed summarized events.
MAX_TIMER_WAIT	Maximum wait time of the timed summarized events.

Example

```

SELECT * FROM events_stages_summary_by_account_by_event_name\G
...
***** 325. row *****
    USER: NULL
    HOST: NULL
    EVENT_NAME: stage/sql/Waiting for event metadata lock
    COUNT_STAR: 0
    SUM_TIMER_WAIT: 0
    MIN_TIMER_WAIT: 0
    AVG_TIMER_WAIT: 0
    MAX_TIMER_WAIT: 0
***** 326. row *****
    USER: NULL
    HOST: NULL
    EVENT_NAME: stage/sql/Waiting for commit lock
    COUNT_STAR: 0
    SUM_TIMER_WAIT: 0
    MIN_TIMER_WAIT: 0
    AVG_TIMER_WAIT: 0
    MAX_TIMER_WAIT: 0
***** 327. row *****
    USER: NULL
    HOST: NULL
    EVENT_NAME: stage/aria/Waiting for a resource
    COUNT_STAR: 0
    SUM_TIMER_WAIT: 0
    MIN_TIMER_WAIT: 0
    AVG_TIMER_WAIT: 0
    MAX_TIMER_WAIT: 0

```

1.1.2.9.2.1.8 Performance Schema events_stages_summary_by_host_by_event_name Table

The table lists stage events, summarized by host and event name.

It contains the following columns:

Column	Description
HOST	Host. Used together with EVENT_NAME for grouping events.
EVENT_NAME	Event name. Used together with HOST for grouping events.
COUNT_STAR	Number of summarized events, which includes all timed and untimed events.
SUM_TIMER_WAIT	Total wait time of the timed summarized events.
MIN_TIMER_WAIT	Minimum wait time of the timed summarized events.
AVG_TIMER_WAIT	Average wait time of the timed summarized events.
MAX_TIMER_WAIT	Maximum wait time of the timed summarized events.

Example

```
SELECT * FROM events_stages_summary_by_host_by_event_name\G
...
***** 216. row *****
HOST: NULL
EVENT_NAME: stage/sql/Waiting for event metadata lock
COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
***** 217. row *****
HOST: NULL
EVENT_NAME: stage/sql/Waiting for commit lock
COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
***** 218. row *****
HOST: NULL
EVENT_NAME: stage/aria/Waiting for a resource
COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
```

1.1.2.9.2.1.9 Performance Schema events_stages_summary_by_thread_by_event_name Table

The table lists stage events, summarized by thread and event name.

It contains the following columns:

Column	Description
THREAD_ID	Thread associated with the event. Together with EVENT_NAME uniquely identifies the row.
EVENT_NAME	Event name. Used together with THREAD_ID for grouping events.
COUNT_STAR	Number of summarized events, which includes all timed and untimed events.
SUM_TIMER_WAIT	Total wait time of the timed summarized events.
MIN_TIMER_WAIT	Minimum wait time of the timed summarized events.
AVG_TIMER_WAIT	Average wait time of the timed summarized events.
MAX_TIMER_WAIT	Maximum wait time of the timed summarized events.

Example

```

SELECT * FROM events_stages_summary_by_thread_by_event_name\G
...
***** 2287. row *****
THREAD_ID: 64
EVENT_NAME: stage/sql/Waiting for event metadata lock
COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
***** 2288. row *****
THREAD_ID: 64
EVENT_NAME: stage/sql/Waiting for commit lock
COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
***** 2289. row *****
THREAD_ID: 64
EVENT_NAME: stage/aria/Waiting for a resource
COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0

```

1.1.2.9.2.1.10 Performance Schema events_stages_summary_by_user_by_event_name Table

The table lists stage events, summarized by user and event name.

It contains the following columns:

Column	Description
USER	User. Used together with EVENT_NAME for grouping events.
EVENT_NAME	Event name. Used together with USER for grouping events.
COUNT_STAR	Number of summarized events, which includes all timed and untimed events.
SUM_TIMER_WAIT	Total wait time of the timed summarized events.
MIN_TIMER_WAIT	Minimum wait time of the timed summarized events.
AVG_TIMER_WAIT	Average wait time of the timed summarized events.
MAX_TIMER_WAIT	Maximum wait time of the timed summarized events.

Example

```

SELECT * FROM events_stages_summary_by_user_by_event_name\G
...
***** 325. row *****
USER: NULL
EVENT_NAME: stage/sql/Waiting for event metadata lock
COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
***** 326. row *****
USER: NULL
EVENT_NAME: stage/sql/Waiting for commit lock
COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
***** 327. row *****
USER: NULL
EVENT_NAME: stage/aria/Waiting for a resource
COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0

```

1.1.2.9.2.1.11 Performance Schema events_stages_summary_global_by_event_name Table

The table lists stage events, summarized by thread and event name.

It contains the following columns:

Column	Description
EVENT_NAME	Event name.
COUNT_STAR	Number of summarized events, which includes all timed and untimed events.
SUM_TIMER_WAIT	Total wait time of the timed summarized events.
MIN_TIMER_WAIT	Minimum wait time of the timed summarized events.
AVG_TIMER_WAIT	Average wait time of the timed summarized events.
MAX_TIMER_WAIT	Maximum wait time of the timed summarized events.

Example

```

SELECT * FROM events_stages_summary_global_by_event_name\G
...
***** 106. row *****
EVENT_NAME: stage/sql/Waiting for trigger metadata lock
COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
***** 107. row *****
EVENT_NAME: stage/sql/Waiting for event metadata lock
COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
***** 108. row *****
EVENT_NAME: stage/sql/Waiting for commit lock
COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
***** 109. row *****
EVENT_NAME: stage/aria/Waiting for a resource
COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0

```

1.1.2.9.2.1.12 Performance Schema events_statements_history Table

The `events_statements_history` table by default contains the ten most recent completed statement events per thread. This number can be adjusted by setting the `performance_schema_events_statements_history_size` system variable when the server starts up.

The table structure is identical to the `events_statements_current` table structure, and contains the following columns:

The table contains the following columns:

Column	Description
THREAD_ID	Thread associated with the event. Together with <code>EVENT_ID</code> uniquely identifies the row.
EVENT_ID	Thread's current event number at the start of the event. Together with <code>THREAD_ID</code> uniquely identifies the row.
END_EVENT_ID	<code>NULL</code> when the event starts, set to the thread's current event number at the end of the event.
EVENT_NAME	Event instrument name and a <code>NAME</code> from the <code>setup_instruments</code> table
SOURCE	Name and line number of the source file containing the instrumented code that produced the event.
TIMER_START	Value in picoseconds when the event timing started or <code>NULL</code> if timing is not collected.
TIMER_END	Value in picoseconds when the event timing ended, or <code>NULL</code> if timing is not collected.
TIMER_WAIT	Value in picoseconds of the event's duration or <code>NULL</code> if timing is not collected.
LOCK_TIME	Time in picoseconds spent waiting for locks. The time is calculated in microseconds but stored in picoseconds for compatibility with other timings.
SQL_TEXT	The SQL statement, or <code>NULL</code> if the command is not associated with an SQL statement.
DIGEST	Statement digest .
DIGEST_TEXT	Statement digest text.
CURRENT_SCHEMA	Statement's default database for the statement, or <code>NULL</code> if there was none.
OBJECT_SCHEMA	Reserved, currently <code>NULL</code>

OBJECT_NAME	Reserved, currently NULL
OBJECT_TYPE	Reserved, currently NULL
OBJECT_INSTANCE_BEGIN	Address in memory of the statement object.
MYSQL_ERRNO	Error code. See MariaDB Error Codes for a full list.
RETURNED_SQLSTATE	The SQLSTATE value.
MESSAGE_TEXT	Statement error message. See MariaDB Error Codes .
ERRORS	0 if SQLSTATE signifies completion (starting with 00) or warning (01), otherwise 1 .
WARNINGS	Number of warnings from the diagnostics area.
ROWS_AFFECTED	Number of rows affected the statement affected.
ROWS_SENT	Number of rows returned.
ROWS_EXAMINED	Number of rows read during the statement's execution.
CREATED_TMP_DISK_TABLES	Number of on-disk temp tables created by the statement.
CREATED_TMP_TABLES	Number of temp tables created by the statement.
SELECT_FULL_JOIN	Number of joins performed by the statement which did not use an index.
SELECT_FULL_RANGE_JOIN	Number of joins performed by the statement which used a range search of the first table.
SELECT_RANGE	Number of joins performed by the statement which used a range of the first table.
SELECT_RANGE_CHECK	Number of joins without keys performed by the statement that check for key usage after each row.
SELECT_SCAN	Number of joins performed by the statement which used a full scan of the first table.
SORT_MERGE_PASSES	Number of merge passes by the sort algorithm performed by the statement. If too high, you may need to increase the sort_buffer_size .
SORT_RANGE	Number of sorts performed by the statement which used a range.
SORT_ROWS	Number of rows sorted by the statement.
SORT_SCAN	Number of sorts performed by the statement which used a full table scan.
NO_INDEX_USED	0 if the statement performed a table scan with an index, 1 if without an index.
NO_GOOD_INDEX_USED	0 if a good index was found for the statement, 1 if no good index was found. See the Range checked for each record description in the EXPLAIN article.
NESTING_EVENT_ID	Reserved, currently NULL .
NESTING_EVENT_TYPE	Reserved, currently NULL .

It is possible to empty this table with a `TRUNCATE TABLE` statement.

`events_statements_current` and `events_statements_history_long` are related tables.

1.1.2.9.2.1.13 Performance Schema `events_statements_history_long` Table

The `events_statements_history_long` table by default contains the ten thousand most recent completed statement events. This number can be adjusted by setting the `performance_schema_events_statements_history_long_size` system variable when the server starts up.

The table structure is identical to the `events_statements_current` table structure, and contains the following columns:

Column	Description
THREAD_ID	Thread associated with the event. Together with <code>EVENT_ID</code> uniquely identifies the row.
EVENT_ID	Thread's current event number at the start of the event. Together with <code>THREAD_ID</code> uniquely identifies the row.
END_EVENT_ID	NULL when the event starts, set to the thread's current event number at the end of the event.
EVENT_NAME	Event instrument name and a <code>NAME</code> from the <code>setup_instruments</code> table
SOURCE	Name and line number of the source file containing the instrumented code that produced the event.

TIMER_START	Value in picoseconds when the event timing started or <code>NULL</code> if timing is not collected.
TIMER_END	Value in picoseconds when the event timing ended, or <code>NULL</code> if timing is not collected.
TIMER_WAIT	Value in picoseconds of the event's duration or <code>NULL</code> if timing is not collected.
LOCK_TIME	Time in picoseconds spent waiting for locks. The time is calculated in microseconds but stored in picoseconds for compatibility with other timings.
SQL_TEXT	The SQL statement, or <code>NULL</code> if the command is not associated with an SQL statement.
DIGEST	Statement digest .
DIGEST_TEXT	Statement digest text.
CURRENT_SCHEMA	Statement's default database for the statement, or <code>NULL</code> if there was none.
OBJECT_SCHEMA	Reserved, currently <code>NULL</code>
OBJECT_NAME	Reserved, currently <code>NULL</code>
OBJECT_TYPE	Reserved, currently <code>NULL</code>
OBJECT_INSTANCE_BEGIN	Address in memory of the statement object.
MYSQL_ERRNO	Error code. See MariaDB Error Codes for a full list.
RETURNED_SQLSTATE	The SQLSTATE value.
MESSAGE_TEXT	Statement error message. See MariaDB Error Codes .
ERRORS	<code>0</code> if <code>SQLSTATE</code> signifies completion (starting with 00) or warning (01), otherwise <code>1</code> .
WARNINGS	Number of warnings from the diagnostics area.
ROWS_AFFECTED	Number of rows affected the statement affected.
ROWS_SENT	Number of rows returned.
ROWS_EXAMINED	Number of rows read during the statement's execution.
CREATED_TMP_DISK_TABLES	Number of on-disk temp tables created by the statement.
CREATED_TMP_TABLES	Number of temp tables created by the statement.
SELECT_FULL_JOIN	Number of joins performed by the statement which did not use an index.
SELECT_FULL_RANGE_JOIN	Number of joins performed by the statement which used a range search of the first table.
SELECT_RANGE	Number of joins performed by the statement which used a range of the first table.
SELECT_RANGE_CHECK	Number of joins without keys performed by the statement that check for key usage after each row.
SELECT_SCAN	Number of joins performed by the statement which used a full scan of the first table.
SORT_MERGE_PASSES	Number of merge passes by the sort algorithm performed by the statement. If too high, you may need to increase the sort_buffer_size .
SORT_RANGE	Number of sorts performed by the statement which used a range.
SORT_ROWS	Number of rows sorted by the statement.
SORT_SCAN	Number of sorts performed by the statement which used a full table scan.
NO_INDEX_USED	<code>0</code> if the statement performed a table scan with an index, <code>1</code> if without an index.
NO_GOOD_INDEX_USED	<code>0</code> if a good index was found for the statement, <code>1</code> if no good index was found. See the <code>Range checked for each record description</code> in the EXPLAIN article.
NESTING_EVENT_ID	Reserved, currently <code>NULL</code> .
NESTING_EVENT_TYPE	Reserved, currently <code>NULL</code> .

It is possible to empty this table with a `TRUNCATE TABLE` statement.

`events_statements_current` and `events_statements_history` are related tables.

1.1.2.9.2.1.14 Performance Schema

events_statements_summary_by_account_by_event_name Table

The [Performance Schema](#) events_statements_summary_by_account_by_event_name table contains statement events summarized by account and event name. It contains the following columns:

Column	Description
USER	User. Used together with HOST and EVENT_NAME for grouping events.
HOST	Host. Used together with USER and EVENT_NAME for grouping events.
EVENT_NAME	Event name. Used together with USER and HOST for grouping events.
COUNT_STAR	Number of summarized events
SUM_TIMER_WAIT	Total wait time of the summarized events that are timed.
MIN_TIMER_WAIT	Minimum wait time of the summarized events that are timed.
AVG_TIMER_WAIT	Average wait time of the summarized events that are timed.
MAX_TIMER_WAIT	Maximum wait time of the summarized events that are timed.
SUM_LOCK_TIME	Sum of the LOCK_TIME column in the events_statements_current table.
SUM_ERRORS	Sum of the ERRORS column in the events_statements_current table.
SUM_WARNINGS	Sum of the WARNINGS column in the events_statements_current table.
SUM_ROWS_AFFECTED	Sum of the ROWS_AFFECTED column in the events_statements_current table.
SUM_ROWS_SENT	Sum of the ROWS_SENT column in the events_statements_current table.
SUM_ROWS_EXAMINED	Sum of the ROWS_EXAMINED column in the events_statements_current table.
SUM_CREATED_TMP_DISK_TABLES	Sum of the CREATED_TMP_DISK_TABLES column in the events_statements_current table.
SUM_CREATED_TMP_TABLES	Sum of the CREATED_TMP_TABLES column in the events_statements_current table.
SUM_SELECT_FULL_JOIN	Sum of the SELECT_FULL_JOIN column in the events_statements_current table.
SUM_SELECT_FULL_RANGE_JOIN	Sum of the SELECT_FULL_RANGE_JOIN column in the events_statements_current table.
SUM_SELECT_RANGE	Sum of the SELECT_RANGE column in the events_statements_current table.
SUM_SELECT_RANGE_CHECK	Sum of the SELECT_RANGE_CHECK column in the events_statements_current table.
SUM_SELECT_SCAN	Sum of the SELECT_SCAN column in the events_statements_current table.
SUM_SORT_MERGE_PASSES	Sum of the SORT_MERGE_PASSES column in the events_statements_current table.
SUM_SORT_RANGE	Sum of the SORT_RANGE column in the events_statements_current table.
SUM_SORT_ROWS	Sum of the SORT_ROWS column in the events_statements_current table.
SUM_SORT_SCAN	Sum of the SORT_SCAN column in the events_statements_current table.
SUM_NO_INDEX_USED	Sum of the NO_INDEX_USED column in the events_statements_current table.
SUM_NO_GOOD_INDEX_USED	Sum of the NO_GOOD_INDEX_USED column in the events_statements_current table.

The *_TIMER_WAIT columns only calculate results for timed events, as non-timed events have a NULL wait time.

Example

```

SELECT * FROM events_statements_summary_by_account_by_event_name\G
...
***** 521. row *****
    USER: NULL
    HOST: NULL
    EVENT_NAME: statement/com/Error
    COUNT_STAR: 0
    SUM_TIMER_WAIT: 0
    MIN_TIMER_WAIT: 0
    AVG_TIMER_WAIT: 0
    MAX_TIMER_WAIT: 0
    SUM_LOCK_TIME: 0
    SUM_ERRORS: 0
    SUM_WARNINGS: 0
    SUM_ROWS_AFFECTED: 0
    SUM_ROWS_SENT: 0
    SUM_ROWS_EXAMINED: 0
SUM_CREATED_TMP_DISK_TABLES: 0
    SUM_CREATED_TMP_TABLES: 0
    SUM_SELECT_FULL_JOIN: 0
SUM_SELECT_FULL_RANGE_JOIN: 0
    SUM_SELECT_RANGE: 0
SUM_SELECT_RANGE_CHECK: 0
    SUM_SELECT_SCAN: 0
SUM_SORT_MERGE_PASSES: 0
    SUM_SORT_RANGE: 0
    SUM_SORT_ROWS: 0
    SUM_SORT_SCAN: 0
SUM_NO_INDEX_USED: 0
SUM_NO_GOOD_INDEX_USED: 0
***** 522. row *****
    USER: NULL
    HOST: NULL
    EVENT_NAME: statement/com/
    COUNT_STAR: 0
    SUM_TIMER_WAIT: 0
    MIN_TIMER_WAIT: 0
    AVG_TIMER_WAIT: 0
    MAX_TIMER_WAIT: 0
    SUM_LOCK_TIME: 0
    SUM_ERRORS: 0
    SUM_WARNINGS: 0
    SUM_ROWS_AFFECTED: 0
    SUM_ROWS_SENT: 0
    SUM_ROWS_EXAMINED: 0
SUM_CREATED_TMP_DISK_TABLES: 0
    SUM_CREATED_TMP_TABLES: 0
    SUM_SELECT_FULL_JOIN: 0
SUM_SELECT_FULL_RANGE_JOIN: 0
    SUM_SELECT_RANGE: 0
SUM_SELECT_RANGE_CHECK: 0
    SUM_SELECT_SCAN: 0
SUM_SORT_MERGE_PASSES: 0
    SUM_SORT_RANGE: 0
    SUM_SORT_ROWS: 0
    SUM_SORT_SCAN: 0
SUM_NO_INDEX_USED: 0
SUM_NO_GOOD_INDEX_USED: 0

```

1.1.2.9.2.1.15 Performance Schema events_statements_summary_by_digest Table

The [Performance Schema digest](#) is a hashed, normalized form of a statement with the specific data values removed. It allows statistics to be gathered for similar kinds of statements.

The [Performance Schema](#) events_statements_summary_by_digest table records statement events summarized by schema and digest. It contains the following columns:

Column	Description
SCHEMA_NAME	Database name. Records are summarised together with DIGEST .
DIGEST	Performance Schema digest. Records are summarised together with SCHEMA_NAME .
DIGEST_TEXT	The unhashed form of the digest.
COUNT_STAR	Number of summarized events
SUM_TIMER_WAIT	Total wait time of the summarized events that are timed.
MIN_TIMER_WAIT	Minimum wait time of the summarized events that are timed.
AVG_TIMER_WAIT	Average wait time of the summarized events that are timed.
MAX_TIMER_WAIT	Maximum wait time of the summarized events that are timed.
SUM_LOCK_TIME	Sum of the LOCK_TIME column in the events_statements_current table.
SUM_ERRORS	Sum of the ERRORS column in the events_statements_current table.
SUM_WARNINGS	Sum of the WARNINGS column in the events_statements_current table.
SUM_ROWS_AFFECTED	Sum of the ROWS_AFFECTED column in the events_statements_current table.
SUM_ROWS_SENT	Sum of the ROWS_SENT column in the events_statements_current table.
SUM_ROWS_EXAMINED	Sum of the ROWS_EXAMINED column in the events_statements_current table.
SUM_CREATED_TMP_DISK_TABLES	Sum of the CREATED_TMP_DISK_TABLES column in the events_statements_current table.
SUM_CREATED_TMP_TABLES	Sum of the CREATED_TMP_TABLES column in the events_statements_current table.
SUM_SELECT_FULL_JOIN	Sum of the SELECT_FULL_JOIN column in the events_statements_current table.
SUM_SELECT_FULL_RANGE_JOIN	Sum of the SELECT_FULL_RANGE_JOIN column in the events_statements_current table.
SUM_SELECT_RANGE	Sum of the SELECT_RANGE column in the events_statements_current table.
SUM_SELECT_RANGE_CHECK	Sum of the SELECT_RANGE_CHECK column in the events_statements_current table.
SUM_SELECT_SCAN	Sum of the SELECT_SCAN column in the events_statements_current table.
SUM_SORT_MERGE_PASSES	Sum of the SORT_MERGE_PASSES column in the events_statements_current table.
SUM_SORT_RANGE	Sum of the SORT_RANGE column in the events_statements_current table.
SUM_SORT_ROWS	Sum of the SORT_ROWS column in the events_statements_current table.
SUM_SORT_SCAN	Sum of the SORT_SCAN column in the events_statements_current table.
SUM_NO_INDEX_USED	Sum of the NO_INDEX_USED column in the events_statements_current table.
SUM_NO_GOOD_INDEX_USED	Sum of the NO_GOOD_INDEX_USED column in the events_statements_current table.
FIRST_SEEN	Time at which the digest was first seen.
LAST_SEEN	Time at which the digest was most recently seen.

The *_TIMER_WAIT columns only calculate results for timed events, as non-timed events have a NULL wait time.

The events_statements_summary_by_digest table is limited in size by the `performance_schema_digests_size` system variable. Once the limit has been reached and the table is full, all entries are aggregated in a row with a NULL digest. The COUNT_STAR value of this NULL row indicates how many digests are recorded in the row and therefore gives an indication of whether `performance_schema_digests_size` should be increased to provide more accurate statistics.

1.1.2.9.2.1.16 Performance Schema events_statements_summary_by_host_by_event_name Table

The Performance Schema events_statements_summary_by_host_by_event_name table contains statement events summarized by host and event name. It contains the following columns:

Column	Description
--------	-------------

HOST	Host. Used together with EVENT_NAME for grouping events.
EVENT_NAME	Event name. Used together with HOST for grouping events.
COUNT_STAR	Number of summarized events
SUM_TIMER_WAIT	Total wait time of the summarized events that are timed.
MIN_TIMER_WAIT	Minimum wait time of the summarized events that are timed.
AVG_TIMER_WAIT	Average wait time of the summarized events that are timed.
MAX_TIMER_WAIT	Maximum wait time of the summarized events that are timed.
SUM_LOCK_TIME	Sum of the LOCK_TIME column in the events_statements_current table.
SUM_ERRORS	Sum of the ERRORS column in the events_statements_current table.
SUM_WARNINGS	Sum of the WARNINGS column in the events_statements_current table.
SUM_ROWS_AFFECTED	Sum of the ROWS_AFFECTED column in the events_statements_current table.
SUM_ROWS_SENT	Sum of the ROWS_SENT column in the events_statements_current table.
SUM_ROWS_EXAMINED	Sum of the ROWS_EXAMINED column in the events_statements_current table.
SUM_CREATED_TMP_DISK_TABLES	Sum of the CREATED_TMP_DISK_TABLES column in the events_statements_current table.
SUM_CREATED_TMP_TABLES	Sum of the CREATED_TMP_TABLES column in the events_statements_current table.
SUM_SELECT_FULL_JOIN	Sum of the SELECT_FULL_JOIN column in the events_statements_current table.
SUM_SELECT_FULL_RANGE_JOIN	Sum of the SELECT_FULL_RANGE_JOIN column in the events_statements_current table.
SUM_SELECT_RANGE	Sum of the SELECT_RANGE column in the events_statements_current table.
SUM_SELECT_RANGE_CHECK	Sum of the SELECT_RANGE_CHECK column in the events_statements_current table.
SUM_SELECT_SCAN	Sum of the SELECT_SCAN column in the events_statements_current table.
SUM_SORT_MERGE_PASSES	Sum of the SORT_MERGE_PASSES column in the events_statements_current table.
SUM_SORT_RANGE	Sum of the SORT_RANGE column in the events_statements_current table.
SUM_SORT_ROWS	Sum of the SORT_ROWS column in the events_statements_current table.
SUM_SORT_SCAN	Sum of the SORT_SCAN column in the events_statements_current table.
SUM_NO_INDEX_USED	Sum of the NO_INDEX_USED column in the events_statements_current table.
SUM_NO_GOOD_INDEX_USED	Sum of the NO_GOOD_INDEX_USED column in the events_statements_current table.

The *_TIMER_WAIT columns only calculate results for timed events, as non-timed events have a NULL wait time.

Example

```

SELECT * FROM events_statements_summary_by_host_by_event_name\G
...
***** 347. row *****
    HOST: NULL
    EVENT_NAME: statement/com/Error
    COUNT_STAR: 0
    SUM_TIMER_WAIT: 0
    MIN_TIMER_WAIT: 0
    AVG_TIMER_WAIT: 0
    MAX_TIMER_WAIT: 0
    SUM_LOCK_TIME: 0
    SUM_ERRORS: 0
    SUM_WARNINGS: 0
    SUM_ROWS_AFFECTED: 0
    SUM_ROWS_SENT: 0
    SUM_ROWS_EXAMINED: 0
SUM_CREATED_TMP_DISK_TABLES: 0
    SUM_CREATED_TMP_TABLES: 0
    SUM_SELECT_FULL_JOIN: 0
SUM_SELECT_FULL_RANGE_JOIN: 0
    SUM_SELECT_RANGE: 0
    SUM_SELECT_RANGE_CHECK: 0
    SUM_SELECT_SCAN: 0
SUM_SORT_MERGE_PASSES: 0
    SUM_SORT_RANGE: 0
    SUM_SORT_ROWS: 0
    SUM_SORT_SCAN: 0
    SUM_NO_INDEX_USED: 0
    SUM_NO_GOOD_INDEX_USED: 0
***** 348. row *****
    HOST: NULL
    EVENT_NAME: statement/com/
    COUNT_STAR: 0
    SUM_TIMER_WAIT: 0
    MIN_TIMER_WAIT: 0
    AVG_TIMER_WAIT: 0
    MAX_TIMER_WAIT: 0
    SUM_LOCK_TIME: 0
    SUM_ERRORS: 0
    SUM_WARNINGS: 0
    SUM_ROWS_AFFECTED: 0
    SUM_ROWS_SENT: 0
    SUM_ROWS_EXAMINED: 0
SUM_CREATED_TMP_DISK_TABLES: 0
    SUM_CREATED_TMP_TABLES: 0
    SUM_SELECT_FULL_JOIN: 0
SUM_SELECT_FULL_RANGE_JOIN: 0
    SUM_SELECT_RANGE: 0
    SUM_SELECT_RANGE_CHECK: 0
    SUM_SELECT_SCAN: 0
SUM_SORT_MERGE_PASSES: 0
    SUM_SORT_RANGE: 0
    SUM_SORT_ROWS: 0
    SUM_SORT_SCAN: 0
    SUM_NO_INDEX_USED: 0
    SUM_NO_GOOD_INDEX_USED: 0

```

1.1.2.9.2.1.17 Performance Schema events_statements_summary_by_program Table

MariaDB starting with 10.5.2

The `events_statements_summary_by_program` table, along with many other new [Performance Schema tables](#), was added in [MariaDB 10.5.2](#).

Each row in the the [Performance Schema](#) `events_statements_summary_by_program` table summarizes events for a particular stored program (stored procedure, stored function, trigger or event).

It contains the following fields.

Column	Type	Null	Description
OBJECT_TYPE	enum('EVENT', 'FUNCTION', 'PROCEDURE', 'TABLE', 'trigger')	YES	Object type for which the summary is generated.
OBJECT_SCHEMA	varchar(64)	NO	The schema of the object for which the summary is generated.
OBJECT_NAME	varchar(64)	NO	The name of the object for which the summary is generated.
COUNT_STAR	bigint(20) unsigned	NO	The number of summarized events (from events_statements_current). This value includes all events, whether timed or nontimed.
SUM_TIMER_WAIT	bigint(20) unsigned	NO	The number of summarized events (from events_statements_current). This value includes all events, whether timed or nontimed.
MIN_TIMER_WAIT	bigint(20) unsigned	NO	The minimum wait time of the summarized timed events.
AVG_TIMER_WAIT	bigint(20) unsigned	NO	The average wait time of the summarized timed events.
MAX_TIMER_WAIT	bigint(20) unsigned	NO	The maximum wait time of the summarized timed events.
COUNT_STATEMENTS	bigint(20) unsigned	NO	Total number of nested statements invoked during stored program execution.
SUM_STATEMENTS_WAIT	bigint(20) unsigned	NO	The total wait time of the summarized timed statements. This value is calculated only for timed statements because nontimed statements have a wait time of NULL. The same is true for the other xxx_STATEMENT_WAIT values.
MIN_STATEMENTS_WAIT	bigint(20) unsigned	NO	The minimum wait time of the summarized timed statements.
AVG_STATEMENTS_WAIT	bigint(20) unsigned	NO	The average wait time of the summarized timed statements.
MAX_STATEMENTS_WAIT	bigint(20) unsigned	NO	The maximum wait time of the summarized timed statements.
SUM_LOCK_TIME	bigint(20) unsigned	NO	The total time spent (in picoseconds) waiting for table locks for the summarized statements.
SUM_ERRORS	bigint(20) unsigned	NO	The total number of errors that occurred for the summarized statements.
SUM_WARNINGS	bigint(20) unsigned	NO	The total number of warnings that occurred for the summarized statements.
SUM_ROWS_AFFECTED	bigint(20) unsigned	NO	The total number of affected rows by the summarized statements.
SUM_ROWS_SENT	bigint(20) unsigned	NO	The total number of rows returned by the summarized statements.
SUM_ROWS_EXAMINED	bigint(20) unsigned	NO	The total number of rows examined by the summarized statements. The total number of affected rows by the summarized statements.
SUM_CREATED_TMP_DISK_TABLES	bigint(20) unsigned	NO	The total number of on-disk temporary tables created by the summarized statements.
SUM_CREATED_TMP_TABLES	bigint(20) unsigned	NO	The total number of in-memory temporary tables created by the summarized statements.
SUM_SELECT_FULL_JOIN	bigint(20) unsigned	NO	The total number of full joins executed by the summarized statements.
SUM_SELECT_FULL_RANGE_JOIN	bigint(20) unsigned	NO	The total number of range search joins executed by the summarized statements.
SUM_SELECT_RANGE	bigint(20) unsigned	NO	The total number of joins that used ranges on the first table executed by the summarized statements.
SUM_SELECT_RANGE_CHECK	bigint(20) unsigned	NO	The total number of joins that check for key usage after each row executed by the summarized statements.
SUM_SELECT_SCAN	bigint(20) unsigned	NO	The total number of joins that did a full scan of the first table executed by the summarized statements.
SUM_SORT_MERGE_PASSES	bigint(20) unsigned	NO	The total number of merge passes that the sort algorithm has had to do for the summarized statements.

SUM_SORT_RANGE	bigint(20) unsigned	NO	The total number of sorts that were done using ranges for the summarized statements.
SUM_SORT_ROWS	bigint(20) unsigned	NO	The total number of sorted rows that were sorted by the summarized statements.
SUM_SORT_SCAN	bigint(20) unsigned	NO	The total number of sorts that were done by scanning the table by the summarized statements.
SUM_NO_INDEX_USED	bigint(20) unsigned	NO	The total number of statements that performed a table scan without using an index.
SUM_NO_GOOD_INDEX_USED	bigint(20) unsigned	NO	The total number of statements where no good index was found.

1.1.2.9.2.1.18 Performance Schema events_statements_summary_by_thread_by_event_name Table

The [Performance Schema](#) `events_statements_summary_by_thread_by_event_name` table contains statement events summarized by thread and event name. It contains the following columns:

Column	Description
THREAD_ID	Thread associated with the event. Together with <code>EVENT_NAME</code> uniquely identifies the row.
EVENT_NAME	Event name. Used together with <code>THREAD_ID</code> for grouping events.
COUNT_STAR	Number of summarized events
SUM_TIMER_WAIT	Total wait time of the summarized events that are timed.
MIN_TIMER_WAIT	Minimum wait time of the summarized events that are timed.
AVG_TIMER_WAIT	Average wait time of the summarized events that are timed.
MAX_TIMER_WAIT	Maximum wait time of the summarized events that are timed.
SUM_LOCK_TIME	Sum of the <code>LOCK_TIME</code> column in the <code>events_statements_current</code> table.
SUM_ERRORS	Sum of the <code>ERRORS</code> column in the <code>events_statements_current</code> table.
SUM_WARNINGS	Sum of the <code>WARNINGS</code> column in the <code>events_statements_current</code> table.
SUM_ROWS_AFFECTED	Sum of the <code>ROWS_AFFECTED</code> column in the <code>events_statements_current</code> table.
SUM_ROWS_SENT	Sum of the <code>ROWS_SENT</code> column in the <code>events_statements_current</code> table.
SUM_ROWS_EXAMINED	Sum of the <code>ROWS_EXAMINED</code> column in the <code>events_statements_current</code> table.
SUM_CREATED_TMP_DISK_TABLES	Sum of the <code>CREATED_TMP_DISK_TABLES</code> column in the <code>events_statements_current</code> table.
SUM_CREATED_TMP_TABLES	Sum of the <code>CREATED_TMP_TABLES</code> column in the <code>events_statements_current</code> table.
SUM_SELECT_FULL_JOIN	Sum of the <code>SELECT_FULL_JOIN</code> column in the <code>events_statements_current</code> table.
SUM_SELECT_FULL_RANGE_JOIN	Sum of the <code>SELECT_FULL_RANGE_JOIN</code> column in the <code>events_statements_current</code> table.
SUM_SELECT_RANGE	Sum of the <code>SELECT_RANGE</code> column in the <code>events_statements_current</code> table.
SUM_SELECT_RANGE_CHECK	Sum of the <code>SELECT_RANGE_CHECK</code> column in the <code>events_statements_current</code> table.
SUM_SELECT_SCAN	Sum of the <code>SELECT_SCAN</code> column in the <code>events_statements_current</code> table.
SUM_SORT_MERGE_PASSES	Sum of the <code>SORT_MERGE_PASSES</code> column in the <code>events_statements_current</code> table.
SUM_SORT_RANGE	Sum of the <code>SORT_RANGE</code> column in the <code>events_statements_current</code> table.
SUM_SORT_ROWS	Sum of the <code>SORT_ROWS</code> column in the <code>events_statements_current</code> table.
SUM_SORT_SCAN	Sum of the <code>SORT_SCAN</code> column in the <code>events_statements_current</code> table.
SUM_NO_INDEX_USED	Sum of the <code>NO_INDEX_USED</code> column in the <code>events_statements_current</code> table.
SUM_NO_GOOD_INDEX_USED	Sum of the <code>NO_GOOD_INDEX_USED</code> column in the <code>events_statements_current</code> table.

The *_TIMER_WAIT columns only calculate results for timed events, as non-timed events have a NULL wait time.

Example

```
SELECT * FROM events_statements_summary_by_thread_by_event_name\G
...
*****3653. row *****
    THREAD_ID: 64
    EVENT_NAME: statement/com/Error
    COUNT_STAR: 0
    SUM_TIMER_WAIT: 0
    MIN_TIMER_WAIT: 0
    AVG_TIMER_WAIT: 0
    MAX_TIMER_WAIT: 0
    SUM_LOCK_TIME: 0
    SUM_ERRORS: 0
    SUM_WARNINGS: 0
    SUM_ROWS_AFFECTED: 0
    SUM_ROWS_SENT: 0
    SUM_ROWS_EXAMINED: 0
    SUM_CREATED_TMP_DISK_TABLES: 0
    SUM_CREATED_TMP_TABLES: 0
    SUM_SELECT_FULL_JOIN: 0
    SUM_SELECT_FULL_RANGE_JOIN: 0
    SUM_SELECT_RANGE: 0
    SUM_SELECT_RANGE_CHECK: 0
    SUM_SELECT_SCAN: 0
    SUM_SORT_MERGE_PASSES: 0
    SUM_SORT_RANGE: 0
    SUM_SORT_ROWS: 0
    SUM_SORT_SCAN: 0
    SUM_NO_INDEX_USED: 0
    SUM_NO_GOOD_INDEX_USED: 0
*****
*****3654. row *****
    THREAD_ID: 64
    EVENT_NAME: statement/com/
    COUNT_STAR: 0
    SUM_TIMER_WAIT: 0
    MIN_TIMER_WAIT: 0
    AVG_TIMER_WAIT: 0
    MAX_TIMER_WAIT: 0
    SUM_LOCK_TIME: 0
    SUM_ERRORS: 0
    SUM_WARNINGS: 0
    SUM_ROWS_AFFECTED: 0
    SUM_ROWS_SENT: 0
    SUM_ROWS_EXAMINED: 0
    SUM_CREATED_TMP_DISK_TABLES: 0
    SUM_CREATED_TMP_TABLES: 0
    SUM_SELECT_FULL_JOIN: 0
    SUM_SELECT_FULL_RANGE_JOIN: 0
    SUM_SELECT_RANGE: 0
    SUM_SELECT_RANGE_CHECK: 0
    SUM_SELECT_SCAN: 0
    SUM_SORT_MERGE_PASSES: 0
    SUM_SORT_RANGE: 0
    SUM_SORT_ROWS: 0
    SUM_SORT_SCAN: 0
    SUM_NO_INDEX_USED: 0
    SUM_NO_GOOD_INDEX_USED: 0
```

1.1.2.9.2.1.19 Performance Schema events_statements_summary_by_user_by_event_name Table

The [Performance Schema](#) `events_statements_summary_by_user_by_event_name` table contains statement events summarized by user and event name. It contains the following columns:

Column	Description
USER	User. Used together with <code>EVENT_NAME</code> for grouping events.
EVENT_NAME	Event name. Used together with <code>USER</code> for grouping events.
COUNT_STAR	Number of summarized events
SUM_TIMER_WAIT	Total wait time of the summarized events that are timed.
MIN_TIMER_WAIT	Minimum wait time of the summarized events that are timed.
AVG_TIMER_WAIT	Average wait time of the summarized events that are timed.
MAX_TIMER_WAIT	Maximum wait time of the summarized events that are timed.
SUM_LOCK_TIME	Sum of the <code>LOCK_TIME</code> column in the <code>events_statements_current</code> table.
SUM_ERRORS	Sum of the <code>ERRORS</code> column in the <code>events_statements_current</code> table.
SUM_WARNINGS	Sum of the <code>WARNINGS</code> column in the <code>events_statements_current</code> table.
SUM_ROWS_AFFECTED	Sum of the <code>ROWS_AFFECTED</code> column in the <code>events_statements_current</code> table.
SUM_ROWS_SENT	Sum of the <code>ROWS_SENT</code> column in the <code>events_statements_current</code> table.
SUM_ROWS_EXAMINED	Sum of the <code>ROWS_EXAMINED</code> column in the <code>events_statements_current</code> table.
SUM_CREATED_TMP_DISK_TABLES	Sum of the <code>CREATED_TMP_DISK_TABLES</code> column in the <code>events_statements_current</code> table.
SUM_CREATED_TMP_TABLES	Sum of the <code>CREATED_TMP_TABLES</code> column in the <code>events_statements_current</code> table.
SUM_SELECT_FULL_JOIN	Sum of the <code>SELECT_FULL_JOIN</code> column in the <code>events_statements_current</code> table.
SUM_SELECT_FULL_RANGE_JOIN	Sum of the <code>SELECT_FULL_RANGE_JOIN</code> column in the <code>events_statements_current</code> table.
SUM_SELECT_RANGE	Sum of the <code>SELECT_RANGE</code> column in the <code>events_statements_current</code> table.
SUM_SELECT_RANGE_CHECK	Sum of the <code>SELECT_RANGE_CHECK</code> column in the <code>events_statements_current</code> table.
SUM_SELECT_SCAN	Sum of the <code>SELECT_SCAN</code> column in the <code>events_statements_current</code> table.
SUM_SORT_MERGE_PASSES	Sum of the <code>SORT_MERGE_PASSES</code> column in the <code>events_statements_current</code> table.
SUM_SORT_RANGE	Sum of the <code>SORT_RANGE</code> column in the <code>events_statements_current</code> table.
SUM_SORT_ROWS	Sum of the <code>SORT_ROWS</code> column in the <code>events_statements_current</code> table.
SUM_SORT_SCAN	Sum of the <code>SORT_SCAN</code> column in the <code>events_statements_current</code> table.
SUM_NO_INDEX_USED	Sum of the <code>NO_INDEX_USED</code> column in the <code>events_statements_current</code> table.
SUM_NO_GOOD_INDEX_USED	Sum of the <code>NO_GOOD_INDEX_USED</code> column in the <code>events_statements_current</code> table.

The `*_TIMER_WAIT` columns only calculate results for timed events, as non-timed events have a `NULL` wait time.

Example

```

SELECT * FROM events_statements_summary_by_user_by_event_name\G
...
***** 521. row *****
    USER: NULL
    EVENT_NAME: statement/com/Error
    COUNT_STAR: 0
    SUM_TIMER_WAIT: 0
    MIN_TIMER_WAIT: 0
    AVG_TIMER_WAIT: 0
    MAX_TIMER_WAIT: 0
    SUM_LOCK_TIME: 0
    SUM_ERRORS: 0
    SUM_WARNINGS: 0
    SUM_ROWS_AFFECTED: 0
    SUM_ROWS_SENT: 0
    SUM_ROWS_EXAMINED: 0
SUM_CREATED_TMP_DISK_TABLES: 0
    SUM_CREATED_TMP_TABLES: 0
    SUM_SELECT_FULL_JOIN: 0
SUM_SELECT_FULL_RANGE_JOIN: 0
    SUM_SELECT_RANGE: 0
    SUM_SELECT_RANGE_CHECK: 0
    SUM_SELECT_SCAN: 0
SUM_SORT_MERGE_PASSES: 0
    SUM_SORT_RANGE: 0
    SUM_SORT_ROWS: 0
    SUM_SORT_SCAN: 0
    SUM_NO_INDEX_USED: 0
    SUM_NO_GOOD_INDEX_USED: 0
***** 522. row *****
    USER: NULL
    EVENT_NAME: statement/com/
    COUNT_STAR: 0
    SUM_TIMER_WAIT: 0
    MIN_TIMER_WAIT: 0
    AVG_TIMER_WAIT: 0
    MAX_TIMER_WAIT: 0
    SUM_LOCK_TIME: 0
    SUM_ERRORS: 0
    SUM_WARNINGS: 0
    SUM_ROWS_AFFECTED: 0
    SUM_ROWS_SENT: 0
    SUM_ROWS_EXAMINED: 0
SUM_CREATED_TMP_DISK_TABLES: 0
    SUM_CREATED_TMP_TABLES: 0
    SUM_SELECT_FULL_JOIN: 0
SUM_SELECT_FULL_RANGE_JOIN: 0
    SUM_SELECT_RANGE: 0
    SUM_SELECT_RANGE_CHECK: 0
    SUM_SELECT_SCAN: 0
SUM_SORT_MERGE_PASSES: 0
    SUM_SORT_RANGE: 0
    SUM_SORT_ROWS: 0
    SUM_SORT_SCAN: 0
    SUM_NO_INDEX_USED: 0
    SUM_NO_GOOD_INDEX_USED: 0

```

1.1.2.9.2.1.20 Performance Schema events_statements_summary_global_by_event_name Table

The [Performance Schema](#) `events_statements_summary_global_by_event_name` table contains statement events summarized by event name. It contains the following columns:

Column	Description
--------	-------------

EVENT_NAME	Event name.
COUNT_STAR	Number of summarized events
SUM_TIMER_WAIT	Total wait time of the summarized events that are timed.
MIN_TIMER_WAIT	Minimum wait time of the summarized events that are timed.
AVG_TIMER_WAIT	Average wait time of the summarized events that are timed.
MAX_TIMER_WAIT	Maximum wait time of the summarized events that are timed.
SUM_LOCK_TIME	Sum of the <code>LOCK_TIME</code> column in the <code>events_statements_current</code> table.
SUM_ERRORS	Sum of the <code>ERRORS</code> column in the <code>events_statements_current</code> table.
SUM_WARNINGS	Sum of the <code>WARNINGS</code> column in the <code>events_statements_current</code> table.
SUM_ROWS_AFFECTED	Sum of the <code>ROWS_AFFECTED</code> column in the <code>events_statements_current</code> table.
SUM_ROWS_SENT	Sum of the <code>ROWS_SENT</code> column in the <code>events_statements_current</code> table.
SUM_ROWS_EXAMINED	Sum of the <code>ROWS_EXAMINED</code> column in the <code>events_statements_current</code> table.
SUM_CREATED_TMP_DISK_TABLES	Sum of the <code>CREATED_TMP_DISK_TABLES</code> column in the <code>events_statements_current</code> table.
SUM_CREATED_TMP_TABLES	Sum of the <code>CREATED_TMP_TABLES</code> column in the <code>events_statements_current</code> table.
SUM_SELECT_FULL_JOIN	Sum of the <code>SELECT_FULL_JOIN</code> column in the <code>events_statements_current</code> table.
SUM_SELECT_FULL_RANGE_JOIN	Sum of the <code>SELECT_FULL_RANGE_JOIN</code> column in the <code>events_statements_current</code> table.
SUM_SELECT_RANGE	Sum of the <code>SELECT_RANGE</code> column in the <code>events_statements_current</code> table.
SUM_SELECT_RANGE_CHECK	Sum of the <code>SELECT_RANGE_CHECK</code> column in the <code>events_statements_current</code> table.
SUM_SELECT_SCAN	Sum of the <code>SELECT_SCAN</code> column in the <code>events_statements_current</code> table.
SUM_SORT_MERGE_PASSES	Sum of the <code>SORT_MERGE_PASSES</code> column in the <code>events_statements_current</code> table.
SUM_SORT_RANGE	Sum of the <code>SORT_RANGE</code> column in the <code>events_statements_current</code> table.
SUM_SORT_ROWS	Sum of the <code>SORT_ROWS</code> column in the <code>events_statements_current</code> table.
SUM_SORT_SCAN	Sum of the <code>SORT_SCAN</code> column in the <code>events_statements_current</code> table.
SUM_NO_INDEX_USED	Sum of the <code>NO_INDEX_USED</code> column in the <code>events_statements_current</code> table.
SUM_NO_GOOD_INDEX_USED	Sum of the <code>NO_GOOD_INDEX_USED</code> column in the <code>events_statements_current</code> table.

The `*_TIMER_WAIT` columns only calculate results for timed events, as non-timed events have a `NULL` wait time.

Example

```

SELECT * FROM events_statements_summary_global_by_event_name\G
...
***** 173. row *****
    EVENT_NAME: statement/com/Error
    COUNT_STAR: 0
    SUM_TIMER_WAIT: 0
    MIN_TIMER_WAIT: 0
    AVG_TIMER_WAIT: 0
    MAX_TIMER_WAIT: 0
    SUM_LOCK_TIME: 0
    SUM_ERRORS: 0
    SUM_WARNINGS: 0
    SUM_ROWS_AFFECTED: 0
        SUM_ROWS_SENT: 0
    SUM_ROWS_EXAMINED: 0
SUM_CREATED_TMP_DISK_TABLES: 0
    SUM_CREATED_TMP_TABLES: 0
        SUM_SELECT_FULL_JOIN: 0
SUM_SELECT_FULL_RANGE_JOIN: 0
    SUM_SELECT_RANGE: 0
    SUM_SELECT_RANGE_CHECK: 0
        SUM_SELECT_SCAN: 0
    SUM_SORT_MERGE_PASSES: 0
        SUM_SORT_RANGE: 0
        SUM_SORT_ROWS: 0
        SUM_SORT_SCAN: 0
    SUM_NO_INDEX_USED: 0
    SUM_NO_GOOD_INDEX_USED: 0
***** 174. row *****
    EVENT_NAME: statement/com/
    COUNT_STAR: 0
    SUM_TIMER_WAIT: 0
    MIN_TIMER_WAIT: 0
    AVG_TIMER_WAIT: 0
    MAX_TIMER_WAIT: 0
    SUM_LOCK_TIME: 0
    SUM_ERRORS: 0
    SUM_WARNINGS: 0
    SUM_ROWS_AFFECTED: 0
        SUM_ROWS_SENT: 0
    SUM_ROWS_EXAMINED: 0
SUM_CREATED_TMP_DISK_TABLES: 0
    SUM_CREATED_TMP_TABLES: 0
        SUM_SELECT_FULL_JOIN: 0
SUM_SELECT_FULL_RANGE_JOIN: 0
    SUM_SELECT_RANGE: 0
    SUM_SELECT_RANGE_CHECK: 0
        SUM_SELECT_SCAN: 0
    SUM_SORT_MERGE_PASSES: 0
        SUM_SORT_RANGE: 0
        SUM_SORT_ROWS: 0
        SUM_SORT_SCAN: 0
    SUM_NO_INDEX_USED: 0
    SUM_NO_GOOD_INDEX_USED: 0

```

1.1.2.9.2.1.21 Performance Schema events_transactions_current Table

MariaDB starting with [10.5.2](#)

The events_transactions_current table was introduced in [MariaDB 10.5.2](#).

The `events_transactions_current` table contains current transaction events for each thread.

The table size cannot be figured, and always stores one row for each thread, showing the current status of the thread's most recent monitored transaction event.

The table contains the following columns:

Column	Type	Description
THREAD_ID	bigint(20) unsigned	The thread associated with the event.
EVENT_ID	bigint(20) unsigned	The event id associated with the event.
END_EVENT_ID	bigint(20) unsigned	This column is set to NULL when the event starts and updated to the thread current event number when the event ends.
EVENT_NAME	varchar(128)	The name of the instrument from which the event was collected. This is a NAME value from the setup_instruments table.
STATE	enum('ACTIVE', 'COMMITTED', 'ROLLED BACK')	The current transaction state. The value is ACTIVE (after START TRANSACTION or BEGIN), COMMITTED (after COMMIT), or ROLLED BACK (after ROLLBACK).
TRX_ID	bigint(20) unsigned	Unused.
GTID	varchar(64)	Transaction GTID , using the format DOMAIN-SERVER_ID-SEQUENCE_NO.
XID_FORMAT_ID	int(11)	XA transaction format ID for GTRID and BQUAL values.
XID_GTRID	varchar(130)	XA global transaction ID.
XID_BQUAL	varchar(130)	XA transaction branch qualifier.
XA_STATE	varchar(64)	The state of the XA transaction. The value is ACTIVE (after XA START), IDLE (after XA END), PREPARED (after XA PREPARE), ROLLED BACK (after XA ROLLBACK), or COMMITTED (after XA COMMIT).
SOURCE	varchar(64)	The name of the source file containing the instrumented code that produced the event and the line number in the file at which the instrumentation occurs.
TIMER_START	bigint(20) unsigned	The unit is picoseconds. When event timing started. NULL if event has no timing information.
TIMER_END	bigint(20) unsigned	The unit is picoseconds. When event timing ended. NULL if event has no timing information.
TIMER_WAIT	bigint(20) unsigned	The unit is picoseconds. Event duration. NULL if event has not timing information.
ACCESS_MODE	enum('READ ONLY', 'READ WRITE')	Transaction access mode.
ISOLATION_LEVEL	varchar(64)	Transaction isolation level. One of: REPEATABLE READ, READ COMMITTED, READ UNCOMMITTED, or SERIALIZABLE.
AUTOCOMMIT	enum('YES','NO')	Whether autocommit mode was enabled when the transaction started.
NUMBER_OF_SAVEPOINTS	bigint(20) unsigned	The number of SAVEPOINT statements issued during the transaction.
NUMBER_OF_ROLLBACK_TO_SAVEPOINT	bigint(20) unsigned	The number of ROLLBACK_TO_SAVEPOINT statements issued during the transaction.
NUMBER_OF_RELEASE_SAVEPOINT	bigint(20) unsigned	The number of RELEASE_SAVEPOINT statements issued during the transaction.
OBJECT_INSTANCE_BEGIN	bigint(20) unsigned	Unused.
NESTING_EVENT_ID	bigint(20) unsigned	The EVENT_ID value of the event within which this event is nested.
NESTING_EVENT_TYPE	enum('TRANSACTION', 'STATEMENT', 'STAGE', 'WAIT')	The nesting event type.

1.1.2.9.2.1.22 Performance Schema events_transactions_history Table

The `events_transactions_history` table was introduced in [MariaDB 10.5.2](#).

The `events_transactions_history` table contains the most recent completed transaction events for each thread.

The number of records stored per thread in the table is determined by the `performance_schema_events_transactions_history_size` system variable, which is autosized on startup.

If adding a completed transaction event would cause the table to exceed this limit, the oldest thread row is discarded.

All of a thread's rows are discarded when the thread ends.

The table contains the following columns:

Column	Type	Description
THREAD_ID	bigint(20) unsigned	The thread associated with the event.
EVENT_ID	bigint(20) unsigned	The event id associated with the event.
END_EVENT_ID	bigint(20) unsigned	This column is set to NULL when the event starts and updated to the thread current event number when the event ends.
EVENT_NAME	varchar(128)	The name of the instrument from which the event was collected. This is a NAME value from the <code>setup_instruments</code> table.
STATE	enum('ACTIVE', 'COMMITTED', 'ROLLED BACK')	The current transaction state. The value is ACTIVE (after START TRANSACTION or BEGIN), COMMITTED (after COMMIT), or ROLLED BACK (after ROLLBACK).
TRX_ID	bigint(20) unsigned	Unused.
GTID	varchar(64)	Transaction GTID , using the format DOMAIN-SERVER_ID-SEQUENCE_NO.
XID_FORMAT_ID	int(11)	XA transaction format ID for GTRID and BQUAL values.
XID_GTRID	varchar(130)	XA global transaction ID.
XID_BQUAL	varchar(130)	XA transaction branch qualifier.
XA_STATE	varchar(64)	The state of the XA transaction. The value is ACTIVE (after XA START), IDLE (after XA END), PREPARED (after XA PREPARE), ROLLED BACK (after XA ROLLBACK), or COMMITTED (after XA COMMIT).
SOURCE	varchar(64)	The name of the source file containing the instrumented code that produced the event and the line number in the file at which the instrumentation occurs.
TIMER_START	bigint(20) unsigned	The unit is picoseconds. When event timing started. NULL if event has no timing information.
TIMER_END	bigint(20) unsigned	The unit is picoseconds. When event timing ended. NULL if event has no timing information.
TIMER_WAIT	bigint(20) unsigned	The unit is picoseconds. Event duration. NULL if event has not timing information.
ACCESS_MODE	enum('READ ONLY', 'READ WRITE')	Transaction access mode.
ISOLATION_LEVEL	varchar(64)	Transaction isolation level. One of: REPEATABLE READ, READ COMMITTED, READ UNCOMMITTED, or SERIALIZABLE.
AUTOCOMMIT	enum('YES', 'NO')	NO
NUMBER_OF_SAVEPOINTS	bigint(20) unsigned	The number of SAVEPOINT statements issued during the transaction.
NUMBER_OF_ROLLBACK_TO_SAVEPOINT	bigint(20) unsigned	The number of ROLLBACK_TO_SAVEPOINT statements issued during the transaction.
NUMBER_OF_RELEASE_SAVEPOINT	bigint(20) unsigned	The number of RELEASE_SAVEPOINT statements issued during the transaction.
OBJECT_INSTANCE_BEGIN	bigint(20) unsigned	Unused.
NESTING_EVENT_ID	bigint(20) unsigned	The EVENT_ID value of the event within which this event is nested.

NESTING_EVENT_TYPE	enum('TRANSACTION', 'STATEMENT', 'STAGE', 'WAIT')	The nesting event type.
--------------------	---	-------------------------

1.1.2.9.2.1.23 Performance Schema events_transactions_history_long Table

MariaDB starting with [10.5.2](#)

The events_transactions_history_long table was introduced in [MariaDB 10.5.2](#).

The events_transactions_history_long table contains the most recent completed transaction events that have ended globally, across all threads.

The number of records stored in the table is determined by the [performance_schema_events_transactions_history_long_size](#) system variable, which is autosized on startup.

If adding a completed transaction would cause the table to exceed this limit, the oldest row, regardless of thread, is discarded.

The table contains the following columns:

Column	Type	Description
THREAD_ID	bigint(20) unsigned	The thread associated with the event.
EVENT_ID	bigint(20) unsigned	The event id associated with the event.
END_EVENT_ID	bigint(20) unsigned	This column is set to NULL when the event starts and updated to the thread current event number when the event ends.
EVENT_NAME	varchar(128)	The name of the instrument from which the event was collected. This is a NAME value from the setup_instruments table.
STATE	enum('ACTIVE', 'COMMITTED', 'ROLLED BACK')	The current transaction state. The value is ACTIVE (after START TRANSACTION or BEGIN), COMMITTED (after COMMIT), or ROLLED BACK (after ROLLBACK).
TRX_ID	bigint(20) unsigned	Unused.
GTID	varchar(64)	Transaction GTID , using the format DOMAIN-SERVER_ID-SEQUENCE_NO.
XID_FORMAT_ID	int(11)	XA transaction format ID for GTRID and BQUAL values.
XID_GTRID	varchar(130)	XA global transaction ID.
XID_BQUAL	varchar(130)	XA transaction branch qualifier.
XA_STATE	varchar(64)	The state of the XA transaction. The value is ACTIVE (after XA START), IDLE (after XA END), PREPARED (after XA PREPARE), ROLLED BACK (after XA ROLLBACK), or COMMITTED (after XA COMMIT).
SOURCE	varchar(64)	The name of the source file containing the instrumented code that produced the event and the line number in the file at which the instrumentation occurs.
TIMER_START	bigint(20) unsigned	The unit is picoseconds. When event timing started. NULL if event has no timing information.
TIMER_END	bigint(20) unsigned	The unit is picoseconds. When event timing ended. NULL if event has no timing information.
TIMER_WAIT	bigint(20) unsigned	The unit is picoseconds. Event duration. NULL if event has not timing information.
ACCESS_MODE	enum('READ ONLY', 'READ WRITE')	Transaction access mode.
ISOLATION_LEVEL	varchar(64)	Transaction isolation level. One of: REPEATABLE READ, READ COMMITTED, READ UNCOMMITTED, or SERIALIZABLE.
AUTOCOMMIT	enum('YES', 'NO')	NO

NUMBER_OF_SAVEPOINTS	bigint(20) unsigned	The number of SAVEPOINT statements issued during the transaction.
NUMBER_OF_ROLLBACK_TO_SAVEPOINT	bigint(20) unsigned	The number of ROLLBACK_TO_SAVEPOINT statements issued during the transaction.
NUMBER_OF_RELEASE_SAVEPOINT	bigint(20) unsigned	The number of RELEASE_SAVEPOINT statements issued during the transaction.
OBJECT_INSTANCE_BEGIN	bigint(20) unsigned	Unused.
NESTING_EVENT_ID	bigint(20) unsigned	The EVENT_ID value of the event within which this event is nested.
NESTING_EVENT_TYPE	enum('TRANSACTION','STATEMENT','STAGE','WAIT')	The nesting event type.

1.1.2.9.2.1.24 Performance Schema events_transactions_summary_by_account_by_event_name Table

MariaDB starting with [10.5.2](#)

The events_transactions_summary_by_account_by_event_name table was introduced in [MariaDB 10.5.2](#).

The `events_transactions_summary_by_account_by_event_name` table contains information on transaction events aggregated by account and event name.

The table contains the following columns:

Column	Type	Description
USER	char(32)	User for which summary is generated.
HOST	char(60)	Host for which summary is generated.
EVENT_NAME	varchar(128)	Event name for which summary is generated.
COUNT_STAR	bigint(20) unsigned	The number of summarized events. This value includes all events, whether timed or nontimed.
SUM_TIMER_WAIT	bigint(20) unsigned	The total wait time of the summarized timed events. This value is calculated only for timed events because nontimed events have a wait time of NULL. The same is true for the other <code>xxx_TIMER_WAIT</code> values.
MIN_TIMER_WAIT	bigint(20) unsigned	The minimum wait time of the summarized timed events.
AVG_TIMER_WAIT	bigint(20) unsigned	The average wait time of the summarized timed events.
MAX_TIMER_WAIT	bigint(20) unsigned	The maximum wait time of the summarized timed events.
COUNT_READ_WRITE	bigint(20) unsigned	The total number of only READ/WRITE transaction events.
SUM_TIMER_READ_WRITE	bigint(20) unsigned	The total wait time of only READ/WRITE transaction events.
MIN_TIMER_READ_WRITE	bigint(20) unsigned	The minimum wait time of only READ/WRITE transaction events.
AVG_TIMER_READ_WRITE	bigint(20) unsigned	The average wait time of only READ/WRITE transaction events.
MAX_TIMER_READ_WRITE	bigint(20) unsigned	The maximum wait time of only READ/WRITE transaction events.
COUNT_READ_ONLY	bigint(20) unsigned	The total number of only READ ONLY transaction events.

SUM_TIMER_READ_ONLY	bigint(20) unsigned	The total wait time of only READ ONLY transaction events.
MIN_TIMER_READ_ONLY	bigint(20) unsigned	The minimum wait time of only READ ONLY transaction events.
AVG_TIMER_READ_ONLY	bigint(20) unsigned	The average wait time of only READ ONLY transaction events.
MAX_TIMER_READ_ONLY	bigint(20) unsigned	The maximum wait time of only READ ONLY transaction events.

1.1.2.9.2.1.25 Performance Schema events_transactions_summary_by_host_by_event_name Table

MariaDB starting with [10.5.2](#)

The `events_transactions_summary_by_host_by_event_name` table was introduced in [MariaDB 10.5.2](#).

The `events_transactions_summary_by_host_by_event_name` table contains information on transaction events aggregated by host and event name.

The table contains the following columns:

Column	Type	Description
HOST	char(60)	Host for which summary is generated.
EVENT_NAME	varchar(128)	Event name for which summary is generated.
COUNT_STAR	bigint(20) unsigned	The number of summarized events. This value includes all events, whether timed or nontimed.
SUM_TIMER_WAIT	bigint(20) unsigned	The total wait time of the summarized timed events. This value is calculated only for timed events because nontimed events have a wait time of NULL. The same is true for the other <code>xxx_TIMER_WAIT</code> values.
MIN_TIMER_WAIT	bigint(20) unsigned	The minimum wait time of the summarized timed events.
AVG_TIMER_WAIT	bigint(20) unsigned	The average wait time of the summarized timed events.
MAX_TIMER_WAIT	bigint(20) unsigned	The maximum wait time of the summarized timed events.
COUNT_READ_WRITE	bigint(20) unsigned	The total number of only READ/WRITE transaction events.
SUM_TIMER_READ_WRITE	bigint(20) unsigned	The total wait time of only READ/WRITE transaction events.
MIN_TIMER_READ_WRITE	bigint(20) unsigned	The minimum wait time of only READ/WRITE transaction events.
AVG_TIMER_READ_WRITE	bigint(20) unsigned	The average wait time of only READ/WRITE transaction events.
MAX_TIMER_READ_WRITE	bigint(20) unsigned	The maximum wait time of only READ/WRITE transaction events.
COUNT_READ_ONLY	bigint(20) unsigned	The total number of only READ ONLY transaction events.
SUM_TIMER_READ_ONLY	bigint(20) unsigned	The total wait time of only READ ONLY transaction events.
MIN_TIMER_READ_ONLY	bigint(20) unsigned	The minimum wait time of only READ ONLY transaction events.

AVG_TIMER_READ_ONLY	bigint(20) unsigned	The average wait time of only READ ONLY transaction events.
MAX_TIMER_READ_ONLY	bigint(20) unsigned	The maximum wait time of only READ ONLY transaction events.

1.1.2.9.2.1.26 Performance Schema events_transactions_summary_by_thread_by_event_name Table

MariaDB starting with [10.5.2](#)

The events_transactions_summary_by_thread_by_event_name table was introduced in [MariaDB 10.5.2](#).

The events_transactions_summary_by_thread_by_event_name table contains information on transaction events aggregated by thread and event name.

The table contains the following columns:

Field	Type	Null	Key	Default	Extra
THREAD_ID	bigint(20) unsigned	NO		NULL	
EVENT_NAME	varchar(128)	NO		NULL	
COUNT_STAR	bigint(20) unsigned	NO		NULL	
SUM_TIMER_WAIT	bigint(20) unsigned	NO		NULL	
MIN_TIMER_WAIT	bigint(20) unsigned	NO		NULL	
AVG_TIMER_WAIT	bigint(20) unsigned	NO		NULL	
MAX_TIMER_WAIT	bigint(20) unsigned	NO		NULL	
COUNT_READ_WRITE	bigint(20) unsigned	NO		NULL	
SUM_TIMER_READ_WRITE	bigint(20) unsigned	NO		NULL	
MIN_TIMER_READ_WRITE	bigint(20) unsigned	NO		NULL	
AVG_TIMER_READ_WRITE	bigint(20) unsigned	NO		NULL	
MAX_TIMER_READ_WRITE	bigint(20) unsigned	NO		NULL	
COUNT_READ_ONLY	bigint(20) unsigned	NO		NULL	
SUM_TIMER_READ_ONLY	bigint(20) unsigned	NO		NULL	
MIN_TIMER_READ_ONLY	bigint(20) unsigned	NO		NULL	
AVG_TIMER_READ_ONLY	bigint(20) unsigned	NO		NULL	
MAX_TIMER_READ_ONLY	bigint(20) unsigned	NO		NULL	

1.1.2.9.2.1.27 Performance Schema events_transactions_summary_by_user_by_event_name Table

MariaDB starting with [10.5.2](#)

The events_transactions_summary_by_user_by_event_name table was introduced in [MariaDB 10.5.2](#).

The events_transactions_summary_by_user_by_event_name table contains information on transaction events aggregated by user and event name.

The table contains the following columns:

Field	Type	Null	Key	Default	Extra
USER	char(32)	YES		NULL	
EVENT_NAME	varchar(128)	NO		NULL	
COUNT_STAR	bigint(20) unsigned	NO		NULL	
SUM_TIMER_WAIT	bigint(20) unsigned	NO		NULL	
MIN_TIMER_WAIT	bigint(20) unsigned	NO		NULL	
AVG_TIMER_WAIT	bigint(20) unsigned	NO		NULL	
MAX_TIMER_WAIT	bigint(20) unsigned	NO		NULL	
COUNT_READ_WRITE	bigint(20) unsigned	NO		NULL	
SUM_TIMER_READ_WRITE	bigint(20) unsigned	NO		NULL	
MIN_TIMER_READ_WRITE	bigint(20) unsigned	NO		NULL	
AVG_TIMER_READ_WRITE	bigint(20) unsigned	NO		NULL	
MAX_TIMER_READ_WRITE	bigint(20) unsigned	NO		NULL	
COUNT_READ_ONLY	bigint(20) unsigned	NO		NULL	
SUM_TIMER_READ_ONLY	bigint(20) unsigned	NO		NULL	
MIN_TIMER_READ_ONLY	bigint(20) unsigned	NO		NULL	
AVG_TIMER_READ_ONLY	bigint(20) unsigned	NO		NULL	
MAX_TIMER_READ_ONLY	bigint(20) unsigned	NO		NULL	

1.1.2.9.2.1.28 Performance Schema events_transactions_summary_global_by_event_name Table

MariaDB starting with [10.5.2](#)

The `events_transactions_summary_global_by_event_name` table was introduced in [MariaDB 10.5.2](#).

The `events_transactions_summary_global_by_event_name` table contains information on transaction events aggregated by event name.

The table contains the following columns:

Field	Type	Null	Key	Default	Extra
EVENT_NAME	varchar(128)	NO		NULL	
COUNT_STAR	bigint(20) unsigned	NO		NULL	
SUM_TIMER_WAIT	bigint(20) unsigned	NO		NULL	
MIN_TIMER_WAIT	bigint(20) unsigned	NO		NULL	
AVG_TIMER_WAIT	bigint(20) unsigned	NO		NULL	
MAX_TIMER_WAIT	bigint(20) unsigned	NO		NULL	
COUNT_READ_WRITE	bigint(20) unsigned	NO		NULL	
SUM_TIMER_READ_WRITE	bigint(20) unsigned	NO		NULL	
MIN_TIMER_READ_WRITE	bigint(20) unsigned	NO		NULL	
AVG_TIMER_READ_WRITE	bigint(20) unsigned	NO		NULL	
MAX_TIMER_READ_WRITE	bigint(20) unsigned	NO		NULL	
COUNT_READ_ONLY	bigint(20) unsigned	NO		NULL	
SUM_TIMER_READ_ONLY	bigint(20) unsigned	NO		NULL	
MIN_TIMER_READ_ONLY	bigint(20) unsigned	NO		NULL	
AVG_TIMER_READ_ONLY	bigint(20) unsigned	NO		NULL	
MAX_TIMER_READ_ONLY	bigint(20) unsigned	NO		NULL	

1.1.2.9.2.1.29 Performance Schema events_waits_current Table

The `events_waits_current` table contains the status of a thread's most recently monitored wait event, listing one event per thread.

The table contains the following columns:

Column	Description
--------	-------------

THREAD_ID	Thread associated with the event. Together with EVENT_ID uniquely identifies the row.
EVENT_ID	Thread's current event number at the start of the event. Together with THREAD_ID uniquely identifies the row.
END_EVENT_ID	NULL when the event starts, set to the thread's current event number at the end of the event.
EVENT_NAME	Event instrument name and a NAME from the setup_instruments table
SOURCE	Name and line number of the source file containing the instrumented code that produced the event.
TIMER_START	Value in picoseconds when the event timing started or NULL if timing is not collected.
TIMER_END	Value in picoseconds when the event timing ended, or NULL if the event has not ended or timing is not collected.
TIMER_WAIT	Value in picoseconds of the event's duration or NULL if the event has not ended or timing is not collected.
SPINS	Number of spin rounds for a mutex, or NULL if spin rounds are not used, or spinning is not instrumented.
OBJECT_SCHEMA	Name of the schema that contains the table for table I/O objects, otherwise NULL for file I/O and synchronization objects.
OBJECT_NAME	File name for file I/O objects, table name for table I/O objects, the socket's IP:PORT value for a socket object or NULL for a synchronization object.
INDEX_NAME	Name of the index, PRIMARY for the primary key, or NULL for no index used.
OBJECT_TYPE	FILE for a file object, TABLE or TEMPORARY TABLE for a table object, or NULL for a synchronization object.
OBJECT_INSTANCE_BEGIN	Address in memory of the object.
NESTING_EVENT_ID	EVENT_ID of event within which this event nests.
NESTING_EVENT_TYPE	Nesting event type. Either statement , stage or wait .
OPERATION	Operation type, for example read, write or lock
NUMBER_OF_BYTES	Number of bytes that the operation read or wrote, or NULL for table I/O waits.
FLAGS	Reserved for use in the future.

It is possible to empty this table with a TRUNCATE TABLE statement.

The related tables, [events_waits_history](#) and [events_waits_history_long](#) derive their values from the current events.

1.1.2.9.2.1.30 Performance Schema events_waits_history Table

The `events_waits_history` table by default contains the ten most recent completed wait events per thread. This number can be adjusted by setting the `performance_schema_events_waits_history_size` system variable when the server starts up.

The table structure is identical to the `events_waits_current` table structure, and contains the following columns:

Column	Description
THREAD_ID	Thread associated with the event. Together with EVENT_ID uniquely identifies the row.
EVENT_ID	Thread's current event number at the start of the event. Together with THREAD_ID uniquely identifies the row.
END_EVENT_ID	NULL when the event starts, set to the thread's current event number at the end of the event.
EVENT_NAME	Event instrument name and a NAME from the setup_instruments table
SOURCE	Name and line number of the source file containing the instrumented code that produced the event.
TIMER_START	Value in picoseconds when the event timing started or NULL if timing is not collected.
TIMER_END	Value in picoseconds when the event timing ended, or NULL if timing is not collected.
TIMER_WAIT	Value in picoseconds of the event's duration or NULL if timing is not collected.
SPINS	Number of spin rounds for a mutex, or NULL if spin rounds are not used, or spinning is not instrumented.
OBJECT_SCHEMA	Name of the schema that contains the table for table I/O objects, otherwise NULL for file I/O and synchronization objects.
OBJECT_NAME	File name for file I/O objects, table name for table I/O objects, the socket's IP:PORT value for a socket object or NULL for a synchronization object.

INDEX_NAME	Name of the index, PRIMARY for the primary key, or NULL for no index used.
OBJECT_TYPE	FILE for a file object, TABLE or TEMPORARY TABLE for a table object, or NULL for a synchronization object.
OBJECT_INSTANCE_BEGIN	Address in memory of the object.
NESTING_EVENT_ID	EVENT_ID of event within which this event nests.
NESTING_EVENT_TYPE	Nesting event type. Either statement , stage or wait .
OPERATION	Operation type, for example read, write or lock
NUMBER_OF_BYTES	Number of bytes that the operation read or wrote, or NULL for table I/O waits.
FLAGS	Reserved for use in the future.

It is possible to empty this table with a TRUNCATE TABLE statement.

[events_waits_current](#) and [events_waits_history_long](#) are related tables.

1.1.2.9.2.1.31 Performance Schema events_waits_summary_by_account_by_event_name Table

The [Performance Schema](#) events_waits_summary_by_account_by_event_name table contains wait events summarized by account and event name. It contains the following columns:

Column	Description
USER	User. Used together with HOST and EVENT_NAME for grouping events.
HOST	Host. Used together with USER and EVENT_NAME for grouping events.
EVENT_NAME	Event name. Used together with USER and HOST for grouping events.
COUNT_STAR	Number of summarized events
SUM_TIMER_WAIT	Total wait time of the summarized events that are timed.
MIN_TIMER_WAIT	Minimum wait time of the summarized events that are timed.
AVG_TIMER_WAIT	Average wait time of the summarized events that are timed.
MAX_TIMER_WAIT	Maximum wait time of the summarized events that are timed.

The *_TIMER_WAIT columns only calculate results for timed events, as non-timed events have a NULL wait time.

Example

```

SELECT * FROM events_waits_summary_by_account_by_event_name\G
...
*****915. row *****
USER: NULL
HOST: NULL
EVENT_NAME: wait/io/socket/sql/server_tcpip_socket
COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
*****916. row *****
USER: NULL
HOST: NULL
EVENT_NAME: wait/io/socket/sql/server_unix_socket
COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
*****917. row *****
USER: NULL
HOST: NULL
EVENT_NAME: wait/io/socket/sql/client_connection
COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
*****918. row *****
USER: NULL
HOST: NULL
EVENT_NAME: idle
COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0

```

1.1.2.9.2.1.32 Performance Schema events_waits_summary_by_host_by_event_name Table

The [Performance Schema](#) `events_waits_summary_by_host_by_event_name` table contains wait events summarized by host and event name. It contains the following columns:

Column	Description
HOST	Host. Used together with <code>EVENT_NAME</code> for grouping events.
EVENT_NAME	Event name. Used together with <code>USER</code> and <code>HOST</code> for grouping events.
COUNT_STAR	Number of summarized events
SUM_TIMER_WAIT	Total wait time of the summarized events that are timed.
MIN_TIMER_WAIT	Minimum wait time of the summarized events that are timed.
AVG_TIMER_WAIT	Average wait time of the summarized events that are timed.
MAX_TIMER_WAIT	Maximum wait time of the summarized events that are timed.

The `*_TIMER_WAIT` columns only calculate results for timed events, as non-timed events have a `NULL` wait time.

Example

```

SELECT * FROM events_waits_summary_by_host_by_event_name\G
...
***** 610. row *****
HOST: NULL
EVENT_NAME: wait/io/socket/sql/server_unix_socket
COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
***** 611. row *****
HOST: NULL
EVENT_NAME: wait/io/socket/sql/client_connection
COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
***** 612. row *****
HOST: NULL
EVENT_NAME: idle
COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0

```

1.1.2.9.2.1.33 Performance Schema events_waits_summary_by_instance Table

The [Performance Schema](#) `events_waits_summary_by_instance` table contains wait events summarized by instance. It contains the following columns:

Column	Description
EVENT_NAME	Event name. Used together with <code>OBJECT_INSTANCE_BEGIN</code> for grouping events.
OBJECT_INSTANCE_BEGIN	If an instrument creates multiple instances, each instance has a unique <code>OBJECT_INSTANCE_BEGIN</code> value to allow for grouping by instance.
COUNT_STAR	Number of summarized events
SUM_TIMER_WAIT	Total wait time of the summarized events that are timed.
MIN_TIMER_WAIT	Minimum wait time of the summarized events that are timed.
AVG_TIMER_WAIT	Average wait time of the summarized events that are timed.
MAX_TIMER_WAIT	Maximum wait time of the summarized events that are timed.

The `*_TIMER_WAIT` columns only calculate results for timed events, as non-timed events have a `NULL` wait time.

Example

```

SELECT * FROM events_waits_summary_by_instance\G
...
***** 202. row *****
  EVENT_NAME: wait/io/file/sql/binlog
OBJECT_INSTANCE_BEGIN: 140578961969856
  COUNT_STAR: 6
  SUM_TIMER_WAIT: 90478331960
  MIN_TIMER_WAIT: 263344
  AVG_TIMER_WAIT: 15079721848
  MAX_TIMER_WAIT: 67760576376
***** 203. row *****
  EVENT_NAME: wait/io/file/sql/dbopt
OBJECT_INSTANCE_BEGIN: 140578961970560
  COUNT_STAR: 6
  SUM_TIMER_WAIT: 39891428472
  MIN_TIMER_WAIT: 387168
  AVG_TIMER_WAIT: 6648571412
  MAX_TIMER_WAIT: 24503293304
***** 204. row *****
  EVENT_NAME: wait/io/file/sql/dbopt
OBJECT_INSTANCE_BEGIN: 140578961971264
  COUNT_STAR: 6
  SUM_TIMER_WAIT: 39902495024
  MIN_TIMER_WAIT: 177888
  AVG_TIMER_WAIT: 6650415692
  MAX_TIMER_WAIT: 21026400404

```

1.1.2.9.2.1.34 Performance Schema events_waits_summary_by_thread_by_event_name Table

The [Performance Schema](#) `events_waits_summary_by_thread_by_event_name` table contains wait events summarized by thread and event name. It contains the following columns:

Column	Description
THREAD_ID	Thread associated with the event. Together with <code>EVENT_NAME</code> uniquely identifies the row.
EVENT_NAME	Event name. Used together with <code>THREAD_ID</code> for grouping events.
COUNT_STAR	Number of summarized events
SUM_TIMER_WAIT	Total wait time of the summarized events that are timed.
MIN_TIMER_WAIT	Minimum wait time of the summarized events that are timed.
AVG_TIMER_WAIT	Average wait time of the summarized events that are timed.
MAX_TIMER_WAIT	Maximum wait time of the summarized events that are timed.

The `*_TIMER_WAIT` columns only calculate results for timed events, as non-timed events have a `NULL` wait time.

Example

```

SELECT * FROM events_waits_summary_by_thread_by_event_name\G
...
*****6424. row *****
  THREAD_ID: 64
  EVENT_NAME: wait/io/socket/sql/server_unix_socket
  COUNT_STAR: 0
  SUM_TIMER_WAIT: 0
  MIN_TIMER_WAIT: 0
  AVG_TIMER_WAIT: 0
  MAX_TIMER_WAIT: 0
*****6425. row *****
  THREAD_ID: 64
  EVENT_NAME: wait/io/socket/sql/client_connection
  COUNT_STAR: 0
  SUM_TIMER_WAIT: 0
  MIN_TIMER_WAIT: 0
  AVG_TIMER_WAIT: 0
  MAX_TIMER_WAIT: 0
*****6426. row *****
  THREAD_ID: 64
  EVENT_NAME: idle
  COUNT_STAR: 73
  SUM_TIMER_WAIT: 2200525216200000
  MIN_TIMER_WAIT: 300000
  AVG_TIMER_WAIT: 30144181000000
  MAX_TIMER_WAIT: 491241757300000

```

1.1.2.9.2.1.35 Performance Schema events_waits_summary_by_user_by_event_name Table

The [Performance Schema](#) `events_waits_summary_by_user_by_event_name` table contains wait events summarized by user and event name. It contains the following columns:

Column	Description
USER	User. Used together with <code>EVENT_NAME</code> for grouping events.
EVENT_NAME	Event name. Used together with <code>USER</code> for grouping events.
COUNT_STAR	Number of summarized events
SUM_TIMER_WAIT	Total wait time of the summarized events that are timed.
MIN_TIMER_WAIT	Minimum wait time of the summarized events that are timed.
AVG_TIMER_WAIT	Average wait time of the summarized events that are timed.
MAX_TIMER_WAIT	Maximum wait time of the summarized events that are timed.

The `*_TIMER_WAIT` columns only calculate results for timed events, as non-timed events have a `NULL` wait time.

Example

```

SELECT * FROM events_waits_summary_by_user_by_event_name\G
...
***** 916. row *****
USER: NULL
EVENT_NAME: wait/io/socket/sql/server_unix_socket
COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
***** 917. row *****
USER: NULL
EVENT_NAME: wait/io/socket/sql/client_connection
COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
***** 918. row *****
USER: NULL
EVENT_NAME: idle
COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0

```

1.1.2.9.2.1.36 Performance Schema events_waits_summary_global_by_event_name Table

The [Performance Schema](#) `events_waits_summary_global_by_event_name` table contains wait events summarized by event name. It contains the following columns:

Column	Description
EVENT_NAME	Event name.
COUNT_STAR	Number of summarized events
SUM_TIMER_WAIT	Total wait time of the summarized events that are timed.
MIN_TIMER_WAIT	Minimum wait time of the summarized events that are timed.
AVG_TIMER_WAIT	Average wait time of the summarized events that are timed.
MAX_TIMER_WAIT	Maximum wait time of the summarized events that are timed.

The `*_TIMER_WAIT` columns only calculate results for timed events, as non-timed events have a `NULL` wait time.

Example

```

SELECT * FROM events_waits_summary_global_by_event_name\G
...
***** 303. row *****
EVENT_NAME: wait/io/socket/sql/server_tcpip_socket
COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
***** 304. row *****
EVENT_NAME: wait/io/socket/sql/server_unix_socket
COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
***** 305. row *****
EVENT_NAME: wait/io/socket/sql/client_connection
COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
***** 306. row *****
EVENT_NAME: idle
COUNT_STAR: 265
SUM_TIMER_WAIT: 46861125181000000
MIN_TIMER_WAIT: 1000000
AVG_TIMER_WAIT: 176834434000000
MAX_TIMER_WAIT: 4912417573000000

```

1.1.2.9.2.1.37 Performance Schema file_instances Table Description

The `file_instances` table lists instances of instruments seen by the Performance Schema when executing file I/O instrumentation, and the associated files. Only files that have been opened, and that have not been deleted, will be listed in the table.

The `performance_schema_max_file_instances` system variable specifies the maximum number of instrumented file objects.

Column	Description
FILE_NAME	File name.
EVENT_NAME	Instrument name associated with the file.
OPEN_COUNT	Open handles on the file. A value of greater than zero means that the file is currently open.

Example

```

SELECT * FROM performance_schema.file_instances WHERE OPEN_COUNT>0;
+-----+-----+-----+
| FILE_NAME          | EVENT_NAME          | OPEN_COUNT |
+-----+-----+-----+
| /var/log/mysql/mariadb-bin.index | wait/io/file/sql/binlog_index | 1          |
| /var/lib/mysql/ibdata1    | wait/io/file/innodb/innodb_data_file | 2          |
| /var/lib/mysql/ib_logfile0   | wait/io/file/innodb/innodb_log_file  | 2          |
| /var/lib/mysql/ib_logfile1   | wait/io/file/innodb/innodb_log_file  | 2          |
| /var/lib/mysql/mysql/gtid_slave_pos.ibd | wait/io/file/innodb/innodb_data_file | 3          |
| /var/lib/mysql/mysql/innodb_index_stats.ibd | wait/io/file/innodb/innodb_data_file | 3          |
| /var/lib/mysql/mysql/innodb_table_stats.ibd | wait/io/file/innodb/innodb_data_file | 3          |
...

```

1.1.2.9.2.1.38 Performance Schema

file_summary_by_event_name Table

The [Performance Schema](#) `file_summary_by_event_name` table contains file events summarized by event name. As of [MariaDB 10.0](#), it contains the following columns:

Column	Description
EVENT_NAME	Event name.
COUNT_STAR	Number of summarized events
SUM_TIMER_WAIT	Total wait time of the summarized events that are timed.
MIN_TIMER_WAIT	Minimum wait time of the summarized events that are timed.
AVG_TIMER_WAIT	Average wait time of the summarized events that are timed.
MAX_TIMER_WAIT	Maximum wait time of the summarized events that are timed.
COUNT_READ	Number of all read operations, including <code>FGETS</code> , <code>FGETC</code> , <code>FREAD</code> , and <code>READ</code> .
SUM_TIMER_READ	Total wait time of all read operations that are timed.
MIN_TIMER_READ	Minimum wait time of all read operations that are timed.
AVG_TIMER_READ	Average wait time of all read operations that are timed.
MAX_TIMER_READ	Maximum wait time of all read operations that are timed.
SUM_NUMBER_OF_BYTES_READ	Bytes read by read operations.
COUNT_WRITE	Number of all write operations, including <code>FPUTS</code> , <code>FPUTC</code> , <code>FPRINTF</code> , <code>VPRINTF</code> , <code>FWRITE</code> , and <code>PWRITE</code> .
SUM_TIMER_WRITE	Total wait time of all write operations that are timed.
MIN_TIMER_WRITE	Minimum wait time of all write operations that are timed.
AVG_TIMER_WRITE	Average wait time of all write operations that are timed.
MAX_TIMER_WRITE	Maximum wait time of all write operations that are timed.
SUM_NUMBER_OF_BYTES_WRITE	Bytes written by write operations.
COUNT_MISC	Number of all miscellaneous operations not counted above, including <code>CREATE</code> , <code>DELETE</code> , <code>OPEN</code> , <code>CLOSE</code> , <code>STREAM_OPEN</code> , <code>STREAM_CLOSE</code> , <code>SEEK</code> , <code>TELL</code> , <code>FLUSH</code> , <code>STAT</code> , <code>FSTAT</code> , <code>CHSIZE</code> , <code>RENAME</code> , and <code>SYNC</code> .
SUM_TIMER_MISC	Total wait time of all miscellaneous operations that are timed.
MIN_TIMER_MISC	Minimum wait time of all miscellaneous operations that are timed.
AVG_TIMER_MISC	Average wait time of all miscellaneous operations that are timed.
MAX_TIMER_MISC	Maximum wait time of all miscellaneous operations that are timed.

Before MariaDB 10, the table contained only the `EVENT_NAME` , `COUNT_READ` , `COUNT_WRITE` , `SUM_NUMBER_OF_BYTES_READ` and `SUM_NUMBER_OF_BYTES_WRITE` columns.

I/O operations can be avoided by caching, in which case they will not be recorded in this table.

You can [TRUNCATE](#) the table, which will reset all counters to zero.

Example

```

SELECT * FROM file_summary_by_event_name\G
...
***** 49. row *****
EVENT_NAME: wait/io/file/aria/MAD
COUNT_STAR: 60
SUM_TIMER_WAIT: 397234368
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 6620224
MAX_TIMER_WAIT: 16808672
COUNT_READ: 0
SUM_TIMER_READ: 0
MIN_TIMER_READ: 0
AVG_TIMER_READ: 0
MAX_TIMER_READ: 0
SUM_NUMBER_OF_BYTES_READ: 0
COUNT_WRITE: 0
SUM_TIMER_WRITE: 0
MIN_TIMER_WRITE: 0
AVG_TIMER_WRITE: 0
MAX_TIMER_WRITE: 0
SUM_NUMBER_OF_BYTES_WRITE: 0
COUNT_MISC: 60
SUM_TIMER_MISC: 397234368
MIN_TIMER_MISC: 0
AVG_TIMER_MISC: 6620224
MAX_TIMER_MISC: 16808672
***** 50. row *****
EVENT_NAME: wait/io/file/aria/control
COUNT_STAR: 3
SUM_TIMER_WAIT: 24055778544
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 8018592848
MAX_TIMER_WAIT: 24027262400
COUNT_READ: 1
SUM_TIMER_READ: 24027262400
MIN_TIMER_READ: 0
AVG_TIMER_READ: 24027262400
MAX_TIMER_READ: 24027262400
SUM_NUMBER_OF_BYTES_READ: 52
COUNT_WRITE: 0
SUM_TIMER_WRITE: 0
MIN_TIMER_WRITE: 0
AVG_TIMER_WRITE: 0
MAX_TIMER_WRITE: 0
SUM_NUMBER_OF_BYTES_WRITE: 0
COUNT_MISC: 2
SUM_TIMER_MISC: 28516144
MIN_TIMER_MISC: 0
AVG_TIMER_MISC: 14258072
MAX_TIMER_MISC: 27262208

```

1.1.2.9.2.1.39 Performance Schema file_summary_by_instance Table

The `Performance Schema` `file_summary_by_instance` table contains file events summarized by instance. As of [MariaDB 10.0](#), it contains the following columns:

Column	Description
<code>FILE_NAME</code>	File name.
<code>EVENT_NAME</code>	Event name.
<code>OBJECT_INSTANCE_BEGIN</code>	Address in memory. Together with <code>FILE_NAME</code> and <code>EVENT_NAME</code> uniquely identifies a row.
<code>COUNT_STAR</code>	Number of summarized events
<code>SUM_TIMER_WAIT</code>	Total wait time of the summarized events that are timed.

MIN_TIMER_WAIT	Minimum wait time of the summarized events that are timed.
AVG_TIMER_WAIT	Average wait time of the summarized events that are timed.
MAX_TIMER_WAIT	Maximum wait time of the summarized events that are timed.
COUNT_READ	Number of all read operations, including <code>FGETS</code> , <code>FGETC</code> , <code>FREAD</code> , and <code>READ</code> .
SUM_TIMER_READ	Total wait time of all read operations that are timed.
MIN_TIMER_READ	Minimum wait time of all read operations that are timed.
AVG_TIMER_READ	Average wait time of all read operations that are timed.
MAX_TIMER_READ	Maximum wait time of all read operations that are timed.
SUM_NUMBER_OF_BYTES_READ	Bytes read by read operations.
COUNT_WRITE	Number of all write operations, including <code>FPUTS</code> , <code>FPUTC</code> , <code>FPRINTF</code> , <code>VPRINTF</code> , <code>FWRITE</code> , and <code>PWRITE</code> .
SUM_TIMER_WRITE	Total wait time of all write operations that are timed.
MIN_TIMER_WRITE	Minimum wait time of all write operations that are timed.
AVG_TIMER_WRITE	Average wait time of all write operations that are timed.
MAX_TIMER_WRITE	Maximum wait time of all write operations that are timed.
SUM_NUMBER_OF_BYTES_WRITE	Bytes written by write operations.
COUNT_MISC	Number of all miscellaneous operations not counted above, including <code>CREATE</code> , <code>DELETE</code> , <code>OPEN</code> , <code>CLOSE</code> , <code>STREAM_OPEN</code> , <code>STREAM_CLOSE</code> , <code>SEEK</code> , <code>TELL</code> , <code>FLUSH</code> , <code>STAT</code> , <code>FSTAT</code> , <code>CHSIZE</code> , <code>RENAME</code> , and <code>SYNC</code> .
SUM_TIMER_MISC	Total wait time of all miscellaneous operations that are timed.
MIN_TIMER_MISC	Minimum wait time of all miscellaneous operations that are timed.
AVG_TIMER_MISC	Average wait time of all miscellaneous operations that are timed.
MAX_TIMER_MISC	Maximum wait time of all miscellaneous operations that are timed.

Before MariaDB 10, the table contained only the `FILE_NAME` , `EVENT_NAME` , `COUNT_READ` , `COUNT_WRITE` , `SUM_NUMBER_OF_BYTES_READ` and `SUM_NUMBER_OF_BYTES_WRITE` columns.

I/O operations can be avoided by caching, in which case they will not be recorded in this table.

You can [TRUNCATE](#) the table, which will reset all counters to zero.

Example

```

SELECT * FROM file_summary_by_instance\G
...
***** row *****
FILE_NAME: /var/lib/mysql/test/db.opt
EVENT_NAME: wait/io/file/sql/dbopt
OBJECT_INSTANCE_BEGIN: 140578961971264
COUNT_STAR: 6
SUM_TIMER_WAIT: 39902495024
MIN_TIMER_WAIT: 177888
AVG_TIMER_WAIT: 6650415692
MAX_TIMER_WAIT: 21026400404
COUNT_READ: 1
SUM_TIMER_READ: 21026400404
MIN_TIMER_READ: 21026400404
AVG_TIMER_READ: 21026400404
MAX_TIMER_READ: 21026400404
SUM_NUMBER_OF_BYTES_READ: 65
COUNT_WRITE: 0
SUM_TIMER_WRITE: 0
MIN_TIMER_WRITE: 0
AVG_TIMER_WRITE: 0
MAX_TIMER_WRITE: 0
SUM_NUMBER_OF_BYTES_WRITE: 0
COUNT_MISC: 5
SUM_TIMER_MISC: 18876094620
MIN_TIMER_MISC: 177888
AVG_TIMER_MISC: 3775218924
MAX_TIMER_MISC: 18864558060
***** row *****
FILE_NAME: /var/log/mysql/mariadb-bin.000157
EVENT_NAME: wait/io/file/sql/binlog
OBJECT_INSTANCE_BEGIN: 140578961971968
COUNT_STAR: 6
SUM_TIMER_WAIT: 73985877680
MIN_TIMER_WAIT: 251136
AVG_TIMER_WAIT: 12330979468
MAX_TIMER_WAIT: 73846656340
COUNT_READ: 0
SUM_TIMER_READ: 0
MIN_TIMER_READ: 0
AVG_TIMER_READ: 0
MAX_TIMER_READ: 0
SUM_NUMBER_OF_BYTES_READ: 0
COUNT_WRITE: 2
SUM_TIMER_WRITE: 62583004
MIN_TIMER_WRITE: 27630192
AVG_TIMER_WRITE: 31291284
MAX_TIMER_WRITE: 34952812
SUM_NUMBER_OF_BYTES_WRITE: 369
COUNT_MISC: 4
SUM_TIMER_MISC: 73923294676
MIN_TIMER_MISC: 251136
AVG_TIMER_MISC: 18480823560
MAX_TIMER_MISC: 73846656340

```

1.1.2.9.2.1.40 Performance Schema global_status Table

MariaDB starting with [10.5.2](#)

The `global_status` table was added in [MariaDB 10.5.2](#).

The `global_status` table contains a list of status variables and their global values. The table only stores status variable statistics for threads which are instrumented, and does not collect statistics for `Com_xxx` variables.

The table contains the following columns:

Column	Description
--------	-------------

VARIABLE_NAME	The global status variable name.
VARIABLE_VALUE	The global status variable value.

[TRUNCATE TABLE](#) resets global status variables, including thread, account, host, and user status, but not those that are never reset by the server.

1.1.2.9.2.1.41 Performance Schema hosts Table

Description

The `hosts` table contains a row for each host used by clients to connect to the server, containing current and total connections.

The size is determined by the [performance_schema_hosts_size](#) system variable, which, if set to zero, will disable connection statistics in the `hosts` table.

It contains the following columns:

Column	Description
HOST	Host name used by the client to connect, <code>NULL</code> for internal threads or user sessions that failed to authenticate.
CURRENT_CONNECTIONS	Current number of the host's connections.
TOTAL_CONNECTIONS	Total number of the host's connections

Example

```
SELECT * FROM hosts;
+-----+-----+-----+
| HOST      | CURRENT_CONNECTIONS | TOTAL_CONNECTIONS |
+-----+-----+-----+
| localhost |              1 |          45 |
| NULL      |             20 |          23 |
+-----+-----+-----+
```

1.1.2.9.2.1.42 Performance Schema memory_summary_by_account_by_event_name Table

MariaDB starting with 10.5.2

The `memory_summary_by_account_by_event_name` table was introduced in [MariaDB 10.5.2](#).

There are five memory summary tables in the Performance Schema that share a number of fields in common. These include:

- `memory_summary_by_account_by_event_name`
- [memory_summary_by_host_by_event_name](#)
- [memory_summary_by_thread_by_event_name](#)
- [memory_summary_by_user_by_event_name](#)
- [memory_global_by_event_name](#)

The `memory_summary_by_account_by_event_name` table contains memory usage statistics aggregated by account and event.

The table contains the following columns:

Field	Type	Null	Default	Description
USER	char(32)	YES	NULL	User portion of the account.
HOST	char(60)	YES	NULL	Host portion of the account.
EVENT_NAME	varchar(128)	NO	NULL	Event name.
COUNT_ALLOC	bigint(20) unsigned	NO	NULL	Total number of allocations to memory.

COUNT_FREE	bigint(20) unsigned	NO	NULL	Total number of attempts to free the allocated memory.
SUM_NUMBER_OF_BYTES_ALLOC	bigint(20) unsigned	NO	NULL	Total number of bytes allocated.
SUM_NUMBER_OF_BYTES_FREE	bigint(20) unsigned	NO	NULL	Total number of bytes freed
LOW_COUNT_USED	bigint(20)	NO	NULL	Lowest number of allocated blocks (lowest value of CURRENT_COUNT_USED).
CURRENT_COUNT_USED	bigint(20)	NO	NULL	Currently allocated blocks that have not been freed (COUNT_ALLOC minus COUNT_FREE).
HIGH_COUNT_USED	bigint(20)	NO	NULL	Highest number of allocated blocks (highest value of CURRENT_COUNT_USED).
LOW_NUMBER_OF_BYTES_USED	bigint(20)	NO	NULL	Lowest number of bytes used.
CURRENT_NUMBER_OF_BYTES_USED	bigint(20)	NO	NULL	Current number of bytes used (total allocated minus total freed).
HIGH_NUMBER_OF_BYTES_USED	bigint(20)	NO	NULL	Highest number of bytes used.

1.1.2.9.2.1.43 Performance Schema memory_summary_by_host_by_event_name Table

MariaDB starting with [10.5.2](#)

The `memory_summary_by_host_by_event_name` table was introduced in [MariaDB 10.5.2](#).

There are five memory summary tables in the Performance Schema that share a number of fields in common. These include:

- [memory_summary_by_account_by_event_name](#)
- [memory_summary_by_host_by_event_name](#)
- [memory_summary_by_thread_by_event_name](#)
- [memory_summary_by_user_by_event_name](#)
- [memory_global_by_event_name](#)

The `memory_summary_by_host_by_event_name` table contains memory usage statistics aggregated by host and event.

The table contains the following columns:

Field	Type	Null	Default	Description
HOST	char(60)	YES	NULL	Host portion of the account.
EVENT_NAME	varchar(128)	NO	NULL	Event name.
COUNT_ALLOC	bigint(20) unsigned	NO	NULL	Total number of allocations to memory.
COUNT_FREE	bigint(20) unsigned	NO	NULL	Total number of attempts to free the allocated memory.
SUM_NUMBER_OF_BYTES_ALLOC	bigint(20) unsigned	NO	NULL	Total number of bytes allocated.
SUM_NUMBER_OF_BYTES_FREE	bigint(20) unsigned	NO	NULL	Total number of bytes freed
LOW_COUNT_USED	bigint(20)	NO	NULL	Lowest number of allocated blocks (lowest value of CURRENT_COUNT_USED).
CURRENT_COUNT_USED	bigint(20)	NO	NULL	Currently allocated blocks that have not been freed (COUNT_ALLOC minus COUNT_FREE).
HIGH_COUNT_USED	bigint(20)	NO	NULL	Highest number of allocated blocks (highest value of CURRENT_COUNT_USED).
LOW_NUMBER_OF_BYTES_USED	bigint(20)	NO	NULL	Lowest number of bytes used.

CURRENT_NUMBEROFBYTESUSED	bigint(20)	NO	NULL	Current number of bytes used (total allocated minus total freed).
HIGH_NUMBER_OF_BYTES_USED	bigint(20)	NO	NULL	Highest number of bytes used.

1.1.2.9.2.1.44 Performance Schema memory_summary_by_thread_by_event_name Table

MariaDB starting with 10.5.2

The `memory_summary_by_thread_by_event_name` table was introduced in MariaDB 10.5.2.

There are five memory summary tables in the Performance Schema that share a number of fields in common. These include:

- [memory_summary_by_account_by_event_name](#)
- [memory_summary_by_host_by_event_name](#)
- [memory_summary_by_thread_by_event_name](#)
- [memory_summary_by_user_by_event_name](#)
- [memory_global_by_event_name](#)

The `memory_summary_by_thread_by_event_name` table contains memory usage statistics aggregated by thread and event.

The table contains the following columns:

Field	Type	Null	Default	Description
THREAD_ID	bigint(20) unsigned	NO	NULL	Thread id.
EVENT_NAME	varchar(128)	NO	NULL	Event name.
COUNT_ALLOC	bigint(20) unsigned	NO	NULL	Total number of allocations to memory.
COUNT_FREE	bigint(20) unsigned	NO	NULL	Total number of attempts to free the allocated memory.
SUM_NUMBER_OF_BYTES_ALLOC	bigint(20) unsigned	NO	NULL	Total number of bytes allocated.
SUM_NUMBER_OF_BYTES_FREE	bigint(20) unsigned	NO	NULL	Total number of bytes freed
LOW_COUNT_USED	bigint(20)	NO	NULL	Lowest number of allocated blocks (lowest value of CURRENT_COUNT_USED).
CURRENT_COUNT_USED	bigint(20)	NO	NULL	Currently allocated blocks that have not been freed (COUNT_ALLOC minus COUNT_FREE).
HIGH_COUNT_USED	bigint(20)	NO	NULL	Highest number of allocated blocks (highest value of CURRENT_COUNT_USED).
LOW_NUMBER_OF_BYTES_USED	bigint(20)	NO	NULL	Lowest number of bytes used.
CURRENT_NUMBER_OF_BYTES_USED	bigint(20)	NO	NULL	Current number of bytes used (total allocated minus total freed).
HIGH_NUMBER_OF_BYTES_USED	bigint(20)	NO	NULL	Highest number of bytes used.

1.1.2.9.2.1.45 Performance Schema memory_summary_by_user_by_event_name Table

MariaDB starting with 10.5.2

The `memory_summary_by_user_by_event_name` table was introduced in MariaDB 10.5.2.

There are five memory summary tables in the Performance Schema that share a number of fields in common. These include:

- [memory_summary_by_account_by_event_name](#)
- [memory_summary_by_host_by_event_name](#)
- [memory_summary_by_thread_by_event_name](#)
- [memory_summary_by_user_by_event_name](#)
- [memory_global_by_event_name](#)

The `memory_summary_by_user_by_event_name` table contains memory usage statistics aggregated by user and event.

The table contains the following columns:

Field	Type	Null	Default	Description
USER	char(32)	YES	NULL	User portion of the account.
EVENT_NAME	varchar(128)	NO	NULL	Event name.
COUNT_ALLOC	bigint(20) unsigned	NO	NULL	Total number of allocations to memory.
COUNT_FREE	bigint(20) unsigned	NO	NULL	Total number of attempts to free the allocated memory.
SUM_NUMBER_OF_BYTES_ALLOC	bigint(20) unsigned	NO	NULL	Total number of bytes allocated.
SUM_NUMBER_OF_BYTES_FREE	bigint(20) unsigned	NO	NULL	Total number of bytes freed
LOW_COUNT_USED	bigint(20)	NO	NULL	Lowest number of allocated blocks (lowest value of CURRENT_COUNT_USED).
CURRENT_COUNT_USED	bigint(20)	NO	NULL	Currently allocated blocks that have not been freed (COUNT_ALLOC minus COUNT_FREE).
HIGH_COUNT_USED	bigint(20)	NO	NULL	Highest number of allocated blocks (highest value of CURRENT_COUNT_USED).
LOW_NUMBER_OF_BYTES_USED	bigint(20)	NO	NULL	Lowest number of bytes used.
CURRENT_NUMBER_OF_BYTES_USED	bigint(20)	NO	NULL	Current number of bytes used (total allocated minus total freed).
HIGH_NUMBER_OF_BYTES_USED	bigint(20)	NO	NULL	Highest number of bytes used.

1.1.2.9.2.1.46 Performance Schema `memory_summary_global_by_event_name` Table

MariaDB starting with [10.5.2](#)

The `memory_summary_global_by_event_name` table was introduced in [MariaDB 10.5.2](#).

There are five memory summary tables in the Performance Schema that share a number of fields in common. These include:

- [memory_summary_by_account_by_event_name](#)
- [memory_summary_by_host_by_event_name](#)
- [memory_summary_by_thread_by_event_name](#)
- [memory_summary_by_user_by_event_name](#)
- [memory_global_by_event_name](#)

The `memory_summary_global_by_event_name` table contains memory usage statistics aggregated by event and event.

The table contains the following columns:

Field	Type	Null	Default	Description
EVENT_NAME	varchar(128)	NO	NULL	Event name.
COUNT_ALLOC	bigint(20) unsigned	NO	NULL	Total number of allocations to memory.
COUNT_FREE	bigint(20) unsigned	NO	NULL	Total number of attempts to free the allocated memory.

SUM_NUMBER_OF_BYTES_ALLOC	bigint(20) unsigned	NO	NULL	Total number of bytes allocated.
SUM_NUMBER_OF_BYTES_FREE	bigint(20) unsigned	NO	NULL	Total number of bytes freed
LOW_COUNT_USED	bigint(20)	NO	NULL	Lowest number of allocated blocks (lowest value of CURRENT_COUNT_USED).
CURRENT_COUNT_USED	bigint(20)	NO	NULL	Currently allocated blocks that have not been freed (COUNT_ALLOC minus COUNT_FREE).
HIGH_COUNT_USED	bigint(20)	NO	NULL	Highest number of allocated blocks (highest value of CURRENT_COUNT_USED).
LOW_NUMBER_OF_BYTES_USED	bigint(20)	NO	NULL	Lowest number of bytes used.
CURRENT_NUMBER_OF_BYTES_USED	bigint(20)	NO	NULL	Current number of bytes used (total allocated minus total freed).
HIGH_NUMBER_OF_BYTES_USED	bigint(20)	NO	NULL	Highest number of bytes used.

Example

Seeing what memory was most often allocated for:

```
SELECT * FROM memory_summary_global_by_event_name
  ORDER BY count_alloc DESC LIMIT 1\G
***** 1. row *****
  EVENT_NAME: memory/sql/QUICK_RANGE_SELECT::alloc
  COUNT_ALLOC: 147976
  COUNT_FREE: 147976
  SUM_NUMBER_OF_BYTES_ALLOC: 600190656
  SUM_NUMBER_OF_BYTES_FREE: 600190656
  LOW_COUNT_USED: 0
  CURRENT_COUNT_USED: 0
  HIGH_COUNT_USED: 68
  LOW_NUMBER_OF_BYTES_USED: 0
  CURRENT_NUMBER_OF_BYTES_USED: 0
  HIGH_NUMBER_OF_BYTES_USED: 275808
```

1.1.2.9.2.1.47 Performance Schema metadata_locks Table

MariaDB starting with [10.5.2](#)

The metadata_locks table was introduced in [MariaDB 10.5.2](#).

The `metadata_locks` table contains [metadata lock](#) information.

To enable metadata lock instrumentation, at runtime:

```
UPDATE performance_schema.setup_instruments SET enabled='YES', timed='YES'
  WHERE name LIKE 'wait/lock/metadata%';
```

or in the [configuration file](#):

```
performance-schema-instrument='wait/lock/metadata/sql/mdl=ON'
```

The table is by default autosized, but the size can be configured with the `performance_schema_max_metadata_locks` system variable.

The table is read-only, and `TRUNCATE TABLE` cannot be used to empty the table.

The table contains the following columns:

Field	Type	Null	Default	Description
-------	------	------	---------	-------------

OBJECT_TYPE	varchar(64)	NO	NULL	Object type. One of BACKUP , COMMIT , EVENT , FUNCTION , GLOBAL , LOCKING SERVICE , PROCEDURE , SCHEMA , TABLE , TABLESPACE , TRIGGER (unused) or USER LEVEL LOCK .
OBJECT_SCHEMA	varchar(64)	YES	NULL	Object schema.
OBJECT_NAME	varchar(64)	YES	NULL	Object name.
OBJECT_INSTANCE_BEGIN	bigint(20) unsigned	NO	NULL	Address in memory of the instrumented object.
LOCK_TYPE	varchar(32)	NO	NULL	Lock type. One of BACKUP_FTWRL1 , BACKUP_START , BACKUP_TRANS_DML , EXCLUSIVE , INTENTION_EXCLUSIVE , SHARED , SHARED_HIGH_PRIO , SHARED_NO_READ_WRITE , SHARED_NO_WRITE , SHARED_READ , SHARED_UPGRADABLE or SHARED_WRITE .
LOCK_DURATION	varchar(32)	NO	NULL	Lock duration. One of EXPLICIT (locks released by explicit action, for example a global lock acquired with FLUSH TABLES WITH READ LOCK) , STATEMENT (locks implicitly released at statement end) or TRANSACTION (locks implicitly released at transaction end).
LOCK_STATUS	varchar(32)	NO	NULL	Lock status. One of GRANTED , KILLED , PENDING , POST_RELEASE_NOTIFY , PRE_ACQUIRE_NOTIFY , TIMEOUT or VICTIM .
SOURCE	varchar(64)	YES	NULL	Source file containing the instrumented code that produced the event, as well as the line number where the instrumentation occurred. This allows one to examine the source code involved.
OWNER_THREAD_ID	bigint(20) unsigned	YES	NULL	Thread that requested the lock.
OWNER_EVENT_ID	bigint(20) unsigned	YES	NULL	Event that requested the lock.

1.1.2.9.2.1.48 Performance Schema mutex_instances Table

Description

The `mutex_instances` table lists all mutexes that the Performance Schema seeing while the server is executing.

A mutex is a code mechanism for ensuring that threads can only access resources one at a time. A second thread attempting to access a resource will find it protected by a mutex, and will wait for it to be unlocked.

The `performance_schema_max_mutex_instances` system variable specifies the maximum number of instrumented mutex instances.

Column	Description
NAME	Instrument name associated with the mutex.
OBJECT_INSTANCE_BEGIN	Memory address of the instrumented mutex.
LOCKED_BY_THREAD_ID	The <code>THREAD_ID</code> of the locking thread if a thread has a mutex locked, otherwise <code>NULL</code> .

1.1.2.9.2.1.49 Performance Schema objects_summary_global_by_type Table

It aggregates object wait events, and contains the following columns:

Column	Description
OBJECT_TYPE	Groups records together with <code>OBJECT_SCHEMA</code> and <code>OBJECT_NAME</code> .
OBJECT_SCHEMA	Groups records together with <code>OBJECT_TYPE</code> and <code>OBJECT_NAME</code> .
OBJECT_NAME	Groups records together with <code>OBJECT_SCHEMA</code> and <code>OBJECT_TYPE</code> .
COUNT_STAR	Number of summarized events
SUM_TIMER_WAIT	Total wait time of the summarized events that are timed.

MIN_TIMER_WAIT	Minimum wait time of the summarized events that are timed.
AVG_TIMER_WAIT	Average wait time of the summarized events that are timed.
MAX_TIMER_WAIT	Maximum wait time of the summarized events that are timed.

You can [TRUNCATE](#) the table, which will reset all counters to zero.

Example

```
SELECT * FROM objects_summary_global_by_type\G
...
***** 101. row *****
OBJECT_TYPE: TABLE
OBJECT_SCHEMA: test
OBJECT_NAME: v
COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
***** 102. row *****
OBJECT_TYPE: TABLE
OBJECT_SCHEMA: test
OBJECT_NAME: xx2
COUNT_STAR: 2
SUM_TIMER_WAIT: 1621920
MIN_TIMER_WAIT: 481344
AVG_TIMER_WAIT: 810960
MAX_TIMER_WAIT: 1140576
```

1.1.2.9.2.1.50 Performance Schema performance_timers Table

Description

The `performance_timers` table lists available event timers.

It contains the following columns:

Column	Description
TIMER_NAME	Time name, used in the setup_timers table.
TIMER_FREQUENCY	Number of timer units per second. Dependent on the processor speed.
TIMER_RESOLUTION	Number of timer units by which timed values increase each time.
TIMER_OVERHEAD	Minimum timer overhead, determined during initialization by calling the timer 20 times and selecting the smallest value. Total overhead will be at least double this, as the timer is called at the beginning and end of each timed event.

Any `NULL` values indicate that that particular timer is not available on your platform. Any timer names with a non-`NULL` value can be used in the [setup_timers](#) table.

Example

```
SELECT * FROM performance_timers;
```

TIMER_NAME	TIMER_FREQUENCY	TIMER_RESOLUTION	TIMER_OVERHEAD
CYCLE	2293651741	1	28
NANOSECOND	1000000000	1	48
MICROSECOND	1000000	1	52
MILLISECOND	1000	1000	9223372036854775807
TICK	106	1	496

1.1.2.9.2.1.51 Performance Schema prepared_statements_instances Table

MariaDB starting with [10.5.2](#)

The prepared_statements_instances table was introduced in [MariaDB 10.5.2](#).

The `prepared_statements_instances` table contains aggregated statistics of prepared statements.

The maximum number of rows in the table is determined by the `performance_schema_max_prepared_statement_instances` system variable, which is by default autosized on startup.

The table contains the following columns:

Field	Type	Null	Key	Default	Extra
OBJECT_INSTANCE_BEGIN	bigint(20) unsigned	NO		NULL	
STATEMENT_ID	bigint(20) unsigned	NO		NULL	
STATEMENT_NAME	varchar(64)	YES		NULL	
SQL_TEXT	longtext	NO		NULL	
OWNER_THREAD_ID	bigint(20) unsigned	NO		NULL	
OWNER_EVENT_ID	bigint(20) unsigned	NO		NULL	
OWNER_OBJECT_TYPE	enum('EVENT','FUNCTION','PROCEDURE','TABLE','TRIGGER')	YES		NULL	
OWNER_OBJECT_SCHEMA	varchar(64)	YES		NULL	
OWNER_OBJECT_NAME	varchar(64)	YES		NULL	
TIMER_PREPARE	bigint(20) unsigned	NO		NULL	
COUNT_REPREPARE	bigint(20) unsigned	NO		NULL	
COUNT_EXECUTE	bigint(20) unsigned	NO		NULL	
SUM_TIMER_EXECUTE	bigint(20) unsigned	NO		NULL	
MIN_TIMER_EXECUTE	bigint(20) unsigned	NO		NULL	
AVG_TIMER_EXECUTE	bigint(20) unsigned	NO		NULL	
MAX_TIMER_EXECUTE	bigint(20) unsigned	NO		NULL	
SUM_LOCK_TIME	bigint(20) unsigned	NO		NULL	
SUM_ERRORS	bigint(20) unsigned	NO		NULL	
SUM_WARNINGS	bigint(20) unsigned	NO		NULL	
SUM_ROWS_AFFECTED	bigint(20) unsigned	NO		NULL	
SUM_ROWS_SENT	bigint(20) unsigned	NO		NULL	
SUM_ROWS_EXAMINED	bigint(20) unsigned	NO		NULL	
SUM_CREATED_TMP_DISK_TABLES	bigint(20) unsigned	NO		NULL	
SUM_CREATED_TMP_TABLES	bigint(20) unsigned	NO		NULL	
SUM_SELECT_FULL_JOIN	bigint(20) unsigned	NO		NULL	
SUM_SELECT_FULL_RANGE_JOIN	bigint(20) unsigned	NO		NULL	
SUM_SELECT_RANGE	bigint(20) unsigned	NO		NULL	
SUM_SELECT_RANGE_CHECK	bigint(20) unsigned	NO		NULL	
SUM_SELECT_SCAN	bigint(20) unsigned	NO		NULL	
SUM_SORT_MERGE_PASSES	bigint(20) unsigned	NO		NULL	
SUM_SORT_RANGE	bigint(20) unsigned	NO		NULL	
SUM_SORT_ROWS	bigint(20) unsigned	NO		NULL	
SUM_SORT_SCAN	bigint(20) unsigned	NO		NULL	
SUM_NO_INDEX_USED	bigint(20) unsigned	NO		NULL	
SUM_NO_GOOD_INDEX_USED	bigint(20) unsigned	NO		NULL	

1.1.2.9.2.1.52 Performance Schema replication_applier_configuration Table

MariaDB starting with 10.5.2

The `replication_applier_configuration` table, along with many other new [Performance Schema tables](#), was added in [MariaDB 10.5.2](#).

The [Performance Schema](#) `replication_applier_configuration` table contains configuration settings affecting replica transactions.

It contains the following fields.

Field	Type	Null	Description
CHANNEL_NAME	char(64)	NO	Replication channel name.
DESIRED_DELAY	int(11)	NO	Target number of seconds the replica should be delayed to the master.

1.1.2.9.2.1.53 Performance Schema replication_applier_status Table

MariaDB starting with 10.5.2

The `replication_applier_status` table, along with many other new [Performance Schema tables](#), was added in [MariaDB 10.5.2](#).

The [Performance Schema](#) `replication_applier_status` table contains information about the general transaction execution status on the replica.

It contains the following fields.

Field	Type	Null	Description
CHANNEL_NAME	char(64)	NO	The replication channel name.
SERVICE_STATE	enum('ON','OFF')	NO	Shows ON when the replication channel's applier threads are active or idle, OFF means that the applier threads are not active.
REMAINING_DELAY	int(10) unsigned	YES	Seconds the replica needs to wait to reach the desired delay from master.
COUNT_TRANSACTIONS_RETRIES	bigint(20) unsigned	NO	The number of retries that were made because the replication SQL thread failed to apply a transaction.

1.1.2.9.2.1.54 Performance Schema replication_applier_status_by_coordinator Table

MariaDB starting with 10.5.2

The `replication_applier_status_by_coordinator` table was added in [MariaDB 10.5.2](#).

The [Performance Schema](#) `replication_applier_status_by_coordinator` table displays the status of the coordinator thread used in multi-threaded replicas to manage multiple worker threads.

It contains the following fields.

Column	Type	Null	Description
CHANNEL_NAME	varchar(256)	NO	Replication channel name.
THREAD_ID	bigint(20) unsigned	YES	The SQL/coordinator thread ID.
SERVICE_STATE	enum('ON','OFF')	NO	ON (thread exists and is active or idle) or OFF (thread no longer exists).
LAST_ERROR_NUMBER	int(11)	NO	Last error number that caused the SQL/coordinator thread to stop.
LAST_ERROR_MESSAGE	varchar(1024)	NO	Last error message that caused the SQL/coordinator thread to stop.
LAST_ERROR_TIMESTAMP	timestamp	NO	Timestamp that shows when the most recent SQL/coordinator error occurred.

LAST_SEEN_TRANSACTION	char(57)	NO	The transaction the worker has last seen.
LAST_TRANS_RETRY_COUNT	int(11)	NO	Total number of retries attempted by last transaction.

1.1.2.9.2.1.55 Performance Schema replication_applier_status_by_worker Table

MariaDB starting with 10.6.0

The `replication_applier_status_by_worker` table was added in MariaDB 10.6.0.

The [Performance Schema](#) `replication_applier_status_by_worker` table displays replica worker thread specific information.

It contains the following fields.

Column	Description
CHANNEL_NAME	Name of replication channel through which the transaction is received.
THREAD_ID	Thread_Id as displayed in the <code>performance_schema.threads</code> table for thread with name 'thread/sql/rpl_parallel_thread'. THREAD_ID will be NULL when worker threads are stopped due to error/force stop.
SERVICE_STATE	Whether or not the thread is running.
LAST_SEEN_TRANSACTION	Last GTID executed by worker
LAST_ERROR_NUMBER	Last Error that occurred on a particular worker.
LAST_ERROR_MESSAGE	Last error specific message.
LAST_ERROR_TIMESTAMP	Time stamp of last error.
WORKER_IDLE_TIME	Total idle time in seconds that the worker thread has spent waiting for work from SQL thread.
LAST_TRANS_RETRY_COUNT	Total number of retries attempted by last transaction.

1.1.2.9.2.1.56 Performance Schema replication_connection_configuration Table

MariaDB starting with 10.5.2

The `replication_connection_configuration` table was added in MariaDB 10.6.0.

The [Performance Schema](#) `replication_connection_configuration` table displays replica's configuration settings used for connecting to the primary.

It contains the following fields.

Column	Type	Null	Description
CHANNEL_NAME	varchar(256)	NO	The replication channel used.
HOST	char(60)	NO	The host name of the source that the replica is connected to.
PORT	int(11)	NO	The port used to connect to the source.
USER	char(32)	NO	The user name of the replication user account used to connect to the source.
USING_GTID	enum('NO', 'CURRENT_POS', 'SLAVE_POS')	NO	Whether replication is using GTIDs or not.
SSL_ALLOWED	enum('YES', 'NO', 'IGNORED')	NO	Whether SSL is allowed for the replica connection.
SSL_CA_FILE	varchar(512)	NO	Path to the file that contains one or more certificates for trusted Certificate Authorities (CA) to use for TLS.

SSL_CA_PATH	varchar(512)	NO	Path to a directory that contains one or more PEM files that contain X509 certificates for a trusted Certificate Authority (CA) to use for TLS.
SSL_CERTIFICATE	varchar(512)	NO	Path to the certificate used to authenticate the master.
SSL_CIPHER	varchar(512)	NO	Which cipher is used for encryption.
SSL_KEY	varchar(512)	NO	Path to the private key used for TLS.
SSL_VERIFY_SERVER_CERTIFICATE	enum('YES','NO')	NO	Whether the server certificate is verified as part of the SSL connection.
SSL_CRL_FILE	varchar(255)	NO	Path to the PEM file containing one or more revoked X.509 certificates.
SSL_CRL_PATH	varchar(255)	NO	PATH to a folder containing PEM files containing one or more revoked X.509 certificates.
CONNECTION_RETRY_INTERVAL	int(11)	NO	The number of seconds between connect retries.
CONNECTION_RETRY_COUNT	bigint(20) unsigned	NO	The number of times the replica can attempt to reconnect to the source in the event of a lost connection.
HEARTBEAT_INTERVAL	double(10,3) unsigned	NO	Number of seconds after which a heartbeat will be sent.
IGNORE_SERVER_IDS	longtext	NO	Binary log events from servers (ids) to ignore.
REPL_DO_DOMAIN_IDS	longtext	NO	Only apply binary logs from these domain ids.
REPL_IGNORE_DOMAIN_IDS	longtext	NO	Binary log events from domains to ignore.

1.1.2.9.2.1.57 Performance Schema rwlock_instances Table

The `rwlock_instances` table lists all read write lock (rwlock) instances that the Performance Schema sees while the server is executing. A read write is a mechanism for ensuring threads can either share access to common resources, or have exclusive access.

The `performance_schema_max_rwlock_instances` system variable specifies the maximum number of instrumented rwlock objects.

The `rwlock_instances` table contains the following columns:

Column	Description
NAME	Instrument name associated with the read write lock
OBJECT_INSTANCE_BEGIN	Address in memory of the instrumented lock
WRITE_LOCKED_BY_THREAD_ID	THREAD_ID of the locking thread if locked in write (exclusive) mode, otherwise NULL .
READ_LOCKED_BY_COUNT	Count of current read locks held

1.1.2.9.2.1.58 Performance Schema session_account_connect_attrs Table

Description

The `session_account_connect_attrs` table shows connection attributes for the current session.

Applications can pass key/value connection attributes to the server when a connection is made. The `session_connect_attrs` and `session_account_connect_attrs` tables provide access to this information, for all sessions and the current session respectively.

The C API functions `mysql_options()` and `mysql_optionsv()` are used for passing connection attributes to the server.

`session_account_connect_attrs` contains the following columns:

Column	Description
PROCESSLIST_ID	Session connection identifier.
ATTR_NAME	Attribute name.
ATTR_VALUE	Attribute value.

ORDINAL_POSITION	Order in which attribute was added to the connection attributes.
------------------	--

Example

```
SELECT * FROM performance_schema.session_account_connect_attrs;
+-----+-----+-----+
| PROCESSLIST_ID | ATTR_NAME      | ATTR_VALUE      | ORDINAL_POSITION |
+-----+-----+-----+
|       45 | _os           | debian-linux-gnu |          0 |
|       45 | _client_name   | libmysql        |          1 |
|       45 | _pid           | 7711            |          2 |
|       45 | _client_version | 10.0.5          |          3 |
|       45 | _platform      | x86_64          |          4 |
|       45 | program_name   | mysql            |          5 |
+-----+-----+-----+
```

1.1.2.9.2.1.59 Performance Schema session_connect_attrs Table

Description

The `session_connect_attrs` table shows connection attributes for all sessions.

Applications can pass key/value connection attributes to the server when a connection is made. The `session_connect_attrs` and `session_account_connect_attrs` tables provide access to this information, for all sessions and the current session respectively.

The C API functions `mysql_options()` and `mysql_optionsv()` are used for passing connection attributes to the server.

`session_connect_attrs` contains the following columns:

Column	Description
PROCESSLIST_ID	Session connection identifier.
ATTR_NAME	Attribute name.
ATTR_VALUE	Attribute value.
ORDINAL_POSITION	Order in which attribute was added to the connection attributes.

Example

Returning the current connection's attributes:

```
SELECT * FROM performance_schema.session_connect_attrs WHERE processlist_id=CONNECTION_ID();
+-----+-----+-----+
| PROCESSLIST_ID | ATTR_NAME      | ATTR_VALUE      | ORDINAL_POSITION |
+-----+-----+-----+
|       45 | _os           | debian-linux-gnu |          0 |
|       45 | _client_name   | libmysql        |          1 |
|       45 | _pid           | 7711            |          2 |
|       45 | _client_version | 10.0.5          |          3 |
|       45 | _platform      | x86_64          |          4 |
|       45 | program_name   | mysql            |          5 |
+-----+-----+-----+
```

1.1.2.9.2.1.60 Performance Schema session_status Table

MariaDB starting with 10.5.2

The `session_status` table was added in MariaDB 10.5.2.

The `session_status` table contains a list of status variables for the current session. The table only stores status variable statistics for threads which are instrumented, and does not collect statistics for `Com_xxx` variables.

The table contains the following columns:

Column	Description
VARIABLE_NAME	The session status variable name.
VARIABLE_VALUE	The session status variable value.

It is not possible to empty this table with a `TRUNCATE TABLE` statement.

1.1.2.9.2.1.61 Performance Schema setup_actors Table

The `setup_actors` table contains information for determining whether monitoring should be enabled for new client connection threads.

The default size is 100 rows, which can be changed by modifying the `performance_schema_setup_actors_size` system variable at server startup.

If a row in the table matches a new foreground thread's client and host, the matching `INSTRUMENTED` column in the `threads` table is set to either `YES` or `NO`, which allows selective application of instrumentation by host, by user, or combination thereof.

Column	Description
HOST	Host name, either a literal, or the <code>%</code> wildcard representing any host.
USER	User name, either a literal or the <code>%</code> wildcard representing any name.
ROLE	Unused

Initially, any user and host is matched:

```
SELECT * FROM performance_schema.setup_actors;
+-----+-----+
| HOST | USER | ROLE |
+-----+-----+
| %   | %   | %   |
+-----+-----+
```

1.1.2.9.2.1.62 Performance Schema setup_consumers Table

Lists the types of consumers for which event information is available.

The `setup_consumers` table contains the following columns:

Column	Description
NAME	Consumer name
ENABLED	YES or NO for whether or not the consumer is enabled. You can modify this column to ensure that event information is added, or is not added.

The table can be modified directly, or the server started with the option enabled, for example:

```
performance-schema-consumer-events-waits-history=ON
```

Example

```
SELECT * FROM performance_schema.setup_consumers;
```

NAME	ENABLED
events_stages_current	NO
events_stages_history	NO
events_stages_history_long	NO
events_statements_current	YES
events_statements_history	NO
events_statements_history_long	NO
events_waits_current	NO
events_waits_history	NO
events_waits_history_long	NO
global_instrumentation	YES
thread_instrumentation	YES
statements_digest	YES

1.1.2.9.2.1.63 Performance Schema setup_instruments Table

The `setup_instruments` table contains a list of instrumented object classes for which it is possible to collect events. There is one row for each instrument in the source code. When an instrument is enabled and executed, instances are created which are then stored in the `cond_instances`, `file_instances`, `mutex_instances`, `rwlock_instances` or `socket_instance` tables.

It contains the following columns:

Column	Description
NAME	Instrument name
ENABLED	Whether or not the instrument is enabled. It can be disabled, and the instrument will produce no events.
TIMED	Whether or not the instrument is timed. It can be set, but if disabled, events produced by the instrument will have <code>NULL</code> values for the corresponding <code>TIMER_START</code> , <code>TIMER_END</code> , and <code>TIMER_WAIT</code> values.

Example

From MariaDB 10.5.7, default settings with the Performance Schema enabled:

```
SELECT * FROM setup_instruments ORDER BY name;
```

NAME	ENABLED	TIMED
idle	YES	YES
memory/csv/blobroot	NO	NO
memory/csv/row	NO	NO
memory/csv/tina_set	NO	NO
memory/csv/TINA_SHARE	NO	NO
memory/csv/Transparent_file	NO	NO
memory/innodb/adaptive hash index	NO	NO
memory/innodb/btr0btr	NO	NO
memory/innodb/btr0buf	NO	NO
memory/innodb/btr0bulk	NO	NO
memory/innodb/btr0cur	NO	NO
memory/innodb/btr0pcur	NO	NO
memory/innodb/btr0sea	NO	NO
memory/innodb/buf0buf	NO	NO
memory/innodb/buf0dblwr	NO	NO
memory/innodb/buf0dump	NO	NO
memory/innodb/buf_buf_pool	NO	NO
memory/innodb/dict0dict	NO	NO
memory/innodb/dict0mem	NO	NO
memory/innodb/dict0stats	NO	NO
memory/innodb/dict_stats_bg_recalc_pool_t	NO	NO
memory/innodb/dict_stats_index_map_t	NO	NO

memory/innodb/dict_stats_n_diff_on_level	NO	NO
memory/innodb/eval0eval	NO	NO
memory/innodb/fil0crypt	NO	NO
memory/innodb/fil0fil	NO	NO
memory/innodb/fsp0file	NO	NO
memory/innodb/fts0ast	NO	NO
memory/innodb/fts0blex	NO	NO
memory/innodb/fts0config	NO	NO
memory/innodb/fts0file	NO	NO
memory/innodb/fts0fts	NO	NO
memory/innodb/fts0opt	NO	NO
memory/innodb/fts0pars	NO	NO
memory/innodb/fts0que	NO	NO
memory/innodb/fts0sql	NO	NO
memory/innodb/fts0tlex	NO	NO
memory/innodb/gis0sea	NO	NO
memory/innodb/handler0alter	NO	NO
memory/innodb/hash0hash	NO	NO
memory/innodb/ha_innodb	NO	NO
memory/innodb/i_s	NO	NO
memory/innodb/lexyy	NO	NO
memory/innodb/lock0lock	NO	NO
memory/innodb/mem0mem	NO	NO
memory/innodb/os0event	NO	NO
memory/innodb/os0file	NO	NO
memory/innodb/other	NO	NO
memory/innodb/pars0lex	NO	NO
memory/innodb/rem0rec	NO	NO
memory/innodb/row0ftsort	NO	NO
memory/innodb/row0import	NO	NO
memory/innodb/row0log	NO	NO
memory/innodb/row0merge	NO	NO
memory/innodb/row0mysql	NO	NO
memory/innodb/row0sel	NO	NO
memory/innodb/row_log_buf	NO	NO
memory/innodb/row_merge_sort	NO	NO
memory/innodb/srv0start	NO	NO
memory/innodb/std	NO	NO
memory/innodb/sync0arr	NO	NO
memory/innodb/sync0debug	NO	NO
memory/innodb/sync0rw	NO	NO
memory/innodb/sync0start	NO	NO
memory/innodb/sync0types	NO	NO
memory/innodb/trx0i_s	NO	NO
memory/innodb/trx0roll	NO	NO
memory/innodb/trx0rseg	NO	NO
memory/innodb/trx0seg	NO	NO
memory/innodb/trx0trx	NO	NO
memory/innodb/trx0undo	NO	NO
memory/innodb/ut0list	NO	NO
memory/innodb/ut0mem	NO	NO
memory/innodb/ut0new	NO	NO
memory/innodb/ut0pool	NO	NO
memory/innodb/ut0rbt	NO	NO
memory/innodb/ut0wqueue	NO	NO
memory/innodb/xtrabackup	NO	NO
memory/memory/HP_INFO	NO	NO
memory/memory/HP_KEYDEF	NO	NO
memory/memory/HP_PTRS	NO	NO
memory/memory/HP_SHARE	NO	NO
memory/myisam/filecopy	NO	NO
memory/myisam/FTB	NO	NO
memory/myisam/FTPARSER_PARAM	NO	NO
memory/myisam/FT_INFO	NO	NO
memory/myisam/ft_memroot	NO	NO
memory/myisam/ft_stopwords	NO	NO
memory/myisam/keycache_thread_var	NO	NO
memory/myisam/MI_DECODE_TREE	NO	NO
memory/myisam/MI_INFO	NO	NO
memory/myisam/MI_INFO::bulk_insert	NO	NO
memory/myisam/MI_INFO::ft1_to_ft2	NO	NO
memory/myisam/MI_SORT_PARAM	NO	NO

memory/myisam/MI_SORT_PARAM::wordroot	NO	NO
memory/myisam/MYISAM_SHARE	NO	NO
memory/myisam/MYISAM_SHARE::decode_tables	NO	NO
memory/myisam/preload_buffer	NO	NO
memory/myisam/record_buffer	NO	NO
memory/myisam/SORT_FT_BUF	NO	NO
memory/myisam/SORT_INFO::buffer	NO	NO
memory/myisam/SORT_KEY_BLOCKS	NO	NO
memory/myisam/stPageList::pages	NO	NO
memory/myisammrg/children	NO	NO
memory/myisammrg/MYRG_INFO	NO	NO
memory/partition/ha_partition::file	NO	NO
memory/partition/ha_partition::part_ids	NO	NO
memory/partition/Partition_admin	NO	NO
memory/partition/Partition_share	NO	NO
memory/partition/partition_sort_buffer	NO	NO
memory/performance_schema/accounts	YES	NO
memory/performance_schema/cond_class	YES	NO
memory/performance_schema/cond_instances	YES	NO
memory/performance_schema/events_stages_history	YES	NO
memory/performance_schema/events_stages_history_long	YES	NO
memory/performance_schema/events_stages_summary_by_account_by_event_name	YES	NO
memory/performance_schema/events_stages_summary_by_host_by_event_name	YES	NO
memory/performance_schema/events_stages_summary_by_thread_by_event_name	YES	NO
memory/performance_schema/events_stages_summary_by_user_by_event_name	YES	NO
memory/performance_schema/events_stages_summary_global_by_event_name	YES	NO
memory/performance_schema/events_statements_current	YES	NO
memory/performance_schema/events_statements_current.sqltext	YES	NO
memory/performance_schema/events_statements_current.tokens	YES	NO
memory/performance_schema/events_statements_history	YES	NO
memory/performance_schema/events_statements_history.sqltext	YES	NO
memory/performance_schema/events_statements_history.tokens	YES	NO
memory/performance_schema/events_statements_history_long	YES	NO
memory/performance_schema/events_statements_history_long.sqltext	YES	NO
memory/performance_schema/events_statements_history_long.tokens	YES	NO
memory/performance_schema/events_statements_summary_by_account_by_event_name	YES	NO
memory/performance_schema/events_statements_summary_by_digest	YES	NO
memory/performance_schema/events_statements_summary_by_digest.tokens	YES	NO
memory/performance_schema/events_statements_summary_by_host_by_event_name	YES	NO
memory/performance_schema/events_statements_summary_by_program	YES	NO
memory/performance_schema/events_statements_summary_by_thread_by_event_name	YES	NO
memory/performance_schema/events_statements_summary_by_user_by_event_name	YES	NO
memory/performance_schema/events_statements_summary_global_by_event_name	YES	NO
memory/performance_schema/events_transactions_history	YES	NO
memory/performance_schema/events_transactions_history_long	YES	NO
memory/performance_schema/events_transactions_summary_by_account_by_event_name	YES	NO
memory/performance_schema/events_transactions_summary_by_host_by_event_name	YES	NO
memory/performance_schema/events_transactions_summary_by_thread_by_event_name	YES	NO
memory/performance_schema/events_transactions_summary_by_user_by_event_name	YES	NO
memory/performance_schema/events_waits_history	YES	NO
memory/performance_schema/events_waits_history_long	YES	NO
memory/performance_schema/events_waits_summary_by_account_by_event_name	YES	NO
memory/performance_schema/events_waits_summary_by_host_by_event_name	YES	NO
memory/performance_schema/events_waits_summary_by_thread_by_event_name	YES	NO
memory/performance_schema/events_waits_summary_by_user_by_event_name	YES	NO
memory/performance_schema/file_class	YES	NO
memory/performance_schema/file_handle	YES	NO
memory/performance_schema/file_instances	YES	NO
memory/performance_schema/hosts	YES	NO
memory/performance_schema/memory_class	YES	NO
memory/performance_schema/memory_summary_by_account_by_event_name	YES	NO
memory/performance_schema/memory_summary_by_host_by_event_name	YES	NO
memory/performance_schema/memory_summary_by_thread_by_event_name	YES	NO
memory/performance_schema/memory_summary_by_user_by_event_name	YES	NO
memory/performance_schema/memory_summary_global_by_event_name	YES	NO
memory/performance_schema/metadata_locks	YES	NO
memory/performance_schema/mutex_class	YES	NO
memory/performance_schema/mutex_instances	YES	NO
memory/performance_schema/prepared_statements_instances	YES	NO
memory/performance_schema/rwlock_class	YES	NO
memory/performance_schema/rwlock_instances	YES	NO

memory/performance_schema/scalable_buffer	YES	NO
memory/performance_schema/session_connect_attrs	YES	NO
memory/performance_schema/setup_actors	YES	NO
memory/performance_schema/setup_objects	YES	NO
memory/performance_schema/socket_class	YES	NO
memory/performance_schema/socket_instances	YES	NO
memory/performance_schema/stage_class	YES	NO
memory/performance_schema/statement_class	YES	NO
memory/performance_schema/table_handles	YES	NO
memory/performance_schema/table_io_waits_summary_by_index_usage	YES	NO
memory/performance_schema/table_lock_waits_summary_by_table	YES	NO
memory/performance_schema/table_shares	YES	NO
memory/performance_schema/threads	YES	NO
memory/performance_schema/thread_class	YES	NO
memory/performance_schema/users	YES	NO
memory/sql/acl_cache	NO	NO
memory/sql/binlog_cache_mngr	NO	NO
memory/sql/binlog_pos	NO	NO
memory/sql/binlog_statement_buffer	NO	NO
memory/sql/binlog_ver_1_event	NO	NO
memory/sql/bison_stack	NO	NO
memory/sql/Blob_mem_storage::storage	NO	NO
memory/sql/DATE_TIME_FORMAT	NO	NO
memory/sql/dboptions_hash	NO	NO
memory/sql/DDL_LOG_MEMORY_ENTRY	NO	NO
memory/sql/display_table_locks	NO	NO
memory/sql/errmsgs	NO	NO
memory/sql/Event_basic::mem_root	NO	NO
memory/sql/Event_queue_element_for_exec::names	NO	NO
memory/sql/Event_scheduler::scheduler_param	NO	NO
memory/sql/Filesort_info::merge	NO	NO
memory/sql/Filesort_info::record_pointers	NO	NO
memory/sql/frm::string	NO	NO
memory/sql/gdl	NO	NO
memory/sql/Gis_read_stream::err_msg	NO	NO
memory/sql/global_system_variables	NO	NO
memory/sql/handler::errmsgs	NO	NO
memory/sql/handlerton	NO	NO
memory/sql/hash_index_key_buffer	NO	NO
memory/sql/host_cache::hostname	NO	NO
memory/sql/ignored_db	NO	NO
memory/sql/JOIN_CACHE	NO	NO
memory/sql/load_env_plugins	NO	NO
memory/sql/Locked_tables_list::m_locked_tables_root	NO	NO
memory/sql/MDL_context::acquire_locks	NO	NO
memory/sql/MPVIO_EXT::auth_info	NO	NO
memory/sql/MYSQL_BIN_LOG::basename	NO	NO
memory/sql/MYSQL_BIN_LOG::index	NO	NO
memory/sql/MYSQL_BIN_LOG::recover	NO	NO
memory/sql/MYSQL_LOCK	NO	NO
memory/sql/MYSQL_LOG::name	NO	NO
memory/sql/mysql_plugin	NO	NO
memory/sql/mysql_plugin_dl	NO	NO
memory/sql/MYSQL_RELAY_LOG::basename	NO	NO
memory/sql/MYSQL_RELAY_LOG::index	NO	NO
memory/sql/my_str_malloc	NO	NO
memory/sql/NAMED_ILINK::name	NO	NO
memory/sql/native_functions	NO	NO
memory/sql/plugin_bookmark	NO	NO
memory/sql/plugin_int_mem_root	NO	NO
memory/sql/plugin_mem_root	NO	NO
memory/sql/Prepared_statement::main_mem_root	NO	NO
memory/sql/Prepared_statement_map	NO	NO
memory/sql/PROFILE	NO	NO
memory/sql/Query_cache	NO	NO
memory/sql/Queue::queue_item	NO	NO
memory/sql/QUICK_RANGE_SELECT::alloc	NO	NO
memory/sql/QUICK_RANGE_SELECT::mrr_buf_desc	NO	NO
memory/sql/Relay_log_info::group_relay_log_name	NO	NO
memory/sql/root	NO	NO
memory/sql/Row_data_memory::memory	NO	NO
memory/sql/rpl_filter_memory	NO	NO

memory/sql/Rpl_info_file::buffer	NO	NO
memory/sql/servers_cache	NO	NO
memory/sql/SLAVE_INFO	NO	NO
memory/sql/Sort_param::tmp_buffer	NO	NO
memory/sql/sp_head::call_mem_root	NO	NO
memory/sql/sp_head::execute_mem_root	NO	NO
memory/sql/sp_head::main_mem_root	NO	NO
memory/sql/sql_acl_mem	NO	NO
memory/sql/sql_acl_memex	NO	NO
memory/sql/String::value	NO	NO
memory/sql/ST_SCHEMA_TABLE	NO	NO
memory/sql/Sys_var_charptr::value	NO	NO
memory/sql/TABLE	NO	NO
memory/sql/table_mapping::m_mem_root	NO	NO
memory/sql/TABLE_RULE_ENT	NO	NO
memory/sql/TABLE_SHARE::mem_root	NO	NO
memory/sql/Table_triggers_list	NO	NO
memory/sql/Table_trigger_dispatcher::m_mem_root	NO	NO
memory/sql/TC_LOG_MMAP::pages	NO	NO
memory/sql/THD::db	NO	NO
memory/sql/THD::handler_tables_hash	NO	NO
memory/sql/thd::main_mem_root	NO	NO
memory/sql/THD::sp_cache	NO	NO
memory/sql/THD::transactions::mem_root	NO	NO
memory/sql/THD::variables	NO	NO
memory/sql/tz_storage	NO	NO
memory/sql/udf_mem	NO	NO
memory/sql/Unique::merge_buffer	NO	NO
memory/sql/Unique::sort_buffer	NO	NO
memory/sql/user_conn	NO	NO
memory/sql/User_level_lock	NO	NO
memory/sql/user_var_entry	NO	NO
memory/sql/user_var_entry::value	NO	NO
memory/sql/XID	NO	NO
stage/aria/Waiting for a resource	NO	NO
stage/innodb/alter table (end)	YES	YES
stage/innodb/alter table (insert)	YES	YES
stage/innodb/alter table (log apply index)	YES	YES
stage/innodb/alter table (log apply table)	YES	YES
stage/innodb/alter table (merge sort)	YES	YES
stage/innodb/alter table (read PK and internal sort)	YES	YES
stage/innodb/buffer pool load	YES	YES
stage/mysys/Waiting for table level lock	NO	NO
stage/sql/After apply log event	NO	NO
stage/sql/After create	NO	NO
stage/sql/After opening tables	NO	NO
stage/sql/After table lock	NO	NO
stage/sql/Allocating local table	NO	NO
stage/sql/altering table	NO	NO
stage/sql/Apply log event	NO	NO
stage/sql/Changing master	NO	NO
stage/sql/Checking master version	NO	NO
stage/sql/checking permissions	NO	NO
stage/sql/checking privileges on cached query	NO	NO
stage/sql/Checking query cache for query	NO	NO
stage/sql/closing tables	NO	NO
stage/sql/Commit	NO	NO
stage/sql/Commit implicit	NO	NO
stage/sql/Committing alter table to storage engine	NO	NO
stage/sql/Connecting to master	NO	NO
stage/sql/Converting HEAP to Aria	NO	NO
stage/sql/copy to tmp table	YES	YES
stage/sql/Copying to group table	NO	NO
stage/sql/Copying to tmp table	NO	NO
stage/sql/Creating delayed handler	NO	NO
stage/sql/Creating sort index	NO	NO
stage/sql/creating table	NO	NO
stage/sql/Creating tmp table	NO	NO
stage/sql/Delete from main table	NO	NO
stage/sql/Delete from reference tables	NO	NO
stage/sql/Discard_or_import_tablespace	NO	NO

stage/sql/Enabling keys	NO	NO	
stage/sql/End of update loop	NO	NO	
stage/sql/Executing	NO	NO	
stage/sql/Execution of init_command	NO	NO	
stage/sql/Explaining	NO	NO	
stage/sql/Filling schema table	NO	NO	
stage/sql/Finding key cache	NO	NO	
stage/sql/Finished reading one binlog; switching to next binlog	NO	NO	
stage/sql/Flushing relay log and master info repository.	NO	NO	
stage/sql/Flushing relay-log info file.	NO	NO	
stage/sql/Freeing items	NO	NO	
stage/sql/Fulltext initialization	NO	NO	
stage/sql/Got handler lock	NO	NO	
stage/sql/Got old table	NO	NO	
stage/sql/init	NO	NO	
stage/sql/init for update	NO	NO	
stage/sql/Insert	NO	NO	
stage/sql/Invalidating query cache entries (table list)	NO	NO	
stage/sql/Invalidating query cache entries (table)	NO	NO	
stage/sql/Killing slave	NO	NO	
stage/sql/Logging slow query	NO	NO	
stage/sql/Making temporary file (append) before replaying LOAD DATA INFILE	NO	NO	
stage/sql/Making temporary file (create) before replaying LOAD DATA INFILE	NO	NO	
stage/sql/Manage keys	NO	NO	
stage/sql/Master has sent all binlog to slave; waiting for more updates	NO	NO	
stage/sql/Opening tables	NO	NO	
stage/sql/Optimizing	NO	NO	
stage/sql/Preparing	NO	NO	
stage/sql/preparing for alter table	NO	NO	
stage/sql/Processing binlog checkpoint notification	NO	NO	
stage/sql/Processing requests	NO	NO	
stage/sql/Purging old relay logs	NO	NO	
stage/sql/Query end	NO	NO	
stage/sql/Queueing master event to the relay log	NO	NO	
stage/sql/Reading event from the relay log	NO	NO	
stage/sql/Reading semi-sync ACK from slave	NO	NO	
stage/sql/Recreating table	NO	NO	
stage/sql/Registering slave on master	NO	NO	
stage/sql/Removing duplicates	NO	NO	
stage/sql/Removing tmp table	NO	NO	
stage/sql/Rename	NO	NO	
stage/sql/Rename result table	NO	NO	
stage/sql/Requesting binlog dump	NO	NO	
stage/sql/Reschedule	NO	NO	
stage/sql/Reset for next command	NO	NO	
stage/sql/Rollback	NO	NO	
stage/sql/Rollback_implicit	NO	NO	
stage/sql/Searching rows for update	NO	NO	
stage/sql/Sending binlog event to slave	NO	NO	
stage/sql/Sending cached result to client	NO	NO	
stage/sql/Sending data	NO	NO	
stage/sql/setup	NO	NO	
stage/sql>Show explain	NO	NO	
stage/sql/Slave has read all relay log; waiting for more updates	NO	NO	
stage/sql/Sorting	NO	NO	
stage/sql/Sorting for group	NO	NO	
stage/sql/Sorting for order	NO	NO	
stage/sql/Sorting result	NO	NO	
stage/sql/startng	NO	NO	
stage/sql/Starting cleanup	NO	NO	
stage/sql/Statistics	NO	NO	
stage/sql/Stopping binlog background thread	NO	NO	
stage/sql/Storing result in query cache	NO	NO	
stage/sql/Storing row into queue	NO	NO	
stage/sql/System lock	NO	NO	
stage/sql/table lock	NO	NO	
stage/sql/Unlocking tables	NO	NO	
stage/sql/Update	NO	NO	
stage/sql/Updating	NO	NO	
stage/sql/Updating main table	NO	NO	
stage/sql/Updating reference tables	NO	NO	
stage/sql/Upgrading lock	NO	NO	

stage/sql/User lock	NO	NO
stage/sql/User sleep	NO	NO
stage/sql/Verifying table	NO	NO
stage/sql/Waiting for background binlog tasks	NO	NO
stage/sql/Waiting for backup lock	NO	NO
stage/sql/Waiting for delay_list	NO	NO
stage/sql/Waiting for event metadata lock	NO	NO
stage/sql/Waiting for GTID to be written to binary log	NO	NO
stage/sql/Waiting for handler insert	NO	NO
stage/sql/Waiting for handler lock	NO	NO
stage/sql/Waiting for handler open	NO	NO
stage/sql/Waiting for INSERT	NO	NO
stage/sql/Waiting for master to send event	NO	NO
stage/sql/Waiting for master update	NO	NO
stage/sql/Waiting for next activation	NO	NO
stage/sql/Waiting for other master connection to process the same GTID	NO	NO
stage/sql/Waiting for parallel replication deadlock handling to complete	NO	NO
stage/sql/Waiting for prior transaction to commit	NO	NO
stage/sql/Waiting for prior transaction to start commit	NO	NO
stage/sql/Waiting for query cache lock	NO	NO
stage/sql/Waiting for requests	NO	NO
stage/sql/Waiting for room in worker thread event queue	NO	NO
stage/sql/Waiting for schema metadata lock	NO	NO
stage/sql/Waiting for semi-sync ACK from slave	NO	NO
stage/sql/Waiting for semi-sync slave connection	NO	NO
stage/sql/Waiting for slave mutex on exit	NO	NO
stage/sql/Waiting for slave thread to start	NO	NO
stage/sql/Waiting for stored function metadata lock	NO	NO
stage/sql/Waiting for stored package body metadata lock	NO	NO
stage/sql/Waiting for stored procedure metadata lock	NO	NO
stage/sql/Waiting for table flush	NO	NO
stage/sql/Waiting for table metadata lock	NO	NO
stage/sql/Waiting for the next event in relay log	NO	NO
stage/sql/Waiting for the scheduler to stop	NO	NO
stage/sql/Waiting for the slave SQL thread to advance position	NO	NO
stage/sql/Waiting for the slave SQL thread to free enough relay log space	NO	NO
stage/sql/Waiting for trigger metadata lock	NO	NO
stage/sql/Waiting for work from SQL thread	NO	NO
stage/sql/Waiting in MASTER_GTID_WAIT()	NO	NO
stage/sql/Waiting in MASTER_GTID_WAIT() (primary waiter)	NO	NO
stage/sql/Waiting on empty queue	NO	NO
stage/sql/Waiting to finalize termination	NO	NO
stage/sql/Waiting until MASTER_DELAY seconds after master executed event	NO	NO
stage/sql/Writing to binlog	NO	NO
statement/abstract/new_packet	YES	YES
statement/abstract/Query	YES	YES
statement/abstract/relay_log	YES	YES
statement/com/Binlog Dump	YES	YES
statement/com/Bulk_execute	YES	YES
statement/com/Change user	YES	YES
statement/com/Close stmt	YES	YES
statement/com/Com_multi	YES	YES
statement/com/Connect	YES	YES
statement/com/Connect Out	YES	YES
statement/com/Create DB	YES	YES
statement/com/Daemon	YES	YES
statement/com/Debug	YES	YES
statement/com/Delayed insert	YES	YES
statement/com/Drop DB	YES	YES
statement/com/Error	YES	YES
statement/com/Execute	YES	YES
statement/com/Fetch	YES	YES
statement/com/Field List	YES	YES
statement/com/Init DB	YES	YES
statement/com/Kill	YES	YES
statement/com/Long Data	YES	YES
statement/com/Ping	YES	YES
statement/com/Prepare	YES	YES
statement/com/Processlist	YES	YES
statement/com/Quit	YES	YES
statement/com/Refresh	YES	YES

statement/com/Register Slave	YES	YES
statement/com/Reset connection	YES	YES
statement/com/Reset stmt	YES	YES
statement/com/Set option	YES	YES
statement/com/Shutdown	YES	YES
statement/com/Slave_IO	YES	YES
statement/com/Slave_SQL	YES	YES
statement/com/Slave_worker	YES	YES
statement/com/Sleep	YES	YES
statement/com/Statistics	YES	YES
statement/com/Table Dump	YES	YES
statement/com/Time	YES	YES
statement/com/Unimpl get tid	YES	YES
statement/scheduler/event	YES	YES
statement/sp/agg_cfetch	YES	YES
statement/sp/cclose	YES	YES
statement/sp/cfetch	YES	YES
statement/sp/copen	YES	YES
statement/sp/cpop	YES	YES
statement/sp/cpush	YES	YES
statement/sp/cursor_copy_struct	YES	YES
statement/sp/error	YES	YES
statement/sp/freturn	YES	YES
statement/sp/hpop	YES	YES
statement/sp/hpush_jump	YES	YES
statement/sp/hreturn	YES	YES
statement/sp/jump	YES	YES
statement/sp/jump_if_not	YES	YES
statement/sp/preturn	YES	YES
statement/sp/set	YES	YES
statement/sp/set_case_expr	YES	YES
statement/sp/set_trigger_field	YES	YES
statement/sp(stmt	YES	YES
statement/sql/	YES	YES
statement/sql/alter_db	YES	YES
statement/sql/alter_db_upgrade	YES	YES
statement/sql/alter_event	YES	YES
statement/sql/alter_function	YES	YES
statement/sql/alter_procedure	YES	YES
statement/sql/alter_sequence	YES	YES
statement/sql/alter_server	YES	YES
statement/sql/alter_table	YES	YES
statement/sql/alter_tablespace	YES	YES
statement/sql/alter_user	YES	YES
statement/sql/analyze	YES	YES
statement/sql/assign_to_keycache	YES	YES
statement/sql/backup	YES	YES
statement/sql/backup_lock	YES	YES
statement/sql/begin	YES	YES
statement/sql/binlog	YES	YES
statement/sql/call_procedure	YES	YES
statement/sql/change_db	YES	YES
statement/sql/change_master	YES	YES
statement/sql/check	YES	YES
statement/sql/checksum	YES	YES
statement/sql/commit	YES	YES
statement/sql/compound_sql	YES	YES
statement/sql/create_db	YES	YES
statement/sql/create_event	YES	YES
statement/sql/create_function	YES	YES
statement/sql/create_index	YES	YES
statement/sql/create_package	YES	YES
statement/sql/create_package_body	YES	YES
statement/sql/create_procedure	YES	YES
statement/sql/create_role	YES	YES
statement/sql/create_sequence	YES	YES
statement/sql/create_server	YES	YES
statement/sql/create_table	YES	YES
statement/sql/create_trigger	YES	YES
statement/sql/create_udf	YES	YES
statement/sql/create_user	YES	YES
statement/sql/create_view	YES	YES

statement/sql/dealloc_sql	YES	YES
statement/sql/delete	YES	YES
statement/sql/delete_multi	YES	YES
statement/sql/do	YES	YES
statement/sql/drop_db	YES	YES
statement/sql/drop_event	YES	YES
statement/sql/drop_function	YES	YES
statement/sql/drop_index	YES	YES
statement/sql/drop_package	YES	YES
statement/sql/drop_package_body	YES	YES
statement/sql/drop_procedure	YES	YES
statement/sql/drop_role	YES	YES
statement/sql/drop_sequence	YES	YES
statement/sql/drop_server	YES	YES
statement/sql/drop_table	YES	YES
statement/sql/drop_trigger	YES	YES
statement/sql/drop_user	YES	YES
statement/sql/drop_view	YES	YES
statement/sql/empty_query	YES	YES
statement/sql/error	YES	YES
statement/sql/execute_immediate	YES	YES
statement/sql/execute_sql	YES	YES
statement/sql/flush	YES	YES
statement/sql/get_diagnostics	YES	YES
statement/sql/grant	YES	YES
statement/sql/grant_role	YES	YES
statement/sql/ha_close	YES	YES
statement/sql/ha_open	YES	YES
statement/sql/ha_read	YES	YES
statement/sql/help	YES	YES
statement/sql/insert	YES	YES
statement/sql/insert_select	YES	YES
statement/sql/install_plugin	YES	YES
statement/sql/kill	YES	YES
statement/sql/load	YES	YES
statement/sql/lock_tables	YES	YES
statement/sql/optimize	YES	YES
statement/sql/preload_keys	YES	YES
statement/sql/prepare_sql	YES	YES
statement/sql/purge	YES	YES
statement/sql/purge_before_date	YES	YES
statement/sql/release_savepoint	YES	YES
statement/sql/rename_table	YES	YES
statement/sql/rename_user	YES	YES
statement/sql/repair	YES	YES
statement/sql/replace	YES	YES
statement/sql/replace_select	YES	YES
statement/sql/reset	YES	YES
statement/sql/resignal	YES	YES
statement/sql/revoke	YES	YES
statement/sql/revoke_all	YES	YES
statement/sql/revoke_role	YES	YES
statement/sql/rollback	YES	YES
statement/sql/rollback_to_savepoint	YES	YES
statement/sql/savepoint	YES	YES
statement/sql/select	YES	YES
statement/sql/set_option	YES	YES
statement/sql/show_authors	YES	YES
statement/sql/show_binlogs	YES	YES
statement/sql/show_binlog_events	YES	YES
statement/sql/show_binlog_status	YES	YES
statement/sql/showCharsets	YES	YES
statement/sql/show_collations	YES	YES
statement/sql/show_contributors	YES	YES
statement/sql/show_create_db	YES	YES
statement/sql/show_create_event	YES	YES
statement/sql/show_create_func	YES	YES
statement/sql/show_create_package	YES	YES
statement/sql/show_create_package_body	YES	YES
statement/sql/show_create_proc	YES	YES
statement/sql/show_create_table	YES	YES

statement/sql/show_create_trigger	YES	YES
statement/sql/show_create_user	YES	YES
statement/sql/show_databases	YES	YES
statement/sql/show_engine_logs	YES	YES
statement/sql/show_engine_mutex	YES	YES
statement/sql/show_engine_status	YES	YES
statement/sql/show_errors	YES	YES
statement/sql/show_events	YES	YES
statement/sql/show_explain	YES	YES
statement/sql/show_fields	YES	YES
statement/sql/show_function_status	YES	YES
statement/sql/show_generic	YES	YES
statement/sql/show_grants	YES	YES
statement/sql/show_keys	YES	YES
statement/sql/show_open_tables	YES	YES
statement/sql/show_package_body_status	YES	YES
statement/sql/show_package_status	YES	YES
statement/sql/show_plugins	YES	YES
statement/sql/show_privileges	YES	YES
statement/sql/show_procedure_status	YES	YES
statement/sql/show_processlist	YES	YES
statement/sql/show_profile	YES	YES
statement/sql/show_profiles	YES	YES
statement/sql/show_relaylog_events	YES	YES
statement/sql/show_slave_hosts	YES	YES
statement/sql/show_slave_status	YES	YES
statement/sql/show_status	YES	YES
statement/sql/show_storage_engines	YES	YES
statement/sql/show_tables	YES	YES
statement/sql/show_table_status	YES	YES
statement/sql/show_triggers	YES	YES
statement/sql/show_variables	YES	YES
statement/sql/show_warnings	YES	YES
statement/sql/shutdown	YES	YES
statement/sql/signal	YES	YES
statement/sql/start_all_slaves	YES	YES
statement/sql/start_slave	YES	YES
statement/sql/stop_all_slaves	YES	YES
statement/sql/stop_slave	YES	YES
statement/sql/truncate	YES	YES
statement/sql/uninstall_plugin	YES	YES
statement/sql/unlock_tables	YES	YES
statement/sql/update	YES	YES
statement/sql/update_multi	YES	YES
statement/sql/xa_commit	YES	YES
statement/sql/xa_end	YES	YES
statement/sql/xa_prepare	YES	YES
statement/sql/xa_recover	YES	YES
statement/sql/xa_rollback	YES	YES
statement/sql/xa_start	YES	YES
transaction	NO	NO
wait/io/file/aria/control	YES	YES
wait/io/file/aria/MAD	YES	YES
wait/io/file/aria/MAI	YES	YES
wait/io/file/aria/translog	YES	YES
wait/io/file/csv/data	YES	YES
wait/io/file/csv/metadata	YES	YES
wait/io/file/csv/update	YES	YES
wait/io/file/innodb/innodb_data_file	YES	YES
wait/io/file/innodb/innodb_log_file	YES	YES
wait/io/file/innodb/innodb_temp_file	YES	YES
wait/io/file/myisam/data_tmp	YES	YES
wait/io/file/myisam/dfile	YES	YES
wait/io/file/myisam/kfile	YES	YES
wait/io/file/myisam/log	YES	YES
wait/io/file/myisammrg/MRG	YES	YES
wait/io/file/mysys/charset	YES	YES
wait/io/file/mysys/cnf	YES	YES
wait/io/file/partition/ha_partition::parfile	YES	YES
wait/io/file/sql/binlog	YES	YES
wait/io/file/sql/binlog_cache	YES	YES
wait/io/file/sql/binlog_index	YES	YES

wait/io/file/sql/binlog_index_cache	YES	YES
wait/io/file/sql/binlog_state	YES	YES
wait/io/file/sql/casetest	YES	YES
wait/io/file/sql/dbopt	YES	YES
wait/io/file/sql/des_key_file	YES	YES
wait/io/file/sql/ERRMSG	YES	YES
wait/io/file/sql/file_parser	YES	YES
wait/io/file/sql/FRM	YES	YES
wait/io/file/sql/global_ddl_log	YES	YES
wait/io/file/sql/init	YES	YES
wait/io/file/sql/io_cache	YES	YES
wait/io/file/sql/load	YES	YES
wait/io/file/sql/LOAD_FILE	YES	YES
wait/io/file/sql/log_event_data	YES	YES
wait/io/file/sql/log_event_info	YES	YES
wait/io/file/sql/map	YES	YES
wait/io/file/sql/master_info	YES	YES
wait/io/file/sql/misc	YES	YES
wait/io/file/sql/partition_ddl_log	YES	YES
wait/io/file/sql/pid	YES	YES
wait/io/file/sql/query_log	YES	YES
wait/io/file/sql/relaylog	YES	YES
wait/io/file/sql/relaylog_cache	YES	YES
wait/io/file/sql/relaylog_index	YES	YES
wait/io/file/sql/relaylog_index_cache	YES	YES
wait/io/file/sql/relay_log_info	YES	YES
wait/io/file/sql/select_to_file	YES	YES
wait/io/file/sql/send_file	YES	YES
wait/io/file/sql/slow_log	YES	YES
wait/io/file/sql/tclog	YES	YES
wait/io/file/sql/trigger	YES	YES
wait/io/file/sql/trigger_name	YES	YES
wait/io/file/sql/wsrep_gra_log	YES	YES
wait/io/socket/sql/client_connection	NO	NO
wait/io/socket/sql/server_tcpip_socket	NO	NO
wait/io/socket/sql/server_unix_socket	NO	NO
wait/io/table/sql/handler	YES	YES
wait/lock/metadata/sql/mdl	NO	NO
wait/lock/table/sql/handler	YES	YES
wait/synch/cond/aria/BITMAP::bitmap_cond	NO	NO
wait/synch/cond/aria/COND_soft_sync	NO	NO
wait/synch/cond/aria/SERVICE_THREAD_CONTROL::COND_control	NO	NO
wait/synch/cond/aria/SHARE::key_del_cond	NO	NO
wait/synch/cond/aria/SORT_INFO::cond	NO	NO
wait/synch/cond/aria/TRANSLOG_BUFFER::prev_sent_to_disk_cond	NO	NO
wait/synch/cond/aria/TRANSLOG_BUFFER::waiting_filling_buffer	NO	NO
wait/synch/cond/aria/TRANSLOG_DESCRIPTOR::log_flush_cond	NO	NO
wait/synch/cond/aria/TRANSLOG_DESCRIPTOR::new_goal_cond	NO	NO
wait/synch/cond/innodb/commit_cond	NO	NO
wait/synch/cond/myisam/MI_SORT_INFO::cond	NO	NO
wait/synch/cond/mysys/COND_alarm	NO	NO
wait/synch/cond/mysys/COND_timer	NO	NO
wait/synch/cond/mysys/IO_CACHE_SHARE::cond	NO	NO
wait/synch/cond/mysys/IO_CACHE_SHARE::cond_writer	NO	NO
wait/synch/cond/mysys/my_thread_var::suspend	NO	NO
wait/synch/cond/mysys/THR_COND_threads	NO	NO
wait/synch/cond/mysys/WT_RESOURCE::cond	NO	NO
wait/synch/cond/sql/Ack_receiver::cond	NO	NO
wait/synch/cond/sql/COND_binlog_send	NO	NO
wait/synch/cond/sql/COND_flush_thread_cache	NO	NO
wait/synch/cond/sql/COND_group_commit_orderer	NO	NO
wait/synch/cond/sql/COND_gtid_ignore_duplicates	NO	NO
wait/synch/cond/sql/COND_manager	NO	NO
wait/synch/cond/sql/COND_parallel_entry	NO	NO
wait/synch/cond/sql/COND_prepare_ordered	NO	NO
wait/synch/cond/sql/COND_queue_state	NO	NO
wait/synch/cond/sql/COND_rpl_thread	NO	NO
wait/synch/cond/sql/COND_rpl_thread_pool	NO	NO
wait/synch/cond/sql/COND_rpl_thread_queue	NO	NO
wait/synch/cond/sql/COND_rpl_thread_stop	NO	NO
wait/synch/cond/sql/COND_server_started	NO	NO

wait/synch/cond/sql/COND_slave_background	NO	NO	
wait/synch/cond/sql/COND_start_thread	NO	NO	
wait/synch/cond/sql/COND_thread_cache	NO	NO	
wait/synch/cond/sql/COND_wait_gtid	NO	NO	
wait/synch/cond/sql/COND_wsrep_donor_monitor	NO	NO	
wait/synch/cond/sql/COND_wsrep_gtid_wait_upto	NO	NO	
wait/synch/cond/sql/COND_wsrep_joinder_monitor	NO	NO	
wait/synch/cond/sql/COND_wsrep_ready	NO	NO	
wait/synch/cond/sql/COND_wsrep_replaying	NO	NO	
wait/synch/cond/sql/COND_wsrep_sst	NO	NO	
wait/synch/cond/sql/COND_wsrep_sst_init	NO	NO	
wait/synch/cond/sql/COND_wsrep_wsrep_slave_threads	NO	NO	
wait/synch/cond/sql/Delayed_insert::cond	NO	NO	
wait/synch/cond/sql/Delayed_insert::cond_client	NO	NO	
wait/synch/cond/sql/Event_scheduler::COND_state	NO	NO	
wait/synch/cond/sql/Item_func_sleep::cond	NO	NO	
wait/synch/cond/sql/Master_info::data_cond	NO	NO	
wait/synch/cond/sql/Master_info::sleep_cond	NO	NO	
wait/synch/cond/sql/Master_info::start_cond	NO	NO	
wait/synch/cond/sql/Master_info::stop_cond	NO	NO	
wait/synch/cond/sql/MDL_context::COND_wait_status	NO	NO	
wait/synch/cond/sql/MYSQL_BIN_LOG::COND_binlog_background_thread	NO	NO	
wait/synch/cond/sql/MYSQL_BIN_LOG::COND_binlog_background_thread_end	NO	NO	
wait/synch/cond/sql/MYSQL_BIN_LOG::COND_bin_log_updated	NO	NO	
wait/synch/cond/sql/MYSQL_BIN_LOG::COND_queue_busy	NO	NO	
wait/synch/cond/sql/MYSQL_BIN_LOG::COND_relay_log_updated	NO	NO	
wait/synch/cond/sql/MYSQL_BIN_LOG::COND_xid_list	NO	NO	
wait/synch/cond/sql/MYSQL_RELAY_LOG::COND_bin_log_updated	NO	NO	
wait/synch/cond/sql/MYSQL_RELAY_LOG::COND_queue_busy	NO	NO	
wait/synch/cond/sql/MYSQL_RELAY_LOG::COND_relay_log_updated	NO	NO	
wait/synch/cond/sql/PAGE::cond	NO	NO	
wait/synch/cond/sql/Query_cache::COND_cache_status_changed	NO	NO	
wait/synch/cond/sql/Relay_log_info::data_cond	NO	NO	
wait/synch/cond/sql/Relay_log_info::log_space_cond	NO	NO	
wait/synch/cond/sql/Relay_log_info::start_cond	NO	NO	
wait/synch/cond/sql/Relay_log_info::stop_cond	NO	NO	
wait/synch/cond/sql/Rpl_group_info::sleep_cond	NO	NO	
wait/synch/cond/sql/show_explain	NO	NO	
wait/synch/cond/sql/TABLE_SHARE::cond	NO	NO	
wait/synch/cond/sql/TABLE_SHARE::COND_rotation	NO	NO	
wait/synch/cond/sql/TABLE_SHARE::tdc.COND_release	NO	NO	
wait/synch/cond/sql/TC_LOG_MMAP::COND_active	NO	NO	
wait/synch/cond/sql/TC_LOG_MMAP::COND_pool	NO	NO	
wait/synch/cond/sql/TC_LOG_MMAP::COND_queue_busy	NO	NO	
wait/synch/cond/sql/THD::COND_wakeup_ready	NO	NO	
wait/synch/cond/sql/THD::COND_wsrep_thd	NO	NO	
wait/synch/cond/sql/User_level_lock::cond	NO	NO	
wait/synch/cond/sql/wait_for_commit::COND_wait_commit	NO	NO	
wait/synch/cond/sql/wsrep_sst_thread	NO	NO	
wait/synch/mutex/aria/LOCK_soft_sync	NO	NO	
wait/synch/mutex/aria/LOCK_trn_list	NO	NO	
wait/synch/mutex/aria/PAGECACHE::cache_lock	NO	NO	
wait/synch/mutex/aria/SERVICE_THREAD_CONTROL::LOCK_control	NO	NO	
wait/synch/mutex/aria/SHARE::bitmap::bitmap_lock	NO	NO	
wait/synch/mutex/aria/SHARE::close_lock	NO	NO	
wait/synch/mutex/aria/SHARE::intern_lock	NO	NO	
wait/synch/mutex/aria/SHARE::key_del_lock	NO	NO	
wait/synch/mutex/aria/SORT_INFO::mutex	NO	NO	
wait/synch/mutex/aria/THR_LOCK_maria	NO	NO	
wait/synch/mutex/aria/TRANSLOG_BUFFER::mutex	NO	NO	
wait/synch/mutex/aria/TRANSLOG_DESCRIPTOR::dirty_buffer_mask_lock	NO	NO	
wait/synch/mutex/aria/TRANSLOG_DESCRIPTOR::file_header_lock	NO	NO	
wait/synch/mutex/aria/TRANSLOG_DESCRIPTOR::log_flush_lock	NO	NO	
wait/synch/mutex/aria/TRANSLOG_DESCRIPTOR::purger_lock	NO	NO	
wait/synch/mutex/aria/TRANSLOG_DESCRIPTOR::sent_to_disk_lock	NO	NO	
wait/synch/mutex/aria/TRANSLOG_DESCRIPTOR::unfinished_files_lock	NO	NO	
wait/synch/mutex/aria/TRN::state_lock	NO	NO	
wait/synch/mutex/csv/tina	NO	NO	
wait/synch/mutex/csv/TINA_SHARE::mutex	NO	NO	
wait/synch/mutex/innodb/buf_dbwr_mutex	NO	NO	
wait/synch/mutex/innodb/buf_pool_mutex	NO	NO	
wait/synch/mutex/innodb/commit_cond_mutex	NO	NO	

wait/synch/mutex/innodb/dict_foreign_err_mutex	NO	NO
wait/synch/mutex/innodb/dict_sys_mutex	NO	NO
wait/synch/mutex/innodb/fil_system_mutex	NO	NO
wait/synch/mutex/innodb/flush_list_mutex	NO	NO
wait/synch/mutex/innodb/fts_delete_mutex	NO	NO
wait/synch/mutex/innodb/fts_doc_id_mutex	NO	NO
wait/synch/mutex/innodb/ibuf_bitmap_mutex	NO	NO
wait/synch/mutex/innodb/ibuf_mutex	NO	NO
wait/synch/mutex/innodb/ibuf_pessimistic_insert_mutex	NO	NO
wait/synch/mutex/innodb/lock_mutex	NO	NO
wait/synch/mutex/innodb/lock_wait_mutex	NO	NO
wait/synch/mutex/innodb/log_flush_order_mutex	NO	NO
wait/synch/mutex/innodb/log_sys_mutex	NO	NO
wait/synch/mutex/innodb/noredo_rseg_mutex	NO	NO
wait/synch/mutex/innodb/page_zip_stat_per_index_mutex	NO	NO
wait/synch/mutex/innodb/pending_checkpoint_mutex	NO	NO
wait/synch/mutex/innodb/purge_sys_pq_mutex	NO	NO
wait/synch/mutex/innodb/recalc_pool_mutex	NO	NO
wait/synch/mutex/innodb/recv_sys_mutex	NO	NO
wait/synch/mutex/innodb/redo_rseg_mutex	NO	NO
wait/synch/mutex/innodb/rtr_active_mutex	NO	NO
wait/synch/mutex/innodb/rtr_match_mutex	NO	NO
wait/synch/mutex/innodb/rtr_path_mutex	NO	NO
wait/synch/mutex/innodb/rw_lock_list_mutex	NO	NO
wait/synch/mutex/innodb/srv_innodb_monitor_mutex	NO	NO
wait/synch/mutex/innodb/srv_misc_tmpfile_mutex	NO	NO
wait/synch/mutex/innodb/srv_monitor_file_mutex	NO	NO
wait/synch/mutex/innodb/srv_threads_mutex	NO	NO
wait/synch/mutex/innodb/trx_mutex	NO	NO
wait/synch/mutex/innodb/trx_pool_manager_mutex	NO	NO
wait/synch/mutex/innodb/trx_pool_mutex	NO	NO
wait/synch/mutex/innodb/trx_sys_mutex	NO	NO
wait/synch/mutex/myisam/MI_CHECK::print_msg	NO	NO
wait/synch/mutex/myisam/MI_SORT_INFO::mutex	NO	NO
wait/synch/mutex/myisam/MYISAM_SHARE::intern_lock	NO	NO
wait/synch/mutex/myisammrg/MYRG_INFO::mutex	NO	NO
wait/synch/mutex/mysys/BITMAP::mutex	NO	NO
wait/synch/mutex/mysys/IO_CACHE::append_buffer_lock	NO	NO
wait/synch/mutex/mysys/IO_CACHE::SHARE_mutex	NO	NO
wait/synch/mutex/mysys/KEY_CACHE::cache_lock	NO	NO
wait/synch/mutex/mysys/LOCK_alarm	NO	NO
wait/synch/mutex/mysys/LOCK_timer	NO	NO
wait/synch/mutex/mysys/LOCK_uuid_generator	NO	NO
wait/synch/mutex/mysys/my_thread_var::mutex	NO	NO
wait/synch/mutex/mysys/THR_LOCK::mutex	NO	NO
wait/synch/mutex/mysys/THR_LOCK_charset	NO	NO
wait/synch/mutex/mysys/THR_LOCK_heap	NO	NO
wait/synch/mutex/mysys/THR_LOCK_lock	NO	NO
wait/synch/mutex/mysys/THR_LOCK_malloc	NO	NO
wait/synch/mutex/mysys/THR_LOCK_myisam	NO	NO
wait/synch/mutex/mysys/THR_LOCK_myisam_mmap	NO	NO
wait/synch/mutex/mysys/THR_LOCK_net	NO	NO
wait/synch/mutex/mysys/THR_LOCK_open	NO	NO
wait/synch/mutex/mysys/THR_LOCK_threads	NO	NO
wait/synch/mutex/mysys/TMPDIR_mutex	NO	NO
wait/synch/mutex/partition/Partition_share::auto_inc_mutex	NO	NO
wait/synch/mutex/sql/Ack_receiver::mutex	NO	NO
wait/synch/mutex/sql/Cversion_lock	NO	NO
wait/synch/mutex/sql/Delayed_insert::mutex	NO	NO
wait/synch/mutex/sql/Event_scheduler::LOCK_scheduler_state	NO	NO
wait/synch/mutex/sql/gtid_waiting::LOCK_gtid_waiting	NO	NO
wait/synch/mutex/sql/hash_filo::lock	NO	NO
wait/synch/mutex/sql/HA_DATA_PARTITION::LOCK_auto_inc	NO	NO
wait/synch/mutex/sql/LOCK_active_mi	NO	NO
wait/synch/mutex/sql/LOCK_after_binlog_sync	NO	NO
wait/synch/mutex/sql/LOCK_audit_mask	NO	NO
wait/synch/mutex/sql/LOCK_binlog	NO	NO
wait/synch/mutex/sql/LOCK_binlog_state	NO	NO
wait/synch/mutex/sql/LOCK_commit_ordered	NO	NO
wait/synch/mutex/sql/LOCK_crypt	NO	NO
wait/synch/mutex/sql/LOCK_delayed_create	NO	NO

wait/synch/mutex/sql/LOCK_delayed_insert	NO	NO	
wait/synch/mutex/sql/LOCK_delayed_status	NO	NO	
wait/synch/mutex/sql/LOCK_des_key_file	NO	NO	
wait/synch/mutex/sql/LOCK_error_log	NO	NO	
wait/synch/mutex/sql/LOCK_error_messages	NO	NO	
wait/synch/mutex/sql/LOCK_event_queue	NO	NO	
wait/synch/mutex/sql/LOCK_gdl	NO	NO	
wait/synch/mutex/sql/LOCK_global_index_stats	NO	NO	
wait/synch/mutex/sql/LOCK_global_system_variables	NO	NO	
wait/synch/mutex/sql/LOCK_global_table_stats	NO	NO	
wait/synch/mutex/sql/LOCK_global_user_client_stats	NO	NO	
wait/synch/mutex/sql/LOCK_item_func_sleep	NO	NO	
wait/synch/mutex/sql/LOCK_load_client_plugin	NO	NO	
wait/synch/mutex/sql/LOCK_manager	NO	NO	
wait/synch/mutex/sql/LOCK_parallel_entry	NO	NO	
wait/synch/mutex/sql/LOCK_plugin	NO	NO	
wait/synch/mutex/sql/LOCK_prepared_stmt_count	NO	NO	
wait/synch/mutex/sql/LOCK_prepare_ordered	NO	NO	
wait/synch/mutex/sql/LOCK_rpl_semi_sync_master_enabled	NO	NO	
wait/synch/mutex/sql/LOCK_rpl_status	NO	NO	
wait/synch/mutex/sql/LOCK_rpl_thread	NO	NO	
wait/synch/mutex/sql/LOCK_rpl_thread_pool	NO	NO	
wait/synch/mutex/sql/LOCK_server_started	NO	NO	
wait/synch/mutex/sql/LOCK_slave_background	NO	NO	
wait/synch/mutex/sql/LOCK_slave_state	NO	NO	
wait/synch/mutex/sql/LOCK_start_thread	NO	NO	
wait/synch/mutex/sql/LOCK_stats	NO	NO	
wait/synch/mutex/sql/LOCK_status	NO	NO	
wait/synch/mutex/sql/LOCK_system_variables_hash	NO	NO	
wait/synch/mutex/sql/LOCK_table_cache	NO	NO	
wait/synch/mutex/sql/LOCK_thread_cache	NO	NO	
wait/synch/mutex/sql/LOCK_thread_id	NO	NO	
wait/synch/mutex/sql/LOCK_unused_shares	NO	NO	
wait/synch/mutex/sql/LOCK_user_conn	NO	NO	
wait/synch/mutex/sql/LOCK_uuid_short_generator	NO	NO	
wait/synch/mutex/sql/LOCK_wsrep_cluster_config	NO	NO	
wait/synch/mutex/sql/LOCK_wsrep_config_state	NO	NO	
wait/synch/mutex/sql/LOCK_wsrep_desync	NO	NO	
wait/synch/mutex/sql/LOCK_wsrep_donor_monitor	NO	NO	
wait/synch/mutex/sql/LOCK_wsrep_group_commit	NO	NO	
wait/synch/mutex/sql/LOCK_wsrep_gtid_wait_upto	NO	NO	
wait/synch/mutex/sql/LOCK_wsrep_joiner_monitor	NO	NO	
wait/synch/mutex/sql/LOCK_wsrep_ready	NO	NO	
wait/synch/mutex/sql/LOCK_wsrep_replaying	NO	NO	
wait/synch/mutex/sql/LOCK_wsrep_slave_threads	NO	NO	
wait/synch/mutex/sql/LOCK_wsrep_SR_pool	NO	NO	
wait/synch/mutex/sql/LOCK_wsrep_SR_store	NO	NO	
wait/synch/mutex/sql/LOCK_wsrep_sst	NO	NO	
wait/synch/mutex/sql/LOCK_wsrep_sst_init	NO	NO	
wait/synch/mutex/sql/LOG::LOCK_log	NO	NO	
wait/synch/mutex/sql/Master_info::data_lock	NO	NO	
wait/synch/mutex/sql/Master_info::run_lock	NO	NO	
wait/synch/mutex/sql/Master_info::sleep_lock	NO	NO	
wait/synch/mutex/sql/Master_info::start_stop_lock	NO	NO	
wait/synch/mutex/sql/MDL_wait::LOCK_wait_status	NO	NO	
wait/synch/mutex/sql/MYSQL_BIN_LOG::LOCK_binlog_background_thread	NO	NO	
wait/synch/mutex/sql/MYSQL_BIN_LOG::LOCK_binlog_end_pos	NO	NO	
wait/synch/mutex/sql/MYSQL_BIN_LOG::LOCK_index	NO	NO	
wait/synch/mutex/sql/MYSQL_BIN_LOG::LOCK_xid_list	NO	NO	
wait/synch/mutex/sql/MYSQL_RELAY_LOG::LOCK_binlog_end_pos	NO	NO	
wait/synch/mutex/sql/MYSQL_RELAY_LOG::LOCK_index	NO	NO	
wait/synch/mutex/sql/PAGE::lock	NO	NO	
wait/synch/mutex/sql/Query_cache::structure_guard_mutex	NO	NO	
wait/synch/mutex/sql/Relay_log_info::data_lock	NO	NO	
wait/synch/mutex/sql/Relay_log_info::log_space_lock	NO	NO	
wait/synch/mutex/sql/Relay_log_info::run_lock	NO	NO	
wait/synch/mutex/sql/Rpl_group_info::sleep_lock	NO	NO	
wait/synch/mutex/sql/Slave_reporting_capability::err_lock	NO	NO	
wait/synch/mutex/sql/TABLE_SHARE::LOCK_ha_data	NO	NO	
wait/synch/mutex/sql/TABLE_SHARE::LOCK_rotation	NO	NO	
wait/synch/mutex/sql/TABLE_SHARE::LOCK_share	NO	NO	
wait/synch/mutex/sql/TABLE_SHARE::tdc.LOCK_table_share	NO	NO	

wait/synch/mutex/sql/TC_LOG_MMAP::LOCK_active	NO	NO
wait/synch/mutex/sql/TC_LOG_MMAP::LOCK_pending_checkpoint	NO	NO
wait/synch/mutex/sql/TC_LOG_MMAP::LOCK_pool	NO	NO
wait/synch/mutex/sql/TC_LOG_MMAP::LOCK_sync	NO	NO
wait/synch/mutex/sql/THD::LOCK_thd_data	NO	NO
wait/synch/mutex/sql/THD::LOCK_thd_kill	NO	NO
wait/synch/mutex/sql/THD::LOCK_wakeup_ready	NO	NO
wait/synch/mutex/sql/tz_LOCK	NO	NO
wait/synch/mutex/sql/wait_for_commit::LOCK_wait_commit	NO	NO
wait/synch/mutex/sql/wsrep_sst_thread	NO	NO
wait/synch/rwlock/aria/KEYINFO::root_lock	NO	NO
wait/synch/rwlock/aria/SHARE::mmap_lock	NO	NO
wait/synch/rwlock/aria/TRANSLOG_DESCRIPTOR::open_files_lock	NO	NO
wait/synch/rwlock/myisam/MYISAM_SHARE::key_root_lock	NO	NO
wait/synch/rwlock/myisam/MYISAM_SHARE::mmap_lock	NO	NO
wait/synch/rwlock/mysys/SAFE_HASH::mutex	NO	NO
wait/synch/rwlock/proxy_proto/rwlock	NO	NO
wait/synch/rwlock/sql/CRYPTO_dynlock_value::lock	NO	NO
wait/synch/rwlock/sql/LOCK_all_status_vars	NO	NO
wait/synch/rwlock/sql/LOCK_dboptions	NO	NO
wait/synch/rwlock/sql/LOCK_grant	NO	NO
wait/synch/rwlock/sql/LOCK_SEQUENCE	NO	NO
wait/synch/rwlock/sql/LOCK_ssl_refresh	NO	NO
wait/synch/rwlock/sql/LOCK_system_variables_hash	NO	NO
wait/synch/rwlock/sql/LOCK_sys_init_connect	NO	NO
wait/synch/rwlock/sql/LOCK_sys_init_slave	NO	NO
wait/synch/rwlock/sql/LOGGER::LOCK_logger	NO	NO
wait/synch/rwlock/sql/MDL_context::LOCK_waiting_for	NO	NO
wait/synch/rwlock/sql/MDL_lock::rwlock	NO	NO
wait/synch/rwlock/sql/Query_cache_query::lock	NO	NO
wait/synch/rwlock/sql/TABLE_SHARE::LOCK_stat_serial	NO	NO
wait/synch/rwlock/sql/THD_list::lock	NO	NO
wait/synch/rwlock/sql/THR_LOCK_servers	NO	NO
wait/synch/rwlock/sql/THR_LOCK_udf	NO	NO
wait/synch/rwlock/sql/Vers_field_stats::lock	NO	NO
wait/synch/sxlock/innodb/btr_search_latch	NO	NO
wait/synch/sxlock/innodb/dict_operation_lock	NO	NO
wait/synch/sxlock/innodb/fil_space_latch	NO	NO
wait/synch/sxlock/innodb/fts_cache_init_rw_lock	NO	NO
wait/synch/sxlock/innodb/fts_cache_rw_lock	NO	NO
wait/synch/sxlock/innodb/index_online_log	NO	NO
wait/synch/sxlock/innodb/index_tree_rw_lock	NO	NO
wait/synch/sxlock/innodb/trx_i_s_cache_lock	NO	NO
wait/synch/sxlock/innodb/trx_purge_latch	NO	NO

996 rows in set (0.005 sec)

1.1.2.9.2.1.64 Performance Schema setup_objects Table

Description

The `setup_objects` table determines whether objects are monitored by the performance schema or not. By default limited to 100 rows, this can be changed by setting the `performance_schema_setup_objects_size` system variable when the server starts.

It contains the following columns:

Column	Description
OBJECT_TYPE	Type of object to instrument, currently only <code>TABLE</code> , for base table.
OBJECT_SCHEMA	Schema containing the object, either the literal or <code>%</code> for any schema.
OBJECT_NAME	Name of the instrumented object, either the literal or <code>%</code> for any object.
ENABLED	Whether the object's events are instrumented or not. Can be disabled, in which case monitoring is not enabled for those objects.
TIMED	Whether the object's events are timed or not. Can be modified.

When the Performance Schema looks for matches in the `setup_objects` , there may be more than one row matching, with different `ENABLED` and

`TIMED` values. It looks for the most specific matches first, that is, it will first look for the specific database and table name combination, then the specific database, only then falling back to a wildcard for both.

Rows can be added or removed from the table, while for existing rows, only the `TIMED` and `ENABLED` columns can be updated. By default, all tables except those in the `performance_schema`, `information_schema` and `mysql` databases are instrumented.

1.1.2.9.2.1.65 Performance Schema setup_timers Table

Description

The `setup_timers` table shows the currently selected event timers.

It contains the following columns:

Column	Description
NAME	Type of instrument the timer is used for.
TIMER_NAME	Timer applying to the instrument type. Can be modified.

The `TIMER_NAME` value can be changed to choose a different timer, and can be any non-NULL value in the `performance_timers.TIMER_NAME` column.

If you modify the table, monitoring is immediately affected, and currently monitored events would use a combination of old and new timers, which is probably undesirable. It is best to reset the Performance Schema statistics if you make changes to this table.

Example

```
SELECT * FROM setup_timers;
+-----+-----+
| NAME      | TIMER_NAME   |
+-----+-----+
| idle      | MICROSECOND |
| wait      | CYCLE        |
| stage     | NANOSECOND   |
| statement | NANOSECOND   |
+-----+-----+
```

1.1.2.9.2.1.66 Performance Schema socket_instances Table

The `socket_instances` table lists active server connections, with each record being a Unix socket file or TCP/IP connection.

The `socket_instances` table contains the following columns:

Column	Description
EVENT_NAME	NAME from the <code>setup_instruments</code> table, and the name of the <code>wait/io/socket/*</code> instrument that produced the event.
OBJECT_INSTANCE_BEGIN	Memory address of the object.
THREAD_ID	Thread identifier that the server assigns to each socket.
SOCKET_ID	The socket's internal file handle.
IP	Client IP address. Blank for Unix socket file, otherwise an IPv4 or IPv6 address. Together with the PORT identifies the connection.
PORT	TCP/IP port number, from 0 to 65535. Together with the IP identifies the connection.
STATE	Socket status, either <code>IDLE</code> if waiting to receive a request from a client, or <code>ACTIVE</code>

1.1.2.9.2.1.67 Performance Schema socket_summary_by_event_name Table

It aggregates timer and byte count statistics for all socket I/O operations by socket instrument.

Column	Description
EVENT_NAME	Socket instrument.
COUNT_STAR	Number of summarized events
SUM_TIMER_WAIT	Total wait time of the summarized events that are timed.
MIN_TIMER_WAIT	Minimum wait time of the summarized events that are timed.
AVG_TIMER_WAIT	Average wait time of the summarized events that are timed.
MAX_TIMER_WAIT	Maximum wait time of the summarized events that are timed.
COUNT_READ	Number of all read operations, including RECV , RECVFROM , and RECVMSG .
SUM_TIMER_READ	Total wait time of all read operations that are timed.
MIN_TIMER_READ	Minimum wait time of all read operations that are timed.
AVG_TIMER_READ	Average wait time of all read operations that are timed.
MAX_TIMER_READ	Maximum wait time of all read operations that are timed.
SUM_NUMBER_OF_BYTES_READ	Bytes read by read operations.
COUNT_WRITE	Number of all write operations, including SEND , SENDTO , and SENDMSG .
SUM_TIMER_WRITE	Total wait time of all write operations that are timed.
MIN_TIMER_WRITE	Minimum wait time of all write operations that are timed.
AVG_TIMER_WRITE	Average wait time of all write operations that are timed.
MAX_TIMER_WRITE	Maximum wait time of all write operations that are timed.
SUM_NUMBER_OF_BYTES_WRITE	Bytes written by write operations.
COUNT_MISC	Number of all miscellaneous operations not counted above, including CONNECT , LISTEN , ACCEPT , CLOSE , and SHUTDOWN .
SUM_TIMER_MISC	Total wait time of all miscellaneous operations that are timed.
MIN_TIMER_MISC	Minimum wait time of all miscellaneous operations that are timed.
AVG_TIMER_MISC	Average wait time of all miscellaneous operations that are timed.
MAX_TIMER_MISC	Maximum wait time of all miscellaneous operations that are timed.

You can [TRUNCATE](#) the table, which will reset all counters to zero.

Example

```

SELECT * FROM socket_summary_by_event_name\G
*****
1. row *****
    EVENT_NAME: wait/io/socket/sql/server_tcpip_socket
    COUNT_STAR: 0
    SUM_TIMER_WAIT: 0
    MIN_TIMER_WAIT: 0
    AVG_TIMER_WAIT: 0
    MAX_TIMER_WAIT: 0
        COUNT_READ: 0
    SUM_TIMER_READ: 0
    MIN_TIMER_READ: 0
    AVG_TIMER_READ: 0
    MAX_TIMER_READ: 0
SUM_NUMBER_OF_BYTES_READ: 0
    COUNT_WRITE: 0
    SUM_TIMER_WRITE: 0
    MIN_TIMER_WRITE: 0
    AVG_TIMER_WRITE: 0
    MAX_TIMER_WRITE: 0
SUM_NUMBER_OF_BYTES_WRITE: 0
    COUNT_MISC: 0
    SUM_TIMER_MISC: 0
    MIN_TIMER_MISC: 0
    AVG_TIMER_MISC: 0
    MAX_TIMER_MISC: 0
*****
2. row *****
    EVENT_NAME: wait/io/socket/sql/server_unix_socket
    COUNT_STAR: 0
    SUM_TIMER_WAIT: 0
    MIN_TIMER_WAIT: 0
    AVG_TIMER_WAIT: 0
    MAX_TIMER_WAIT: 0
        COUNT_READ: 0
    SUM_TIMER_READ: 0
    MIN_TIMER_READ: 0
    AVG_TIMER_READ: 0
    MAX_TIMER_READ: 0
SUM_NUMBER_OF_BYTES_READ: 0
    COUNT_WRITE: 0
    SUM_TIMER_WRITE: 0
    MIN_TIMER_WRITE: 0
    AVG_TIMER_WRITE: 0
    MAX_TIMER_WRITE: 0
SUM_NUMBER_OF_BYTES_WRITE: 0
    COUNT_MISC: 0
    SUM_TIMER_MISC: 0
    MIN_TIMER_MISC: 0
    AVG_TIMER_MISC: 0
    MAX_TIMER_MISC: 0
...

```

1.1.2.9.2.1.68 Performance Schema socket_summary_by_instance Table

It aggregates timer and byte count statistics for all socket I/O operations by socket instance.

Column	Description
EVENT_NAME	Socket instrument.
OBJECT_INSTANCE_BEGIN	Address in memory.
COUNT_STAR	Number of summarized events
SUM_TIMER_WAIT	Total wait time of the summarized events that are timed.
MIN_TIMER_WAIT	Minimum wait time of the summarized events that are timed.
AVG_TIMER_WAIT	Average wait time of the summarized events that are timed.

MAX_TIMER_WAIT	Maximum wait time of the summarized events that are timed.
COUNT_READ	Number of all read operations, including <code>RECV</code> , <code>RECVFROM</code> , and <code>RECVMSG</code> .
SUM_TIMER_READ	Total wait time of all read operations that are timed.
MIN_TIMER_READ	Minimum wait time of all read operations that are timed.
AVG_TIMER_READ	Average wait time of all read operations that are timed.
MAX_TIMER_READ	Maximum wait time of all read operations that are timed.
SUM_NUMBER_OF_BYTES_READ	Bytes read by read operations.
COUNT_WRITE	Number of all write operations, including <code>SEND</code> , <code>SENDTO</code> , and <code>SENDMSG</code> .
SUM_TIMER_WRITE	Total wait time of all write operations that are timed.
MIN_TIMER_WRITE	Minimum wait time of all write operations that are timed.
AVG_TIMER_WRITE	Average wait time of all write operations that are timed.
MAX_TIMER_WRITE	Maximum wait time of all write operations that are timed.
SUM_NUMBER_OF_BYTES_WRITE	Bytes written by write operations.
COUNT_MISC	Number of all miscellaneous operations not counted above, including <code>CONNECT</code> , <code>LISTEN</code> , <code>ACCEPT</code> , <code>CLOSE</code> , and <code>SHUTDOWN</code> .
SUM_TIMER_MISC	Total wait time of all miscellaneous operations that are timed.
MIN_TIMER_MISC	Minimum wait time of all miscellaneous operations that are timed.
AVG_TIMER_MISC	Average wait time of all miscellaneous operations that are timed.
MAX_TIMER_MISC	Maximum wait time of all miscellaneous operations that are timed.

The corresponding row in the table is deleted when a connection terminates.

You can [TRUNCATE](#) the table, which will reset all counters to zero.

1.1.2.9.2.1.69 Performance Schema status_by_thread Table

MariaDB starting with [10.5.2](#)

The `session_status` table was added in [MariaDB 10.5.2](#).

The `status_by_thread` table contains status variable information about active foreground threads. The table does not collect statistics for `Com_xxx` variables.

The table contains the following columns:

Column	Description
THREAD_ID	The thread identifier of the session in which the status variable is defined.
VARIABLE_NAME	Status variable name.
VARIABLE_VALUE	Aggregated status variable value.

If [TRUNCATE TABLE](#) is run, will aggregate the status for all threads to the global status and account status, then reset the thread status. If account statistics are not collected but host and user status are, the session status is added to host and user status.

1.1.2.9.2.1.70 Performance Schema table_io_waits_summary_by_index_usage Table

The `table_io_waits_summary_by_index_usage` table records table I/O waits by index.

Column	Description
OBJECT_TYPE	<code>TABLE</code> in the case of all indexes.

OBJECT_SCHEMA	Schema name.
OBJECT_NAME	Table name.
INDEX_NAME	Index name, or PRIMARY for the primary index, NULL for no index (inserts are counted in this case).
COUNT_STAR	Number of summarized events and the sum of the x_READ and x_WRITE columns.
SUM_TIMER_WAIT	Total wait time of the summarized events that are timed.
MIN_TIMER_WAIT	Minimum wait time of the summarized events that are timed.
AVG_TIMER_WAIT	Average wait time of the summarized events that are timed.
MAX_TIMER_WAIT	Maximum wait time of the summarized events that are timed.
COUNT_READ	Number of all read operations, and the sum of the equivalent x_FETCH columns.
SUM_TIMER_READ	Total wait time of all read operations that are timed.
MIN_TIMER_READ	Minimum wait time of all read operations that are timed.
AVG_TIMER_READ	Average wait time of all read operations that are timed.
MAX_TIMER_READ	Maximum wait time of all read operations that are timed.
COUNT_WRITE	Number of all write operations, and the sum of the equivalent x_INSERT , x_UPDATE and x_DELETE columns.
SUM_TIMER_WRITE	Total wait time of all write operations that are timed.
MIN_TIMER_WRITE	Minimum wait time of all write operations that are timed.
AVG_TIMER_WRITE	Average wait time of all write operations that are timed.
MAX_TIMER_WRITE	Maximum wait time of all write operations that are timed.
COUNT_FETCH	Number of all fetch operations.
SUM_TIMER_FETCH	Total wait time of all fetch operations that are timed.
MIN_TIMER_FETCH	Minimum wait time of all fetch operations that are timed.
AVG_TIMER_FETCH	Average wait time of all fetch operations that are timed.
MAX_TIMER_FETCH	Maximum wait time of all fetch operations that are timed.
COUNT_INSERT	Number of all insert operations.
SUM_TIMER_INSERT	Total wait time of all insert operations that are timed.
MIN_TIMER_INSERT	Minimum wait time of all insert operations that are timed.
AVG_TIMER_INSERT	Average wait time of all insert operations that are timed.
MAX_TIMER_INSERT	Maximum wait time of all insert operations that are timed.
COUNT_UPDATE	Number of all update operations.
SUM_TIMER_UPDATE	Total wait time of all update operations that are timed.
MIN_TIMER_UPDATE	Minimum wait time of all update operations that are timed.
AVG_TIMER_UPDATE	Average wait time of all update operations that are timed.
MAX_TIMER_UPDATE	Maximum wait time of all update operations that are timed.
COUNT_DELETE	Number of all delete operations.
SUM_TIMER_DELETE	Total wait time of all delete operations that are timed.
MIN_TIMER_DELETE	Minimum wait time of all delete operations that are timed.
AVG_TIMER_DELETE	Average wait time of all delete operations that are timed.
MAX_TIMER_DELETE	Maximum wait time of all delete operations that are timed.

You can [TRUNCATE](#) the table, which will reset all counters to zero. The table is also truncated if the [table_io_waits_summary_by_table](#) table is truncated.

If a table's index structure is changed, index statistics recorded in this table may also be reset.

1.1.2.9.2.1.71 Performance Schema table_io_waits_summary_by_table Table

The `table_io_waits_summary_by_table` table records table I/O waits by table.

Column	Description
OBJECT_TYPE	Since this table records waits by table, always set to <code>TABLE</code> .
OBJECT_SCHEMA	Schema name.
OBJECT_NAME	Table name.
COUNT_STAR	Number of summarized events and the sum of the <code>x_READ</code> and <code>x_WRITE</code> columns.
SUM_TIMER_WAIT	Total wait time of the summarized events that are timed.
MIN_TIMER_WAIT	Minimum wait time of the summarized events that are timed.
AVG_TIMER_WAIT	Average wait time of the summarized events that are timed.
MAX_TIMER_WAIT	Maximum wait time of the summarized events that are timed.
COUNT_READ	Number of all read operations, and the sum of the equivalent <code>x_FETCH</code> columns.
SUM_TIMER_READ	Total wait time of all read operations that are timed.
MIN_TIMER_READ	Minimum wait time of all read operations that are timed.
AVG_TIMER_READ	Average wait time of all read operations that are timed.
MAX_TIMER_READ	Maximum wait time of all read operations that are timed.
COUNT_WRITE	Number of all write operations, and the sum of the equivalent <code>x_INSERT</code> , <code>x_UPDATE</code> and <code>x_DELETE</code> columns.
SUM_TIMER_WRITE	Total wait time of all write operations that are timed.
MIN_TIMER_WRITE	Minimum wait time of all write operations that are timed.
AVG_TIMER_WRITE	Average wait time of all write operations that are timed.
MAX_TIMER_WRITE	Maximum wait time of all write operations that are timed.
COUNT_FETCH	Number of all fetch operations.
SUM_TIMER_FETCH	Total wait time of all fetch operations that are timed.
MIN_TIMER_FETCH	Minimum wait time of all fetch operations that are timed.
AVG_TIMER_FETCH	Average wait time of all fetch operations that are timed.
MAX_TIMER_FETCH	Maximum wait time of all fetch operations that are timed.
COUNT_INSERT	Number of all insert operations.
SUM_TIMER_INSERT	Total wait time of all insert operations that are timed.
MIN_TIMER_INSERT	Minimum wait time of all insert operations that are timed.
AVG_TIMER_INSERT	Average wait time of all insert operations that are timed.
MAX_TIMER_INSERT	Maximum wait time of all insert operations that are timed.
COUNT_UPDATE	Number of all update operations.
SUM_TIMER_UPDATE	Total wait time of all update operations that are timed.
MIN_TIMER_UPDATE	Minimum wait time of all update operations that are timed.
AVG_TIMER_UPDATE	Average wait time of all update operations that are timed.
MAX_TIMER_UPDATE	Maximum wait time of all update operations that are timed.
COUNT_DELETE	Number of all delete operations.
SUM_TIMER_DELETE	Total wait time of all delete operations that are timed.

MIN_TIMER_DELETE	Minimum wait time of all delete operations that are timed.
AVG_TIMER_DELETE	Average wait time of all delete operations that are timed.
MAX_TIMER_DELETE	Maximum wait time of all delete operations that are timed.

You can [TRUNCATE](#) the table, which will reset all counters to zero. Truncating this table will also truncate the [table_io_waits_summary_by_index_usage](#) table.

1.1.2.9.2.1.72 Performance Schema table_lock_waits_summary_by_table Table

The `table_lock_waits_summary_by_table` table records table lock waits by table.

Column	Description
OBJECT_TYPE	Since this table records waits by table, always set to TABLE .
OBJECT_SCHEMA	Schema name.
OBJECT_NAME	Table name.
COUNT_STAR	Number of summarized events and the sum of the <code>x_READ</code> and <code>x_WRITE</code> columns.
SUM_TIMER_WAIT	Total wait time of the summarized events that are timed.
MIN_TIMER_WAIT	Minimum wait time of the summarized events that are timed.
AVG_TIMER_WAIT	Average wait time of the summarized events that are timed.
MAX_TIMER_WAIT	Maximum wait time of the summarized events that are timed.
COUNT_READ	Number of all read operations, and the sum of the equivalent <code>x_READ_NORMAL</code> , <code>x_READ_WITH_SHARED_LOCKS</code> , <code>x_READ_HIGH_PRIORITY</code> and <code>x_READ_NO_INSERT</code> columns.
SUM_TIMER_READ	Total wait time of all read operations that are timed.
MIN_TIMER_READ	Minimum wait time of all read operations that are timed.
AVG_TIMER_READ	Average wait time of all read operations that are timed.
MAX_TIMER_READ	Maximum wait time of all read operations that are timed.
COUNT_WRITE	Number of all write operations, and the sum of the equivalent <code>x_WRITE_ALLOW_WRITE</code> , <code>x_WRITE_CONCURRENT_INSERT</code> , <code>x_WRITE_DELAYED</code> , <code>x_WRITE_LOW_PRIORITY</code> and <code>x_WRITE_NORMAL</code> columns.
SUM_TIMER_WRITE	Total wait time of all write operations that are timed.
MIN_TIMER_WRITE	Minimum wait time of all write operations that are timed.
AVG_TIMER_WRITE	Average wait time of all write operations that are timed.
MAX_TIMER_WRITE	Maximum wait time of all write operations that are timed.
COUNT_READ_NORMAL	Number of all internal read normal locks.
SUM_TIMER_READ_NORMAL	Total wait time of all internal read normal locks that are timed.
MIN_TIMER_READ_NORMAL	Minimum wait time of all internal read normal locks that are timed.
AVG_TIMER_READ_NORMAL	Average wait time of all internal read normal locks that are timed.
MAX_TIMER_READ_NORMAL	Maximum wait time of all internal read normal locks that are timed.
COUNT_READ_WITH_SHARED_LOCKS	Number of all internal read with shared locks.
SUM_TIMER_READ_WITH_SHARED_LOCKS	Total wait time of all internal read with shared locks that are timed.
MIN_TIMER_READ_WITH_SHARED_LOCKS	Minimum wait time of all internal read with shared locks that are timed.
AVG_TIMER_READ_WITH_SHARED_LOCKS	Average wait time of all internal read with shared locks that are timed.
MAX_TIMER_READ_WITH_SHARED_LOCKS	Maximum wait time of all internal read with shared locks that are timed.
COUNT_READ_HIGH_PRIORITY	Number of all internal read high priority locks.
SUM_TIMER_READ_HIGH_PRIORITY	Total wait time of all internal read high priority locks that are timed.

MIN_TIMER_READ_HIGH_PRIORITY	Minimum wait time of all internal read high priority locks that are timed.
AVG_TIMER_READ_HIGH_PRIORITY	Average wait time of all internal read high priority locks that are timed.
MAX_TIMER_READ_HIGH_PRIORITY	Maximum wait time of all internal read high priority locks that are timed.
COUNT_READ_NO_INSERT	Number of all internal read no insert locks.
SUM_TIMER_READ_NO_INSERT	Total wait time of all internal read no insert locks that are timed.
MIN_TIMER_READ_NO_INSERT	Minimum wait time of all internal read no insert locks that are timed.
AVG_TIMER_READ_NO_INSERT	Average wait time of all internal read no insert locks that are timed.
MAX_TIMER_READ_NO_INSERT	Maximum wait time of all internal read no insert locks that are timed.
COUNT_READ_EXTERNAL	Number of all external read locks.
SUM_TIMER_READ_EXTERNAL	Total wait time of all external read locks that are timed.
MIN_TIMER_READ_EXTERNAL	Minimum wait time of all external read locks that are timed.
AVG_TIMER_READ_EXTERNAL	Average wait time of all external read locks that are timed.
MAX_TIMER_READ_EXTERNAL	Maximum wait time of all external read locks that are timed.
COUNT_WRITE_ALLOW_WRITE	Number of all internal write allow write locks.
SUM_TIMER_WRITE_ALLOW_WRITE	Total wait time of all internal write allow write locks that are timed.
MIN_TIMER_WRITE_ALLOW_WRITE	Minimum wait time of all internal write allow write locks that are timed.
AVG_TIMER_WRITE_ALLOW_WRITE	Average wait time of all internal write allow write locks that are timed.
MAX_TIMER_WRITE_ALLOW_WRITE	Maximum wait time of all internal write allow write locks that are timed.
COUNT_WRITE_CONCURRENT_INSERT	Number of all internal concurrent insert write locks.
SUM_TIMER_WRITE_CONCURRENT_INSERT	Total wait time of all internal concurrent insert write locks that are timed.
MIN_TIMER_WRITE_CONCURRENT_INSERT	Minimum wait time of all internal concurrent insert write locks that are timed.
AVG_TIMER_WRITE_CONCURRENT_INSERT	Average wait time of all internal concurrent insert write locks that are timed.
MAX_TIMER_WRITE_CONCURRENT_INSERT	Maximum wait time of all internal concurrent insert write locks that are timed.
COUNT_WRITE_DELAYED	Number of all internal write delayed locks.
SUM_TIMER_WRITE_DELAYED	Total wait time of all internal write delayed locks that are timed.
MIN_TIMER_WRITE_DELAYED	Minimum wait time of all internal write delayed locks that are timed.
AVG_TIMER_WRITE_DELAYED	Average wait time of all internal write delayed locks that are timed.
MAX_TIMER_WRITE_DELAYED	Maximum wait time of all internal write delayed locks that are timed.
COUNT_WRITE_LOW_PRIORITY	Number of all internal write low priority locks.
SUM_TIMER_WRITE_LOW_PRIORITY	Total wait time of all internal write low priority locks that are timed.
MIN_TIMER_WRITE_LOW_PRIORITY	Minimum wait time of all internal write low priority locks that are timed.
AVG_TIMER_WRITE_LOW_PRIORITY	Average wait time of all internal write low priority locks that are timed.
MAX_TIMER_WRITE_LOW_PRIORITY	Maximum wait time of all internal write low priority locks that are timed.
COUNT_WRITE_NORMAL	Number of all internal write normal locks.
SUM_TIMER_WRITE_NORMAL	Total wait time of all internal write normal locks that are timed.
MIN_TIMER_WRITE_NORMAL	Minimum wait time of all internal write normal locks that are timed.
AVG_TIMER_WRITE_NORMAL	Average wait time of all internal write normal locks that are timed.
MAX_TIMER_WRITE_NORMAL	Maximum wait time of all internal write normal locks that are timed.
COUNT_WRITE_EXTERNAL	Number of all external write locks.
SUM_TIMER_WRITE_EXTERNAL	Total wait time of all external write locks that are timed.
MIN_TIMER_WRITE_EXTERNAL	Minimum wait time of all external write locks that are timed.

AVG_TIMER_WRITE_EXTERNAL	Average wait time of all external write locks that are timed.
MAX_TIMER_WRITE_EXTERNAL	Maximum wait time of all external write locks that are timed.

You can [TRUNCATE](#) the table, which will reset all counters to zero.

1.1.2.9.2.1.73 Performance Schema threads Table

Each server thread is represented as a row in the `threads` table.

The `threads` table contains the following columns:

Column	Description	Added
THREAD_ID	A unique thread identifier.	MariaDB 5.5
NAME	Name associated with the server's thread instrumentation code, for example <code>thread/sql/main</code> for the server's <code>main()</code> function, and <code>thread/sql/one_connection</code> for a user connection.	MariaDB 5.5
TYPE	<code>FOREGROUND</code> or <code>BACKGROUND</code> , depending on the thread type. User connection threads are <code>FOREGROUND</code> , internal server threads are <code>BACKGROUND</code> .	MariaDB 10.0
PROCESSLIST_ID	The <code>PROCESSLIST.ID</code> value for threads displayed in the <code>INFORMATION_SCHEMA.PROCESSLIST</code> table, or <code>0</code> for background threads. Also corresponds with the <code>CONNECTION_ID()</code> return value for the thread.	MariaDB 5.5
PROCESSLIST_USER	Foreground thread user, or <code>NULL</code> for a background thread.	MariaDB 10.0
PROCESSLIST_HOST	Foreground thread host, or <code>NULL</code> for a background thread.	MariaDB 10.0
PROCESSLIST_DB	Thread's default database, or <code>NULL</code> if none exists.	MariaDB 10.0
PROCESSLIST_COMMAND	Type of command executed by the thread. These correspond to the the <code>COM_xxx</code> client/server protocol commands, and the <code>Com_xxx</code> status variables. See Thread Command Values .	MariaDB 10.0
PROCESSLIST_TIME	Time in seconds the thread has been in its current state.	MariaDB 10.0
PROCESSLIST_STATE	Action, event or state indicating what the thread is doing.	MariaDB 10.0
PROCESSLIST_INFO	Statement being executed by the thread, or <code>NULL</code> if a statement is not being executed. If a statement results in calling other statements, such as for a stored procedure , the innermost statement from the stored procedure is shown here.	MariaDB 10.0
PARENT_THREAD_ID	<code>THREAD_ID</code> of the parent thread, if any. Subthreads can for example be spawned as a result of INSERT DELAYED statements.	MariaDB 10.0
ROLE	Unused.	MariaDB 10.0
INSTRUMENTED	YES or NO for Whether the thread is instrumented or not. For foreground threads, the initial value is determined by whether there's a user/host match in the <code>setup_actors</code> table. Subthreads are again matched, while for background threads, this will be set to YES by default. To monitor events that the thread executes, <code>INSTRUMENTED</code> must be YES and the <code>thread_instrumentation</code> consumer in the <code>setup_consumers</code> table must also be YES .	MariaDB 10.0
HISTORY	YES or NO for Whether to log historical events for the thread. For foreground threads, the initial value is determined by whether there's a user/host match in the <code>setup_actors</code> table. Subthreads are again matched, while for background threads, this will be set to YES by default. To monitor events that the thread executes, <code>INSTRUMENTED</code> must be YES and the <code>thread_instrumentation</code> consumer in the <code>setup_consumers</code> table must also be YES .	MariaDB 10.5
CONNECTION_TYPE	The protocol used to establish the connection, or <code>NULL</code> for background threads.	MariaDB 10.5
THREAD_OS_ID	The thread or task identifier as defined by the underlying operating system, if there is one.	MariaDB 10.5

Example

```
SELECT * FROM performance_schema.threads\G;
*****
THREAD_ID: 1
    NAME: thread/sql/main
    TYPE: BACKGROUND
PROCESSLIST_ID: NULL
PROCESSLIST_USER: NULL
PROCESSLIST_HOST: NULL
PROCESSLIST_DB: NULL
PROCESSLIST_COMMAND: NULL
PROCESSLIST_TIME: 215859
PROCESSLIST_STATE: Table lock
PROCESSLIST_INFO: INTERNAL DDL LOG RECOVER IN PROGRESS
PARENT_THREAD_ID: NULL
    ROLE: NULL
    INSTRUMENTED: YES
...
*****
THREAD_ID: 64
    NAME: thread/sql/one_connection
    TYPE: FOREGROUND
PROCESSLIST_ID: 44
PROCESSLIST_USER: root
PROCESSLIST_HOST: localhost
PROCESSLIST_DB: NULL
PROCESSLIST_COMMAND: Query
PROCESSLIST_TIME: 0
PROCESSLIST_STATE: Sending data
PROCESSLIST_INFO: SELECT * FROM performance_schema.threads
PARENT_THREAD_ID: NULL
    ROLE: NULL
    INSTRUMENTED: YES
```

1.1.2.9.2.1.74 Performance Schema users Table

Description

Each user that connects to the server is stored as a row in the `users` table, along with current and total connections.

The table size is determined at startup by the value of the `performance_schema_users_size` system variable. If this is set to `0`, user statistics will be disabled.

Column	Description
USER	The connection's client user name for the connection, or <code>NULL</code> if an internal thread.
CURRENT_CONNECTIONS	Current connections for the user.
TOTAL_CONNECTIONS	Total connections for the user.

Example

```
SELECT * FROM performance_schema.users;
+-----+-----+-----+
| USER      | CURRENT_CONNECTIONS | TOTAL_CONNECTIONS |
+-----+-----+-----+
| debian-sys-maint |          0 |          35 |
| NULL        |         20 |          23 |
| root        |          1 |           2 |
+-----+-----+-----+
```

1.1.2.9.2.2 Performance Schema Overview

The Performance Schema is a feature for monitoring server performance.

Contents

1. [Introduction](#)
2. [Activating the Performance Schema](#)
3. [Enabling the Performance Schema](#)
4. [Listing Performance Schema Variables](#)
5. [Column Comments](#)
6. [See Also](#)

Introduction

It is implemented as a storage engine, and so will appear in the list of storage engines available.

SHOW ENGINES;				
Engine	Support	Comment	Transactions	XA
...				
PERFORMANCE_SCHEMA	YES	Performance Schema	NO	NO
...				

However, `performance_schema` is not a regular storage engine for storing data, it's a mechanism for implementing the Performance Schema feature.

The storage engine contains a database called `performance_schema`, which in turn consists of a number of tables that can be queried with regular SQL statements, returning specific performance information.

```
USE performance_schema
```

SHOW TABLES;	
Tables_in_performance_schema	
accounts	
...	
users	
80 rows in set (0.00 sec)	

See [List of Performance Schema Tables](#) for a full list and links to detailed descriptions of each table. From MariaDB 10.5, there are 80 Performance Schema tables, while until MariaDB 10.4, there are 52.

Activating the Performance Schema

The performance schema is disabled by default for performance reasons. You can check its current status by looking at the value of the `performance_schema` system variable.

SHOW VARIABLES LIKE 'performance_schema';	
Variable_name	Value
performance_schema	ON

The performance schema cannot be activated at runtime - it must be set when the server starts by adding the following line in your `my.cnf` configuration file.

```
performance_schema=ON
```

Until MariaDB 10.4, all memory used by the Performance Schema is allocated at startup. From MariaDB 10.5, some memory is allocated dynamically, depending on load, number of connections, number of tables open etc.

Enabling the Performance Schema

You need to set up all consumers (starting collection of data) and instrumentations (what to collect):

```
UPDATE performance_schema.setup_consumers SET ENABLED = 'YES';
UPDATE performance_schema.setup_instruments SET ENABLED = 'YES', TIMED = 'YES';
```

You can decide what to enable/disable with `WHERE NAME like "%what_to_enable"`; You can disable instrumentations by setting `ENABLED` to "NO".

You can also do this in your my.cnf file. The following enables all instrumentation of all stages (computation units) in MariaDB:

```
[mysqld]
performance_schema=ON
performance-schema-instrument='stage/%=ON'
performance-schema-consumer-events-stages-current=ON
performance-schema-consumer-events-stages-history=ON
performance-schema-consumer-events-stages-history-long=ON
```

List Performance Schema Variables

```
SHOW VARIABLES LIKE "perf%";
```

Variable_name	Value
performance_schema	ON
...	
performance_schema_users_size	100

See [Performance Schema System Variables](#) for a full list of available system variables.

Note that the "consumer" events are not shown on this list, as they are only available as options, not as system variables, and they can only be enabled at [startup](#).

Column Comments

MariaDB starting with [10.7.1](#)

From [MariaDB 10.7.1](#), comments have been added to table columns in the Performance Schema. These can be viewed with, for example:

```
SELECT column_name, column_comment FROM information_schema.columns
  WHERE table_schema='performance_schema' AND table_name='file_instances';
...
***** 2. row *****
  column_name: EVENT_NAME
column_comment: Instrument name associated with the file.
***** 3. row *****
  column_name: OPEN_COUNT
column_comment: Open handles on the file. A value of greater than zero means
                that the file is currently open.
...
```

See Also

- [Performance schema options](#)
- [SHOW ENGINE STATUS](#)
- [SHOW PROFILE](#)
- [ANALYZE STATEMENT](#)
- [Performance schema in MySQL 5.6](#). All things here should also work for MariaDB.

1.1.2.9.2.3 Performance Schema Status Variables

This page documents status variables related to the [Performance Schema](#). See [Server Status Variables](#) for a complete list of status variables that can be viewed with [SHOW STATUS](#).

See also the [Full list of MariaDB options, system and status variables](#).

Performance_schema_accounts_lost

- **Description:** Number of times a row could not be added to the performance schema accounts table due to it being full. The global value can be flushed by [FLUSH STATUS](#).
- **Scope:** Global, Session
- **Data Type:** numeric

Performance_schema_cond_classes_lost

- **Description:** Number of condition instruments that could not be loaded.
- **Scope:** Global, Session
- **Data Type:** numeric

Performance_schema_cond_instances_lost

- **Description:** Number of instances a condition object could not be created. The global value can be flushed by [FLUSH STATUS](#).
- **Scope:** Global, Session
- **Data Type:** numeric

Performance_schema_digest_lost

- **Description:** The global value can be flushed by [FLUSH STATUS](#).
- **Scope:** Global, Session
- **Data Type:** numeric

Performance_schema_file_classes_lost

- **Description:** Number of file instruments that could not be loaded.
- **Scope:** Global, Session
- **Data Type:** numeric

Performance_schema_file_handles_lost

- **Description:** Number of instances a file object could not be opened. The global value can be flushed by [FLUSH STATUS](#).
- **Scope:** Global, Session
- **Data Type:** numeric

Performance_schema_file_instances_lost

- **Description:** Number of instances a file object could not be created. The global value can be flushed by [FLUSH STATUS](#).
- **Scope:** Global, Session
- **Data Type:** numeric

Performance_schema_hosts_lost

- **Description:** Number of times a row could not be added to the performance schema hosts table due to it being full. The global value can be flushed by [FLUSH STATUS](#).
- **Scope:** Global, Session
- **Data Type:** numeric

Performance_schema_index_stat_lost

Contents

1. [Performance_schema_accounts_lost](#)
2. [Performance_schema_cond_classes_lost](#)
3. [Performance_schema_cond_instances_lost](#)
4. [Performance_schema_digest_lost](#)
5. [Performance_schema_file_classes_lost](#)
6. [Performance_schema_file_handles_lost](#)
7. [Performance_schema_file_instances_lost](#)
8. [Performance_schema_hosts_lost](#)
9. [Performance_schema_index_stat_lost](#)
10. [Performance_schema_locker_lost](#)
11. [Performance_schema_memory_classes_lost](#)
12. [Performance_schema_metadata_lock_lost](#)
13. [Performance_schema_mutex_classes_lost](#)
14. [Performance_schema_mutex_instances_lost](#)
15. [Performance_schema_nested_statement_instances_lost](#)
16. [Performance_schema_prepared_statement_instances_lost](#)
17. [Performance_schema_program_lost](#)
18. [Performance_schema_rwlock_classes_lost](#)
19. [Performance_schema_rwlock_instances_lost](#)
20. [Performance_schema_session_connect_at_instances_lost](#)
21. [Performance_schema_socket_classes_lost](#)
22. [Performance_schema_socket_instances_lost](#)
23. [Performance_schema_stage_classes_lost](#)
24. [Performance_schema_statement_classes_lost](#)
25. [Performance_schema_table_handles_lost](#)
26. [Performance_schema_table_instances_lost](#)
27. [Performance_schema_table_lock_stat_lost](#)
28. [Performance_schema_thread_classes_lost](#)
29. [Performance_schema_thread_instances_lost](#)
30. [Performance_schema_users_lost](#)

- **Description:**
 - **Scope:** Global, Session
 - **Data Type:** numeric
 - **Introduced:** MariaDB 10.5.2
-

Performance_schema_locker_lost

- **Description:** Number of events not recorded, due to either being recursive, or having a deeper nested events stack than the implementation limit. The global value can be flushed by [FLUSH STATUS](#).
 - **Scope:** Global, Session
 - **Data Type:** numeric
-

Performance_schema_memory_classes_lost

- **Description:**
 - **Scope:** Global, Session
 - **Data Type:** numeric
 - **Introduced:** MariaDB 10.5.2
-

Performance_schema_metadata_lock_lost

- **Description:**
 - **Scope:** Global, Session
 - **Data Type:** numeric
 - **Introduced:** MariaDB 10.5.2
-

Performance_schema_mutex_classes_lost

- **Description:** Number of mutual exclusion instruments that could not be loaded.
 - **Scope:** Global, Session
 - **Data Type:** numeric
-

Performance_schema_mutex_instances_lost

- **Description:** Number of instances a mutual exclusion object could not be created. The global value can be flushed by [FLUSH STATUS](#).
 - **Scope:** Global, Session
 - **Data Type:** numeric
-

Performance_schema_nested_statement_lost

- **Description:**
 - **Scope:** Global, Session
 - **Data Type:** numeric
 - **Introduced:** MariaDB 10.5.2
-

Performance_schema_prepared_statements_lost

- **Description:**
 - **Scope:** Global, Session
 - **Data Type:** numeric
 - **Introduced:** MariaDB 10.5.2
-

Performance_schema_program_lost

- **Description:**

- **Scope:** Global, Session
 - **Data Type:** numeric
 - **Introduced:** MariaDB 10.5.2
-

Performance_schema_rwlock_classes_lost

- **Description:** Number of read/write lock instruments that could not be loaded.
 - **Scope:** Global, Session
 - **Data Type:** numeric
-

Performance_schema_rwlock_instances_lost

- **Description:** Number of instances a read/write lock object could not be created. The global value can be flushed by [FLUSH STATUS](#).
 - **Scope:** Global, Session
 - **Data Type:** numeric
-

Performance_schema_session_connect_attrs_lost

- **Description:** Number of connections for which connection attribute truncation has occurred. The global value can be flushed by [FLUSH STATUS](#).
 - **Scope:** Global, Session
 - **Data Type:** numeric
-

Performance_schema_socket_classes_lost

- **Description:**
 - **Scope:** Global, Session
 - **Data Type:** numeric
-

Performance_schema_socket_instances_lost

- **Description:** Number of instances a socket object could not be created. The global value can be flushed by [FLUSH STATUS](#).
 - **Scope:** Global, Session
 - **Data Type:** numeric
-

Performance_schema_stage_classes_lost

- **Description:** Number of stage event instruments that could not be loaded. The global value can be flushed by [FLUSH STATUS](#).
 - **Scope:** Global, Session
 - **Data Type:** numeric
-

Performance_schema_statement_classes_lost

- **Description:** Number of statement instruments that could not be loaded. The global value can be flushed by [FLUSH STATUS](#).
 - **Scope:** Global, Session
 - **Data Type:** numeric
-

Performance_schema_table_handles_lost

- **Description:** Number of instances a table object could not be opened. The global value can be flushed by [FLUSH STATUS](#).
 - **Scope:** Global, Session
 - **Data Type:** numeric
-

Performance_schema_table_instances_lost

- **Description:** Number of instances a table object could not be created. The global value can be flushed by [FLUSH STATUS](#).
 - **Scope:** Global, Session
 - **Data Type:** numeric
-

Performance_schema_table_lock_stat_lost

- **Description:**
 - **Scope:** Global, Session
 - **Data Type:** numeric
 - **Introduced:** MariaDB 10.5.2
-

Performance_schema_thread_classes_lost

- **Description:** Number of thread instruments that could not be loaded.
 - **Scope:** Global, Session
 - **Data Type:** numeric
-

Performance_schema_thread_instances_lost

- **Description:** Number of instances thread object could not be created. The global value can be flushed by [FLUSH STATUS](#).
 - **Scope:** Global, Session
 - **Data Type:** numeric
-

Performance_schema_users_lost

- **Description:** Number of times a row could not be added to the performance schema users table due to it being full. The global value can be flushed by [FLUSH STATUS](#).
 - **Scope:** Global, Session
 - **Data Type:** numeric
-

1.1.2.9.2.4 Performance Schema System Variables

The following variables are used with MariaDB's [Performance Schema](#). See [Performance Schema Options](#) for Performance Schema options that are not system variables. See [Server System Variables](#) for a complete list of system variables and instructions on setting them.

See also the [Full list of MariaDB options, system and status variables](#).

performance_schema

- **Description:** If set to 1 (0 is default), enables the Performance Schema
 - **Commandline:** --performance-schema=#
 - **Scope:** Global
 - **Dynamic:** No
 - **Data Type:** boolean
 - **Default Value:** OFF
-

performance_schema_accounts_size

- **Description:** Maximum number of rows in the `performance_schema.accounts` table. If set to 0, the [Performance Schema](#) will not store statistics in the accounts table. Use -1 (the default) for automated sizing.
- **Commandline:** --performance-schema-accounts-size=#
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** numeric
- **Default Value:** -1

- **Range:** -1 to 1048576

performance_schema_digests_size

- **Description:** Maximum number of rows that can be stored in the `events_statements_summary_by_digest` table. 0 for disabling, -1 (the default) for automated sizing.
- **Commandline:** --performance-schema-digests-size=#
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** numeric
- **Default Value:** -1
- **Range:** -1 to 200

performance_schema_events_stages_history_long_size

- **Description:** Number of rows in the `events_stages_history_long` table. 0 for disabling, -1 (the default) for automated sizing.
- **Commandline:** --performance-schema-events-stages-history-long-size=#
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** numeric
- **Default Value:** -1
- **Range:** -1 to 1048576

performance_schema_events_stages_history_size

- **Description:** Number of rows per thread in the `events_stages_history` table. 0 for disabling, -1 (the default) for automated sizing.
- **Commandline:** --performance-schema-events-stages-history-size=#
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** numeric
- **Default Value:** -1
- **Range:** -1 to 1024

performance_schema_events_statements_history_long_size

- **Description:** Number of rows in the `events_statements_history_long` table. 0 for disabling, -1 (the default) for automated sizing.
- **Commandline:** --performance-schema-events-statements-history-long-size=#
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** numeric
- **Default Value:** -1
- **Range:** -1 to 1048576

performance_schema_events_statements_history_size

- **Description:** Number of rows per thread in the `events_statements_history` table. 0 for disabling, -1 (the default) for automated sizing.
- **Commandline:** --performance-schema-events-statements-history-size=#
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** numeric
- **Default Value:** -1
- **Range:** -1 to 1024

Contents

1. [performance_schema](#)
2. [performance_schema_accounts_size](#)
3. [performance_schema_digests_size](#)
4. [performance_schema_events_stages_hist](#)
5. [performance_schema_events_stages_histc](#)
6. [performance_schema_events_statements_](#)
7. [performance_schema_events_statements_](#)
8. [performance_schema_events_transactions](#)
9. [performance_schema_events_transactions](#)
10. [performance_schema_events_waits_hist](#)
11. [performance_schema_events_waits_histc](#)
12. [performance_schema_hosts_size](#)
13. [performance_schema_max_cond_classes](#)
14. [performance_schema_max_cond_instance](#)
15. [performance_schema_max_digest_length](#)
16. [performance_schema_max_file_classes](#)
17. [performance_schema_max_file_handles](#)
18. [performance_schema_max_file_instances](#)
19. [performance_schema_max_index_stat](#)
20. [performance_schema_max_memory_classe](#)
21. [performance_schema_max_metadata_lock](#)
22. [performance_schema_max_mutex_classes](#)
23. [performance_schema_max_mutex_instance](#)
24. [performance_schema_max_prepared_state](#)
25. [performance_schema_max_program_instanc](#)
26. [performance_schema_max_rwlock_classes](#)
27. [performance_schema_max_rwlock_instances](#)
28. [performance_schema_max_socket_classes](#)
29. [performance_schema_max_socket_instances](#)
30. [performance_schema_max_sql_text_length](#)
31. [performance_schema_max_stage_classes](#)
32. [performance_schema_max_statement_clas](#)
33. [performance_schema_max_statement_stac](#)
34. [performance_schema_max_table_handles](#)
35. [performance_schema_max_table_instance](#)
36. [performance_schema_max_table_lock_stat](#)
37. [performance_schema_max_thread_classes](#)
38. [performance_schema_max_thread_instances](#)
39. [performance_schema_session_connect_at](#)
40. [performance_schema_setup_actors_size](#)
41. [performance_schema_setup_objects_size](#)
42. [performance_schema_users_size](#)

performance_schema_events_transactions_history_long_size

- **Description:** Number of rows in [events_transactions_history_long](#) table. Use 0 to disable, -1 for automated sizing.
 - **Commandline:** --performance-schema-events-transactions-history-long-size=#
 - **Scope:** Global
 - **Dynamic:** No
 - **Data Type:** numeric
 - **Default Value:** -1
 - **Range:** -1 to 1048576
 - **Introduced:** MariaDB 10.5.2
-

performance_schema_events_transactions_history_size

- **Description:** Number of rows per thread in [events_transactions_history](#). Use 0 to disable, -1 for automated sizing.
 - **Commandline:** --performance-schema-events-transactions-history-size=#
 - **Scope:** Global
 - **Dynamic:** No
 - **Data Type:** numeric
 - **Default Value:** -1
 - **Range:** -1 to 1024
 - **Introduced:** MariaDB 10.5.2
-

performance_schema_events_waits_history_long_size

- **Description:** Number of rows contained in the [events_waits_history_long](#) table. 0 for disabling, -1 (the default) for automated sizing.
 - **Commandline:** --performance-schema-events-waits-history-long-size=#
 - **Scope:** Global
 - **Dynamic:** No
 - **Data Type:** numeric
 - **Default Value:** -1
 - **Range:** -1 to 1048576
-

performance_schema_events_waits_history_size

- **Description:** Number of rows per thread contained in the [events_waits_history](#) table. 0 for disabling, -1 (the default) for automated sizing.
 - **Commandline:** --performance-schema-events-waits-history-size=#
 - **Scope:** Global
 - **Dynamic:** No
 - **Data Type:** numeric
 - **Default Value:** -1
 - **Range:** -1 to 1024
-

performance_schema_hosts_size

- **Description:** Number of rows stored in the [hosts](#) table. If set to zero, no connection statistics are kept for the hosts table. -1 (the default) for automated sizing.
 - **Commandline:** --performance-schema-hosts-size=#
 - **Scope:** Global
 - **Dynamic:** No
 - **Data Type:** numeric
 - **Default Value:** -1
 - **Range:** -1 to 1048576
-

performance_schema_max_cond_classes

- **Description:** Specifies the maximum number of condition instruments.
- **Commandline:** --performance-schema-max-cond-classes=#
- **Scope:** Global

- **Dynamic:** No
 - **Data Type:** numeric
 - **Default Value:** 90 (>= MariaDB 10.5.1), 80 (<= MariaDB 10.5.0)
 - **Range:** 0 to 256
-

performance_schema_max_cond_instances

- **Description:** Specifies the maximum number of instrumented condition objects. 0 for disabling, -1 (the default) for automated sizing.
 - **Commandline:** --performance-schema-max-cond-instances=#
 - **Scope:** Global
 - **Dynamic:** No
 - **Data Type:** numeric
 - **Default Value:** -1
 - **Range:** -1 to 1048576
-

performance_schema_max_digest_length

- **Description:** Maximum length considered for digest text, when stored in performance_schema tables.
 - **Commandline:** --performance-schema-max-digest-length=#
 - **Scope:** Global
 - **Dynamic:** No
 - **Data Type:** numeric
 - **Default Value:** 1024
 - **Range:** 0 to 1048576
-

performance_schema_max_file_classes

- **Description:** Specifies the maximum number of file instruments.
 - **Commandline:** --performance-schema-max-file-classes=#
 - **Scope:** Global
 - **Dynamic:** No
 - **Data Type:** numeric
 - **Default Value:** 80 (>= MariaDB 10.5.2), 50 (<= MariaDB 10.5.1)
 - **Range:** 0 to 256
-

performance_schema_max_file_handles

- **Description:** Specifies the maximum number of opened file objects. Should always be higher than [open_files_limit](#).
 - **Commandline:** --performance-schema-max-file-handles=#
 - **Scope:** Global
 - **Dynamic:** No
 - **Data Type:** numeric
 - **Default Value:** 32768
 - **Range:** -1 to 32768
-

performance_schema_max_file_instances

- **Description:** Specifies the maximum number of instrumented file objects. 0 for disabling, -1 (the default) for automated sizing.
 - **Commandline:** --performance-schema-max-file-instances=#
 - **Scope:** Global
 - **Dynamic:** No
 - **Data Type:** numeric
 - **Default Value:** -1
 - **Range:** -1 to 1048576
-

performance_schema_max_index_stat

- **Description:** Maximum number of index statistics for instrumented tables. Use 0 to disable, -1 for automated scaling.
 - **Commandline:** --performance-schema-max-index-stat=#
 - **Scope:** Global
 - **Dynamic:** No
 - **Data Type:** numeric
 - **Default Value:** -1
 - **Range:** -1 to 1048576
 - **Introduced:** MariaDB 10.5.2
-

performance_schema_max_memory_classes

- **Description:** Maximum number of memory pool instruments.
 - **Commandline:** --performance-schema-max-memory-classes=#
 - **Scope:** Global
 - **Dynamic:** No
 - **Data Type:** numeric
 - **Default Value:** 320
 - **Range:** 0 to 1024
 - **Introduced:** MariaDB 10.5.2
-

performance_schema_max_metadata_locks

- **Description:** Maximum number of [Performance Schema metadata locks](#). Use 0 to disable, -1 for automated scaling.
 - **Commandline:** --performance-schema-max-metadata-locks=#
 - **Scope:** Global
 - **Dynamic:** No
 - **Data Type:** numeric
 - **Default Value:** -1
 - **Range:** -1 to 104857600
 - **Introduced:** MariaDB 10.5.2
-

performance_schema_max_mutex_classes

- **Description:** Specifies the maximum number of mutex instruments.
 - **Commandline:** --performance-schema-max-mutex-classes=#
 - **Scope:** Global
 - **Dynamic:** No
 - **Data Type:** numeric
 - **Default Value:** 210 (>= MariaDB 10.5.2), 200 (<= MariaDB 10.5.1)
 - **Range:** 0 to 256
-

performance_schema_max_mutex_instances

- **Description:** Specifies the maximum number of instrumented mutex instances. 0 for disabling, -1 (the default) for automated sizing.
 - **Commandline:** --performance-schema-max-mutex-instances=#
 - **Scope:** Global
 - **Dynamic:** No
 - **Data Type:** numeric
 - **Default Value:** -1
 - **Range:** -1 to 104857600
-

performance_schema_max_prepared_statement_instances

- **Description:** Maximum number of instrumented prepared statements. Use 0 to disable, -1 for automated scaling.
- **Commandline:** --performance-schema-max-prepared-statement-instances=#
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** numeric

- **Default Value:** -1
 - **Range:** -1 to 1048576
 - **Introduced:** MariaDB 10.5.2
-

performance_schema_max_program_instances

- **Description:** Maximum number of instrumented programs. Use 0 to disable, -1 for automated scaling.
 - **Commandline:** --performance-schema-max-program-instances=#
 - **Scope:** Global
 - **Dynamic:** No
 - **Data Type:** numeric
 - **Default Value:** -1
 - **Range:** -1 to 1048576
 - **Introduced:** MariaDB 10.5.2
-

performance_schema_max_rwlock_classes

- **Description:** Specifies the maximum number of rwlock instruments.
 - **Commandline:** --performance-schema-max-rwlock-classes=#
 - **Scope:** Global
 - **Dynamic:** No
 - **Data Type:** numeric
 - **Default Value:** 50 (>= MariaDB 10.5.2), 40 (<= MariaDB 10.5.1)
 - **Range:** 0 to 256
-

performance_schema_max_rwlock_instances

- **Description:** Specifies the maximum number of instrumented rwlock objects. 0 for disabling, -1 (the default) for automated sizing.
 - **Commandline:** --performance-schema-max-rwlock-instances=#
 - **Scope:** Global
 - **Dynamic:** No
 - **Data Type:** numeric
 - **Default Value:** -1
 - **Range:** -1 to 104857600
-

performance_schema_max_socket_classes

- **Description:** Specifies the maximum number of socket instruments.
 - **Commandline:** --performance-schema-max-socket-classes=#
 - **Scope:** Global
 - **Dynamic:** No
 - **Data Type:** numeric
 - **Default Value:** 10
 - **Range:** 0 to 256
-

performance_schema_max_socket_instances

- **Description:** Specifies the maximum number of instrumented socket objects. 0 for disabling, -1 (the default) for automated sizing.
 - **Commandline:** --performance-schema-max-socket-instances=#
 - **Scope:** Global
 - **Dynamic:** No
 - **Data Type:** numeric
 - **Default Value:** -1
 - **Range:** -1 to 1048576
-

performance_schema_max_sql_text_length

- **Description:** Maximum length of displayed sql text.
 - **Commandline:** --performance-schema-max-sql-text-length=#
 - **Scope:** Global
 - **Dynamic:** No
 - **Data Type:** numeric
 - **Default Value:** 1024
 - **Range:** 0 to 1048576
 - **Introduced:** MariaDB 10.5.2
-

performance_schema_max_stage_classes

- **Description:** Specifies the maximum number of stage instruments.
 - **Commandline:** --performance-schema-max-stage-classes=#
 - **Scope:** Global
 - **Dynamic:** No
 - **Data Type:** numeric
 - **Default Value:** 160 (>= MariaDB 10.3.3), 150 (<= MariaDB 10.3.2)
 - **Range:** 0 to 256
-

performance_schema_max_statement_classes

- **Description:** Specifies the maximum number of statement instruments. Automatically calculated at server build based on the number of available statements. Should be left as either autosized or disabled, as changing to any positive value has no benefit and will most likely allocate unnecessary memory. Setting to zero disables all statement instrumentation, and no memory will be allocated for this purpose.
 - **Commandline:** --performance-schema-max-statement-classes=#
 - **Scope:** Global
 - **Dynamic:** No
 - **Data Type:** numeric
 - **Default Value:** Autosized (see description)
 - **Range:** 0 to 256
-

performance_schema_max_statement_stack

- **Description:** Number of rows per thread in EVENTS_STATEMENTS_CURRENT.
 - **Commandline:** --performance-schema-max-statement-stack=#
 - **Scope:** Global
 - **Dynamic:** No
 - **Data Type:** numeric
 - **Default Value:** 10
 - **Range:** 1 to 256
 - **Introduced:** MariaDB 10.5.2
-

performance_schema_max_table_handles

- **Description:** Specifies the maximum number of opened table objects. 0 for disabling, -1 (the default) for automated sizing.
 - **Commandline:** --performance-schema-max-table-handles=#
 - **Scope:** Global
 - **Dynamic:** No
 - **Data Type:** numeric
 - **Default Value:** -1
 - **Range:** -1 to 1048576
-

performance_schema_max_table_instances

- **Description:** Specifies the maximum number of instrumented table objects. 0 for disabling, -1 (the default) for automated sizing.
- **Commandline:** --performance-schema-max-table-instances=#
- **Scope:** Global
- **Dynamic:** No

- **Data Type:** numeric
 - **Default Value:** -1
 - **Range:** -1 to 1048576
-

performance_schema_max_table_lock_stat

- **Description:** Maximum number of lock statistics for instrumented tables. Use 0 to disable, -1 for automated scaling.
 - **Commandline:** --performance-schema-max-table-lock-stat=#
 - **Scope:** Global
 - **Dynamic:** No
 - **Data Type:** numeric
 - **Default Value:** -1
 - **Range:** -1 to 1048576
 - **Introduced:** MariaDB 10.5.2
-

performance_schema_max_thread_classes

- **Description:** Specifies the maximum number of thread instruments.
 - **Commandline:** --performance-schema-max-thread-classes=#
 - **Scope:** Global
 - **Dynamic:** No
 - **Data Type:** numeric
 - **Default Value:** 50
 - **Range:** 0 to 256
-

performance_schema_max_thread_instances

- **Description:** Specifies how many of the running server threads (see [max_connections](#) and [max_delayed_threads](#)) can be instrumented. Should be greater than the sum of max_connections and max_delayed_threads. 0 for disabling, -1 (the default) for automated sizing.
 - **Commandline:** --performance-schema-max-thread-instances=#
 - **Scope:** Global
 - **Dynamic:** No
 - **Data Type:** numeric
 - **Default Value:** -1
 - **Range:** -1 to 1048576
-

performance_schema_session_connect_attrs_size

- **Description:** Per thread preallocated memory for holding connection attribute strings. Incremented if the strings are larger than the reserved space. 0 for disabling, -1 (the default) for automated sizing.
 - **Commandline:** --performance-schema-session-connect-attrs-size=#
 - **Scope:** Global
 - **Dynamic:** No
 - **Data Type:** numeric
 - **Default Value:** -1
 - **Range:** -1 to 1048576
-

performance_schema_setup_actors_size

- **Description:** The maximum number of rows to store in the performance schema [setup_actors](#) table. -1 (from MariaDB 10.5.2) denotes automated sizing.
 - **Commandline:** --performance-schema-setup-actors-size=#
 - **Scope:** Global
 - **Dynamic:** No
 - **Data Type:** numeric
 - **Default Value:** -1 (>= MariaDB 10.5.2), 100 (<= MariaDB 10.5.1)
 - **Range:** -1 to 1024 (>= MariaDB 10.5.2), 0 to 1024 (<= MariaDB 10.5.1)
-

performance_schema_setup_objects_size

- **Description:** The maximum number of rows that can be stored in the performance schema `setup_objects` table. -1 (from [MariaDB 10.5.2](#)) denotes automated sizing.
- **Commandline:** --performance-schema-setup-objects-size=#
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** numeric
- **Default Value:** -1 (>= [MariaDB 10.5.2](#)), 100 (<= [MariaDB 10.5.1](#))
- **Range:** -1 to 1048576 (>= [MariaDB 10.5.2](#)), 0 to 1048576 (<= [MariaDB 10.5.1](#))

performance_schema_users_size

- **Description:** Number of rows in the `performance_schema.users` table. If set to 0, the [Performance Schema](#) will not store connection statistics in the users table. -1 (the default) for automated sizing.
- **Commandline:** --performance-schema-users-size=#
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** numeric
- **Default Value:** -1
- **Range:** -1 to 1048576

1.1.2.9.2.5 Performance Schema Digests

The Performance Schema digest is a normalized form of a statement, with the specific data values removed. It allows statistics to be gathered for similar kinds of statements.

For example:

```
SELECT * FROM customer WHERE age < 20
SELECT * FROM customer WHERE age < 30
```

With the data values removed, both of these statements normalize to:

```
SELECT * FROM customer WHERE age < ?
```

which is the digest text. The digest text is then MD5 hashed, resulting in a digest. For example:

```
DIGEST_TEXT: SELECT * FROM `performance_schema` . `users`
DIGEST: 0f70cec4015f2a346df4ac0e9475d9f1
```

By contrast, the following two statements would not have the same digest as, while the data values are the same, they call upon different tables.

```
SELECT * FROM customer1 WHERE age < 20
SELECT * FROM customer2 WHERE age < 20
```

The digest text is limited to 1024 bytes. Queries exceeding this limit are truncated with '...', meaning that long queries that would otherwise have different digests may share the same digest.

Digest information is used in a number of performance scheme tables. These include

- `events_statements_current`
- `events_statements_history`
- `events_statements_history_long`
- `events_statements_summary_by_digest` (a summary table by schema and digest)

1.1.2.9.2.6 PERFORMANCE_SCHEMA Storage Engine

If you run `SHOW ENGINES`, you'll see the following storage engine listed.

```
SHOW ENGINES\G
...
  Engine: PERFORMANCE_SCHEMA
  Support: YES
  Comment: Performance Schema
Transactions: NO
  XA: NO
  Savepoints: NO
...
```

The PERFORMANCE_SCHEMA is not a regular storage engine for storing data, it's a mechanism for implementing the [Performance Schema](#) feature.

The [SHOW ENGINE PERFORMANCE_SCHEMA STATUS](#) statement is also available, which shows how much memory is used by the tables and internal buffers.

See [Performance Schema](#) for more details.

1.1.2.9.3 The mysql Database Tables AREA

A synonym for [ST_AREA](#).

CENTROID

A synonym for [ST_CENTROID](#).

ExteriorRing

A synonym for [ST_ExteriorRing](#).

InteriorRingN

A synonym for [ST_InteriorRingN](#).

NumInteriorRings

A synonym for [ST_NumInteriorRings](#).

ST_AREA

Syntax

```
ST_Area(poly)
Area(poly)
```

Description

Returns a double-precision number the area of the Polygon value `poly`, as measured in its spatial reference system.

`ST_Area()` and `Area()` are synonyms.

Examples

```

SET @poly = 'Polygon((0 0,0 3,3 0,0 0),(1 1,1 2,2 1,1 1))';
SELECT Area(GeomFromText(@poly));
+-----+
| Area(GeomFromText(@poly)) |
+-----+
| 4 |
+-----+

```

ST_CENTROID

Syntax

```

ST_Centroid(mpoly)
Centroid(mpoly)

```

Description

Returns a point reflecting the mathematical centroid (geometric center) for the [MultiPolygon](#) *mpoly*. The resulting point will not necessarily be on the MultiPolygon.

`ST_Centroid()` and `Centroid()` are synonyms.

Examples

```

SET @poly = ST_GeomFromText('POLYGON((0 0,20 0,20 20,0 20,0 0))';
SELECT ST_AsText(ST_Centroid(@poly)) AS center;
+-----+
| center      |
+-----+
| POINT(10 10) |
+-----+

```

ST_ExteriorRing

Syntax

```

ST_ExteriorRing(poly)
ExteriorRing(poly)

```

Description

Returns the exterior ring of the Polygon value `poly` as a LineString.

`ST_ExteriorRing()` and `ExteriorRing()` are synonyms.

Examples

```

SET @poly = 'Polygon((0 0,0 3,3 0,0 0),(1 1,1 2,2 2,1,1 1))';
SELECT AsText(ExteriorRing(GeomFromText(@poly)));
+-----+
| AsText(ExteriorRing(GeomFromText(@poly))) |
+-----+
| LINESTRING(0 0,0 3,3 3,3 0,0 0)           |
+-----+

```

ST_InteriorRingN

Syntax

```
ST_InteriorRingN(poly,N)
InteriorRingN(poly,N)
```

Description

Returns the N-th interior ring for the Polygon value `poly` as a LineString. Rings are numbered beginning with 1.

`ST_InteriorRingN()` and `InteriorRingN()` are synonyms.

Examples

```
SET @poly = 'Polygon((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1))';

SELECT AsText(InteriorRingN(GeomFromText(@poly),1));
+-----+
| AsText(InteriorRingN(GeomFromText(@poly),1)) |
+-----+
| LINESTRING(1 1,1 2,2 2,2 1,1 1)             |
+-----+
```

ST_NumInteriorRings

Syntax

```
ST_NumInteriorRings(poly)
NumInteriorRings(poly)
```

Description

Returns an integer containing the number of interior rings in the Polygon value `poly`.

Note that according the the OpenGIS standard, a `POLYGON` should have exactly one `ExteriorRing` and all other rings should lie within that `ExteriorRing` and thus be the `InteriorRings`. Practically, however, some systems, including MariaDB's, permit polygons to have several 'ExteriorRings'. In the case of there being multiple, non-overlapping exterior rings `ST_NumInteriorRings()` will return 1.

`ST_NumInteriorRings()` and `NumInteriorRings()` are synonyms.

Examples

```
SET @poly = 'Polygon((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1))';

SELECT NumInteriorRings(GeomFromText(@poly));
+-----+
| NumInteriorRings(GeomFromText(@poly)) |
+-----+
|                               1 |
```

Non-overlapping 'polygon':

```
SELECT ST_NumInteriorRings(ST_PolyFromText('POLYGON((0 0,10 0,10 10,0 10,0 0),
(-1 -1,-5 -5,-1 -5,-1 -1))')) AS NumInteriorRings;
+-----+
| NumInteriorRings |
+-----+
|          1 |
+-----+
```

WKT Definition

Description

The Well-Known Text (WKT) representation of Geometry is designed to exchange geometry data in ASCII form. Examples of the basic geometry types include:

Geometry Types
POINT
LINESTRING
POLYGON
MULTIPOINT
MULTILINESTRING
MULTIPOLYGON
GEOMETRYCOLLECTION
GEOMETRY

See Also

- [Geometry Types](#)

AsText

A synonym for [ST_AsText\(\)](#).

AsWKT

A synonym for [ST_AsText\(\)](#).

GeomCollFromText

A synonym for [ST_GeomCollFromText](#).

GeometryCollectionFromText

A synonym for [ST_GeomCollFromText](#).

GeometryFromText

A synonym for [ST_GeomFromText](#).

GeomFromText

A synonym for [ST_GeomFromText](#).

LineFromText

A synonym for [ST_LineFromText](#).

LineStringFromText

A synonym for [ST_LineFromText](#).

MLineFromText

Syntax

```
MLineFromText(wkt[,srid])
MultiLineStringFromText(wkt[,srid])
```

Description

Constructs a [MULTILINESTRING](#) value using its [WKT](#) representation and [SRID](#).

`MLineFromText()` and `MultiLineStringFromText()` are synonyms.

Examples

```
CREATE TABLE gis_multi_line (g MULTILINESTRING);
SHOW FIELDS FROM gis_multi_line;
INSERT INTO gis_multi_line VALUES
(MultiLineStringFromText('MULTILINESTRING((10 48,10 21,10 0),(16 0,16 23,16 48))'),
(MLineFromText('MULTILINESTRING((10 48,10 21,10 0))')),
(MLineFromWKB(AsWKB(MultiLineString(LineString(Point(1, 2), Point(3, 5)), LineString(Point(2, 5), Point(5, 8), Point(21, 7))))));
```

MPointFromText

Syntax

```
MPointFromText(wkt[,srid])
MultiPointFromText(wkt[,srid])
```

Description

Constructs a [MULTIPOINT](#) value using its [WKT](#) representation and [SRID](#).

`MPointFromText()` and `MultiPointFromText()` are synonyms.

Examples

```
CREATE TABLE gis_multi_point (g MULTIPOINT);
SHOW FIELDS FROM gis_multi_point;
INSERT INTO gis_multi_point VALUES
(MultiPointFromText('MULTIPOINT(0 0,10 10,10 20,20 20)'),
(MPointFromText('MULTIPOINT(1 1,11 11,11 21,21 21)'),
(MPointFromWKB(AsWKB(MultiPoint(Point(3, 6), Point(4, 10))))));
```

MPolyFromText

Syntax

```
MPolyFromText(wkt[,srid])
MultiPolygonFromText(wkt[,srid])
```

Description

Constructs a **MULTIPOLYGON** value using its [WKT](#) representation and [SRID](#).

`MPolyFromText()` and `MultiPolygonFromText()` are synonyms.

Examples

```
CREATE TABLE gis_multi_polygon (g MULTIPOINT);
SHOW FIELDS FROM gis_multi_polygon;
INSERT INTO gis_multi_polygon VALUES
(MultiPolygonFromText('MULTIPOLYGON(((28 26,28 0,84 0,84 42,28 26),(52 18,66 23,73 9,48 6,52 18)),((59 18,67 18,67 13,59 13,59 18),(59 18,67 18,67 13,59 13,59 18))),((59 18,67 18,67 13,59 13,59 18))'));
(MPolyFromText('MULTIPOLYGON(((28 26,28 0,84 0,84 42,28 26),(52 18,66 23,73 9,48 6,52 18)),((59 18,67 18,67 13,59 13,59 18))))');
(MPolyFromWKB(AsWKB(MultiPolygon(Polygon(LineString(Point(0, 3), Point(3, 3), Point(3, 0), Point(0, 3)))))));
```

MultiLineStringFromText

A synonym for [MLineFromText](#).

MultiPointFromText

A synonym for [MPointFromText](#).

MultiPolygonFromText

A synonym for [MPolyFromText](#).

PointFromText

A synonym for [ST_PointFromText](#).

PolyFromText

A synonym for [ST_PolyFromText](#).

PolygonFromText

A synonym for [ST_PolyFromText](#).

ST_AsText

Syntax

```
ST_AsText(g)
AsText(g)
ST_AsWKT(g)
AsWKT(g)
```

Description

Converts a value in internal geometry format to its [WKT](#) representation and returns the string result.

`ST_AsText()` , `AsText()` , `ST_AsWKT()` and `AsWKT()` are all synonyms.

Examples

```
SET @g = 'LineString(1 1,4 4,6 6)';

SELECT ST_AsText(ST_GeomFromText(@g));
+-----+
| ST_AsText(ST_GeomFromText(@g)) |
+-----+
| LINESTRING(1 1,4 4,6 6)      |
+-----+
```

ST_ASWKT

A synonym for [ST_ASTEXT\(\)](#).

ST_GeomCollFromText

Syntax

```
ST_GeomCollFromText(wkt[,srid])
ST_GeometryCollectionFromText(wkt[,srid])
GeomCollFromText(wkt[,srid])
GeometryCollectionFromText(wkt[,srid])
```

Description

Constructs a [GEOMETRYCOLLECTION](#) value using its [WKT](#) representation and [SRID](#).

`ST_GeomCollFromText()`, `ST_GeometryCollectionFromText()`, `GeomCollFromText()` and `GeometryCollectionFromText()` are all synonyms.

Example

```
CREATE TABLE gis_geometrycollection (g GEOMETRYCOLLECTION);
SHOW FIELDS FROM gis_geometrycollection;
INSERT INTO gis_geometrycollection VALUES
(GeomCollFromText('GEOMETRYCOLLECTION(POINT(0 0), LINESTRING(0 0,10 10)))'),
(GeometryFromWKB(AsWKB(GeometryCollection(Point(44, 6), LineString(Point(3, 6), Point(7, 9))))),
(GeomFromText('GeometryCollection(')),
(GeomFromText('GeometryCollection EMPTY'));
```

ST_GeometryCollectionFromText

A synonym for [ST_GeomCollFromText](#).

ST_GeometryFromText

A synonym for [ST_GeomFromText](#).

ST_GeomFromText

Syntax

```
ST_GeomFromText(wkt[,srid])
ST_GeometryFromText(wkt[,srid])
GeomFromText(wkt[,srid])
GeometryFromText(wkt[,srid])
```

Description

Constructs a geometry value of any type using its [WKT](#) representation and [SRID](#).

`GeomFromText()`, `GeometryFromText()`, `ST_GeomFromText()` and `ST_GeometryFromText()` are all synonyms.

Example

```
SET @g = ST_GeomFromText('POLYGON((1 1,1 5,4 9,6 9,9 3,7 2,1 1));
```

ST_LineFromText

Syntax

```
ST_LineFromText(wkt[,srid])
ST_LineStringFromText(wkt[,srid])
LineFromText(wkt[,srid])
LineStringFromText(wkt[,srid])
```

Description

Constructs a [LINESTRING](#) value using its [WKT](#) representation and [SRID](#).

`ST_LineFromText()`, `ST_LineStringFromText()`, `ST_LineFromText()` and `ST_LineStringFromText()` are all synonyms.

Examples

```
CREATE TABLE gis_line (g LINESTRING);
SHOW FIELDS FROM gis_line;
INSERT INTO gis_line VALUES
  (LineFromText('LINESTRING(0 0,0 10,10 0)'),
   (LineStringFromText('LINESTRING(10 10,20 10,20 20,10 20,10 10)'), 
    (LineStringFromWKB(AsWKB(LineString(Point(10, 10), Point(40, 10))))));
```

ST_LineStringFromText

A synonym for [ST_LineFromText](#).

ST_PointFromText

Syntax

```
ST_PointFromText(wkt[,srid])
PointFromText(wkt[,srid])
```

Description

Constructs a [POINT](#) value using its [WKT](#) representation and [SRID](#).

`ST_PointFromText()` and `PointFromText()` are synonyms.

Examples

```

CREATE TABLE gis_point  (g POINT);
SHOW FIELDS FROM gis_point;
INSERT INTO gis_point VALUES
  (PointFromText('POINT(10 10'))),
  (PointFromText('POINT(20 10'))),
  (PointFromText('POINT(20 20'))),
  (PointFromWKB(AsWKB(PointFromText('POINT(10 20')))));

```

ST_PolyFromText

Syntax

```

ST_PolyFromText(wkt[,srid])
ST_PolygonFromText(wkt[,srid])
PolyFromText(wkt[,srid])
PolygonFromText(wkt[,srid])

```

Description

Constructs a POLYGON value using its [WKT](#) representation and [SRID](#).

`ST_PolyFromText()` , `ST_PolygonFromText()` , `PolyFromText()` and `ST_PolygonFromText()` are all synonyms.

Examples

```

CREATE TABLE gis_polygon    (g POLYGON);
INSERT INTO gis_polygon VALUES
  (PolygonFromText('POLYGON((10 10,20 10,20 20,10 20,10 10))')),
  (PolyFromText('POLYGON((0 0,50 0,50 50,0 50,0 0), (10 10,20 10,20 20,10 20,10 10))'));

```

ST_PolygonFromText

A synonym for [ST_PolyFromText](#).

Addition Operator (+)

Syntax

+

Description

Addition.

If both operands are integers, the result is calculated with [BIGINT](#) precision. If either integer is unsigned, the result is also an unsigned integer.

For real or string operands, the operand with the highest precision determines the result precision.

Contents

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

Examples

```
SELECT 3+5;
+-----+
| 3+5 |
+-----+
|   8 |
+-----+
```

See Also

- [Type Conversion](#)
- [Subtraction Operator \(-\)](#)
- [Multiplication Operator \(*\)](#)
- [Division Operator \(/\)](#)

Subtraction Operator (-)

Syntax

```
-
```

Description

Subtraction. The operator is also used as the unary minus for changing sign.

If both operands are integers, the result is calculated with `BIGINT` precision. If either integer is unsigned, the result is also an unsigned integer, unless the `NO_UNSIGNED_SUBTRACTION SQL_MODE` is enabled, in which case the result is always signed.

For real or string operands, the operand with the highest precision determines the result precision.

Examples

```
SELECT 96-9;
+-----+
| 96-9 |
+-----+
|   87 |
+-----+

SELECT 15-17;
+-----+
| 15-17 |
+-----+
|    -2 |
+-----+

SELECT 3.66 + 1.333;
+-----+
| 3.66 + 1.333 |
+-----+
|      4.993 |
+-----+
```

Unary minus:

```
SELECT - (3+5);
+-----+
| - (3+5) |
+-----+
|    -8 |
+-----+
```

See Also

- [Type Conversion](#)
- [Addition Operator \(+\)](#)
- [Multiplication Operator \(*\)](#)
- [Division Operator \(/\)](#)

Division Operator (/)

Syntax

```
/
```

Description

Division operator. Dividing by zero will return NULL. By default, returns four digits after the decimal. This is determined by the server system variable `div_precision_increment` which by default is four. It can be set from 0 to 30.

Dividing by zero returns `NULL`. If the `ERROR_ON_DIVISION_BY_ZERO SQL_MODE` is used (the default since [MariaDB 10.2.4](#)), a division by zero also produces a warning.

Contents

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

Examples

```
SELECT 4/5;
+-----+
| 4/5   |
+-----+
| 0.8000 |
+-----+

SELECT 300/(2-2);
+-----+
| 300/(2-2) |
+-----+
|      NULL  |
+-----+

SELECT 300/7;
+-----+
| 300/7    |
+-----+
| 42.8571  |
+-----+
```

Changing `div_precision_increment` for the session from the default of four to six:

```
SET div_precision_increment = 6;

SELECT 300/7;
+-----+
| 300/7    |
+-----+
| 42.857143 |
+-----+

SELECT 300/7;
+-----+
| 300/7    |
+-----+
| 42.857143 |
+-----+
```

See Also

- [Type Conversion](#)
- [Addition Operator \(+\)](#)
- [Subtraction Operator \(-\)](#)
- [Multiplication Operator \(*\)](#)

Multiplication Operator (*)

Syntax

```
*
```

Description

Multiplication operator.

Examples

```
SELECT 7*6;
+-----+
| 7*6 |
+-----+
|   42 |
+-----+

SELECT 1234567890*9876543210;
+-----+
| 1234567890*9876543210 |
+-----+
| -6253480962446024716 |
+-----+

SELECT 18014398509481984*18014398509481984.0;
+-----+
| 18014398509481984*18014398509481984.0 |
+-----+
| 324518553658426726783156020576256.0 |
+-----+

SELECT 18014398509481984*18014398509481984;
+-----+
| 18014398509481984*18014398509481984 |
+-----+
|          0 |
+-----+
```

Contents

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

See Also

- [Type Conversion](#)
- [Addition Operator \(+\)](#)
- [Subtraction Operator \(-\)](#)
- [Division Operator \(/\)](#)

Modulo Operator (%)

Syntax

```
N % M
```

Description

Modulo operator. Returns the remainder of `N` divided by `M`. See also [MOD](#).

Examples

```
SELECT 1042 % 50;
+-----+
| 1042 % 50 |
+-----+
|      42 |
+-----+
```

DIV

Syntax

```
DIV
```

Description

Integer division. Similar to [FLOOR\(\)](#), but is safe with `BIGINT` values. Incorrect results may occur for non-integer operands that exceed `BIGINT` range.

If the `ERROR_ON_DIVISION_BY_ZERO SQL_MODE` is used, a division by zero produces an error. Otherwise, it returns `NULL`.

The remainder of a division can be obtained using the [MOD](#) operator.

Examples

```
SELECT 300 DIV 7;
+-----+
| 300 DIV 7 |
+-----+
|      42 |
+-----+

SELECT 300 DIV 0;
+-----+
| 300 DIV 0 |
+-----+
|      NULL |
+-----+
```

ABS

Syntax

```
ABS(X)
```

Description

Returns the absolute (non-negative) value of `x`. If `x` is not a number, it is converted to a numeric type.

Examples

Contents

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

```
SELECT ABS(42);
+-----+
| ABS(42) |
+-----+
|      42 |
+-----+

SELECT ABS(-42);
+-----+
| ABS(-42) |
+-----+
|      42 |
+-----+

SELECT ABS(DATE '1994-01-01');
+-----+
| ABS(DATE '1994-01-01') |
+-----+
|          19940101 |
+-----+
```

See Also

- [SIGN\(\)](#)

ACOS

Syntax

ACOS(X)

Description

Returns the arc cosine of x , that is, the value whose cosine is x . Returns `NULL` if x is not in the range -1 to 1 .

Contents

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

Examples

```

SELECT ACOS(1);
+-----+
| ACOS(1) |
+-----+
|      0   |
+-----+

SELECT ACOS(1.0001);
+-----+
| ACOS(1.0001) |
+-----+
|      NULL    |
+-----+

SELECT ACOS(0);
+-----+
| ACOS(0)      |
+-----+
| 1.5707963267949 |
+-----+

SELECT ACOS(0.234);
+-----+
| ACOS(0.234)      |
+-----+
| 1.33460644244679 |
+-----+

```

ASIN

Syntax

ASIN(X)

Description

Returns the arc sine of X, that is, the value whose sine is X. Returns NULL if X is not in the range -1 to 1.

Contents

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

Examples

```

SELECT ASIN(0.2);
+-----+
| ASIN(0.2)      |
+-----+
| 0.2013579207903308 |
+-----+

SELECT ASIN('foo');
+-----+
| ASIN('foo')      |
+-----+
|      0   |
+-----+

SHOW WARNINGS;
+-----+
| Level  | Code | Message           |
+-----+
| Warning | 1292 | Truncated incorrect DOUBLE value: 'foo' |
+-----+

```

ATAN

Syntax

```
ATAN(X)
```

Description

Returns the arc tangent of X, that is, the value whose tangent is X.

Contents

- [Syntax](#)
- [Description](#)
- [Examples](#)

Examples

```
SELECT ATAN(2);
+-----+
| ATAN(2)           |
+-----+
| 1.1071487177940904 |
+-----+

SELECT ATAN(-2);
+-----+
| ATAN(-2)          |
+-----+
| -1.1071487177940904 |
+-----+
```

ATAN2

Syntax

```
ATAN(Y,X), ATAN2(Y,X)
```

Description

Returns the arc tangent of the two variables X and Y. It is similar to calculating the arc tangent of Y / X, except that the signs of both arguments are used to determine the quadrant of the result.

Examples

```
SELECT ATAN(-2,2);
+-----+
| ATAN(-2,2)          |
+-----+
| -0.7853981633974483 |
+-----+

SELECT ATAN2(PI(),0);
+-----+
| ATAN2(PI(),0)        |
+-----+
| 1.5707963267948966 |
+-----+
```

CEIL

Syntax

```
CEIL(X)
```

Description

`CEIL()` is a synonym for `CEILING()`.

CEILING

Syntax

```
CEILING(X)
```

Description

Returns the smallest integer value not less than X.

Examples

```
SELECT CEILING(1.23);
+-----+
| CEILING(1.23) |
+-----+
|          2   |
+-----+  
  
SELECT CEILING(-1.23);
+-----+
| CEILING(-1.23) |
+-----+
|          -1   |
+-----+
```

CONV

Syntax

```
CONV(N,from_base,to_base)
```

Description

Converts numbers between different number bases. Returns a string representation of the number `N`, converted from base `from_base` to base `to_base`.

Returns `NULL` if any argument is `NULL`, or if the second or third argument are not in the allowed range.

The argument `N` is interpreted as an integer, but may be specified as an integer or a string. The minimum base is 2 and the maximum base is 36. If `to_base` is a negative number, `N` is regarded as a signed number. Otherwise, `N` is treated as unsigned. `CONV()` works with 64-bit precision.

Some shortcuts for this function are also available: `BIN()`, `OCT()`, `HEX()`, `UNHEX()`. Also, MariaDB allows `binary` literal values and `hexadecimal` literal values.

Examples

```

SELECT CONV('a',16,2);
+-----+
| CONV('a',16,2) |
+-----+
| 1010           |
+-----+

SELECT CONV('6E',18,8);
+-----+
| CONV('6E',18,8) |
+-----+
| 172            |
+-----+

SELECT CONV(-17,10,-18);
+-----+
| CONV(-17,10,-18) |
+-----+
| -H              |
+-----+

SELECT CONV(12+'10'+'10'+0xa,10,10);
+-----+
| CONV(12+'10'+'10'+0xa,10,10) |
+-----+
| 42               |
+-----+

```

COS

Syntax

`COS(X)`

Description

Returns the cosine of X, where X is given in radians.

Examples

```

SELECT COS(PI());
+-----+
| COS(PI()) |
+-----+
|      -1   |
+-----+

```

COT

Syntax

`COT(X)`

Description

Returns the cotangent of X.

Examples

```
SELECT COT(42);
+-----+
| COT(42) |
+-----+
| 0.4364167060752729 |
+-----+

SELECT COT(12);
+-----+
| COT(12) |
+-----+
| -1.5726734063976893 |
+-----+

SELECT COT(0);
ERROR 1690 (22003): DOUBLE value is out of range in 'cot(0)'
```

CRC32

Syntax

<= MariaDB 10.7

```
CRC32(expr)
```

From MariaDB 10.8

```
CRC32([par,]expr)
```

Description

Computes a cyclic redundancy check (CRC) value and returns a 32-bit unsigned value. The result is NULL if the argument is NULL. The argument is expected to be a string and (if possible) is treated as one if it is not.

Uses the ISO 3309 polynomial that used by zlib and many others. [MariaDB 10.8](#) introduced the [CRC32C\(\)](#) function, which uses the alternate Castagnoli polynomia.

MariaDB starting with 10.8

Often, CRC is computed in pieces. To facilitate this, [MariaDB 10.8.0](#) introduced an optional parameter:

```
CRC32('MariaDB')=CRC32(CRC32('Maria'), 'DB').
```

Examples

```
SELECT CRC32('MariaDB');
+-----+
| CRC32('MariaDB') |
+-----+
| 4227209140 |
+-----+

SELECT CRC32('mariadb');
+-----+
| CRC32('mariadb') |
+-----+
| 2594253378 |
+-----+
```

```
SELECT CRC32(CRC32('Maria'), 'DB');
```

CRC32(CRC32('Maria'), 'DB')
4227209140

See Also

- [CRC32C\(\)](#)

DEGREES

Syntax

```
DEGREES(X)
```

Description

Returns the argument x , converted from radians to degrees.

This is the converse of the [RADIAN\(\)](#) function.

Examples

```
SELECT DEGREES(PI());
```

DEGREES(PI())
180


```
SELECT DEGREES(PI() / 2);
```

DEGREES(PI() / 2)
90


```
SELECT DEGREES(45);
```

DEGREES(45)
2578.3100780887

EXP

Syntax

```
EXP(X)
```

Description

Returns the value of e (the base of natural logarithms) raised to the power of X. The inverse of this function is [LOG\(\)](#) (using a single argument only) or [LN\(\)](#).

If X is NULL , this function returns NULL .

Examples

```
SELECT EXP(2);
+-----+
| EXP(2)      |
+-----+
| 7.38905609893065 |
+-----+

SELECT EXP(-2);
+-----+
| EXP(-2)      |
+-----+
| 0.1353352832366127 |
+-----+

SELECT EXP(0);
+-----+
| EXP(0)      |
+-----+
| 1           |
+-----+

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

FLOOR

Syntax

```
FLOOR(X)
```

Description

Returns the largest integer value not greater than X.

Examples

```
SELECT FLOOR(1.23);
+-----+
| FLOOR(1.23) |
+-----+
| 1           |
+-----+

SELECT FLOOR(-1.23);
+-----+
| FLOOR(-1.23) |
+-----+
| -2          |
+-----+
```

LN

Syntax

```
LN(X)
```

Description

Returns the natural logarithm of X; that is, the base-e logarithm of X. If X is less than or equal to 0, or `NULL`, then `NULL` is returned.

The inverse of this function is [EXP\(\)](#).

Examples

```
SELECT LN(2);
+-----+
| LN(2)      |
+-----+
| 0.693147180559945 |
+-----+

SELECT LN(-2);
+-----+
| LN(-2)   |
+-----+
| NULL    |
+-----+
```

LOG

Syntax

```
LOG(X), LOG(B,X)
```

Description

If called with one parameter, this function returns the natural logarithm of X. If X is less than or equal to 0, then `NULL` is returned.

If called with two parameters, it returns the logarithm of X to the base B. If B is ≤ 1 or X ≤ 0 , the function returns `NULL`.

If any argument is `NULL`, the function returns `NULL`.

The inverse of this function (when called with a single argument) is the [EXP\(\)](#) function.

Examples

LOG(X):

```
SELECT LOG(2);
+-----+
| LOG(2)      |
+-----+
| 0.693147180559945 |
+-----+

SELECT LOG(-2);
+-----+
| LOG(-2)   |
+-----+
| NULL    |
+-----+
```

LOG(B,X)

```
SELECT LOG(2,16);
+-----+
| LOG(2,16) |
+-----+
|      4 |
+-----+

SELECT LOG(3,27);
+-----+
| LOG(3,27) |
+-----+
|      3 |
+-----+

SELECT LOG(3,1);
+-----+
| LOG(3,1) |
+-----+
|      0 |
+-----+

SELECT LOG(3,0);
+-----+
| LOG(3,0) |
+-----+
|    NULL |
+-----+
```

LOG10

Syntax

LOG10(X)

Description

Returns the base-10 logarithm of X.

Examples

```
SELECT LOG10(2);
+-----+
| LOG10(2)          |
+-----+
| 0.301029995663981 |
+-----+

SELECT LOG10(100);
+-----+
| LOG10(100)        |
+-----+
|      2 |
+-----+

SELECT LOG10(-100);
+-----+
| LOG10(-100)       |
+-----+
|    NULL |
+-----+
```

LOG2

Syntax

```
LOG2(X)
```

Description

Returns the base-2 logarithm of X.

Examples

```
SELECT LOG2(4398046511104);
+-----+
| LOG2(4398046511104) |
+-----+
|          42 |
+-----+  
  
SELECT LOG2(65536);
+-----+
| LOG2(65536) |
+-----+
|       16 |
+-----+  
  
SELECT LOG2(-100);
+-----+
| LOG2(-100) |
+-----+
|      NULL |
+-----+
```

MOD

Syntax

```
MOD(N,M), N % M, N MOD M
```

Description

Modulo operation. Returns the remainder of N divided by M. See also [Modulo Operator](#).

If the `ERROR_ON_DIVISION_BY_ZERO SQL_MODE` is used, any number modulus zero produces an error. Otherwise, it returns NULL.

The integer part of a division can be obtained using [DIV](#).

Examples

```

SELECT 1042 % 50;
+-----+
| 1042 % 50 |
+-----+
|      42 |
+-----+

SELECT MOD(234, 10);
+-----+
| MOD(234, 10) |
+-----+
|          4 |
+-----+

SELECT 253 % 7;
+-----+
| 253 % 7 |
+-----+
|       1 |
+-----+

SELECT MOD(29,9);
+-----+
| MOD(29,9) |
+-----+
|       2 |
+-----+

SELECT 29 MOD 9;
+-----+
| 29 MOD 9 |
+-----+
|       2 |
+-----+

```

OCT

Syntax

`OCT(N)`

Description

Returns a string representation of the octal value of N, where N is a longlong ([BIGINT](#)) number. This is equivalent to [CONV\(N,10,8\)](#). Returns NULL if N is NULL.

Contents

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

Examples

```

SELECT OCT(34);
+-----+
| OCT(34) |
+-----+
|      42 |
+-----+

SELECT OCT(12);
+-----+
| OCT(12) |
+-----+
|      14 |
+-----+

```

See Also

- [CONV\(\)](#)
- [BIN\(\)](#)
- [HEX\(\)](#)

PI

Syntax

```
PI()
```

Description

Returns the value of π (pi). The default number of decimal places displayed is six, but MariaDB uses the full double-precision value internally.

Examples

```
SELECT PI();
+-----+
| PI()      |
+-----+
| 3.141593 |
+-----+

SELECT PI()+0.00000000000000000000;
+-----+
| PI()+0.00000000000000000000 |
+-----+
|      3.1415926535897931159980 |
+-----+
```

POW

Syntax

```
POW(X,Y)
```

Description

Returns the value of X raised to the power of Y.

POWER() is a synonym.

Examples

```
SELECT POW(2,3);
+-----+
| POW(2,3) |
+-----+
|      8   |
+-----+

SELECT POW(2,-2);
+-----+
| POW(2,-2) |
+-----+
|     0.25  |
+-----+
```

POWER

Syntax

```
POWER(X,Y)
```

Description

This is a synonym for [POW\(\)](#), which returns the value of X raised to the power of Y.

RADIANS

Syntax

```
RADIANS(X)
```

Description

Returns the argument x , converted from degrees to radians. Note that π radians equals 180 degrees.

This is the converse of the [DEGREES\(\)](#) function.

Examples

```

SELECT RADIANS(45);
+-----+
| RADIANS(45)      |
+-----+
| 0.785398163397448 |
+-----+

SELECT RADIANS(90);
+-----+
| RADIANS(90)      |
+-----+
| 1.5707963267949 |
+-----+

SELECT RADIANS(PI());
+-----+
| RADIANS(PI())      |
+-----+
| 0.0548311355616075 |
+-----+

SELECT RADIANS(180);
+-----+
| RADIANS(180)      |
+-----+
| 3.14159265358979 |
+-----+

```

RAND

Syntax

`RAND()`, `RAND(N)`

Description

Returns a random `DOUBLE` precision floating point value v in the range $0 \leq v < 1.0$. If a constant integer argument N is specified, it is used as the seed value, which produces a repeatable sequence of column values. In the example below, note that the sequences of values produced by `RAND(3)` is the same both places where it occurs.

In a WHERE clause, `RAND()` is evaluated each time the WHERE is executed.

Statements using the `RAND()` function are not [safe for statement-based replication](#).

Contents

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

Practical uses

The expression to get a random integer from a given range is the following:

```
FLOOR(min_value + RAND() * (max_value - min_value +1))
```

`RAND()` is often used to read random rows from a table, as follows:

```
SELECT * FROM my_table ORDER BY RAND() LIMIT 10;
```

Note, however, that this technique should never be used on a large table as it will be extremely slow. MariaDB will read all rows in the table, generate a random value for each of them, order them, and finally will apply the `LIMIT` clause.

Examples

```

CREATE TABLE t (i INT);

INSERT INTO t VALUES(1),(2),(3);

SELECT i, RAND() FROM t;
+-----+
| i    | RAND()          |
+-----+
| 1   | 0.255651095188829 |
| 2   | 0.833920199269355 |
| 3   | 0.40264774151393 |
+-----+

SELECT i, RAND(3) FROM t;
+-----+
| i    | RAND(3)          |
+-----+
| 1   | 0.90576975597606 |
| 2   | 0.373079058130345 |
| 3   | 0.148086053457191 |
+-----+

SELECT i, RAND() FROM t;
+-----+
| i    | RAND()          |
+-----+
| 1   | 0.511478140495232 |
| 2   | 0.349447508668012 |
| 3   | 0.212803152588013 |
+-----+

```

Using the same seed, the same sequence will be returned:

```

SELECT i, RAND(3) FROM t;
+-----+
| i    | RAND(3)          |
+-----+
| 1   | 0.90576975597606 |
| 2   | 0.373079058130345 |
| 3   | 0.148086053457191 |
+-----+

```

Generating a random number from 5 to 15:

```

SELECT FLOOR(5 + (RAND() * 11));

```

See Also

- [Techniques for Efficiently Finding a Random Row](#)
- [rand_seed1 and rand_seed2 system variables](#)

ROUND

Syntax

```

ROUND(X), ROUND(X,D)

```

Description

Rounds the argument `X` to `D` decimal places. The rounding algorithm depends on the data type of `X`. `D` defaults to `0` if not specified. `D` can be negative to cause `D` digits left of the decimal point of the value `X` to become zero.

Examples

```
SELECT ROUND(-1.23);
+-----+
| ROUND(-1.23) |
+-----+
|          -1 |
+-----+

SELECT ROUND(-1.58);
+-----+
| ROUND(-1.58) |
+-----+
|          -2 |
+-----+

SELECT ROUND(1.58);
+-----+
| ROUND(1.58) |
+-----+
|          2 |
+-----+

SELECT ROUND(1.298, 1);
+-----+
| ROUND(1.298, 1) |
+-----+
|          1.3 |
+-----+

SELECT ROUND(1.298, 0);
+-----+
| ROUND(1.298, 0) |
+-----+
|          1 |
+-----+

SELECT ROUND(23.298, -1);
+-----+
| ROUND(23.298, -1) |
+-----+
|          20 |
+-----+
```

SIGN

Syntax

```
SIGN(X)
```

Description

Returns the sign of the argument as -1, 0, or 1, depending on whether X is negative, zero, or positive.

Examples

```
SELECT SIGN(-32);
```

```
+-----+
| SIGN(-32) |
+-----+
|      -1   |
+-----+
```

```
SELECT SIGN(0);
```

```
+-----+
| SIGN(0) |
+-----+
|      0   |
+-----+
```

```
SELECT SIGN(234);
```

```
+-----+
| SIGN(234) |
+-----+
|      1   |
+-----+
```

See Also

- [ABS\(\)](#)

SIN

Syntax

```
SIN(X)
```

Description

Returns the sine of X, where X is given in radians.

Examples

```
SELECT SIN(1.5707963267948966);
```

```
+-----+
| SIN(1.5707963267948966) |
+-----+
|      1   |
+-----+
```

```
SELECT SIN(PI());
```

```
+-----+
| SIN(PI())      |
+-----+
| 1.22460635382238e-16 |
+-----+
```

```
SELECT ROUND(SIN(PI()));
```

```
+-----+
| ROUND(SIN(PI())) |
+-----+
|      0   |
+-----+
```

SQRT

Syntax

```
SQRT(X)
```

Description

Returns the square root of X. If X is negative, NULL is returned.

Examples

```
SELECT SQRT(4);
+-----+
| SQRT(4) |
+-----+
|      2   |
+-----+

SELECT SQRT(20);
+-----+
| SQRT(20)        |
+-----+
| 4.47213595499958 |
+-----+

SELECT SQRT(-16);
+-----+
| SQRT(-16) |
+-----+
|    NULL   |
+-----+

SELECT SQRT(1764);
+-----+
| SQRT(1764) |
+-----+
|      42   |
+-----+
```

TAN

Syntax

```
TAN(X)
```

Description

Returns the tangent of X, where X is given in radians.

Examples

```

SELECT TAN(0.7853981633974483);
+-----+
| TAN(0.7853981633974483) |
+-----+
| 0.9999999999999999 |
+-----+

SELECT TAN(PI());
+-----+
| TAN(PI()) |
+-----+
| -1.22460635382238e-16 |
+-----+

SELECT TAN(PI()+1);
+-----+
| TAN(PI()+1) |
+-----+
| 1.5574077246549 |
+-----+

SELECT TAN(RADIANS(PI()));
+-----+
| TAN(RADIANS(PI())) |
+-----+
| 0.0548861508080033 |
+-----+

```

TRUNCATE

This page documents the TRUNCATE function. See [TRUNCATE TABLE](#) for the DDL statement.

Syntax

```
TRUNCATE(X,D)
```

Description

Returns the number X, truncated to D decimal places. If D is 0, the result has no decimal point or fractional part. D can be negative to cause D digits left of the decimal point of the value X to become zero.

Examples

Contents

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

```

SELECT TRUNCATE(1.223,1);
+-----+
| TRUNCATE(1.223,1) |
+-----+
|      1.2 |
+-----+

SELECT TRUNCATE(1.999,1);
+-----+
| TRUNCATE(1.999,1) |
+-----+
|      1.9 |
+-----+

SELECT TRUNCATE(1.999,0);
+-----+
| TRUNCATE(1.999,0) |
+-----+
|      1 |
+-----+

SELECT TRUNCATE(-1.999,1);
+-----+
| TRUNCATE(-1.999,1) |
+-----+
|     -1.9 |
+-----+

SELECT TRUNCATE(122,-2);
+-----+
| TRUNCATE(122,-2) |
+-----+
|      100 |
+-----+

SELECT TRUNCATE(10.28*100,0);
+-----+
| TRUNCATE(10.28*100,0) |
+-----+
|      1028 |
+-----+

```

See Also

- [TRUNCATE TABLE](#)

Plugin Overview

Plugins are server components that enhance MariaDB in some way. These can be anything from new storage engines, plugins for enhancing full-text parsing, or even small enhancements, such as a plugin to get a timestamp as an integer.

Querying Plugin Information

There are a number of ways to see which plugins are currently active.

A server almost always has a large number of active plugins, because the server contains a large number of built-in plugins, which are active by default and cannot be uninstalled.

Querying Plugin Information with SHOW PLUGINS

The [SHOW PLUGINS](#) statement can be used to query information about all active plugins.

For example:

```

SHOW PLUGINS\G;
*****
1. row *****
  Name: binlog
  Status: ACTIVE
  Type: STORAGE ENGINE
  Library: NULL
  License: GPL
*****
2. row *****
  Name: mysql_native_password
  Status: ACTIVE
  Type: AUTHENTICATION
  Library: NULL
  License: GPL
*****
3. row *****
  Name: mysql_old_password
  Status: ACTIVE
  Type: AUTHENTICATION
  Library: NULL
  License: GPL
...

```

If a plugin's `Library` column has a `NULL` value, then the plugin is built-in, and it cannot be uninstalled.

Querying Plugin Information with `information_schema.PLUGINS`

The `information_schema.PLUGINS` table can be queried to get more detailed information about plugins.

For example:

```

SELECT * FROM information_schema.PLUGINS\G
...
*****
6. row *****
  PLUGIN_NAME: CSV
  PLUGIN_VERSION: 1.0
  PLUGIN_STATUS: ACTIVE
  PLUGIN_TYPE: STORAGE ENGINE
  PLUGIN_TYPE_VERSION: 100003.0
  PLUGIN_LIBRARY: NULL
  PLUGIN_LIBRARY_VERSION: NULL
  PLUGIN_AUTHOR: Brian Aker, MySQL AB
  PLUGIN_DESCRIPTION: CSV storage engine
  PLUGIN_LICENSE: GPL
  LOAD_OPTION: FORCE
  PLUGIN_MATURITY: Stable
  PLUGIN_AUTH_VERSION: 1.0
*****
7. row *****
  PLUGIN_NAME: MEMORY
  PLUGIN_VERSION: 1.0
  PLUGIN_STATUS: ACTIVE
  PLUGIN_TYPE: STORAGE ENGINE
  PLUGIN_TYPE_VERSION: 100003.0
  PLUGIN_LIBRARY: NULL
  PLUGIN_LIBRARY_VERSION: NULL
  PLUGIN_AUTHOR: MySQL AB
  PLUGIN_DESCRIPTION: Hash based, stored in memory, useful for temporary tables
  PLUGIN_LICENSE: GPL
  LOAD_OPTION: FORCE
  PLUGIN_MATURITY: Stable
  PLUGIN_AUTH_VERSION: 1.0
...

```

If a plugin's `PLUGIN_LIBRARY` column has the `NULL` value, then the plugin is built-in, and it cannot be uninstalled.

Contents

- 1. [Querying Plugin Information](#)
 - 1. [Querying Plugin Information with `SHOW PLUGINS`](#)
 - 2. [Querying Plugin Information with `information_schema.PLUGINS`](#)
 - 3. [Querying Plugin Information with `mysql.plugin`](#)
- 2. [Installing a Plugin](#)
 - 1. [Installing a Plugin Dynamically](#)
 - 1. [Installing a Plugin with `INSTALL SONAME`](#)
 - 2. [Installing a Plugin with `INSTALL PLUGIN`](#)
 - 2. [Installing a Plugin with Plugin Load Options](#)
 - 1. [Installing a Plugin with `--plugin-load-add`](#)
 - 2. [Installing a Plugin with `--plugin-load`](#)
 - 3. [Specifying Multiple Plugin Load Options](#)
 - 3. [Installing a Plugin with `mysql_plugin`](#)
 - 4. [Configuring the Plugin Directory](#)
 - 5. [Configuring the Minimum Plugin Maturity](#)
 - 6. [Configuring Plugin Activation at Server Startup](#)
 - 3. [Uninstalling Plugins](#)
 - 4. [See Also](#)

Querying Plugin Information with mysql.plugin

The `mysql.plugin` table can be queried to get information about installed plugins.

This table only contains information about plugins that have been installed via the following methods:

- The `INSTALL SONAME` statement.
- The `INSTALL PLUGIN` statement.
- The `mysql_plugin` utility.

This table does not contain information about:

- Built-in plugins.
- Plugins loaded with the `--plugin-load-add` option.
- Plugins loaded with the `--plugin-load` option.

This table only contains enough information to reload the plugin when the server is restarted, which means it only contains the plugin name and the plugin library.

For example:

```
SELECT * FROM mysql.plugin;

+-----+-----+
| name | d1   |
+-----+-----+
| PBXT | libpbxt.so |
+-----+-----+
```

Installing a Plugin

There are three primary ways to install a plugin:

- A plugin can be installed dynamically with an SQL statement.
- A plugin can be installed with a `mysqld` option, but it requires a server restart.
- A plugin can be installed with the `mysql_plugin` utility, while the server is completely offline.

When you are installing a plugin, you also have to ensure that:

- The server's plugin directory is properly configured, and the plugin's library is in the plugin directory.
- The server's minimum plugin maturity is properly configured, and the plugin is mature enough to be installed.

Installing a Plugin Dynamically

A plugin can be installed dynamically by executing either the `INSTALL SONAME` or the `INSTALL PLUGIN` statement.

If a plugin is installed with one of these statements, then a record will be added to the `mysql.plugins` table for the plugin. This means that the plugin will automatically be loaded every time the server restarts, unless specifically uninstalled or deactivated.

Installing a Plugin with `INSTALL SONAME`

You can install a plugin dynamically by executing the `INSTALL SONAME` statement. `INSTALL SONAME` installs all plugins from the given plugin library. This could be required for some plugin libraries.

For example, to install all plugins in the `server_audit` plugin library (which is currently only the `server_audit` audit plugin), you could execute the following:

```
INSTALL SONAME 'server_audit';
```

Installing a Plugin with `INSTALL PLUGIN`

You can install a plugin dynamically by executing the `INSTALL PLUGIN` statement. `INSTALL PLUGIN` installs a single plugin from the given plugin library.

For example, to install the `server_audit` audit plugin from the `server_audit` plugin library, you could execute the following:

```
INSTALL PLUGIN server_audit SONAME 'server_audit';
```

Installing a Plugin with Plugin Load Options

A plugin can be installed with a `mysqld` option by providing either the `--plugin-load-add` or the `--plugin-load` option.

If a plugin is installed with one of these options, then a record will **not** be added to the `mysql.plugins` table for the plugin. This means that if the server is restarted without the same option set, then the plugin will **not** automatically be loaded.

Installing a Plugin with `--plugin-load-add`

You can install a plugin with the `--plugin-load-add` option by specifying the option as a command-line argument to `mysqld` or by specifying the option in a relevant server [option group](#) in an [option file](#).

The `--plugin-load-add` option uses the following format:

- Plugins can be specified in the format `name=library`, where `name` is the plugin name and `library` is the plugin library. This format installs a single plugin from the given plugin library.
- Plugins can also be specified in the format `library`, where `library` is the plugin library. This format installs all plugins from the given plugin library.
- Multiple plugins can be specified by separating them with semicolons.

For example, to install all plugins in the `server_audit` plugin library (which is currently only the `server_audit` audit plugin) and also the `ed25519` authentication plugin from the `auth_ed25519` plugin library, you could set the option to the following values on the command-line:

```
$ mysqld --user=mysql --plugin-load-add='server_audit' --plugin-load-add='ed25519=auth_ed25519'
```

You could also set the option to the same values in an [option file](#):

```
[mariadb]
...
plugin_load_add = server_audit
plugin_load_add = ed25519=auth_ed25519
```

Special care must be taken when specifying both the `--plugin-load` option and the `--plugin-load-add` option together. The `--plugin-load` option resets the plugin load list, and this can cause unexpected problems if you are not aware. The `--plugin-load-add` option does **not** reset the plugin load list, so it is much safer to use. See [Specifying Multiple Plugin Load Options](#) for more information.

Installing a Plugin with `--plugin-load`

You can install a plugin with the `--plugin-load` option by specifying the option as a command-line argument to `mysqld` or by specifying the option in a relevant server [option group](#) in an [option file](#).

The `--plugin-load` option uses the following format:

- Plugins can be specified in the format `name=library`, where `name` is the plugin name and `library` is the plugin library. This format installs a single plugin from the given plugin library.
- Plugins can also be specified in the format `library`, where `library` is the plugin library. This format installs all plugins from the given plugin library.
- Multiple plugins can be specified by separating them with semicolons.

For example, to install all plugins in the `server_audit` plugin library (which is currently only the `server_audit` audit plugin) and also the `ed25519` authentication plugin from the `auth_ed25519` plugin library, you could set the option to the following values on the command-line:

```
$ mysqld --user=mysql --plugin-load='server_audit;ed25519=auth_ed25519'
```

You could also set the option to the same values in an [option file](#):

```
[mariadb]
...
plugin_load = server_audit;ed25519=auth_ed25519
```

Special care must be taken when specifying the `--plugin-load` option multiple times, or when specifying both the `--plugin-load` option and the `--plugin-load-add` option together. The `--plugin-load` option resets the plugin load list, and this can cause unexpected problems if you are not aware. The `--plugin-load-add` option does **not** reset the plugin load list, so it is much safer to use. See [Specifying Multiple Plugin Load Options](#) for more information.

[Load Options](#) for more information.

Specifying Multiple Plugin Load Options

Special care must be taken when specifying the `--plugin-load` option multiple times, or when specifying both the `--plugin-load` option and the `--plugin-load-add` option. The `--plugin-load` option resets the plugin load list, and this can cause unexpected problems if you are not aware. The `--plugin-load-add` option does **not** reset the plugin load list, so it is much safer to use.

This can have the following consequences:

- If the `--plugin-load` option is specified **multiple times**, then only the last instance will have any effect. For example, in the following case, the first instance of the option is reset:

```
[mariadb]
...
plugin_load = server_audit
plugin_load = ed25519=auth_ed25519
```

- If the `--plugin-load` option is specified **after** the `--plugin-load-add` option, then it will also reset the changes made by that option. For example, in the following case, the `--plugin-load-add` option does not do anything, because the subsequent `--plugin-load` option resets the plugin load list:

```
[mariadb]
...
plugin_load_add = server_audit
plugin_load = ed25519=auth_ed25519
```

- In contrast, if the `--plugin-load` option is specified **before** the `--plugin-load-add` option, then it will work fine, because the `--plugin-load-add` option does not reset the plugin load list. For example, in the following case, both plugins are properly loaded:

```
[mariadb]
...
plugin_load = server_audit
plugin_load_add = ed25519=auth_ed25519
```

Installing a Plugin with `mysql_plugin`

A plugin can be installed with the `mysql_plugin` utility if the server is completely offline.

The syntax is:

```
mysql_plugin [options] <plugin> ENABLE|DISABLE
```

For example, to install the `server_audit` audit plugin, you could execute the following:

```
mysql_plugin server_audit ENABLE
```

If a plugin is installed with this utility, then a record will be added to the `mysql.plugins` table for the plugin. This means that the plugin will automatically be loaded every time the server restarts, unless specifically uninstalled or deactivated.

Configuring the Plugin Directory

When a plugin is being installed, the server looks for the plugin's library in the server's plugin directory. This directory is configured by the `plugin_dir` system variable. This can be specified as a command-line argument to `mysqld` or it can be specified in a relevant server `option group` in an `option file`. For example:

```
[mariadb]
...
plugin_dir = /usr/lib64/mysql/plugin
```

Configuring the Minimum Plugin Maturity

When a plugin is being installed, the server compares the plugin's maturity level against the server's minimum allowed plugin maturity. This can help prevent users from using unstable plugins on production servers. This minimum plugin maturity is configured by the `plugin_maturity` system variable. This can be specified as a command-line argument to `mysqld` or it can be specified in a relevant server `option group` in an `option file`. For example:

```
[mariadb]
...
plugin_maturity = stable
```

Configuring Plugin Activation at Server Startup

A plugin will be loaded by default when the server starts if:

- The plugin was installed with the `INSTALL SONAME` statement.
- The plugin was installed with the `INSTALL PLUGIN` statement.
- The plugin was installed with the `mysql_plugin` utility.
- The server is configured to load the plugin with the `--plugin-load-add` option.
- The server is configured to load the plugin with the `--plugin-load` option.

This behavior can be changed with special options that take the form `--plugin-name`. For example, for the `server_audit` audit plugin, the special option is called `--server-audit`.

The possible values for these special options are:

Option Value	Description
OFF	Disables the plugin without removing it from the <code>mysql.plugins</code> table.
ON	Enables the plugin. If the plugin cannot be initialized, then the server will still continue starting up, but the plugin will be disabled.
FORCE	Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error.
FORCE_PLUS_PERMANENT	Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error. In addition, the plugin cannot be uninstalled with <code>UNINSTALL SONAME</code> or <code>UNINSTALL PLUGIN</code> while the server is running.

A plugin's status can be found by looking at the `PLUGIN_STATUS` column of the `information_schema.PLUGINS` table.

Uninstalling Plugins

Plugins that are found in the `mysql.plugin` table, that is those that were installed with `INSTALL SONAME`, `INSTALL PLUGIN` or `mysql_plugin` can be uninstalled in one of two ways:

- The `UNINSTALL SONAME` or the `UNINSTALL PLUGIN` statement while the server is running
- With `mysql_plugin` while the server is offline.

Plugins that were enabled as a `--plugin-load` option do not need to be uninstalled. If `--plugin-load` is omitted the next time the server starts, or the plugin is not listed as one of the `--plugin-load` entries, the plugin will not be loaded.

`UNINSTALL PLUGIN` uninstalls a single installed plugin, while `UNINSTALL SONAME` uninstalls all plugins belonging to a given library.

See Also

- [List of Plugins](#)
- [INSTALL PLUGIN](#)
- [INSTALL SONAME](#)
- [UNINSTALL PLUGIN](#)
- [UNINSTALL SONAME](#)
- [SHOW PLUGINS](#)
- [INFORMATION_SCHEMA.PLUGINS Table](#)
- [mysql_plugin](#)

MBR Definition

Description

The MBR (Minimum Bounding Rectangle), or Envelope is the bounding geometry, formed by the minimum and maximum (X,Y) coordinates:

Examples

```
((MINX MINY, MAXX MINY, MAXX MAXY, MINX MAXY, MINX MINY))
```

MBRContains

Syntax

```
MBRContains(g1,g2)
```

Description

Returns 1 or 0 to indicate whether the Minimum Bounding Rectangle of g1 contains the Minimum Bounding Rectangle of g2. This tests the opposite relationship as [MBRWithin\(\)](#).

Examples

```
SET @g1 = GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0));  
SET @g2 = GeomFromText('Point(1 1)');  
  
SELECT MBRContains(@g1,@g2), MBRContains(@g2,@g1);  
+-----+-----+  
| MBRContains(@g1,@g2) | MBRContains(@g2,@g1) |  
+-----+-----+  
| 1 | 0 |  
+-----+-----+
```

MBREqual

Syntax

```
MBREqual(g1,g2)
```

Description

Returns 1 or 0 to indicate whether the Minimum Bounding Rectangles of the two geometries g1 and g2 are the same.

Examples

```

SET @g1=GEOMFROMTEXT('LINESTRING(0 0, 1 2)');
SET @g2=GEOMFROMTEXT('POLYGON((0 0, 0 2, 1 2, 1 0, 0 0))');
SELECT MbrEqual(@g1,@g2);
+-----+
| MbrEqual(@g1,@g2) |
+-----+
|          1          |
+-----+

SET @g1=GEOMFROMTEXT('LINESTRING(0 0, 1 3)');
SET @g2=GEOMFROMTEXT('POLYGON((0 0, 0 2, 1 4, 1 0, 0 0))');
SELECT MbrEqual(@g1,@g2);
+-----+
| MbrEqual(@g1,@g2) |
+-----+
|          0          |
+-----+

```

MBRWithin

Syntax

```
MBRWithin(g1,g2)
```

Description

Returns 1 or 0 to indicate whether the Minimum Bounding Rectangle of g1 is within the Minimum Bounding Rectangle of g2. This tests the opposite relationship as [MBRContains\(\)](#).

Examples

```

SET @g1 = GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
SET @g2 = GeomFromText('Polygon((0 0,0 5,5 5,5 0,0 0))');
SELECT MBRWithin(@g1,@g2), MBRWithin(@g2,@g1);
+-----+
| MBRWithin(@g1,@g2) | MBRWithin(@g2,@g1) |
+-----+
|          1          |          0          |
+-----+

```

IF Function

Syntax

```
IF(expr1,expr2,expr3)
```

Description

If `expr1` is TRUE (`expr1 <> 0` and `expr1 <> NULL`) then `IF()` returns `expr2` ; otherwise it returns `expr3` . `IF()` returns a numeric or string value, depending on the context in which it is used.

Note: There is also an [IF statement](#) which differs from the `IF()` function described here.

Contents

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

Examples

```
SELECT IF(1>2,2,3);
+-----+
| IF(1>2,2,3) |
+-----+
|      3 |
+-----+
```

```
SELECT IF(1<2,'yes','no');
+-----+
| IF(1<2,'yes','no') |
+-----+
| yes                |
+-----+
```

```
SELECT IF(STRCMP('test','test1'),'no','yes');
+-----+
| IF(STRCMP('test','test1'),'no','yes') |
+-----+
| no                                |
+-----+
```

See Also

There is also an [IF statement](#), which differs from the `IF()` function described above.

MBRDisjoint

Syntax

```
MBRDisjoint(g1,g2)
```

Description

Returns 1 or 0 to indicate whether the Minimum Bounding Rectangles of the two geometries g1 and g2 are disjoint. Two geometries are disjoint if they do not intersect, that is touch or overlap.

Examples

```
SET @g1 = GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
SET @g2 = GeomFromText('Polygon((4 4,4 7,7 7,7 4,4 4))');
SELECT mbrdisjoint(@g1,@g2);
+-----+
| mbrdisjoint(@g1,@g2) |
+-----+
|      1 |
+-----+  
  
SET @g1 = GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
SET @g2 = GeomFromText('Polygon((3 3,3 6,6 6,6 3,3 3))');
SELECT mbrdisjoint(@g1,@g2);
+-----+
| mbrdisjoint(@g1,@g2) |
+-----+
|      0 |
+-----+
```

MBRIntersects

Syntax

```
MBRIntersects(g1,g2)
```

Description

Returns 1 or 0 to indicate whether the Minimum Bounding Rectangles of the two geometries g1 and g2 intersect.

Examples

```
SET @g1 = GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
SET @g2 = GeomFromText('Polygon((3 3,3 6,6 6,6 3,3 3))');
SELECT mbrintersects(@g1,@g2);
+-----+
| mbrintersects(@g1,@g2) |
+-----+
|          1          |
+-----+

SET @g1 = GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))';
SET @g2 = GeomFromText('Polygon((4 4,4 7,7 7,7 4,4 4))');
SELECT mbrintersects(@g1,@g2);
+-----+
| mbrintersects(@g1,@g2) |
+-----+
|          0          |
+-----+
```

MBROverlaps

Syntax

```
MBROverlaps(g1,g2)
```

Description

Returns 1 or 0 to indicate whether the Minimum Bounding Rectangles of the two geometries g1 and g2 overlap. The term spatially overlaps is used if two geometries intersect and their intersection results in a geometry of the same dimension but not equal to either of the given geometries.

Examples

```

SET @g1 = GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
SET @g2 = GeomFromText('Polygon((4 4,4 7,7 7,7 4,4 4))');
SELECT mbroverlaps(@g1,@g2);
+-----+
| mbroverlaps(@g1,@g2) |
+-----+
|          0 |
+-----+

SET @g1 = GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
SET @g2 = GeomFromText('Polygon((3 3,3 6,6 6,6 3,3 3))');
SELECT mbroverlaps(@g1,@g2);
+-----+
| mbroverlaps(@g1,@g2) |
+-----+
|          0 |
+-----+

SET @g1 = GeomFromText('Polygon((0 0,0 4,4 4,4 0,0 0))');
SET @g2 = GeomFromText('Polygon((3 3,3 6,6 6,6 3,3 3))');
SELECT mbroverlaps(@g1,@g2);
+-----+
| mbroverlaps(@g1,@g2) |
+-----+
|          1 |
+-----+

```

MBRTouches

Syntax

```
MBRTouches(g1,g2)
```

Description

Returns 1 or 0 to indicate whether the Minimum Bounding Rectangles of the two geometries `g1` and `g2` touch. Two geometries spatially touch if the interiors of the geometries do not intersect, but the boundary of one of the geometries intersects either the boundary or the interior of the other.

Examples

```

SET @g1 = GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
SET @g2 = GeomFromText('Polygon((4 4,4 7,7 7,7 4,4 4))');
SELECT mbrtouches(@g1,@g2);
+-----+
| mbrtouches(@g1,@g2) |
+-----+
|          0 |
+-----+

SET @g1 = GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
SET @g2 = GeomFromText('Polygon((3 3,3 6,6 6,6 3,3 3))');
SELECT mbrtouches(@g1,@g2);
+-----+
| mbrtouches(@g1,@g2) |
+-----+
|          1 |
+-----+

SET @g1 = GeomFromText('Polygon((0 0,0 4,4 4,4 0,0 0))');
SET @g2 = GeomFromText('Polygon((3 3,3 6,6 6,6 3,3 3))');
SELECT mbrtouches(@g1,@g2);
+-----+
| mbrtouches(@g1,@g2) |
+-----+
|          0 |
+-----+

```

CASE OPERATOR

Syntax

```

CASE value WHEN [compare_value] THEN result [WHEN [compare_value] THEN
result ...] [ELSE result] END

CASE WHEN [condition] THEN result [WHEN [condition] THEN result ...]
[ELSE result] END

```

Description

The first version returns the result where value=compare_value. The second version returns the result for the first condition that is true. If there was no matching result value, the result after ELSE is returned, or NULL if there is no ELSE part.

There is also a [CASE statement](#), which differs from the CASE operator described here.

Contents

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

Examples

```

SELECT CASE 1 WHEN 1 THEN 'one' WHEN 2 THEN 'two' ELSE 'more' END;
+-----+
| CASE 1 WHEN 1 THEN 'one' WHEN 2 THEN 'two' ELSE 'more' END |
+-----+
| one
+-----+


SELECT CASE WHEN 1>0 THEN 'true' ELSE 'false' END;
+-----+
| CASE WHEN 1>0 THEN 'true' ELSE 'false' END |
+-----+
| true
+-----+


SELECT CASE BINARY 'B' WHEN 'a' THEN 1 WHEN 'b' THEN 2 END;
+-----+
| CASE BINARY 'B' WHEN 'a' THEN 1 WHEN 'b' THEN 2 END |
+-----+
| NULL
+-----+

```

IFNULL

Syntax

```

IFNULL(expr1,expr2)
NVL(expr1,expr2)

```

Description

If *expr1* is not NULL, IFNULL() returns *expr1*; otherwise it returns *expr2*. IFNULL() returns a numeric or string value, depending on the context in which it is used.

From MariaDB 10.3, NVL() is an alias for IFNULL().

Contents

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

Examples

```
SELECT IFNULL(1,0);
+-----+
| IFNULL(1,0) |
+-----+
|      1      |
+-----+

SELECT IFNULL(NULL,10);
+-----+
| IFNULL(NULL,10) |
+-----+
|      10      |
+-----+

SELECT IFNULL(1/0,10);
+-----+
| IFNULL(1/0,10) |
+-----+
|    10.0000   |
+-----+

SELECT IFNULL(1/0,'yes');
+-----+
| IFNULL(1/0,'yes') |
+-----+
| yes               |
+-----+
```

See Also

- [NULL values](#)
- [IS NULL operator](#)
- [IS NOT NULL operator](#)
- [COALESCE function](#)
- [NULLIF function](#)
- [CONNECT data types](#)

NULLIF

Syntax

```
NULLIF(expr1,expr2)
```

Description

Returns NULL if expr1 = expr2 is true, otherwise returns expr1. This is the same as CASE WHEN expr1 = expr2 THEN NULL ELSE expr1 END.

Examples

Contents

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

```

SELECT NULLIF(1,1);
+-----+
| NULLIF(1,1) |
+-----+
|      NULL   |
+-----+

SELECT NULLIF(1,2);
+-----+
| NULLIF(1,2) |
+-----+
|          1   |
+-----+

```

See Also

- [NULL values](#)
- [IS NULL operator](#)
- [IS NOT NULL operator](#)
- [COALESCE function](#)
- [IFNULL function](#)
- [CONNECT data types](#)

COMMIT

The `COMMIT` statement ends a transaction, saving any changes to the data so that they become visible to subsequent transactions. Also, [unlocks metadata](#) changed by current transaction. If `autocommit` is set to 1, an implicit commit is performed after each statement. Otherwise, all transactions which don't end with an explicit `COMMIT` are implicitly rolled back and the changes are lost. The `ROLLBACK` statement can be used to do this explicitly.

The required syntax for the `COMMIT` statement is as follows:

```
COMMIT [WORK] [AND [NO] CHAIN] [[NO] RELEASE]
```

`COMMIT` is the more important transaction terminator, as well as the more interesting one. The basic form of the `COMMIT` statement is simply the keyword `COMMIT` (the keyword `WORK` is simply noise and can be omitted without changing the effect).

The optional `AND CHAIN` clause is a convenience for initiating a new transaction as soon as the old transaction terminates. If `AND CHAIN` is specified, then there is effectively nothing between the old and new transactions, although they remain separate. The characteristics of the new transaction will be the same as the characteristics of the old one — that is, the new transaction will have the same access mode, isolation level and diagnostics area size (we'll discuss all of these shortly) as the transaction just terminated.

`RELEASE` tells the server to disconnect the client immediately after the current transaction.

There are `NO RELEASE` and `AND NO CHAIN` options. By default, commits do not `RELEASE` or `CHAIN`, but it's possible to change this default behavior with the `completion_type` server system variable. In this case, the `AND NO CHAIN` and `NO RELEASE` options override the server default.

See Also

- [autocommit](#) - server system variable that determines whether statements are automatically committed.
- [completion_type](#) - server system variable that determines whether COMMIT's are standard, COMMIT AND CHAIN or COMMIT RELEASE.
- [SQL statements that cause an implicit commit](#)

DEALLOCATE / DROP PREPARE

Syntax

```
{DEALLOCATE | DROP} PREPARE stmt_name
```

Description

To deallocate a prepared statement produced with `PREPARE`, use a `DEALLOCATE PREPARE` statement that refers to the prepared statement name.

A prepared statement is implicitly deallocated when a new `PREPARE` command is issued. In that case, there is no need to use `DEALLOCATE`.

Attempting to execute a prepared statement after deallocating it results in an error, as if it was not prepared at all:

```
ERROR 1243 (HY000): Unknown prepared statement handler (stmt_name) given to EXECUTE
```

If the specified statement has not been PREPARED, an error similar to the following will be produced:

```
ERROR 1243 (HY000): Unknown prepared statement handler (stmt_name) given to DEALLOCATE PREPARE
```

Example

See [example in PREPARE](#).

See Also

- [PREPARE Statement](#)
- [EXECUTE Statement](#)
- [EXECUTE IMMEDIATE](#)

EXECUTE Statement

Syntax

```
EXECUTE stmt_name  
[USING expression[, expression] ...]
```

MariaDB starting with 10.2.3

`EXECUTE` with expression as parameters was introduced in [MariaDB 10.2.3](#). Before that one could only use variables (@var_name) as parameters.

Contents

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

Description

After preparing a statement with `PREPARE`, you execute it with an `EXECUTE` statement that refers to the prepared statement name. If the prepared statement contains any parameter markers, you must supply a `USING` clause that lists user variables containing the values to be bound to the parameters. Parameter values can be supplied only by user variables, and the `USING` clause must name exactly as many variables as the number of parameter markers in the statement.

You can execute a given prepared statement multiple times, passing different variables to it or setting the variables to different values before each execution.

If the specified statement has not been PREPARED, an error similar to the following is produced:

```
ERROR 1243 (HY000): Unknown prepared statement handler (stmt_name) given to EXECUTE
```

Example

See [example in PREPARE](#).

See Also

- [EXECUTE IMMEDIATE](#)

SAVEPOINT

Syntax

```
SAVEPOINT identifier  
ROLLBACK [WORK] TO [SAVEPOINT] identifier  
RELEASE SAVEPOINT identifier
```

Description

InnoDB supports the SQL statements `SAVEPOINT`, `ROLLBACK TO SAVEPOINT`, `RELEASE SAVEPOINT` and the optional `WORK` keyword for `ROLLBACK`.

Each savepoint must have a legal [MariaDB identifier](#). A savepoint is a named sub-transaction.

Normally `ROLLBACK` undoes the changes performed by the whole transaction. When used with the `TO` clause, it undoes the changes performed after the specified savepoint, and erases all subsequent savepoints. However, all locks that have been acquired after the save point will survive. `RELEASE SAVEPOINT` does not rollback or commit any changes, but removes the specified savepoint.

When the execution of a trigger or a stored function begins, it is not possible to use statements which reference a savepoint which was defined from out of that stored program.

When a `COMMIT` (including implicit commits) or a `ROLLBACK` statement (with no `TO` clause) is performed, they act on the whole transaction, and all savepoints are removed.

Errors

If `COMMIT` or `ROLLBACK` is issued and no transaction was started, no error is reported.

If `SAVEPOINT` is issued and no transaction was started, no error is reported but no savepoint is created. When `ROLLBACK TO SAVEPOINT` or `RELEASE SAVEPOINT` is called for a savepoint that does not exist, an error like this is issued:

```
ERROR 1305 (42000): SAVEPOINT svp_name does not exist
```

UNLOCK TABLES

Syntax

```
UNLOCK TABLES
```

Description

`UNLOCK TABLES` explicitly releases any table locks held by the current session. See [LOCK TABLES](#) for more information.

In addition to releasing table locks acquired by the [LOCK TABLES](#) statement, the `UNLOCK TABLES` statement also releases the global read lock acquired by the `FLUSH TABLES WITH READ LOCK` statement. The `FLUSH TABLES WITH READ LOCK` statement is very useful for performing backups. See [FLUSH](#) for more information about `FLUSH TABLES WITH READ LOCK`.

XA Transactions

Overview

The MariaDB XA implementation is based on the X/Open CAE document Distributed Transaction Processing: The XA Specification. This document is published by The Open Group and available at <http://www.opengroup.org/public/pubs/catalog/c193.htm>.

XA transactions are designed to allow distributed transactions, where a transaction manager (the application) controls a transaction which involves multiple resources. Such resources are usually DBMSs, but could be resources of any type. The whole set of required transactional operations is called a global transaction. Each subset of operations which involve a single resource is called a local transaction. XA uses a 2-phases commit (2PC). With the first commit, the transaction manager tells each resource to prepare an effective commit, and waits for a confirm message. The changes are not still made effective at this point. If any of

Contents

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

Contents

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

Contents

1. [Overview](#)
2. [Internal XA vs External XA](#)
3. [Transaction Coordinator Log](#)
4. [Syntax](#)
5. [XA RECOVER](#)
6. [Examples](#)
7. [Known Issues](#)
 1. [MariaDB Galera Cluster](#)

the resources encountered an error, the transaction manager will rollback the global transaction. If all resources communicate that the first commit is successful, the transaction manager can require a second commit, which makes the changes effective.

In MariaDB, XA transactions can only be used with storage engines that support them. At least [InnoDB](#), [TokuDB](#), [SPIDER](#) and [MyRocks](#) support them. For InnoDB, until [MariaDB 10.2](#), XA transactions can be disabled by setting the `innodb_support_xa` server system variable to 0. From [MariaDB 10.3](#), XA transactions are always supported.

Like regular transactions, XA transactions create [metadata locks](#) on accessed tables.

XA transactions require [REPEATABLE READ](#) as a minimum isolation level. However, distributed transactions should always use [SERIALIZABLE](#).

Trying to start more than one XA transaction at the same time produces a 1400 error ([SQLSTATE 'XAE09'](#)). The same error is produced when attempting to start an XA transaction while a regular transaction is in effect. Trying to start a regular transaction while an XA transaction is in effect produces a 1399 error ([SQLSTATE 'XAE07'](#)).

The [statements that cause an implicit COMMIT](#) for regular transactions produce a 1400 error ([SQLSTATE 'XAE09'](#)) if a XA transaction is in effect.

Internal XA vs External XA

XA transactions are an overloaded term in MariaDB. If a [storage engine](#) is XA-capable, it can mean one or both of these:

- It supports MariaDB's internal two-phase commit API. This is transparent to the user. Sometimes this is called "internal XA", since MariaDB's internal [transaction coordinator log](#) can handle coordinating these transactions.
- It supports XA transactions, with the `XA START`, `XA PREPARE`, `XA COMMIT`, etc. statements. Sometimes this is called "external XA", since it requires the use of an external transaction coordinator to use this feature properly.

Transaction Coordinator Log

If you have two or more XA-capable storage engines enabled, then a transaction coordinator log must be available.

There are currently two implementations of the transaction coordinator log:

- Binary log-based transaction coordinator log
- Memory-mapped file-based transaction coordinator log

If the [binary log](#) is enabled on a server, then the server will use the binary log-based transaction coordinator log. Otherwise, it will use the memory-mapped file-based transaction coordinator log.

See [Transaction Coordinator Log](#) for more information.

Syntax

```
XA {START|BEGIN} xid [JOIN|RESUME]  
  
XA END xid [SUSPEND [FOR MIGRATE]]  
  
XA PREPARE xid  
  
XA COMMIT xid [ONE PHASE]  
  
XA ROLLBACK xid  
  
XA RECOVER [FORMAT=['RAW' | 'SQL']]  
  
xid: gtrid [, bqual [, formatID ]]
```

The interface to XA transactions is a set of SQL statements starting with `XA`. Each statement changes a transaction's state, determining which actions it can perform. A transaction which does not exist is in the `NON-EXISTING` state.

`XA START` (or `BEGIN`) starts a transaction and defines its `xid` (a transaction identifier). The `JOIN` or `RESUME` keywords have no effect. The new transaction will be in `ACTIVE` state.

The `xid` can have 3 components, though only the first one is mandatory. `gtrid` is a quoted string representing a global transaction identifier. `bqual` is a quoted string representing a local transaction identifier. `formatID` is an unsigned integer indicating the format used for the first two components; if not specified, defaults to 1. MariaDB does not interpret in any way these components, and only uses them to identify a transaction. `xid`s of transactions in effect must be unique.

`XA END` declares that the specified `ACTIVE` transaction is finished and it changes its state to `IDLE`. `SUSPEND [FOR MIGRATE]` has no effect.

XA PREPARE prepares an IDLE transaction for commit, changing its state to PREPARED . This is the first commit.

XA COMMIT definitely commits and terminates a transaction which has already been PREPARED . If the ONE PHASE clause is specified, this statement performs a 1-phase commit on an IDLE transaction.

XA ROLLBACK rolls back and terminates an IDLE or PREPARED transaction.

XA RECOVER shows information about all PREPARED transactions.

When trying to execute an operation which is not allowed for the transaction's current state, an error is produced:

```
XA COMMIT 'test' ONE PHASE;
ERROR 1399 (XAE07): XAER_RMFAIL: The command cannot be executed when global transaction is in the ACTIVE state

XA COMMIT 'test2';
ERROR 1399 (XAE07): XAER_RMFAIL: The command cannot be executed when global transaction is in the NON-EXISTING state
```

XA RECOVER

The XA RECOVER statement shows information about all transactions which are in the PREPARED state. It does not matter which connection created the transaction: if it has been PREPARED , it appears. But this does not mean that a connection can commit or rollback a transaction which was started by another connection. Note that transactions using a 1-phase commit are never in the PREPARED state, so they cannot be shown by XA RECOVER .

XA RECOVER produces four columns:

```
XA RECOVER;
+-----+-----+-----+-----+
| formatID | gtrid_length | bqual_length | data |
+-----+-----+-----+-----+
|      1 |          4 |          0 | test |
+-----+-----+-----+-----+
```

MariaDB starting with 10.3.3

You can use XA RECOVER FORMAT='SQL' to get the data in a human readable form that can be directly copy-pasted into XA COMMIT or XA ROLLBACK . This is particularly useful for binary xid generated by some transaction coordinators.

formatID is the formatID part of xid .

data are the gtrid and bqual parts of xid , concatenated.

gtrid_length and bqual_length are the lengths of gtrid and bqual , respectively.

Examples

2-phases commit:

```
XA START 'test';

INSERT INTO t VALUES (1,2);

XA END 'test';

XA PREPARE 'test';

XA COMMIT 'test';
```

1-phase commit:

```

XA START 'test';

INSERT INTO t VALUES (1,2);

XA END 'test';

XA COMMIT 'test' ONE PHASE;

```

Human-readable:

```

xa start '12\r34\t67\v78', 'abc\ndef', 3;

insert t1 values (40);

xa end '12\r34\t67\v78', 'abc\ndef', 3;

xa prepare '12\r34\t67\v78', 'abc\ndef', 3;

xa recover format='RAW';
+-----+-----+-----+
| formatID | gtrid_length | bqual_length | data           |
+-----+-----+-----+
| 34      |       11        |          7   | 12
def |
+-----+-----+-----+

xa recover format='SQL';
+-----+-----+-----+
| formatID | gtrid_length | bqual_length | data           |
+-----+-----+-----+
| 3       |       11        |          7   | X'31320d3334093637763738',X'6162630a646566',3 |
+-----+-----+-----+

xa rollback X'31320d3334093637763738',X'6162630a646566',3;

```

Known Issues

MariaDB Galera Cluster

MariaDB Galera Cluster does not support XA transactions.

However, MariaDB Galera Cluster builds include a built-in plugin called `wsrep`. Prior to MariaDB 10.4.3, this plugin was internally considered an **XA-capable storage engine**. Consequently, these MariaDB Galera Cluster builds have multiple XA-capable storage engines by default, even if the only "real" storage engine that supports external XA transactions enabled on these builds by default is `InnoDB`. Therefore, when using one of these builds MariaDB would be forced to use a `transaction coordinator log` by default, which could have performance implications.

See [Transaction Coordinator Log Overview: MariaDB Galera Cluster](#) for more information.

Authentication from MariaDB 10.4

MariaDB starting with 10.4

MariaDB 10.4 introduces a number of changes to the authentication process, intended to make things easier and more intuitive.

Overview

There are four **new main features in 10.4** relating to authentication:

- 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.
- The `root@localhost` user account created by `mysql_install_db` is created with the ability to use two `authentication plugins`.
 - First, it is configured to try to use the `unix_socket` authentication plugin. This allows the `root@localhost` user to login without a password via the local Unix socket file defined by the `socket` system variable, as long as the login is attempted from a process owned by the operating system `root` user account.

- Second, if authentication fails with the `unix_socket` authentication plugin, then it is configured to try to use the `mysql_native_password` authentication plugin. However, an invalid password is initially set, so in order to authenticate this way, a password must be set with `SET PASSWORD`.
- However, just using the `unix_socket` authentication plugin may be fine for many users, and it is very secure. You may want to try going without password authentication to see how well it works for you. Remember, the best way to keep your password safe is not to have one!
- All user accounts, passwords, and global privileges are now stored in the `mysql.global_priv` table. The `mysql.user` table still exists and has exactly the same set of columns as before, but it's now a view that references the `mysql.global_priv` table. Tools that analyze the `mysql.user` table should continue to work as before. From MariaDB 10.4.13, the dedicated `mariadb.sys` user is created as the definer of this view. Previously `root` was the definer, which resulted in privilege problems when this username was changed.
- MariaDB 10.4 adds support for [User Password Expiry](#), which is not active by default.

Contents

1. [Overview](#)
2. [Description](#)
3. [Cookbook](#)
4. [Reverting to the Previous Authentication Method for root@localhost](#)
 1. [Configuring mysql_install_db to Revert to the Previous Authentication Method](#)
 2. [Altering the User Account to Revert to the Previous Authentication Method](#)
5. [See Also](#)

Description

As a result of the above changes, the open-for-everyone all-powerful root account is finally gone. And installation scripts will no longer demand that you “PLEASE REMEMBER TO SET A PASSWORD FOR THE MariaDB root USER !”, because the root account is securely created automatically.

Two all-powerful accounts are created by default — root and the OS user that owns the data directory, typically `mysql`. They are created as:

```
CREATE USER root@localhost IDENTIFIED VIA unix_socket OR mysql_native_password USING 'invalid'
CREATE USER mysql@localhost IDENTIFIED VIA unix_socket OR mysql_native_password USING 'invalid'
```

Using `unix_socket` means that if you are the system root user, you can log in as `root@localhost` without a password. This technique was pioneered by Otto Kekäläinen in Debian MariaDB packages and has been successfully [used in Debian](#) since as early as MariaDB 10.0.

It is based on a simple fact that asking the system root for a password adds no extra security — root has full access to all the data files and all process memory anyway. But not asking for a password means, there is no root password to forget (no need for the numerous tutorials on “how to reset MariaDB root password”). And if you want to script some tedious database work, there is no need to store the root password in plain text for the script to use (no need for `debian-sys-maint` user).

Still, some users may wish to log in as MariaDB root without using sudo. Hence the old authentication method — conventional MariaDB password — is still available. By default it is disabled (“invalid” is not a valid password hash), but one can set the password with a usual `SET PASSWORD` statement. And still retain the password-less access via sudo.

If you install MariaDB locally (say from a tarball), you would not want to use sudo to be able to log in. This is why MariaDB creates a second all-powerful user with the same name as a system user that owns the data directory. In local (not system-wide) installations, this will be the user who installed MariaDB — they automatically get convenient password-less root-like access, because they can access all the data files anyway.

Even if MariaDB is installed system-wide, you may not want to run your database maintenance scripts as system root — now you can run them as system `mysql` user. And you will know that they will never destroy your entire system, even if you make a typo in a shell script.

However, seasoned MariaDB DBAs who are used to the old ways do need to make some changes. See the examples below for common tasks.

Cookbook

After installing MariaDB system-wide the first thing you've got used to doing is logging in into the unprotected root account and protecting it, that is, setting the root password:

```
$ sudo dnf install MariaDB-server
$ mysql -uroot
...
MariaDB> set password = password("XH4VmT3_jt");
```

This is not only unnecessary now, it will simply not work — there is no unprotected root account. To log in as root use

```
$ sudo dnf install MariaDB-server
$ sudo mysql
```

Note that it implies you are connecting via the unix socket, not tcp. If you happen to have `protocol=tcp` in a system-wide `/etc/my.cnf` file, use `sudo mysql --protocol=socket`.

After installing MariaDB locally you've also used to connect to the unprotected root account using `mysql -uroot`. This will not work either, simply use `mysql` without specifying a username.

If you've forgotten your root password, no problem — you can still connect using sudo and change the password. And if you've also removed unix_socket authentication, to restore access do as follows:

- restart MariaDB with `--skip-grant-tables`
- login into the unprotected server
- run `FLUSH PRIVILEGES` (note, before 10.4 this would've been the last step, not anymore). This disables `--skip-grant-tables` and allows you to change the stored authentication method
- run `SET PASSWORD FOR root@localhost` to change the root password.

To view inside privilege tables, the old `mysql.user` table still exists. You can select from it as before, although you cannot update it anymore. It doesn't show alternative authentication plugins and this was one of the reasons for switching to the `mysql.global_priv` table — complex authentication rules did not fit into rigid structure of a relational table. You can select from the new table, for example:

```
select concat(user, '@', host, ' => ', json_detailed(priv)) from mysql.global_priv;
```

Reverting to the Previous Authentication Method for `root@localhost`

If you don't want the `root@localhost` user account created by `mysql_install_db` to use `unix_socket` authentication by default, then there are a few ways to revert to the previous `mysql_native_password` authentication method for this user account.

Configuring `mysql_install_db` to Revert to the Previous Authentication Method

One way to revert to the previous `mysql_native_password` authentication method for the `root@localhost` user account is to execute `mysql_install_db` with a special option. If `mysql_install_db` is executed while `--auth-root-authentication-method=normal` is specified, then it will create the default user accounts using the default behavior of [MariaDB 10.3](#) and before.

This means that the `root@localhost` user account will use `mysql_native_password` authentication by default. There are some other differences as well. See [mysql_install_db: User Accounts Created by Default](#) for more information.

For example, the option can be set on the command-line while running `mysql_install_db`:

```
mysql_install_db --user=mysql --datadir=/var/lib/mysql --auth-root-authentication-method=normal
```

The option can also be set in an [option file](#) in an [option group](#) supported by `mysql_install_db`. For example:

```
[mysql_install_db]
auth_root_authentication_method=normal
```

If the option is set in an [option file](#) and if `mysql_install_db` is executed, then `mysql_install_db` will read this option from the [option file](#), and it will automatically set this option.

Altering the User Account to Revert to the Previous Authentication Method

If you have already installed MariaDB, and if the `root@localhost` user account is already using `unix_socket` authentication, then you can revert to the old `mysql_native_password` authentication method for the user account by executing the following:

```
ALTER USER root@localhost IDENTIFIED VIA mysql_native_password USING PASSWORD("verysecret")
```

See Also

- [Authentication from MariaDB 10.4 video tutorial](#)
- [Authentication in MariaDB 10.4 — understanding the changes \(mariadb.org\)](#)

ST_X

Syntax

```
ST_X(p)
X(p)
```

Description

Returns the X-coordinate value for the point `p` as a double-precision number.

`ST_X()` and `X()` are synonyms.

Examples

```
SET @pt = 'Point(56.7 53.34)';

SELECT X(GeomFromText(@pt));
+-----+
| X(GeomFromText(@pt)) |
+-----+
|      56.7 |
+-----+
```

ST_Y

Syntax

```
ST_Y(p)
Y(p)
```

Description

Returns the Y-coordinate value for the point `p` as a double-precision number.

`ST_Y()` and `Y()` are synonyms.

Examples

```
SET @pt = 'Point(56.7 53.34)';

SELECT Y(GeomFromText(@pt));
+-----+
| Y(GeomFromText(@pt)) |
+-----+
|      53.34 |
+-----+
```

EXECUTE IMMEDIATE

MariaDB starting with [10.2.3](#)

`EXECUTE IMMEDIATE` was introduced in [MariaDB 10.2.3](#).

Syntax

```
EXECUTE IMMEDIATE statement
```

Description

EXECUTE IMMEDIATE executes a dynamic SQL statement created on the fly, which can reduce performance overhead.

For example:

```
EXECUTE IMMEDIATE 'SELECT 1'
```

which is shorthand for:

```
prepare stmt from "select 1";
execute stmt;
deallocate prepare stmt;
```

EXECUTE IMMEDIATE supports complex expressions as prepare source and parameters:

```
EXECUTE IMMEDIATE CONCAT('SELECT COUNT(*) FROM ', 't1', ' WHERE a=?') USING 5+5;
```

Limitations: subselects and stored function calls are not supported as a prepare source.

The following examples return an error:

```
CREATE OR REPLACE FUNCTION f1() RETURNS VARCHAR(64) RETURN 'SELECT * FROM t1';
EXECUTE IMMEDIATE f1();
ERROR 1970 (42000): EXECUTE IMMEDIATE does not support subqueries or stored functions

EXECUTE IMMEDIATE (SELECT 'SELECT * FROM t1');
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that
corresponds to your MariaDB server version for the right syntax to use near
'SELECT 'SELECT * FROM t1'' at line 1

CREATE OR REPLACE FUNCTION f1() RETURNS INT RETURN 10;
EXECUTE IMMEDIATE 'SELECT * FROM t1 WHERE a=?' USING f1();
ERROR 1970 (42000): EXECUTE..USING does not support subqueries or stored functions

EXECUTE IMMEDIATE 'SELECT * FROM t1 WHERE a=?' USING (SELECT 10);
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that
corresponds to your MariaDB server version for the right syntax to use near
'SELECT 10)' at line 1
```

One can use a user or an SP variable as a workaround:

```
CREATE OR REPLACE FUNCTION f1() RETURNS VARCHAR(64) RETURN 'SELECT * FROM t1';
SET @stmt=f1();
EXECUTE IMMEDIATE @stmt;

SET @stmt=(SELECT 'SELECT 1');
EXECUTE IMMEDIATE @stmt;

CREATE OR REPLACE FUNCTION f1() RETURNS INT RETURN 10;
SET @param=f1();
EXECUTE IMMEDIATE 'SELECT * FROM t1 WHERE a=?' USING @param;

SET @param=(SELECT 10);
EXECUTE IMMEDIATE 'SELECT * FROM t1 WHERE a=?' USING @param;
```

EXECUTE IMMEDIATE supports user variables and SP variables as OUT parameters

```

DELIMITER $$ 
CREATE OR REPLACE PROCEDURE p1(OUT a INT)
BEGIN
    SET a:= 10;
END;
$$
DELIMITER ;
SET @a=2;
EXECUTE IMMEDIATE 'CALL p1(?)' USING @a;
SELECT @a;
+----+
| @a |
+----+
|   10 |
+----+

```

Similar to PREPARE, EXECUTE IMMEDIATE is allowed in stored procedures but is not allowed in stored functions.

This example uses EXECUTE IMMEDIATE inside a stored procedure:

```

DELIMITER $$ 
CREATE OR REPLACE PROCEDURE p1()
BEGIN
    EXECUTE IMMEDIATE 'SELECT 1';
END;
$$
DELIMITER ;
CALL p1;
+---+
| 1 |
+---+
| 1 |
+---+

```

This script returns an error:

```

DELIMITER $$ 
CREATE FUNCTION f1() RETURNS INT
BEGIN
    EXECUTE IMMEDIATE 'DO 1';
    RETURN 1;
END;
$$
ERROR 1336 (0A000): Dynamic SQL is not allowed in stored function or trigger

```

EXECUTE IMMEDIATE can use DEFAULT and IGNORE indicators as bind parameters:

```

CREATE OR REPLACE TABLE t1 (a INT DEFAULT 10);
EXECUTE IMMEDIATE 'INSERT INTO t1 VALUES (?)' USING DEFAULT;
SELECT * FROM t1;
+----+
| a |
+----+
|   10 |
+----+

```

EXECUTE IMMEDIATE increments the [Com_execute_immediate](#) status variable, as well as the [Com_stmt_prepare](#), [Com_stmt_execute](#) and [Com_stmt_close](#) status variables.

Note, EXECUTE IMMEDIATE does not increment the [Com_execute_sql](#) status variable. [Com_execute_sql](#) is used only for [PREPARE...EXECUTE](#).

This session screenshot demonstrates how EXECUTE IMMEDIATE affects status variables:

```

SELECT * FROM INFORMATION_SCHEMA.SESSION_STATUS WHERE VARIABLE_NAME RLIKE
('COM_(EXECUTE|STMT_PREPARE|STMT_EXECUTE|STMT_CLOSE)');

+-----+-----+
| VARIABLE_NAME | VARIABLE_VALUE |
+-----+-----+
| COM_EXECUTE_IMMEDIATE | 0 |
| COM_EXECUTE_SQL | 0 |
| COM_STMT_CLOSE | 0 |
| COM_STMT_EXECUTE | 0 |
| COM_STMT_PREPARE | 0 |
+-----+-----+

EXECUTE IMMEDIATE 'SELECT 1';
+---+
| 1 |
+---+
| 1 |
+---+


SELECT * FROM INFORMATION_SCHEMA.SESSION_STATUS WHERE VARIABLE_NAME RLIKE
('COM_(EXECUTE|STMT_PREPARE|STMT_EXECUTE|STMT_CLOSE)');
+-----+-----+
| VARIABLE_NAME | VARIABLE_VALUE |
+-----+-----+
| COM_EXECUTE_IMMEDIATE | 1 |
| COM_EXECUTE_SQL | 0 |
| COM_STMT_CLOSE | 1 |
| COM_STMT_EXECUTE | 1 |
| COM_STMT_PREPARE | 1 |
+-----+-----+

```

Metadata Locking

MariaDB supports metadata locking. This means that when a transaction (including [XA transactions](#)) uses a table, it locks its metadata until the end of transaction. Non-transactional tables are also locked, as well as views and objects which are related to locked tables/views (stored functions, triggers, etc). When a connection tries to use a DDL statement (like an [ALTER TABLE](#)) which modifies a table that is locked, that connection is queued, and has to wait until it's unlocked. Using savepoints and performing a partial rollback does not release metadata locks.

[LOCK TABLES ... WRITE](#) are also queued. Some wrong statements which produce an error may not need to wait for the lock to be freed.

The metadata lock's timeout is determined by the value of the [lock_wait_timeout](#) server system variable (in seconds). However, note that its default value is 31536000 (1 year, MariaDB <= 10.2.3), or 86400 (1 day, MariaDB >= 10.2.4). If this timeout is exceeded, the following error is returned:

```
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
```

If the [metadata_lock_info](#) plugin is installed, the [Information Schema metadata_lock_info](#) table stores information about existing metadata locks.

MariaDB starting with 10.5.2

From [MariaDB 10.5](#), the [Performance Schema metadata_locks](#) table contains metadata lock information.

Example

Let's use the following MEMORY (non-transactional) table:

```
CREATE TABLE t (a INT) ENGINE = MEMORY;
```

Connection 1 starts a transaction, and INSERTs a row into t:

```
START TRANSACTION;

INSERT INTO t SET a=1;
```

`t`'s metadata is now locked by connection 1. Connection 2 tries to alter `t`, but has to wait:

```
ALTER TABLE t ADD COLUMN b INT;
```

Connection 2's prompt is blocked now.

Now connection 1 ends the transaction:

```
COMMIT;
```

...and connection 2 finally gets the output of its command:

```
Query OK, 1 row affected (35.23 sec)
Records: 1 Duplicates: 0 Warnings: 0
```

X

A synonym for [ST_X](#).

Y

A synonym for [ST_Y](#).

AES_DECRYPT

Syntax

```
AES_DECRYPT(crypt_str,key_str)
```

Description

This function allows decryption of data using the official AES (Advanced Encryption Standard) algorithm. For more information, see the description of [AES_ENCRYPT\(\)](#).

DECODE

Syntax

```
DECODE(crypt_str,pass_str)
```

In Oracle mode from MariaDB 10.3.2:

```
DECODE(expr, search_expr, result_expr [, search_expr2, result_expr2 ...] [default_expr])
```

In all modes from MariaDB 10.3.2:

```
DECODE_ORACLE(expr, search_expr, result_expr [, search_expr2, result_expr2 ...] [default_expr])
```

Description

In the default mode, `DECODE` decrypts the encrypted string `crypt_str` using `pass_str` as the password. `crypt_str` should be a string returned from [ENCODE\(\)](#). The resulting string will be the original string only if `pass_str` is the same.

In Oracle mode from MariaDB 10.3.2, `DECODE` compares `expr` to the search expressions, in order. If it finds a match, the corresponding result expression is returned. If no matches are found, the default expression is returned, or NULL if no default is provided.

NULLs are treated as equivalent.

`DECODE_ORACLE` is a synonym for the Oracle-mode version of the function, and is available in all modes.

Examples

From MariaDB 10.3.2:

```
SELECT DECODE_ORACLE(2+1,3*1,'found1',3*2,'found2','default');
+-----+
| DECODE_ORACLE(2+1,3*1,'found1',3*2,'found2','default') |
+-----+
| found1 |
+-----+

SELECT DECODE_ORACLE(2+4,3*1,'found1',3*2,'found2','default');
+-----+
| DECODE_ORACLE(2+4,3*1,'found1',3*2,'found2','default') |
+-----+
| found2 |
+-----+

SELECT DECODE_ORACLE(2+2,3*1,'found1',3*2,'found2','default');
+-----+
| DECODE_ORACLE(2+2,3*1,'found1',3*2,'found2','default') |
+-----+
| default |
+-----+
```

Nulls are treated as equivalent:

```
SELECT DECODE_ORACLE(NULL,NULL,'Nulls are equivalent','Nulls are not equivalent');
+-----+
| DECODE_ORACLE(NULL,NULL,'Nulls are equivalent','Nulls are not equivalent') |
+-----+
| Nulls are equivalent |
+-----+
```

LOCK TABLES

Syntax

```
LOCK TABLE[S]
  tbl_name [[AS] alias] lock_type
  [, tbl_name [[AS] alias] lock_type] ...
  [WAIT n|NOWAIT]

lock_type:
  READ [LOCAL]
  | [LOW_PRIORITY] WRITE
  | WRITE CONCURRENT

UNLOCK TABLES
```

Description

The `lock_type` can be one of:

Option	Description
READ	Read lock, no writes allowed
READ LOCAL	Read lock, but allow concurrent inserts
WRITE	Exclusive write lock. No other connections can read or write to this table

Contents

1. [Syntax](#)
2. [Description](#)
 1. [WAIT/NOWAIT](#)
 3. [Limitations](#)
 4. [See Also](#)

LOW_PRIORITY WRITE	Exclusive write lock, but allow new read locks on the table until we get the write lock.
WRITE CONCURRENT	Exclusive write lock, but allow READ LOCAL locks to the table.

MariaDB enables client sessions to acquire table locks explicitly for the purpose of cooperating with other sessions for access to tables, or to prevent other sessions from modifying tables during periods when a session requires exclusive access to them. A session can acquire or release locks only for itself. One session cannot acquire locks for another session or release locks held by another session.

Locks may be used to emulate transactions or to get more speed when updating tables.

`LOCK TABLES` explicitly acquires table locks for the current client session. Table locks can be acquired for base tables or views. To use `LOCK TABLES`, you must have the `LOCK TABLES` privilege, and the `SELECT` privilege for each object to be locked. See [GRANT](#)

For view locking, `LOCK TABLES` adds all base tables used in the view to the set of tables to be locked and locks them automatically. If you lock a table explicitly with `LOCK TABLES`, any tables used in triggers are also locked implicitly, as described in [Triggers and Implicit Locks](#).

`UNLOCK TABLES` explicitly releases any table locks held by the current session.

MariaDB starting with [10.3.0](#)

WAIT/NOWAIT

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

Limitations

- `LOCK TABLES` [doesn't work when using Galera cluster](#). You may experience crashes or locks when used with Galera.
- `LOCK TABLES` works on XtraDB/InnoDB tables only if the `innodb_table_locks` system variable is set to 1 (the default) and `autocommit` is set to 0 (1 is default). Please note that no error message will be returned on `LOCK TABLES` with `innodb_table_locks = 0`.
- `LOCK TABLES` [implicitly commits](#) the active transaction, if any. Also, starting a transaction always releases all table locks acquired with `LOCK TABLES`. This means that there is no way to have table locks and an active transaction at the same time. The only exceptions are the transactions in `autocommit` mode. To preserve the data integrity between transactional and non-transactional tables, the `GET_LOCK()` function can be used.
- When using `LOCK TABLES` on a `TEMPORARY` table, it will always be locked with a `WRITE` lock.
- While a connection holds an explicit read lock on a table, it cannot modify it. If you try, the following error will be produced:

ERROR 1099 (HY000): Table 'tab_name' was locked with a READ lock and can't be updated

- While a connection holds an explicit lock on a table, it cannot access a non-locked table. If you try, the following error will be produced:

ERROR 1100 (HY000): Table 'tab_name' was not locked with LOCK TABLES

- While a connection holds an explicit lock on a table, it cannot issue the following: `INSERT DELAYED`, `CREATE TABLE`, `CREATE TABLE ... LIKE`, and DDL statements involving stored programs and views (except for triggers). If you try, the following error will be produced:

ERROR 1192 (HY000): Can't execute the given command because you have active locked tables or an active transaction

- `LOCK TABLES` can not be used in stored routines - if you try, the following error will be produced on creation:

ERROR 1314 (0A000): LOCK is not allowed in stored procedures

See Also

- [UNLOCK TABLES](#)

PURGE BINARY LOGS

Syntax

```
PURGE { BINARY | MASTER } LOGS  
{ TO 'log_name' | BEFORE datetime_expr }
```

Description

The `PURGE BINARY LOGS` statement deletes all the [binary log](#) files listed in the log index file prior to the specified log file name or date. `BINARY` and `MASTER` are synonyms. Deleted log files also are removed from the list recorded in the index file, so that the given log file becomes the first in the list.

The datetime expression is in the format 'YYYY-MM-DD hh:mm:ss'.

If a replica is active but has yet to read from a binary log file you attempt to delete, the statement will fail with an error. However, if the replica is not connected and has yet to read from a log file you delete, the file will be deleted, but the replica will be unable to continue replicating once it connects again.

This statement has no effect if the server was not started with the `--log-bin` option to enable binary logging.

To list the binary log files on the server, use [SHOW BINARY LOGS](#). To see which files they are reading, use [SHOW SLAVE STATUS](#) (or [SHOW REPLICAS STATUS](#) from [MariaDB 10.5.1](#)). You can only delete the files that are older than the oldest file that is used by the slaves.

To delete all binary log files, use [RESET MASTER](#). To move to a new log file (for example if you want to remove the current log file), use [FLUSH LOGS](#) before you execute `PURGE LOGS`.

If the `expire_logs_days` server system variable is not set to 0, the server automatically deletes binary log files after the given number of days. From [MariaDB 10.6](#), the `binlog_expire_logs_seconds` variable allows more precise control over binlog deletion, and takes precedence if both are non-zero.

Requires the [SUPER](#) privilege or, from [MariaDB 10.5.2](#), the [BINLOG ADMIN](#) privilege, to run.

Examples

```
PURGE BINARY LOGS TO 'mariadb-bin.000063';
```

```
PURGE BINARY LOGS BEFORE '2013-04-21';
```

```
PURGE BINARY LOGS BEFORE '2013-04-22 09:55:22';
```

See Also

- [Using and Maintaining the Binary Log](#)
- [FLUSH LOGS](#).

MD5

Syntax

```
MD5(str)
```

Description

Calculates an MD5 128-bit checksum for the string.

The return value is a 32-hex digit string, and as of [MariaDB 5.5](#), is a nonbinary string in the connection [character set and collation](#), determined by the values of the [character_set_connection](#) and [collation_connection](#) system variables. Before 5.5, the return value was a binary string.

NULL is returned if the argument was NULL.

Examples

```
SELECT MD5('testing');
+-----+
| MD5('testing') |
+-----+
| ae2b1fc... |
+-----+
```

PASSWORD

Syntax

```
PASSWORD(str)
```

Description

The `PASSWORD()` function is used for hashing passwords for use in authentication by the MariaDB server. It is not intended for use in other applications.

Contents

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

Calculates and returns a hashed password string from the plaintext password *str*. Returns an empty string (>= [MariaDB 10.0.4](#)) if the argument was `NULL`.

The return value is a nonbinary string in the connection [character set and collation](#), determined by the values of the `character_set_connection` and `collation_connection` system variables.

This is the function that is used for hashing MariaDB passwords for storage in the Password column of the [user table](#) (see [privileges](#)), usually used with the `SET PASSWORD` statement. It is not intended for use in other applications.

Until [MariaDB 10.3](#), the return value is 41-bytes in length, and the first character is always `''`. From [MariaDB 10.4](#), the function takes into account the authentication plugin where applicable (A `CREATE USER` or `SET PASSWORD` statement). For example, when used in conjunction with a user authenticated by the [ed25519 plugin](#), the statement will create a longer hash:

```
CREATE USER edtest@localhost IDENTIFIED VIA ed25519 USING PASSWORD('secret');

CREATE USER edtest2@localhost IDENTIFIED BY 'secret';

SELECT CONCAT(user, '@', host, ' => ', JSON_DETAILED(priv)) FROM mysql.global_priv
 WHERE user LIKE 'edtest%'\G
 *****
 1. row *****
 CONCAT(user, '@', host, ' => ', JSON_DETAILED(priv)): edtest@localhost => {
...
  "plugin": "ed25519",
  "authentication_string": "ZIgUREUg5PVgQ6LskhXm0+eZLS0nC8be6HPjYWR4YJY",
...
}
*****
 2. row *****
 CONCAT(user, '@', host, ' => ', JSON_DETAILED(priv)): edtest2@localhost => {
...
  "plugin": "mysql_native_password",
  "authentication_string": "*14E65567ABDB5135D0CFD9A70B3032C179A49EE7",
...
}
```

The behavior of this function is affected by the value of the `old_passwords` system variable. If this is set to `1` (`0` is default), MariaDB reverts to using the [mysql_old_password authentication plugin](#) by default for newly created users and passwords.

Examples

```
SELECT PASSWORD('notagoodpwd');
+-----+
| PASSWORD('notagoodpwd') |
+-----+
| *3A70EE9FC6594F88CE9E959CD51C5A1C002DC937 |
+-----+
```

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

See Also

- [Password Validation Plugins](#) - permits the setting of basic criteria for passwords
- [OLD_PASSWORD\(\)](#) - pre-MySQL 4.1 password function

ENDPOINT

A synonym for [ST_ENDPOINT](#).

LENGTH

Syntax

```
GLength(ls)
```

Description

Returns as a double-precision number the length of the [LineString](#) value *ls* in its associated spatial reference.

Examples

```
SET @ls = 'LineString(1 1,2 2,3 3)';
SELECT GLength(GeomFromText(@ls));
+-----+
| GLength(GeomFromText(@ls)) |
+-----+
| 2.82842712474619 |
+-----+
```

See Also

- [ST_LENGTH\(\)](#) is the OpenGIS equivalent.

ROLLBACK

The `ROLLBACK` statement rolls back (ends) a transaction, destroying any changes to SQL-data so that they never become visible to subsequent transactions. The required syntax for the `ROLLBACK` statement is as follows.

```
ROLLBACK [ WORK ] [ AND [ NO ] CHAIN ]
[ TO [ SAVEPOINT ] {<savepoint name> | <simple target specification>} ]
```

The `ROLLBACK` statement will either end a transaction, destroying all data changes that happened during any of the transaction, or it will just destroy any data changes that happened since you established a savepoint. The basic form of the `ROLLBACK` statement is just the keyword `ROLLBACK` (the keyword `WORK` is simply noise and can be omitted without changing the effect).

The optional `AND CHAIN` clause is a convenience for initiating a new transaction as soon as the old transaction terminates. If `AND CHAIN` is specified, then there is effectively nothing between the old and new transactions, although they remain separate. The characteristics of the new transaction will

be the same as the characteristics of the old one — that is, the new transaction will have the same access mode, isolation level and diagnostics area size (we'll discuss all of these shortly) as the transaction just terminated. The `AND NO CHAIN` option just tells your DBMS to end the transaction — that is, these four SQL statements are equivalent:

```
ROLLBACK;  
ROLLBACK WORK;  
ROLLBACK AND NO CHAIN;  
ROLLBACK WORK AND NO CHAIN;
```

All of them end a transaction without saving any transaction characteristics. The only other options, the equivalent statements:

```
ROLLBACK AND CHAIN;  
ROLLBACK WORK AND CHAIN;
```

both tell your DBMS to end a transaction, but to save that transaction's characteristics for the next transaction.

`ROLLBACK` is much simpler than `COMMIT` : it may involve no more than a few deletions (of Cursors, locks, prepared SQL statements and log-file entries). It's usually assumed that `ROLLBACK` can't fail, although such a thing is conceivable (for example, an encompassing transaction might reject an attempt to `ROLLBACK` because it's lining up for a `COMMIT`).

`ROLLBACK` cancels all effects of a transaction. It does not cancel effects on objects outside the DBMS's control (for example the values in host program variables or the settings made by some SQL/CLI function calls). But in general, it is a convenient statement for those situations when you say "oops, this isn't working" or when you simply don't care whether your temporary work becomes permanent or not.

Here is a moot question. If all you've been doing is `SELECT`s, so that there have been no data changes, should you end the transaction with `ROLLBACK` or `COMMIT`? It shouldn't really matter because both `ROLLBACK` and `COMMIT` do the same transaction-terminating job. However, the popular conception is that `ROLLBACK` implies failure, so after a successful series of `SELECT` statements the convention is to end the transaction with `COMMIT` rather than `ROLLBACK`.

MariaDB (and most other DBMSs) supports rollback of SQL-data change statements, but not of SQL-Schema statements. This means that if you use any of `CREATE`, `ALTER`, `DROP`, `GRANT`, `REVOKE`, you are implicitly committing at execution time.

```
INSERT INTO Table_2 VALUES(5);  
DROP TABLE Table_3 CASCADE;  
ROLLBACK;
```

The result will be that both the `INSERT` and the `DROP` will go through as separate transactions so the `ROLLBACK` will have no effect.

NumPoints

A synonym for [ST_NumPoints](#).

PointN

A synonym for [ST_PointN](#).

STARTPOINT

A synonym for [ST_StartPoint](#).

ST_ENDPOINT

Syntax

```
ST_EndPoint(ls)  
EndPoint(ls)
```

Description

Returns the `Point` that is the endpoint of the `LineString` value `ls`.

`ST_EndPoint()` and `EndPoint()` are synonyms.

Examples

```
SET @ls = 'LineString(1 1,2 2,3 3)';

SELECT AsText(EndPoint(GeomFromText(@ls)));
+-----+
| AsText(EndPoint(GeomFromText(@ls))) |
+-----+
| POINT(3 3)                         |
+-----+
```

ST_NUMPOINTS

Syntax

```
ST_NumPoints(ls)
NumPoints(ls)
```

Description

Returns the number of [Point](#) objects in the [LineString](#) value `ls`.

`ST_NumPoints()` and `NumPoints()` are synonyms.

Examples

```
SET @ls = 'LineString(1 1,2 2,3 3)';

SELECT NumPoints(GeomFromText(@ls));
+-----+
| NumPoints(GeomFromText(@ls)) |
+-----+
|                      3 |
+-----+
```

ST_POINTN

Syntax

```
ST_PointN(ls,N)
PointN(ls,N)
```

Description

Returns the N-th [Point](#) in the [LineString](#) value `ls`. Points are numbered beginning with `1`.

`ST_PointN()` and `PointN()` are synonyms.

Examples

```

SET @ls = 'LineString(1 1,2 2,3 3)';

SELECT AsText(PointN(GeomFromText(@ls),2));
+-----+
| AsText(PointN(GeomFromText(@ls),2)) |
+-----+
| POINT(2 2)                         |
+-----+

```

PREPARE Statement

Syntax

```
PREPARE stmt_name FROM preparable_stmt
```

Description

The `PREPARE` statement prepares a statement and assigns it a name, `stmt_name`, by which to refer to the statement later. Statement names are not case sensitive. `preparable_stmt` is either a string literal or a [user variable](#) (not a [local variable](#), an SQL expression or a subquery) that contains the text of the statement. The text must represent a single SQL statement, not multiple statements. Within the statement, "?" characters can be used as parameter markers to indicate where data values are to be bound to the query later when you execute it. The "?" characters should not be enclosed within quotes, even if you intend to bind them to string values. Parameter markers can be used only where expressions should appear, not for SQL keywords, identifiers, and so forth.

The scope of a prepared statement is the session within which it is created. Other sessions cannot see it.

If a prepared statement with the given name already exists, it is deallocated implicitly before the new statement is prepared. This means that if the new statement contains an error and cannot be prepared, an error is returned and no statement with the given name exists.

Prepared statements can be PREPARED and EXECUTED in a stored procedure, but not in a stored function or trigger. Also, even if the statement is PREPARED in a procedure, it will not be deallocated when the procedure execution ends.

A prepared statement can access [user-defined variables](#), but not [local variables](#) or procedure's parameters.

If the prepared statement contains a syntax error, `PREPARE` will fail. As a side effect, stored procedures can use it to check if a statement is valid. For example:

```

CREATE PROCEDURE `test_stmt` (IN sql_text TEXT)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        SELECT CONCAT(sql_text, ' is not valid');
    END;
    SET @SQL := sql_text;
    PREPARE stmt FROM @SQL;
    DEALLOCATE PREPARE stmt;
END;

```

The [FOUND_ROWS\(\)](#) and [ROW_COUNT\(\)](#) functions, if called immediately after `EXECUTE`, return the number of rows read or affected by the prepared statements; however, if they are called after `DEALLOCATE PREPARE`, they provide information about this statement. If the prepared statement produces errors or warnings, [GET DIAGNOSTICS](#) return information about them. `DEALLOCATE PREPARE` shouldn't clear the [diagnostics area](#), unless it produces an error.

A prepared statement is executed with `EXECUTE` and released with `DEALLOCATE PREPARE`.

The `max_prepared_stmt_count` server system variable determines the number of allowed prepared statements that can be prepared on the server. If it is set to 0, prepared statements are not allowed. If the limit is reached, an error similar to the following will be produced:

```
ERROR 1461 (42000): Can't create more than max_prepared_stmt_count statements
(current value: 0)
```

Contents

1. [Syntax](#)
2. [Description](#)
 1. [Oracle Mode](#)
3. [Permitted Statements](#)
4. [Example](#)
5. [See Also](#)

MariaDB starting with 10.3

In Oracle mode from MariaDB 10.3, PREPARE stmt FROM 'SELECT :1, :2' is used, instead of ? .

Permitted Statements

Not all statements can be prepared. Only the following SQL commands are permitted:

- `ALTER TABLE`
- `ANALYZE TABLE`
- `BINLOG`
- `CACHE INDEX`
- `CALL`
- `CHANGE MASTER`
- `CHECKSUM {TABLE | TABLES}`
- `COMMIT`
- `{CREATE | DROP} DATABASE`
- `{CREATE | DROP} INDEX`
- `{CREATE | RENAME | DROP} TABLE`
- `{CREATE | RENAME | DROP} USER`
- `{CREATE | DROP} VIEW`
- `DELETE`
- `DESCRIBE`
- `DO`
- `EXPLAIN`
- `FLUSH {TABLE | TABLES | TABLES WITH READ LOCK | HOSTS | PRIVILEGES | LOGS | STATUS | MASTER | SLAVE | DES_KEY_FILE | USER_RESOURCES | QUERY CACHE | TABLE_STATISTICS | INDEX_STATISTICS | USER_STATISTICS | CLIENT_STATISTICS}`
- `GRANT`
- `INSERT`
- `INSTALL {PLUGIN | SONAME}`
- `HANDLER READ`
- `KILL`
- `LOAD INDEX INTO CACHE`
- `OPTIMIZE TABLE`
- `REPAIR TABLE`
- `REPLACE`
- `RESET {MASTER | SLAVE | QUERY CACHE}`
- `REVOKE`
- `ROLLBACK`
- `SELECT`
- `SET`
- `SET GLOBAL SQL_SLAVE_SKIP_COUNTER`
- `SET ROLE`
- `SET SQL_LOG_BIN`
- `SET TRANSACTION ISOLATION LEVEL`
- `SHOW EXPLAIN`
- `SHOW {DATABASES | TABLES | OPEN TABLES | TABLE STATUS | COLUMNS | INDEX | TRIGGERS | EVENTS | GRANTS | CHARACTER SET | COLLATION | ENGINES | PLUGINS [SONAME] | PRIVILEGES | PROCESSLIST | PROFILE | PROFILES | VARIABLES | STATUS | WARNINGS | ERRORS | TABLE_STATISTICS | INDEX_STATISTICS | USER_STATISTICS | CLIENT_STATISTICS | AUTHORS | CONTRIBUTORS}`
- `SHOW CREATE {DATABASE | TABLE | VIEW | PROCEDURE | FUNCTION | TRIGGER | EVENT}`
- `SHOW {FUNCTION | PROCEDURE} CODE`
- `SHOW BINLOG EVENTS`
- `SHOW SLAVE HOSTS`
- `SHOW {MASTER | BINARY} LOGS`
- `SHOW {MASTER | SLAVE | TABLES | INNODB | FUNCTION | PROCEDURE} STATUS`
- `SLAVE {START | STOP}`
- `TRUNCATE TABLE`
- `SHUTDOWN`
- `UNINSTALL {PLUGIN | SONAME}`
- `UPDATE`

Synonyms are not listed here, but can be used. For example, `DESC` can be used instead of `DESCRIBE`.

MariaDB starting with 10.1.1

Compound statements can be prepared too.

Note that if a statement can be run in a stored routine, it will work even if it is called by a prepared statement. For example, `SIGNAL` can't be directly prepared. However, it is allowed in [stored routines](#). If the `x()` procedure contains `SIGNAL`, you can still prepare and execute the '`CALL x();`' prepared statement.

MariaDB starting with 10.2.3

`PREPARE` now supports most kinds of expressions as well, for example:

```
PREPARE stmt FROM CONCAT('SELECT * FROM ', table_name);
```

MariaDB starting with 10.6.2

All statements can be prepared, except `PREPARE`, `EXECUTE`, and `DEALLOCATE / DROP PREPARE`.

When `PREPARE` is used with a statement which is not supported, the following error is produced:

```
ERROR 1295 (HY000): This command is not supported in the prepared statement protocol yet
```

Example

```
create table t1 (a int,b char(10));
insert into t1 values (1,"one"),(2, "two"),(3,"three");
prepare test from "select * from t1 where a=?";
set @param=2;
execute test using @param;
+-----+
| a    | b     |
+-----+
| 2   | two  |
+-----+
set @param=3;
execute test using @param;
+-----+
| a    | b     |
+-----+
| 3   | three |
+-----+
deallocate prepare test;
```

Since identifiers are not permitted as prepared statements parameters, sometimes it is necessary to dynamically compose an SQL statement. This technique is called *dynamic SQL*). The following example shows how to use dynamic SQL:

```
CREATE PROCEDURE teststmt_test(IN tab_name VARCHAR(64))
BEGIN
  SET @sql = CONCAT('SELECT COUNT(*) FROM ', tab_name);
  PREPARE stmt FROM @sql;
  EXECUTE stmt;
  DEALLOCATE PREPARE stmt;
END;

CALL teststmt_test('mysql.user');
+-----+
| COUNT(*) |
+-----+
|      4   |
+-----+
```

Use of variables in prepared statements:

```

PREPARE stmt FROM 'SELECT @x;';

SET @x = 1;

EXECUTE stmt;
+-----+
| @x   |
+-----+
|    1  |
+-----+

SET @x = 0;

EXECUTE stmt;
+-----+
| @x   |
+-----+
|    0  |
+-----+

DEALLOCATE PREPARE stmt;

```

See Also

- [Out parameters in PREPARE](#)
- [EXECUTE Statement](#)
- [DEALLOCATE / DROP Prepared Statement](#)
- [EXECUTE IMMEDIATE](#)
- [Oracle mode from MariaDB 10.3](#)

INET_ATON

Syntax

```
INET_ATON(expr)
```

Description

Given the dotted-quad representation of an IPv4 network address as a string, returns an integer that represents the numeric value of the address. Addresses may be 4- or 8-byte addresses.

Returns NULL if the argument is not understood.

Examples

```

SELECT INET_ATON('192.168.1.1');
+-----+
| INET_ATON('192.168.1.1') |
+-----+
|      3232235777 |
+-----+

```

This is calculated as follows: $192 \times 256^3 + 168 \times 256^2 + 1 \times 256 + 1$

See Also

- [INET6_ATON\(\)](#)
- [INET_NTOA\(\)](#)

INET_NTOA

Syntax

```
INET_NTOA(expr)
```

Description

Given a numeric IPv4 network address in network byte order (4 or 8 byte), returns the dotted-quad representation of the address as a string.

Examples

```
SELECT INET_NTOA(3232235777);
+-----+
| INET_NTOA(3232235777) |
+-----+
| 192.168.1.1           |
+-----+
```

192.168.1.1 corresponds to 3232235777 since $192 \times 256^3 + 168 \times 256^2 + 1 \times 256 + 1 = 3232235777$

See Also

- [INET6_NTOA\(\)](#)
- [INET_ATON\(\)](#)

IS_FREE_LOCK

Syntax

```
IS_FREE_LOCK(str)
```

Description

Checks whether the lock named `str` is free to use (that is, not locked). Returns `1` if the lock is free (no one is using the lock), `0` if the lock is in use, and `NULL` if an error occurs (such as an incorrect argument, like an empty string or `NULL`). `str` is case insensitive.

Contents

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

If the `metadata_lock_info` plugin is installed, the `Information Schema metadata_lock_info` table contains information about locks of this kind (as well as `metadata locks`).

Statements using the `IS_FREE_LOCK` function are [not safe for statement-based replication](#).

See Also

- [GET_LOCK](#)
- [RELEASE_LOCK](#)
- [IS_USED_LOCK](#)
- [RELEASE_ALL_LOCKS](#)

START TRANSACTION

Syntax

```

START TRANSACTION [transaction_property [, transaction_property] ...] | BEGIN [WORK]
COMMIT [WORK] [AND [NO] CHAIN] [[NO] RELEASE]
ROLLBACK [WORK] [AND [NO] CHAIN] [[NO] RELEASE]
SET autocommit = {0 | 1}

transaction_property:
  WITH CONSISTENT SNAPSHOT
  | READ WRITE
  | READ ONLY

```

Description

The `START TRANSACTION` or `BEGIN` statement begins a new transaction. `COMMIT` commits the current transaction, making its changes permanent. `ROLLBACK` rolls back the current transaction, canceling its changes. The `SET autocommit` statement disables or enables the default autocommit mode for the current session.

`START TRANSACTION` and `SET autocommit = 1` implicitly commit the current transaction, if any.

The optional `WORK` keyword is supported for `COMMIT` and `ROLLBACK`, as are the `CHAIN` and `RELEASE` clauses. `CHAIN` and `RELEASE` can be used for additional control over transaction completion. The value of the `completion_type` system variable determines the default completion behavior.

The `AND CHAIN` clause causes a new transaction to begin as soon as the current one ends, and the new transaction has the same isolation level as the just-terminated transaction. The `RELEASE` clause causes the server to disconnect the current client session after terminating the current transaction. Including the `NO` keyword suppresses `CHAIN` or `RELEASE` completion, which can be useful if the `completion_type` system variable is set to cause chaining or release completion by default.

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 `SET TRANSACTION` statement, in which case the specified mode is valid for all sessions, or for all subsequent transaction used by the current session.

autocommit

By default, MariaDB runs with `autocommit` mode enabled. This means that as soon as you execute a statement that updates (modifies) a table, MariaDB stores the update on disk to make it permanent. To disable autocommit mode, use the following statement:

```
SET autocommit=0;
```

After disabling autocommit mode by setting the autocommit variable to zero, changes to transaction-safe tables (such as those for InnoDB or NDBCLUSTER) are not made permanent immediately. You must use `COMMIT` to store your changes to disk or `ROLLBACK` to ignore the changes.

To disable autocommit mode for a single series of statements, use the `START TRANSACTION` statement.

DDL Statements

DDL statements (`CREATE`, `ALTER`, `DROP`) and administrative statements (`FLUSH`, `RESET`, `OPTIMIZE`, `ANALYZE`, `CHECK`, `REPAIR`, `CACHE INDEX`), transaction management statements (`BEGIN`, `START TRANSACTION`) and `LOAD DATA INFILE`, cause an implicit `COMMIT` and start a new transaction. An exception to this rule are the DDL that operate on temporary tables: you can `CREATE`, `ALTER` and `DROP` them without causing any `COMMIT`, but those actions cannot be rolled back. This means that if you call `ROLLBACK`, the temporary tables you created in the transaction will remain, while the rest of the transaction will be rolled back.

Transactions cannot be used in Stored Functions or Triggers. In Stored Procedures and Events `BEGIN` is not allowed, so you should use `START TRANSACTION` instead.

A transaction acquires a `metadata lock` on every table it accesses to prevent other connections from altering their structure. The lock is released at the end of the transaction. This happens even with non-transactional storage engines (like `MEMORY` or `CONNECT`), so it makes sense to use

Contents

1. [Syntax](#)
2. [Description](#)
 1. [Access Mode](#)
 2. [autocommit](#)
 3. [DDL Statements](#)
 4. [in_transaction](#)
 5. [WITH CONSISTENT SNAPSHOT](#)
3. [Examples](#)
4. [See Also](#)

transactions with non-transactional tables.

in_transaction

The `in_transaction` system variable is a session-only, read-only variable that returns `1` inside a transaction, and `0` if not in a transaction.

WITH CONSISTENT SNAPSHOT

The `WITH CONSISTENT SNAPSHOT` option starts a consistent read for storage engines such as InnoDB that can do so, the same as if a `START TRANSACTION` followed by a `SELECT` from any InnoDB table was issued.

See [Enhancements for START TRANSACTION WITH CONSISTENT SNAPSHOT](#).

Examples

```
START TRANSACTION;
SELECT @A:=SUM(salary) FROM table1 WHERE type=1;
UPDATE table2 SET summary=@A WHERE type=1;
COMMIT;
```

See Also

- [Enhancements for START TRANSACTION WITH CONSISTENT SNAPSHOT](#)
- [MyRocks and START TRANSACTION WITH CONSISTENT SNAPSHOT](#)

IS_IPV6

Syntax

```
IS_IPV6(expr)
```

Description

Returns 1 if the expression is a valid IPv6 address specified as a string, otherwise returns 0. Does not consider IPv4 addresses to be valid IPv6 addresses.

Examples

```
SELECT IS_IPV6('48f3::d432:1431:ba23:846f');
+-----+
| IS_IPV6('48f3::d432:1431:ba23:846f') |
+-----+
|          1 |
+-----+
1 row in set (0.02 sec)

SELECT IS_IPV6('10.0.1.1');
+-----+
| IS_IPV6('10.0.1.1') |
+-----+
|          0 |
+-----+
```

See Also

- [INET6 data type](#)
- [INET6_ATON](#)
- [INET6_NTOA](#)

Transaction Timeouts

MariaDB has always had the `wait_timeout` and `interactive_timeout` settings, which close connections after a certain period of inactivity.

However, these are by default set to a long wait period. In situations where transactions may be started, but not committed or rolled back, more granular control and a shorter timeout may be desirable so as to avoid locks being held for too long.

MariaDB 10.3 introduced three new variables to handle this situation.

- `idle_transaction_timeout` (all transactions)
- `idle_write_transaction_timeout` (write transactions - called `idle_readwrite_transaction_timeout` until MariaDB 10.3.2)
- `idle_READONLY_transaction_timeout` (read transactions)

These accept a time in seconds to time out, by closing the connection, transactions that are idle for longer than this period. By default all are set to zero, or no timeout.

`idle_transaction_timeout` affects all transactions, `idle_write_transaction_timeout` affects write transactions only and `idle_READONLY_transaction_timeout` affects read transactions only. The latter two variables work independently. However, if either is set along with `idle_transaction_timeout`, the settings for `idle_write_transaction_timeout` or `idle_READONLY_transaction_timeout` will take precedence.

Examples

```
SET SESSION idle_transaction_timeout=2;
BEGIN;
SELECT * FROM t;
Empty set (0.000 sec)
## wait 3 seconds
SELECT * FROM t;
ERROR 2006 (HY000): MySQL server has gone away
```

```
SET SESSION idle_write_transaction_timeout=2;
BEGIN;
SELECT * FROM t;
Empty set (0.000 sec)
## wait 3 seconds
SELECT * FROM t;
Empty set (0.000 sec)
INSERT INTO t VALUES(1);
## wait 3 seconds
SELECT * FROM t;
ERROR 2006 (HY000): MySQL server has gone away
```

```
SET SESSION idle_transaction_timeout=2, SESSION idle_READONLY_transaction_timeout=10;
BEGIN;
SELECT * FROM t;
Empty set (0.000 sec)
## wait 3 seconds
SELECT * FROM t;
Empty set (0.000 sec)
## wait 11 seconds
SELECT * FROM t;
ERROR 2006 (HY000): MySQL server has gone away
```

WAIT and NOWAIT

MariaDB starting with 10.3.0

MariaDB 10.3.0 introduced extended syntax so that it is possible to set `innodb_lock_wait_timeout` and `lock_wait_timeout` for the following statements:

Syntax

```
ALTER TABLE tbl_name [WAIT n|NOWAIT] ...
CREATE ... INDEX ON tbl_name (index_col_name, ...) [WAIT n|NOWAIT] ...
DROP INDEX ... [WAIT n|NOWAIT]
DROP TABLE tbl_name [WAIT n|NOWAIT] ...
LOCK TABLE ... [WAIT n|NOWAIT]
OPTIMIZE TABLE tbl_name [WAIT n|NOWAIT]
RENAME TABLE tbl_name [WAIT n|NOWAIT] ...
SELECT ... FOR UPDATE [WAIT n|NOWAIT]
SELECT ... LOCK IN SHARE MODE [WAIT n|NOWAIT]
TRUNCATE TABLE tbl_name [WAIT n|NOWAIT]
```

Description

The lock wait timeout can be explicitly set in the statement by using either `WAIT n` (to set the wait in seconds) or `NOWAIT`, in which case the statement will immediately fail if the lock cannot be obtained. `WAIT 0` is equivalent to `NOWAIT`.

See Also

- [Query Limits and Timeouts](#)
- [ALTER TABLE](#)
- [CREATE INDEX](#)
- [DROP INDEX](#)
- [DROP TABLE](#)
- [LOCK TABLES and UNLOCK TABLES](#)
- [OPTIMIZE TABLE](#)
- [RENAME TABLE](#)
- [SELECT](#)
- [TRUNCATE TABLE](#)

Account Locking

MariaDB starting with 10.4.2

Account locking was introduced in [MariaDB 10.4.2](#).

Description

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).

User accounts can be locked at creation, with the [CREATE USER](#) statement, or modified after creation with the [ALTER USER](#) statement. For example:

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

or

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

The server will return an `ER_ACCOUNT_HAS_BEEN_LOCKED` error when locked users attempt to connect:

```
mysql -ulorin
ERROR 4151 (HY000): Access denied, this account is locked
```

The [ALTER USER](#) statement is also used to unlock a user:

```
ALTER USER 'lorin'@'localhost' ACCOUNT UNLOCK;
```

The [SHOW CREATE USER](#) statement will show whether the account is locked:

Contents

1. [Description](#)
- [See Also](#)

```
SHOW CREATE USER 'marijn'@'localhost';
+-----+
| CREATE USER for marijn@localhost          |
+-----+
| CREATE USER 'marijn'@'localhost' ACCOUNT LOCK |
+-----+
```

as well as querying the [mysql.global_priv](#) table:

```
SELECT CONCAT(user, '@', host, ' => ', JSON_DETAILED(priv)) FROM mysql.global_priv
WHERE user='marijn';
+-----+
| CONCAT(user, '@', host, ' => ', JSON_DETAILED(priv))           |
+-----+
| marijn@localhost => {
    "access": 0,
    "plugin": "mysql_native_password",
    "authentication_string": "",
    "account_locked": true,
    "password_last_changed": 1558017158
} |
+-----+
```

See Also

- [Account Locking and Password Expiry](#) video tutorial

IS_USED_LOCK

Syntax

```
IS_USED_LOCK(str)
```

Description

Checks whether the lock named `str` is in use (that is, locked). If so, it returns the connection identifier of the client that holds the lock. Otherwise, it returns `NULL`. `str` is case insensitive.

If the [metadata_lock_info](#) plugin is installed, the [Information Schema metadata_lock_info](#) table contains information about locks of this kind (as well as [metadata locks](#)).

Statements using the `IS_USED_LOCK` function are [not safe for statement-based replication](#).

Contents

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

See Also

- [GET_LOCK](#)
- [RELEASE_LOCK](#)
- [IS_FREE_LOCK](#)
- [RELEASE_ALL_LOCKS](#)

User Password Expiry

MariaDB starting with [10.4.3](#)

User password expiry was introduced in [MariaDB 10.4.3](#).

Password expiry permits administrators to expire user passwords, either manually or automatically.

System Variables

There are two system variables which affect password expiry: `default_password_lifetime`, which determines the amount of time between requiring the user to change their password. `0`, the default, means automatic password expiry is not active.

The second variable, `disconnect_on_expired_password` determines whether a client is permitted to connect if their password has expired, or whether they are permitted to connect in sandbox mode, able to perform a limited subset of queries related to resetting the password, in particular `SET PASSWORD` and `SET`.

Contents

1. System Variables
2. Setting a Password Expiry Limit for a User
3. SHOW CREATE USER
4. Checking When Passwords Expire
5. --connect-expired-password Client Option
6. See Also

Setting a Password Expiry Limit for a User

Besides automatic password expiry, as determined by `default_password_lifetime`, password expiry times can be set on an individual user basis, overriding the global using the `CREATE USER` or `ALTER USER` statements, for example:

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

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

Limits can be disabled by use of the `NEVER` keyword, for example:

```
CREATE USER 'monty'@'localhost' PASSWORD EXPIRE NEVER;
```

```
ALTER USER 'monty'@'localhost' PASSWORD EXPIRE NEVER;
```

A manually set limit can be restored the system default by use of `DEFAULT`, for example:

```
CREATE USER 'monty'@'localhost' PASSWORD EXPIRE DEFAULT;
```

```
ALTER USER 'monty'@'localhost' PASSWORD EXPIRE DEFAULT;
```

SHOW CREATE USER

The `SHOW CREATE USER` statement will display information about the password expiry status of the user. Unlike MySQL, it will not display if the user is unlocked, or if the password expiry is set to default.

```
CREATE USER 'monty'@'localhost' PASSWORD EXPIRE INTERVAL 120 DAY;
CREATE USER 'konstantin'@'localhost' PASSWORD EXPIRE NEVER;
CREATE USER 'amse'@'localhost' PASSWORD EXPIRE DEFAULT;

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

SHOW CREATE USER 'konstantin'@'localhost';
+-----+
| CREATE USER for konstantin@localhost      |
+-----+
| CREATE USER 'konstantin'@'localhost' PASSWORD EXPIRE NEVER |
+-----+

SHOW CREATE USER 'amse'@'localhost';
+-----+
| CREATE USER for amse@localhost           |
+-----+
| CREATE USER 'amse'@'localhost'           |
+-----+
```

Checking When Passwords Expire

The following query can be used to check when the current passwords expire for all users:

```
WITH password_expiration_info AS (
  SELECT User, Host,
  IF(
    IFNULL(JSON_EXTRACT(Priv, '$.password_lifetime'), -1) = -1,
    @@global.default_password_lifetime,
    JSON_EXTRACT(Priv, '$.password_lifetime')
  ) AS password_lifetime,
  JSON_EXTRACT(Priv, '$.password_last_changed') AS password_last_changed
  FROM mysql.global_priv
)
SELECT pei.User, pei.Host,
  pei.password_lifetime,
  FROM_UNIXTIME(pei.password_last_changed) AS password_last_changed_datetime,
  FROM_UNIXTIME(
    pei.password_last_changed +
    (pei.password_lifetime * 60 * 60 * 24)
  ) AS password_expiration_datetime
  FROM password_expiration_info pei
  WHERE pei.password_lifetime != 0
  AND pei.password_last_changed IS NOT NULL
UNION
SELECT pei.User, pei.Host,
  pei.password_lifetime,
  FROM_UNIXTIME(pei.password_last_changed) AS password_last_changed_datetime,
  0 AS password_expiration_datetime
  FROM password_expiration_info pei
  WHERE pei.password_lifetime = 0
  OR pei.password_last_changed IS NULL;
```

--connect-expired-password Client Option

The `mysql client --connect-expired-password` option notifies the server that the client is prepared to handle expired password sandbox mode (even if the `--batch` option was specified).

See Also

- [Account Locking and Password Expiry](#) video tutorial

NAME_CONST

Syntax

```
NAME_CONST(name,value)
```

Description

Returns the given value. When used to produce a result set column, `NAME_CONST()` causes the column to have the given name. The arguments should be constants.

This function is used internally when replicating stored procedures. It makes little sense to use it explicitly in SQL statements, and it was not supposed to be used like that.

```
SELECT NAME_CONST('myname', 14);
+-----+
| myname |
+-----+
|    14   |
+-----+
```

SLEEP

Syntax

```
SLEEP(duration)
```

Description

Sleeps (pauses) for the number of seconds given by the duration argument, then returns `0`. If `SLEEP()` is interrupted, it returns `1`. The duration may have a fractional part given in microseconds.

Statements using the `SLEEP()` function are not [safe for replication](#).

Example

```
SELECT SLEEP(5.5);
+-----+
| SLEEP(5.5) |
+-----+
|      0 |
+-----+
1 row in set (5.50 sec)
```

UUID_SHORT

Syntax

```
UUID_SHORT()
```

Description

Returns a "short" universally unique identifier as a 64-bit unsigned integer (rather than a string-form 128-bit identifier as returned by the `UUID()` function).

The value of `UUID_SHORT()` is guaranteed to be unique if the following conditions hold:

- The `server_id` of the current host is unique among your set of master and slave servers
- `server_id` is between 0 and 255
- You don't set back your system time for your server between mysqld restarts
- You do not invoke `UUID_SHORT()` on average more than 16 million times per second between mysqld restarts

Contents

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

The `UUID_SHORT()` return value is constructed this way:

```
(server_id & 255) << 56
+ (server_startup_time_in_seconds << 24)
+ incremented_variable++;
```

Statements using the `UUID_SHORT()` function are not [safe for statement-based replication](#).

Examples

```
SELECT UUID_SHORT();
+-----+
| UUID_SHORT()      |
+-----+
| 21517162376069120 |
+-----+
```

```
create table t1 (a bigint unsigned default(uuid_short()) primary key);
insert into t1 values(),();
select * from t1;
+-----+
| a      |
+-----+
| 98113699159474176 |
| 98113699159474177 |
+-----+
```

See Also

- [UUID\(\)](#) ; Return full (128 bit) Universally Unique Identifier
- [AUTO_INCREMENT](#)
- [Sequences](#) - an alternative to auto_increment available from [MariaDB 10.3](#)
- [SYS_GUID](#) - UUID without the - character for Oracle compatibility
- [UUID data type](#)

!

Syntax

NOT, !

Description

Logical NOT. Evaluates to 1 if the operand is 0, to 0 if the operand is non-zero, and NOT NULL returns NULL.

By default, the `!` operator has a [higher precedence](#). If the `HIGH_NOT_PRECEDENCE` `SQL_MODE` flag is set, `NOT` and `!` have the same precedence.

Contents

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

Examples

```
SELECT NOT 10;
```

```
+-----+
| NOT 10 |
+-----+
|      0 |
+-----+
```

```
SELECT NOT 0;
```

```
+-----+
| NOT 0 |
+-----+
|      1 |
+-----+
```

```
SELECT NOT NULL;
```

```
+-----+
| NOT NULL |
+-----+
|      NULL |
+-----+
```

```
SELECT ! (1+1);
```

```
+-----+
| ! (1+1) |
+-----+
|      0 |
+-----+
```

```
SELECT ! 1+1;
```

```
+-----+
| ! 1+1 |
+-----+
|      1 |
+-----+
```

&&

Syntax

AND, &&

Description

Logical AND. Evaluates to 1 if all operands are non-zero and not NULL, to 0 if one or more operands are 0, otherwise NULL is returned.

For this operator, [short-circuit evaluation](#) can be used.

Contents

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

Examples

```
SELECT 1 && 1;
+-----+
| 1 && 1 |
+-----+
|      1 |
+-----+

SELECT 1 && 0;
+-----+
| 1 && 0 |
+-----+
|      0 |
+-----+

SELECT 1 && NULL;
+-----+
| 1 && NULL |
+-----+
|      NULL |
+-----+

SELECT 0 && NULL;
+-----+
| 0 && NULL |
+-----+
|      0 |
+-----+

SELECT NULL && 0;
+-----+
| NULL && 0 |
+-----+
|      0 |
+-----+
```

Assignment Operator (:=)

Syntax

```
var_name := expr
```

Description

Assignment operator for assigning a value. The value on the right is assigned to the variable on left.

Unlike the [= operator](#), `:=` can always be used to assign a value to a variable.

This operator works with both [user-defined variables](#) and [local variables](#).

When assigning the same value to several variables, [LAST_VALUE\(\)](#) can be useful.

Examples

```

SELECT @x := 10;
+-----+
| @x := 10 |
+-----+
|      10 |
+-----+

SELECT @x, @y := @x;
+-----+
| @x    | @y := @x |
+-----+
|   10  |      10 |
+-----+

```

GROUP_CONCAT

Syntax

`GROUP_CONCAT(expr)`

Description

This function returns a string result with the concatenated non-NULL values from a group. It returns `NULL` if there are no non-NULL values.

The maximum returned length in bytes is determined by the `group_concat_max_len` server system variable, which defaults to 1M (>= MariaDB 10.2.4) or 1K (<= MariaDB 10.2.3).

If `group_concat_max_len <= 512`, the return type is `VARBINARY` or `VARCHAR`; otherwise, the return type is `BLOB` or `TEXT`. The choice between binary or non-binary types depends from the input.

The full syntax is as follows:

```

GROUP_CONCAT([DISTINCT] expr [,expr ...]
            [ORDER BY {unsigned_integer | col_name | expr}
             [ASC | DESC] [,col_name ...]]
            [SEPARATOR str_val]
            [LIMIT {[offset,] row_count | row_count OFFSET offset}])

```

`DISTINCT` eliminates duplicate values from the output string.

`ORDER BY` determines the order of returned values.

`SEPARATOR` specifies a separator between the values. The default separator is a comma (,). It is possible to avoid using a separator by specifying an empty string.

LIMIT

MariaDB starting with 10.3.3

Until MariaDB 10.3.2, it was not possible to use the `LIMIT` clause with `GROUP_CONCAT`. This restriction was lifted in MariaDB 10.3.3.

Contents

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

Examples

```

SELECT student_name,
       GROUP_CONCAT(test_score)
  FROM student
 GROUP BY student_name;

```

Get a readable list of MariaDB users from the `mysql.user` table:

```
SELECT GROUP_CONCAT(DISTINCT User ORDER BY User SEPARATOR '\n')
  FROM mysql.user;
```

In the former example, `DISTINCT` is used because the same user may occur more than once. The new line (`\n`) used as a `SEPARATOR` makes the results easier to read.

Get a readable list of hosts from which each user can connect:

```
SELECT User, GROUP_CONCAT(Host ORDER BY Host SEPARATOR ', ')
  FROM mysql.user GROUP BY User ORDER BY User;
```

The former example shows the difference between the `GROUP_CONCAT`'s `ORDER BY` (which sorts the concatenated hosts), and the `SELECT`'s `ORDER BY` (which sorts the rows).

From MariaDB 10.3.3, `LIMIT` can be used with `GROUP_CONCAT`, so, for example, given the following table:

```
CREATE TABLE d (dd DATE, cc INT);

INSERT INTO d VALUES ('2017-01-01',1);
INSERT INTO d VALUES ('2017-01-02',2);
INSERT INTO d VALUES ('2017-01-04',3);
```

the following query:

```
SELECT SUBSTRING_INDEX(GROUP_CONCAT(CONCAT_WS(":",dd,cc) ORDER BY cc DESC),":",1) FROM d;
+-----+
| SUBSTRING_INDEX(GROUP_CONCAT(CONCAT_WS(":",dd,cc) ORDER BY cc DESC),":",1) |
+-----+
| 2017-01-04:3 |
+-----+
```

can be more simply rewritten as:

```
SELECT GROUP_CONCAT(CONCAT_WS(":",dd,cc) ORDER BY cc DESC LIMIT 1) FROM d;
+-----+
| GROUP_CONCAT(CONCAT_WS(":",dd,cc) ORDER BY cc DESC LIMIT 1) |
+-----+
| 2017-01-04:3 |
+-----+
```

See Also

- [CONCAT\(\)](#)
- [CONCAT_WS\(\)](#)
- [SELECT](#)
- [ORDER BY](#)

STDDEV_SAMP

Syntax

```
STDDEV_SAMP(expr)
```

Description

Returns the sample standard deviation of `expr` (the square root of [VAR_SAMP\(\)](#)).

It is an [aggregate function](#), and so can be used with the `GROUP BY` clause.

From MariaDB 10.2.2, `STDDEV_SAMP()` can be used as a [window function](#).

`STDDEV_SAMP()` returns `NULL` if there were no matching rows.

BENCHMARK

Syntax

```
BENCHMARK(count,expr)
```

Description

The BENCHMARK() function executes the expression `expr` repeatedly `count` times. It may be used to time how quickly MariaDB processes the expression. The result value is always 0. The intended use is from within the [mysql client](#), which reports query execution times.

Examples

```
SELECT BENCHMARK(1000000,ENCODE('hello','goodbye'));
+-----+
| BENCHMARK(1000000,ENCODE('hello','goodbye')) |
+-----+
|          0 |
+-----+
1 row in set (0.21 sec)
```

Roles Overview

MariaDB starting with 10.0.5

Roles were introduced in [MariaDB 10.0.5](#).

Contents

- [Description](#)
- [System Tables](#)
- [Examples](#)
- [Roles and Views \(and Stored Routines\)](#)
- [Other Resources](#)

Description

A role bundles a number of privileges together. It assists larger organizations where, typically, a number of users would have the same privileges, and, previously, the only way to change the privileges for a group of users was by changing each user's privileges individually.

Alternatively, multiple external users could have been assigned the same user, and there would have been no way to see which actual user was responsible for which action.

With roles, managing this is easy. For example, there could be a number of users assigned to a journalist role, with identical privileges. Changing the privileges for all the journalists is a matter of simply changing the role's privileges, while the individual user is still linked with any changes that take place.

Roles are created with the [CREATE ROLE](#) statement, and dropped with the [DROP ROLE](#) statement. Roles are then assigned to a user with an extension to the [GRANT](#) statement, while privileges are assigned to a role in the regular way with [GRANT](#). Similarly, the [REVOKE](#) statement can be used to both revoke a role from a user, or revoke a privilege from a role.

Once a user has connected, he can obtain all privileges associated with a role by [setting](#) a role with the [SET ROLE](#) statement. The [CURRENT_ROLE](#) function returns the currently set role for the session, if any.

Only roles granted directly to a user can be set, roles granted to other roles cannot. Instead the privileges granted to a role, which is, in turn, granted to another role (grantee), will be immediately available to any user who sets this second grantee role.

Roles were implemented as a GSoC 2013 project by Vicentiu Ciorbaru.

MariaDB starting with 10.1.1

The [SET DEFAULT ROLE](#) statement allows one to set a default role for a user. A default role is automatically enabled when a user connects (an implicit SET ROLE statement is executed immediately after a connection is established).

System Tables

Information about roles and who they've been granted to can be found in the [Information Schema APPLICABLE_ROLES table](#) as well as the [mysql.ROLES_MAPPING table](#).

The [Information Schema ENABLED_ROLES table](#) shows the enabled roles for the current session.

Examples

Creating a role and granting a privilege:

```
CREATE ROLE journalist;

GRANT SHOW DATABASES ON *.* TO journalist;

GRANT journalist TO hulda;
```

Note, that hulda has no `SHOW DATABASES` privilege, even though she was granted the journalist role. She needs to `set` the role first:

```
SHOW DATABASES;
+-----+
| Database      |
+-----+
| information_schema |
+-----+

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

SET ROLE journalist;

SELECT CURRENT_ROLE;
+-----+
| CURRENT_ROLE |
+-----+
| journalist   |
+-----+

SHOW DATABASES;
+-----+
| Database      |
+-----+
| ...          |
| information_schema |
| mysql         |
| performance_schema |
| test          |
| ...          |
+-----+

SET ROLE NONE;
```

Roles can be granted to roles:

```
CREATE ROLE writer;

GRANT SELECT ON data.* TO writer;

GRANT writer TO journalist;
```

But one does not need to set a role granted to a role. For example, hulda will automatically get all writer privileges when she sets the journalist role:

```

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

SHOW TABLES FROM data;
Empty set (0.01 sec)

SET ROLE journalist;

SELECT CURRENT_ROLE;
+-----+
| CURRENT_ROLE |
+-----+
| journalist   |
+-----+

SHOW TABLES FROM data;
+-----+
| Tables_in_data |
+-----+
| set1          |
| ...           |
+-----+

```

Roles and Views (and Stored Routines)

When a user sets a role, he, in a sense, has two identities with two associated sets of privileges. But a view (or a stored routine) can have only one definer. So, when a view (or a stored routine) is created with the `SQL SECURITY DEFINER`, one can specify whether the definer should be `CURRENT_USER` (and the view will have none of the privileges of the user's role) or `CURRENT_ROLE` (in this case, the view will use role's privileges, but none of the user's privileges). As a result, sometimes one can create a view that is impossible to use.

```

CREATE ROLE r1;

GRANT ALL ON db1.* TO r1;

GRANT r1 TO foo@localhost;

GRANT ALL ON db.* TO foo@localhost;

SELECT CURRENT_USER
+-----+
| current_user |
+-----+
| foo@localhost |
+-----+

SET ROLE r1;

CREATE TABLE db1.t1 (i int);

CREATE VIEW db.v1 AS SELECT * FROM db1.t1;

SHOW CREATE VIEW db.v1;
+-----+
| View | Create View
+-----+
| v1   | CREATE ALGORITHM=UNDEFINED DEFINER='foo'@'localhost` SQL SECURITY DEFINER VIEW `db`.`v1` AS SELECT `db1`.`t1`.`i` AS `i` fr
+-----+

CREATE DEFINER=CURRENT_ROLE VIEW db.v2 AS SELECT * FROM db1.t1;

SHOW CREATE VIEW db.b2;
+-----+
| View | Create View
+-----+
| v2   | CREATE ALGORITHM=UNDEFINED DEFINER='r1` SQL SECURITY DEFINER VIEW `db`.`v2` AS select `db1`.`t1`.`a` AS `a` from `db1`.`t1`
+-----+

```

Other Resources

- [Roles Review](#) by Peter Gulutzan

BINLOG_GTIID_POS

Syntax

```
BINLOG_GTIID_POS(binlog_filename,binlog_offset)
```

Description

The BINLOG_GTIID_POS() function takes as input an old-style [binary log](#) position in the form of a file name and a file offset. It looks up the position in the current binlog, and returns a string representation of the corresponding [GTID](#) position. If the position is not found in the current binlog, NULL is returned.

Examples

```
SELECT BINLOG_GTIID_POS("master-bin.000001", 600);
```

See Also

- [SHOW BINLOG EVENTS](#) - Show events and their positions in the binary log

COLLATION

Syntax

```
COLLATION(str)
```

Description

Returns the collation of the string argument. If `str` is not a string, it is considered as a binary string (so the function returns 'binary'). This applies to `NULL`, too. The return value is a string in the utf8 character set.

See [Character Sets and Collations](#).

Contents

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

Examples

```
SELECT COLLATION('abc');
+-----+
| COLLATION('abc') |
+-----+
| latin1_swedish_ci |
+-----+

SELECT COLLATION(_utf8'abc');
+-----+
| COLLATION(_utf8'abc') |
+-----+
| utf8_general_ci       |
+-----+
```

See Also

- [String literals](#)
- [CAST\(\)](#)
- [CONVERT\(\)](#)

AES_ENCRYPT

Syntax

```
AES_ENCRYPT(str,key_str)
```

Description

`AES_ENCRYPT()` and `AES_DECRYPT()` allow encryption and decryption of data using the official AES (Advanced Encryption Standard) algorithm, previously known as "Rijndael." Encoding with a 128-bit key length is used, but you can extend it up to 256 bits by modifying the source. We chose 128 bits because it is much faster and it is secure enough for most purposes.

`AES_ENCRYPT()` encrypts a string `str` using the key `key_str`, and returns a binary string.

`AES_DECRYPT()` decrypts the encrypted string and returns the original string.

The input arguments may be any length. If either argument is `NULL`, the result of this function is also `NULL`.

Because AES is a block-level algorithm, padding is used to encode uneven length strings and so the result string length may be calculated using this formula:

```
16 x (trunc(string_length / 16) + 1)
```

If `AES_DECRYPT()` detects invalid data or incorrect padding, it returns `NULL`. However, it is possible for `AES_DECRYPT()` to return a non-`NULL` value (possibly garbage) if the input data or the key is invalid.

Examples

```
INSERT INTO t VALUES (AES_ENCRYPT('text',SHA2('password',512)));
```

COMPRESS

Syntax

```
COMPRESS(string_to_compress)
```

Description

Compresses a string and returns the result as a binary string. This function requires MariaDB to have been compiled with a compression library such as zlib. Otherwise, the return value is always `NULL`. The compressed string can be uncompressed with `UNCOMPRESS()`.

The `have_compress` server system variable indicates whether a compression library is present.

Examples

```
SELECT LENGTH(COMPRESS(REPEAT('a',1000)));
+-----+
| LENGTH(COMPRESS(REPEAT('a',1000))) |
+-----+
|                               21 |
+-----+


SELECT LENGTH(COMPRESS(''));
+-----+
| LENGTH(COMPRESS('')) |
+-----+
|          0 |
+-----+


SELECT LENGTH(COMPRESS('a'));
+-----+
| LENGTH(COMPRESS('a')) |
+-----+
|           13 |
+-----+


SELECT LENGTH(COMPRESS(REPEAT('a',16)));
+-----+
| LENGTH(COMPRESS(REPEAT('a',16))) |
+-----+
|                           15 |
+-----+
```

CONNECTION_ID

Syntax

```
CONNECTION_ID()
```

Description

Returns the connection ID (thread ID) for the connection. Every thread (including events) has an ID that is unique among the set of currently connected clients.

Until MariaDB 10.3.1, returns `MYSQL_TYPE_LONGLONG`, or `bigint(10)`, in all cases. From MariaDB 10.3.1, returns `MYSQL_TYPE_LONG`, or `int(10)`, when the result would fit within 32-bits.

Examples

```
SELECT CONNECTION_ID();
+-----+
| CONNECTION_ID() |
+-----+
|          3 |
+-----+
```

See Also

- [SHOW PROCESSLIST](#)
- [INFORMATION_SCHEMA.PROCESSLIST](#)

SCHEMA

Syntax

```
SCHEMA()
```

Description

This function is a synonym for [DATABASE\(\)](#).

SESSION_USER

Syntax

```
SESSION_USER()
```

Description

`SESSION_USER()` is a synonym for [USER\(\)](#).

SYSTEM_USER

Syntax

```
SYSTEM_USER()
```

Description

`SYSTEM_USER()` is a synonym for [USER\(\)](#).

<

Syntax

<

Description

Less than operator. Evaluates both SQL expressions and returns 1 if the left value is less than the right value and 0 if it is not, or `NULL` if either expression is `NULL`. If the expressions return different data types, (for instance, a number and a string), performs type conversion.

When used in row comparisons these two queries return the same results:

```
SELECT (t1.a, t1.b) < (t2.x, t2.y)
FROM t1 INNER JOIN t2;

SELECT (t1.a < t2.x) OR ((t1.a = t2.x) AND (t1.b < t2.y))
FROM t1 INNER JOIN t2;
```

Examples

```
SELECT 2 < 2;
+-----+
| 2 < 2 |
+-----+
|      0 |
+-----+
```

Type conversion:

```
SELECT 3<'4';
+-----+
| 3<'4' |
+-----+
|      1 |
+-----+
```

Case insensitivity - see [Character Sets and Collations](#):

```
SELECT 'a'<'A';
+-----+
| 'a'<'A' |
+-----+
|      0 |
+-----+
```

<=

Syntax

<=

Description

Less than or equal operator. Evaluates both SQL expressions and returns 1 if the left value is less than or equal to the right value and 0 if it is not, or `NULL` if either expression is `NULL`. If the expressions return different data types, (for instance, a number and a string), performs type conversion.

When used in row comparisons these two queries return the same results:

Contents

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

Contents

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

```

SELECT (t1.a, t1.b) <= (t2.x, t2.y)
FROM t1 INNER JOIN t2;

SELECT (t1.a < t2.x) OR ((t1.a = t2.x) AND (t1.b <= t2.y))
FROM t1 INNER JOIN t2;

```

Examples

```

SELECT 0.1 <= 2;
+-----+
| 0.1 <= 2 |
+-----+
|      1 |
+-----+

```

```

SELECT 'a'<='A';
+-----+
| 'a'<='A' |
+-----+
|      1 |
+-----+

```

BETWEEN AND

Syntax

```
expr BETWEEN min AND max
```

Description

If expr is greater than or equal to min and expr is less than or equal to max, BETWEEN returns 1, otherwise it returns 0. This is equivalent to the expression $(min \leqslant \text{expr} \text{ AND } \text{expr} \leqslant max)$ if all the arguments are of the same type. Otherwise type conversion takes place according to the rules described at [Type Conversion](#), but applied to all the three arguments.

Contents

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

Examples

```

SELECT 1 BETWEEN 2 AND 3;
+-----+
| 1 BETWEEN 2 AND 3 |
+-----+
|      0 |
+-----+

```

```

SELECT 'b' BETWEEN 'a' AND 'c';
+-----+
| 'b' BETWEEN 'a' AND 'c' |
+-----+
|      1 |
+-----+

```

```

SELECT 2 BETWEEN 2 AND '3';
+-----+
| 2 BETWEEN 2 AND '3' |
+-----+
|      1 |
+-----+

```

```

SELECT 2 BETWEEN 2 AND 'x-3';
+-----+
| 2 BETWEEN 2 AND 'x-3' |
+-----+
|          0 |
+-----+
1 row in set, 1 warning (0.00 sec)

Warning (Code 1292): Truncated incorrect DOUBLE value: 'x-3'

```

NULL:

```

SELECT 1 BETWEEN 1 AND NULL;
+-----+
| 1 BETWEEN 1 AND NULL |
+-----+
|          NULL |
+-----+

```

DATE, DATETIME and TIMESTAMP examples. Omitting the time component compares against 00:00 , so later times on the same date are not returned:

```

CREATE TABLE `x` (
  a date ,
  b datetime,
  c timestamp
)

INSERT INTO x VALUES
('2018-11-11', '2018-11-11 05:15', '2018-11-11 05:15'),
('2018-11-12', '2018-11-12 05:15', '2018-11-12 05:15');

SELECT * FROM x WHERE a BETWEEN '2018-11-11' AND '2018-11-12';
+-----+-----+-----+
| a      | b      | c      |
+-----+-----+-----+
| 2018-11-11 | 2018-11-11 05:15:00 | 2018-11-11 05:15:00 |
| 2018-11-12 | 2018-11-12 05:15:00 | 2018-11-12 05:15:00 |
+-----+-----+-----+

SELECT * FROM x WHERE b BETWEEN '2018-11-11' AND '2018-11-12';
+-----+-----+-----+
| a      | b      | c      |
+-----+-----+-----+
| 2018-11-11 | 2018-11-11 05:15:00 | 2018-11-11 05:15:00 |
+-----+-----+-----+

SELECT * FROM x WHERE c BETWEEN '2018-11-11' AND '2018-11-12';
+-----+-----+-----+
| a      | b      | c      |
+-----+-----+-----+
| 2018-11-11 | 2018-11-11 05:15:00 | 2018-11-11 05:15:00 |
+-----+-----+-----+

```

GREATEST

Syntax

```
GREATEST(value1,value2,...)
```

Description

With two or more arguments, returns the largest (maximum-valued) argument. The arguments are compared using the same rules as for [LEAST\(\)](#).

Examples

```
SELECT GREATEST(2,0);
+-----+
| GREATEST(2,0) |
+-----+
|      2      |
+-----+
```

```
SELECT GREATEST(34.0,3.0,5.0,767.0);
+-----+
| GREATEST(34.0,3.0,5.0,767.0) |
+-----+
|          767.0      |
+-----+
```

```
SELECT GREATEST('B','A','C');
+-----+
| GREATEST('B','A','C') |
+-----+
|  C                  |
+-----+
```

DES_DECRYPT

Syntax

```
DES_DECRYPT(crypt_str[,key_str])
```

Description

Decrypts a string encrypted with [DES_ENCRYPT\(\)](#). If an error occurs, this function returns `NULL`.

This function works only if MariaDB has been configured with [TLS support](#).

If no `key_str` argument is given, `DES_DECRYPT()` examines the first byte of the encrypted string to determine the DES key number that was used to encrypt the original string, and then reads the key from the DES key file to decrypt the message. For this to work, the user must have the SUPER privilege. The key file can be specified with the `--des-key-file` server option.

If you pass this function a `key_str` argument, that string is used as the key for decrypting the message.

If the `crypt_str` argument does not appear to be an encrypted string, MariaDB returns the given `crypt_str`.

DES_ENCRYPT

Syntax

```
DES_ENCRYPT(str[,{key_num|key_str}])
```

Description

Encrypts the string with the given key using the Triple-DES algorithm.

This function works only if MariaDB has been configured with [TLS support](#).

The encryption key to use is chosen based on the second argument to `DES_ENCRYPT()`, if one was given. With no argument, the first key from the DES key file is used. With a `key_num` argument, the given key number (0-9) from the DES key file is used. With a `key_str` argument, the given key string is used to encrypt `str`.

The key file can be specified with the `--des-key-file` server option.

The return string is a binary string where the first character is `CHAR(128 | key_num)`. If an error occurs, `DES_ENCRYPT()` returns `NULL`.

The 128 is added to make it easier to recognize an encrypted key. If you use a string key, `key_num` is 127.

The string length for the result is given by this formula:

```
new_len = orig_len + (8 - (orig_len % 8)) + 1
```

Each line in the DES key file has the following format:

```
key_num des_key_str
```

Each `key_num` value must be a number in the range from 0 to 9. Lines in the file may be in any order. `des_key_str` is the string that is used to encrypt the message. There should be at least one space between the number and the key. The first key is the default key that is used if you do not specify any key argument to `DES_ENCRYPT()`.

You can tell MariaDB to read new key values from the key file with the `FLUSH DES_KEY_FILE` statement. This requires the `RELOAD` privilege.

One benefit of having a set of default keys is that it gives applications a way to check for the existence of encrypted column values, without giving the end user the right to decrypt those values.

Examples

```
SELECT customer_address FROM customer_table  
WHERE encrypted_credit_card = DES_ENCRYPT('credit_card_number');
```

See Also

- [DES_DECRYPT\(\)](#)

ENCODE

Syntax

```
ENCODE(str,pass_str)
```

Description

ENCODE is not considered cryptographically secure, and should not be used for password encryption.

Encrypt `str` using `pass_str` as the password. To decrypt the result, use `DECODE()`.

The result is a binary string of the same length as `str`.

The strength of the encryption is based on how good the random generator is.

It is not recommended to rely on the encryption performed by the ENCODE function. Using a salt value (changed when a password is updated) will improve matters somewhat, but for storing passwords, consider a more cryptographically secure function, such as [SHA2\(\)](#).

Examples

```
ENCODE('not so secret text', CONCAT('random_salt','password'))
```

ENCRYPT

Syntax

```
ENCRYPT(str[,salt])
```

Description

Encrypts a string using the Unix crypt() system call, returning an encrypted binary string. The `salt` argument should be a string with at least two characters or the returned result will be NULL. If no salt argument is given, a random value of sufficient length is used.

It is not recommended to use ENCRYPT() with utf16, utf32 or ucs2 multi-byte character sets because the crypt() system call expects a string terminated with a zero byte.

Note that the underlying crypt() system call may have some limitations, such as ignoring all but the first eight characters.

If the `have_crypt` system variable is set to `NO` (because the crypt() system call is not available), the ENCRYPT function will always return NULL.

Examples

```
SELECT ENCRYPT('encrypt me');
+-----+
| ENCRYPT('encrypt me') |
+-----+
| 4I5BsEx0lqTDk |
+-----+
```

INTERVAL

Syntax

```
INTERVAL(N,N1,N2,N3,...)
```

Description

Returns the index of the last argument that is less than the first argument or is NULL.

Returns 0 if N < N1, 1 if N < N2, 2 if N < N3 and so on or -1 if N is NULL. All arguments are treated as integers. It is required that N1 < N2 < N3 < ... < Nn for this function to work correctly. This is because a fast binary search is used.

Examples

```

SELECT INTERVAL(23, 1, 15, 17, 30, 44, 200);
+-----+
| INTERVAL(23, 1, 15, 17, 30, 44, 200) |
+-----+
|                                3 |
+-----+

SELECT INTERVAL(10, 1, 10, 100, 1000);
+-----+
| INTERVAL(10, 1, 10, 100, 1000) |
+-----+
|                               2 |
+-----+

SELECT INTERVAL(22, 23, 30, 44, 200);
+-----+
| INTERVAL(22, 23, 30, 44, 200) |
+-----+
|                                0 |
+-----+

SELECT INTERVAL(10, 2, NULL);
+-----+
| INTERVAL(10, 2, NULL) |
+-----+
|                               2 |
+-----+

```

IS NOT NULL

Syntax

IS NOT NULL

Description

Tests whether a value is not NULL. See also [NULL Values in MariaDB](#).

Examples

```

SELECT 1 IS NOT NULL, 0 IS NOT NULL, NULL IS NOT NULL;
+-----+-----+-----+
| 1 IS NOT NULL | 0 IS NOT NULL | NULL IS NOT NULL |
+-----+-----+-----+
|           1 |           1 |          0 |
+-----+-----+-----+

```

See also

- [NULL values](#)
- [IS NULL operator](#)
- [COALESCE function](#)
- [IFNULL function](#)
- [NULLIF function](#)
- [CONNECT data types](#)

ISNULL

Syntax

```
ISNULL(expr)
```

Description

If `expr` is NULL, `ISNULL()` returns 1, otherwise it returns 0.

See also [NULL Values in MariaDB](#).

Examples

```
SELECT ISNULL(1+1);
+-----+
| ISNULL(1+1) |
+-----+
|          0 |
+-----+  
  
SELECT ISNULL(1/0);
+-----+
| ISNULL(1/0) |
+-----+
|          1 |
+-----+
```

Operator Precedence

The precedence is the order in which the SQL operators are evaluated.

The following list shows the SQL operator precedence. Operators that appear first in the list have a higher precedence. Operators which are listed together have the same precedence.

- `INTERVAL`
- `BINARY`, `COLLATE`
- `!`
- `-` (unary minus), `[[bitwise-not]]` (unary bit inversion)
- `||` (string concatenation)
- `^`
- `*`, `/`, `DIV`, `%`, `MOD`
- `-`, `+`
- `<<`, `>>`
- `&`
- `|`
- `=` (comparison), `<=`, `>=`, `>`, `<=`, `<`, `<>`, `!=`, `IS`, `LIKE`, `REGEXP`, `IN`
- `BETWEEN`, `CASE`, `WHEN`, `THEN`, `ELSE`, `END`
- `NOT`
- `&&`, `AND`
- `XOR`
- `||` (logical or), `OR`
- `=` (assignment), `:=`

Functions precedence is always higher than operators precedence.

In this page `CASE` refers to the [CASE operator](#), not to the [CASE statement](#).

If the `HIGH_NOT_PRECEDENCE SQL_MODE` is set, `NOT` has the same precedence as `!`.

The `||` operator's precedence, as well as its meaning, depends on the `PIPES_AS_CONCAT SQL_MODE` flag: if it is on, `||` can be used to concatenate strings (like the `CONCAT()` function) and has a higher precedence.

The `=` operator's precedence depends on the context - it is higher when `=` is used as a comparison operator.

`Parenthesis` can be used to modify the operators precedence in an expression.

Short-circuit evaluation

The `AND`, `OR`, `&&` and `||` operators support short-circuit evaluation. This means that, in some cases, the expression on the right of those operators is not evaluated, because its result cannot affect the result. In the following cases, short-circuit evaluation is used and `x()` is not evaluated:

- `FALSE AND x()`
- `FALSE && x()`
- `TRUE OR x()`
- `TRUE || x()`
- `NULL BETWEEN x() AND x()`

Note however that the short-circuit evaluation does *not* apply to `NULL AND x()`. Also, `BETWEEN`'s right operands are not evaluated if the left operand is `NULL`, but in all other cases all the operands are evaluated.

This is a speed optimization. Also, since functions can have side-effects, this behavior can be used to choose whether execute them or not using a concise syntax:

```
SELECT some_function() OR log_error();
```

&

Syntax

```
&
```

Description

Bitwise AND. Converts the values to binary and compares bits. Only if both the corresponding bits are 1 is the resulting bit also 1.

See also [bitwise OR](#).

Examples

```
SELECT 2&1;
+----+
| 2&1 |
+----+
|   0 |
+----+

SELECT 3&1;
+----+
| 3&1 |
+----+
|   1 |
+----+

SELECT 29 & 15;
+-----+
| 29 & 15 |
+-----+
|      13 |
+-----+
```

OLD_PASSWORD

Syntax

```
OLD_PASSWORD(str)
```

Description

`OLD_PASSWORD()` was added to MySQL when the implementation of `PASSWORD()` was changed to improve security. `OLD_PASSWORD()` returns the value of the old (pre-MySQL 4.1) implementation of `PASSWORD()` as a string, and is intended to permit you to reset passwords for any pre-4.1 clients that need to connect to a more recent MySQL server version, or any version of MariaDB, without locking them out.

As of MariaDB 5.5, the return value is a nonbinary string in the connection `character set and collation`, determined by the values of the `character_set_connection` and `collation_connection` system variables. Before 5.5, the return value was a binary string.

The return value is 16 bytes in length, or NULL if the argument was NULL.

See Also

- [PASSWORD\(\)](#)
- [MySQL manual on password hashing](#)

SHA1

Syntax

```
SHA1(str), SHA(str)
```

Description

Calculates an SHA-1 160-bit checksum for the string `str`, as described in RFC 3174 (Secure Hash Algorithm).

The value is returned as a string of 40 hex digits, or NULL if the argument was NULL. As of MariaDB 5.5, the return value is a nonbinary string in the connection `character set and collation`, determined by the values of the `character_set_connection` and `collation_connection` system variables. Before 5.5, the return value was a binary string.

Examples

```
SELECT SHA1('some boring text');
+-----+
| SHA1('some boring text')           |
+-----+
| af969fc2085b1bb6d31e517d5c456def5cdd7093 |
+-----+
```

SHA2

Syntax

```
SHA2(str,hash_len)
```

Description

Given a string `str`, calculates an SHA-2 checksum, which is considered more cryptographically secure than its [SHA-1](#) equivalent. The SHA-2 family includes SHA-224, SHA-256, SHA-384, and SHA-512, and the `hash_len` must correspond to one of these, i.e. 224, 256, 384 or 512. 0 is equivalent to 256.

The return value is a nonbinary string in the connection `character set and collation`, determined by the values of the `character_set_connection` and `collation_connection` system variables.

NULL is returned if the hash length is not valid, or the string `str` is NULL.

SHA2 will only work if MariaDB was has been configured with [TLS support](#).

Contents

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

Examples

```
SELECT SHA2('Maria',224);
+-----+
| SHA2('Maria',224) |
+-----+
| 6cc67add32286412efcab9d0e1675a43a5c2ef3cec8879f81516ff83 |
+-----+  
  
SELECT SHA2('Maria',256);
+-----+
| SHA2('Maria',256) |
+-----+
| 9ff18ebe7449349f358e3af0b57cf7a032c1c6b2272cb2656ff85eb112232f16 |
+-----+  
  
SELECT SHA2('Maria',0);
+-----+
| SHA2('Maria',0) |
+-----+
| 9ff18ebe7449349f358e3af0b57cf7a032c1c6b2272cb2656ff85eb112232f16 |
+-----+
```

UNCOMPRESS

Syntax

```
UNCOMPRESS(string_to_uncompress)
```

Description

Uncompresses a string compressed by the [COMPRESS\(\)](#) function. If the argument is not a compressed value, the result is `NULL`. This function requires MariaDB to have been compiled with a compression library such as zlib. Otherwise, the return value is always `NULL`. The [have_compress](#) server system variable indicates whether a compression library is present.

Examples

```
SELECT UNCOMPRESS(COMPRESS('a string'));
+-----+
| UNCOMPRESS(COMPRESS('a string')) |
+-----+
| a string |
+-----+  
  
SELECT UNCOMPRESS('a string');
+-----+
| UNCOMPRESS('a string') |
+-----+
| NULL |
+-----+
```

UNCOMPRESSED_LENGTH

Syntax

```
UNCOMPRESSED_LENGTH(compressed_string)
```

Description

Returns the length that the compressed string had before being compressed with `COMPRESS()`.

`UNCOMPRESSED_LENGTH()` returns `NULL` or an incorrect result if the string is not compressed.

Until MariaDB 10.3.1, returns `MYSQL_TYPE_LONGLONG`, or `bigint(10)`, in all cases. From MariaDB 10.3.1, returns `MYSQL_TYPE_LONG`, or `int(10)`, when the result would fit within 32-bits.

Examples

```
SELECT UNCOMPRESSED_LENGTH(COMPRESS(REPEAT('a',30)));
+-----+
| UNCOMPRESSED_LENGTH(COMPRESS(REPEAT('a',30))) |
+-----+
|          30 |
+-----+
```

<<

Syntax

```
value1 << value2
```

Description

Converts a longlong (`BIGINT`) number (`value1`) to binary and shifts `value2` units to the left.

Examples

```
SELECT 1 << 2;
+-----+
| 1 << 2 |
+-----+
|      4 |
+-----+
```

>>

Syntax

```
value1 >> value2
```

Description

Converts a longlong (`BIGINT`) number (`value1`) to binary and shifts `value2` units to the right.

Examples

```
SELECT 4 >> 2;
+-----+
| 4 >> 2 |
+-----+
|      1 |
+-----+
```

BIT_COUNT

Syntax

```
BIT_COUNT(N)
```

Description

Returns the number of bits that are set in the argument N.

Examples

```
SELECT BIT_COUNT(29), BIT_COUNT(b'101010');
+-----+-----+
| BIT_COUNT(29) | BIT_COUNT(b'101010') |
+-----+-----+
|          4 |             3 |
+-----+-----+
```

Λ

Syntax

```
^
```

Description

Bitwise XOR. Converts the values to binary and compares bits. If one (and only one) of the corresponding bits is 1 is the resulting bit also 1.

Examples

```
SELECT 1 ^ 1;
+-----+
| 1 ^ 1 |
+-----+
|      0 |
+-----+

SELECT 1 ^ 0;
+-----+
| 1 ^ 0 |
+-----+
|      1 |
+-----+

SELECT 11 ^ 3;
+-----+
| 11 ^ 3 |
+-----+
|       8 |
+-----+
```

Syntax

|

Description

Bitwise OR. Converts the values to binary and compares bits. If either of the corresponding bits has a value of 1, the resulting bit is also 1.

See also [bitwise AND](#).

Examples

```
SELECT 2|1;
+-----+
| 2|1 |
+-----+
|      3 |
+-----+

SELECT 29 | 15;
+-----+
| 29 | 15 |
+-----+
|      31 |
+-----+
```

GET_LOCK

Syntax

```
GET_LOCK(str,timeout)
```

Description

Tries to obtain a lock with a name given by the string `str`, using a timeout of `timeout` seconds. Returns `1` if the lock was obtained successfully, `0`

if the attempt timed out (for example, because another client has previously locked the name), or `NULL` if an error occurred (such as running out of memory or the thread was killed with `mysqladmin kill`).

A lock is released with `RELEASE_LOCK()`, when the connection terminates (either normally or abnormally), or before MariaDB 10.0.2, when the connection executes another `GET_LOCK` statement. From MariaDB 10.0.2, a connection can hold multiple locks at the same time, so a lock that is no longer needed needs to be explicitly released.

Contents

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

The `IS_FREE_LOCK` function returns whether a specified lock is free or not, and the `IS_USED_LOCK` whether the function is in use or not.

Locks obtained with `GET_LOCK()` do not interact with transactions. That is, committing a transaction does not release any such locks obtained during the transaction.

From MariaDB 10.0.2, it is also possible to recursively set the same lock. If a lock with the same name is set `n` times, it needs to be released `n` times as well.

`str` is case insensitive for `GET_LOCK()` and related functions. If `str` is an empty string or `NULL`, `GET_LOCK()` returns `NULL` and does nothing. From MariaDB 10.2.2, `timeout` supports microseconds. Before then, it was rounded to the closest integer.

If the `metadata_lock_info` plugin is installed, locks acquired with this function are visible in the [Information Schema METADATA_LOCK_INFO](#) table.

This function can be used to implement application locks or to simulate record locks. Names are locked on a server-wide basis. If a name has been locked by one client, `GET_LOCK()` blocks any request by another client for a lock with the same name. This allows clients that agree on a given lock name to use the name to perform cooperative advisory locking. But be aware that it also allows a client that is not among the set of cooperating clients to lock a name, either inadvertently or deliberately, and thus prevent any of the cooperating clients from locking that name. One way to reduce the likelihood of this is to use lock names that are database-specific or application-specific. For example, use lock names of the form `db_name.str` or `app_name.str`.

Statements using the `GET_LOCK` function are [not safe for statement-based replication](#).

The patch to permit multiple locks was [contributed by Konstantin "Kostja" Osipov \(MDEV-3917\)](#).

Examples

```
SELECT GET_LOCK('lock1',10);
+-----+
| GET_LOCK('lock1',10) |
+-----+
|          1          |
+-----+

SELECT IS_FREE_LOCK('lock1'), IS_USED_LOCK('lock1');
+-----+
| IS_FREE_LOCK('lock1') | IS_USED_LOCK('lock1') |
+-----+
|          0          |         46          |
+-----+

SELECT IS_FREE_LOCK('lock2'), IS_USED_LOCK('lock2');
+-----+
| IS_FREE_LOCK('lock2') | IS_USED_LOCK('lock2') |
+-----+
|          1          |      NULL       |
+-----+
```

From MariaDB 10.0.2, multiple locks can be held:

```

SELECT GET_LOCK('lock2',10);
+-----+
| GET_LOCK('lock2',10) |
+-----+
|          1 |
+-----+


SELECT IS_FREE_LOCK('lock1'), IS_FREE_LOCK('lock2');
+-----+-----+
| IS_FREE_LOCK('lock1') | IS_FREE_LOCK('lock2') |
+-----+-----+
|          0 |          0 |
+-----+-----+


SELECT RELEASE_LOCK('lock1'), RELEASE_LOCK('lock2');
+-----+-----+
| RELEASE_LOCK('lock1') | RELEASE_LOCK('lock2') |
+-----+-----+
|          1 |          1 |
+-----+-----+

```

Before [MariaDB 10.0.2](#), a connection could only hold a single lock:

```

SELECT GET_LOCK('lock2',10);
+-----+
| GET_LOCK('lock2',10) |
+-----+
|          1 |
+-----+


SELECT IS_FREE_LOCK('lock1'), IS_FREE_LOCK('lock2');
+-----+-----+
| IS_FREE_LOCK('lock1') | IS_FREE_LOCK('lock2') |
+-----+-----+
|          1 |          0 |
+-----+-----+


SELECT RELEASE_LOCK('lock1'), RELEASE_LOCK('lock2');
+-----+-----+
| RELEASE_LOCK('lock1') | RELEASE_LOCK('lock2') |
+-----+-----+
|      NULL |          1 |
+-----+-----+

```

From [MariaDB 10.0.2](#), it is possible to hold the same lock recursively. This example is viewed using the [metadata_lock_info](#) plugin:

```
SELECT GET_LOCK('lock3',10);
+-----+
| GET_LOCK('lock3',10) |
+-----+
|          1 |
+-----+


SELECT GET_LOCK('lock3',10);
+-----+
| GET_LOCK('lock3',10) |
+-----+
|          1 |
+-----+


SELECT * FROM INFORMATION_SCHEMA.METADATA_LOCK_INFO;
+-----+-----+-----+-----+-----+
| THREAD_ID | LOCK_MODE           | LOCK_DURATION | LOCK_TYPE | TABLE_SCHEMA | TABLE_NAME
+-----+-----+-----+-----+-----+
|      46 | MDL_SHARED_NO_WRITE | NULL          | User lock  | lock3       |
+-----+-----+-----+-----+-----+


SELECT RELEASE_LOCK('lock3');
+-----+
| RELEASE_LOCK('lock3') |
+-----+
|          1 |
+-----+


SELECT * FROM INFORMATION_SCHEMA.METADATA_LOCK_INFO;
+-----+-----+-----+-----+-----+
| THREAD_ID | LOCK_MODE           | LOCK_DURATION | LOCK_TYPE | TABLE_SCHEMA | TABLE_NAME
+-----+-----+-----+-----+-----+
|      46 | MDL_SHARED_NO_WRITE | NULL          | User lock  | lock3       |
+-----+-----+-----+-----+-----+


SELECT RELEASE_LOCK('lock3');
+-----+
| RELEASE_LOCK('lock3') |
+-----+
|          1 |
+-----+


SELECT * FROM INFORMATION_SCHEMA.METADATA_LOCK_INFO;
Empty set (0.000 sec)
```

Timeout example: Connection 1:

```
SELECT GET_LOCK('lock4',10);
```

+	-----+
	GET_LOCK('lock4',10)
+	-----+
	1
+	-----+

Connection 2:

```
SELECT GET_LOCK('lock4',10);
```

After 10 seconds...

```
+-----+
| GET_LOCK('lock4',10) |
+-----+
|                                         0 |
+-----+
```

Deadlocks are automatically detected and resolved. Connection 1:

```
SELECT GET_LOCK('lock5',10);
+-----+
| GET_LOCK('lock5',10) |
+-----+
|          1 |
+-----+
```

Connection 2:

```
SELECT GET_LOCK('lock6',10);
+-----+
| GET_LOCK('lock6',10) |
+-----+
|          1 |
+-----+
```

Connection 1:

```
SELECT GET_LOCK('lock6',10);
+-----+
| GET_LOCK('lock6',10) |
+-----+
|          0 |
+-----+
```

Connection 2:

```
SELECT GET_LOCK('lock5',10);
ERROR 1213 (40001): Deadlock found when trying to get lock; try restarting transaction
```

See Also

- [RELEASE_LOCK](#)
- [IS_FREE_LOCK](#)
- [IS_USED_LOCK](#)
- [RELEASE_ALL_LOCKS](#)

INET6_ATON

Syntax

```
INET6_ATON(expr)
```

Description

Given an IPv6 or IPv4 network address as a string, returns a binary string that represents the numeric value of the address.

No trailing zone ID's or trailing network masks are permitted. For IPv4 addresses, or IPv6 addresses with IPv4 address parts, no classful addresses or trailing port numbers are permitted and octal numbers are not supported.

The returned binary string will be [VARBINARY\(16\)](#) or [VARBINARY\(4\)](#) for IPv6 and IPv4 addresses respectively.

Returns NULL if the argument is not understood.

MariaDB starting with 10.5.0

From MariaDB 10.5.0, INET6_ATON can take INET6 as an argument.

Examples

```

SELECT HEX(INET6_ATON('10.0.1.1'));
+-----+
| HEX(INET6_ATON('10.0.1.1')) |
+-----+
| 0A000101 |
+-----+

SELECT HEX(INET6_ATON('48f3::d432:1431:ba23:846f'));
+-----+
| HEX(INET6_ATON('48f3::d432:1431:ba23:846f')) |
+-----+
| 48F300000000000D4321431BA23846F |
+-----+

```

See Also

- [INET6_NTOA\(\)](#)
- [INET_ATON\(\)](#)
- [INET6 Data Type](#)

INET6_NTOA

Syntax

`INET6_NTOA(expr)`

Contents

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

Description

Given an IPv6 or IPv4 network address as a numeric binary string, returns the address as a nonbinary string in the connection character set.

The return string is lowercase, and is platform independent, since it does not use functions specific to the operating system. It has a maximum length of 39 characters.

Returns NULL if the argument is not understood.

Examples

```

SELECT INET6_NTOA(UNHEX('0A000101'));
+-----+
| INET6_NTOA(UNHEX('0A000101')) |
+-----+
| 10.0.1.1 |
+-----+

SELECT INET6_NTOA(UNHEX('48F300000000000D4321431BA23846F'));
+-----+
| INET6_NTOA(UNHEX('48F300000000000D4321431BA23846F')) |
+-----+
| 48f3::d432:1431:ba23:846f |
+-----+

```

See Also

- [INET6_ATON\(\)](#)
- [INET_NTOA\(\)](#)

IS_IPV4

Syntax

```
IS_IPV4(expr)
```

Description

If the expression is a valid IPv4 address, returns 1, otherwise returns 0.

`IS_IPV4()` is stricter than [INET_ATON\(\)](#), but as strict as [INET6_ATON\(\)](#), in determining the validity of an IPv4 address. This implies that if `IS_IPV4` returns 1, the same expression will always return a non-NULL result when passed to `INET_ATON()`, but that the reverse may not apply.

Examples

```
SELECT IS_IPV4('1110.0.1.1');
+-----+
| IS_IPV4('1110.0.1.1') |
+-----+
|          0 |
+-----+  
  
SELECT IS_IPV4('48f3::d432:1431:ba23:846f');
+-----+
| IS_IPV4('48f3::d432:1431:ba23:846f') |
+-----+
|          0 |
+-----+
```

IS_IPV4_COMPAT

Syntax

```
IS_IPV4_COMPAT(expr)
```

Description

Returns 1 if a given numeric binary string IPv6 address, such as returned by [INET6_ATON\(\)](#), is IPv4-compatible, otherwise returns 0.

MariaDB starting with 10.5.0

From [MariaDB 10.5.0](#), when the argument is not INET6, automatic implicit `CAST` to INET6 is applied. As a consequence, `IS_IPV4_COMPAT` now understands arguments in both text representation and binary(16) representation. Before [MariaDB 10.5.0](#), the function understood only binary(16) representation.

Examples

```
SELECT IS_IPV4_COMPAT(INET6_ATON '::10.0.1.1');
+-----+
| IS_IPV4_COMPAT(INET6_ATON '::10.0.1.1') |
+-----+
|          1 |
+-----+  
  
SELECT IS_IPV4_COMPAT(INET6_ATON '::48f3::d432:1431:ba23:846f');
+-----+
| IS_IPV4_COMPAT(INET6_ATON '::48f3::d432:1431:ba23:846f') |
+-----+
|          0 |
+-----+
```

IS_IPV4_MAPPED

Syntax

```
IS_IPV4_MAPPED(expr)
```

Description

Returns 1 if a given a numeric binary string IPv6 address, such as returned by [INET6_ATON\(\)](#), is a valid IPv4-mapped address, otherwise returns 0.

MariaDB starting with 10.5.0

From MariaDB 10.5.0, when the argument is not INET6, automatic implicit CAST to INET6 is applied. As a consequence, `IS_IPV4_MAPPED` now understands arguments in both text representation and binary(16) representation. Before MariaDB 10.5.0, the function understood only binary(16) representation.

Examples

```
SELECT IS_IPV4_MAPPED(INET6_ATON('::10.0.1.1'));
+-----+
| IS_IPV4_MAPPED(INET6_ATON('::10.0.1.1')) |
+-----+
|          0 |
+-----+
SELECT IS_IPV4_MAPPED(INET6_ATON '::ffff:10.0.1.1');
+-----+
| IS_IPV4_MAPPED(INET6_ATON '::ffff:10.0.1.1') |
+-----+
|          1 |
+-----+
```

~

Syntax

```
~
```

Description

Bitwise NOT. Converts the value to 4 bytes binary and inverts all bits.

Examples

```
SELECT 3 & ~1;
```

```
+-----+
| 3 & ~1 |
+-----+
|      2 |
+-----+
```

```
SELECT 5 & ~1;
```

```
+-----+
| 5 & ~1 |
+-----+
|      4 |
+-----+
```

TRUE FALSE

Description

The constants TRUE and FALSE evaluate to 1 and 0, respectively. The constant names can be written in any lettercase.

Examples

```
SELECT TRUE, true, FALSE, false;
+-----+-----+-----+
| TRUE | TRUE | FALSE | FALSE |
+-----+-----+-----+
|    1 |    1 |     0 |     0 |
+-----+-----+-----+
```

BLOB and TEXT Data Types

Description

A BLOB is a binary large object that can hold a variable amount of data. The four BLOB types are

- [TINYBLOB](#),
- [BLOB](#),
- [MEDIUMBLOB](#), and
- [LONGBLOB](#).

These differ only in the maximum length of the values they can hold.

The TEXT types are

- [TINYTEXT](#),
- [TEXT](#),
- [MEDIUMTEXT](#), and
- [LONGTEXT](#).
- [JSON](#) (alias for LONGTEXT)

These correspond to the four BLOB types and have the same maximum lengths and [storage requirements](#).

MariaDB starting with [10.2.1](#)

Starting from [MariaDB 10.2.1](#), BLOB and TEXT columns can have a DEFAULT value.

MariaDB starting with [10.4.3](#)

From [MariaDB 10.4](#), it is possible to set a unique index on columns that use the BLOB or TEXT data types.

MASTER_GTID_WAIT

Syntax

```
MASTER_GTID_WAIT(gtid-list[, timeout])
```

Description

This function takes a string containing a comma-separated list of [global transaction id's](#) (similar to the value of, for example, [gtid_binlog_pos](#)). It waits until the value of [gtid_slave_pos](#) has the same or higher seq_no within all replication domains specified in the gtid-list; in other words, it waits until the slave has reached the specified GTID position.

An optional second argument gives a timeout in seconds. If the timeout expires before the specified GTID position is reached, then the function returns -1. Passing NULL or a negative number for the timeout means no timeout, and the function will wait indefinitely.

If the wait completes without a timeout, 0 is returned. Passing NULL for the gtid-list makes the function return NULL immediately, without waiting.

The gtid-list may be the empty string, in which case `MASTER_GTID_WAIT()` returns immediately. If the gtid-list contains fewer domains than [gtid_slave_pos](#), then only those domains are waited upon. If gtid-list contains a domain that is not present in `@@gtid_slave_pos`, then `MASTER_GTID_WAIT()` will wait until an event containing such domain_id arrives on the slave (or until timed out or killed).

`MASTER_GTID_WAIT()` can be useful to ensure that a slave has caught up to a master. Simply take the value of [gtid_binlog_pos](#) on the master, and use it in a `MASTER_GTID_WAIT()` call on the slave; when the call completes, the slave will have caught up with that master position.

`MASTER_GTID_WAIT()` can also be used in client applications together with the [last_gtid](#) session variable. This is useful in a read-scaleout [replication](#) setup, where the application writes to a single master but divides the reads out to a number of slaves to distribute the load. In such a setup, there is a risk that an application could first do an update on the master, and then a bit later do a read on a slave, and if the slave is not fast enough, the data read from the slave might not include the update just made, possibly confusing the application and/or the end-user. One way to avoid this is to request the value of [last_gtid](#) on the master just after the update. Then before doing the read on the slave, do a `MASTER_GTID_WAIT()` on the value obtained from the master; this will ensure that the read is not performed until the slave has replicated sufficiently far for the update to have become visible.

Note that `MASTER_GTID_WAIT()` can be used even if the slave is configured not to use GTID for connections ([CHANGE MASTER TO master_use_gtid=no](#)). This is because from MariaDB 10, GTIDs are always logged on the master server, and always recorded on the slave servers.

Differences to MASTER_POS_WAIT()

- `MASTER_GTID_WAIT()` is global; it waits for any master connection to reach the specified GTID position. [MASTER_POS_WAIT\(\)](#) works only against a specific connection. This also means that while `MASTER_POS_WAIT()` aborts if its master connection is terminated with [STOP SLAVE](#) or due to an error, `MASTER_GTID_WAIT()` continues to wait while slaves are stopped.
- `MASTER_GTID_WAIT()` can take its timeout as a floating-point value, so a timeout in fractional seconds is supported, eg. `MASTER_GTID_WAIT("0-1-100", 0.5)`. (The minimum wait is one microsecond, 0.000001 seconds).
- `MASTER_GTID_WAIT()` allows one to specify a timeout of zero in order to do a non-blocking check to see if the slaves have progressed to a specific GTID position (`MASTER_POS_WAIT()` takes a zero timeout as meaning an infinite wait). To do an infinite `MASTER_GTID_WAIT()`, specify a negative timeout, or omit the timeout argument.
- `MASTER_GTID_WAIT()` does not return the number of events executed since the wait started, nor does it return NULL if a slave thread is stopped. It always returns either 0 for successful wait completed, or -1 for timeout reached (or NULL if the specified gtid-pos is NULL).

Since `MASTER_GTID_WAIT()` looks only at the seq_no part of the GTIDs, not the server_id, care is needed if a slave becomes diverged from another server so that two different GTIDs with the same seq_no (in the same domain) arrive at the same server. This situation is in any case best avoided; setting [gtid_strict_mode](#) is recommended, as this will prevent any such out-of-order sequence numbers from ever being replicated on a slave.

MASTER_POS_WAIT

Syntax

```
MASTER_POS_WAIT(log_name, log_pos[, timeout, ["connection_name"]])
```

Description

This function is useful in [replication](#) for controlling primary/replica synchronization. It blocks until the replica has read and applied all updates up to the specified position (`log_name, log_pos`) in the primary log. The return value is the number of log events the replica had to wait for to advance to the specified position. The function returns NULL if the replica SQL thread is not started, the replica's primary information is not initialized, the arguments are incorrect, or an error occurs. It returns -1 if the timeout has been exceeded. If the replica SQL thread stops while `MASTER_POS_WAIT()` is waiting, the function returns NULL. If the replica is past the specified position, the function returns immediately.

If a `timeout` value is specified, `MASTER_POS_WAIT()` stops waiting when `timeout` seconds have elapsed. `timeout` must be greater than 0; a zero or negative `timeout` means no `timeout`.

The `connection_name` is used when you are using [multi-source-replication](#). If you don't specify it, it's set to the value of the `default_master_connection` system variable.

Statements using the `MASTER_POS_WAIT()` function are not [safe for replication](#).

CHAR BYTE

Description

The `CHAR BYTE` data type is an alias for the `BINARY` data type. This is a compatibility feature.

LONGBLOB

Syntax

LONGBLOB

Description

A `BLOB` column with a maximum length of 4,294,967,295 bytes or 4GB ($2^{32} - 1$). The effective maximum length of LONGBLOB columns depends on the configured maximum packet size in the client/server protocol and available memory. Each LONGBLOB value is stored using a four-byte length prefix that indicates the number of bytes in the value.

Oracle Mode

MariaDB starting with 10.3

In [Oracle mode from MariaDB 10.3](#), `BLOB` is a synonym for `LONGBLOB`.

See Also

- [BLOB](#)
- [BLOB and TEXT Data Types](#)
- [Data Type Storage Requirements](#)
- [Oracle mode from MariaDB 10.3](#)

MEDIUMBLOB

Syntax

MEDIUMBLOB

Description

A `BLOB` column with a maximum length of 16,777,215 ($2^{24} - 1$) bytes. Each MEDIUMBLOB value is stored using a three-byte length prefix that

indicates the number of bytes in the value.

See Also

- [BLOB](#)
- [BLOB and TEXT Data Types](#)
- [Data Type Storage Requirements](#)

MEDIUMTEXT

Syntax

```
MEDIUMTEXT [CHARACTER SET charset_name] [COLLATE collation_name]
```

Description

A [TEXT](#) column with a maximum length of 16,777,215 ($2^{24} - 1$) characters. The effective maximum length is less if the value contains multi-byte characters. Each MEDIUMTEXT value is stored using a three-byte length prefix that indicates the number of bytes in the value.

See Also

- [TEXT](#)
- [BLOB and TEXT Data Types](#)
- [Data Type Storage Requirements](#)

RELEASE_LOCK

Syntax

```
RELEASE_LOCK(str)
```

Description

Releases the lock named by the string `str` that was obtained with [GET_LOCK\(\)](#). Returns 1 if the lock was released, 0 if the lock was not established by this thread (in which case the lock is not released), and `NULL` if the named lock did not exist. The lock does not exist if it was never obtained by a call to `GET_LOCK()` or if it has previously been released.

Contents

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

`str` is case insensitive. If `str` is an empty string or `NULL`, `RELEASE_LOCK()` returns `NULL` and does nothing.

Statements using the `RELEASE_LOCK()` function are not [safe for replication](#).

The [DO statement](#) is convenient to use with `RELEASE_LOCK()`.

Examples

Connection1:

```
SELECT GET_LOCK('lock1',10);
+-----+
| GET_LOCK('lock1',10) |
+-----+
|          1          |
+-----+
```

Connection 2:

```
SELECT GET_LOCK('lock2',10);
```

+	-+-----+
	GET_LOCK('lock2',10)
+	-+-----+
	1
+	-+-----+

Connection 1:

```
SELECT RELEASE_LOCK('lock1'), RELEASE_LOCK('lock2'), RELEASE_LOCK('lock3');
+-----+-----+-----+
| RELEASE_LOCK('lock1') | RELEASE_LOCK('lock2') | RELEASE_LOCK('lock3') |
+-----+-----+-----+
|           1 |                 0 |          NULL |
+-----+-----+-----+
```

From MariaDB 10.0.2, it is possible to hold the same lock recursively. This example is viewed using the `metadata_lock_info` plugin:

```
SELECT GET_LOCK('lock3',10);
+-----+
| GET_LOCK('lock3',10) |
+-----+
|          1 |
+-----+


SELECT GET_LOCK('lock3',10);
+-----+
| GET_LOCK('lock3',10) |
+-----+
|          1 |
+-----+


SELECT * FROM INFORMATION_SCHEMA.METADATA_LOCK_INFO;
+-----+-----+-----+-----+-----+
| THREAD_ID | LOCK_MODE           | LOCK_DURATION | LOCK_TYPE | TABLE_SCHEMA | TABLE_NAME |
+-----+-----+-----+-----+-----+
|      46 | MDL_SHARED_NO_WRITE | NULL          | User lock  | lock3       |
+-----+-----+-----+-----+-----+


SELECT RELEASE_LOCK('lock3');
+-----+
| RELEASE_LOCK('lock3') |
+-----+
|          1 |
+-----+


SELECT * FROM INFORMATION_SCHEMA.METADATA_LOCK_INFO;
+-----+-----+-----+-----+-----+
| THREAD_ID | LOCK_MODE           | LOCK_DURATION | LOCK_TYPE | TABLE_SCHEMA | TABLE_NAME |
+-----+-----+-----+-----+-----+
|      46 | MDL_SHARED_NO_WRITE | NULL          | User lock  | lock3       |
+-----+-----+-----+-----+-----+


SELECT RELEASE_LOCK('lock3');
+-----+
| RELEASE_LOCK('lock3') |
+-----+
|          1 |
+-----+


SELECT * FROM INFORMATION_SCHEMA.METADATA_LOCK_INFO;
Empty set (0.000 sec)
```

See Also

- GET_LOCK
 - IS_FREE_LOCK

- [IS_USED_LOCK](#)
- [RELEASE_ALL_LOCKS](#)

UUID

Syntax

```
UUID()
```

Description

Returns a Universally Unique Identifier (UUID).

A UUID is designed as a number that is globally unique in space and time. Two calls to `UUID()` are expected to generate two different values, even if these calls are performed on two separate computers that are not connected to each other.

Contents

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

`UUID()` results are intended to be unique, but cannot always be relied upon to unpredictable and unguessable, so should not be relied upon for these purposes.

A UUID is a 128-bit number represented by a utf8 string of five hexadecimal numbers in `aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeee` format:

- The first three numbers are generated from a timestamp.
- The fourth number preserves temporal uniqueness in case the timestamp value loses monotonicity (for example, due to daylight saving time).
- The fifth number is an IEEE 802 node number that provides spatial uniqueness. A random number is substituted if the latter is not available (for example, because the host computer has no Ethernet card, or we do not know how to find the hardware address of an interface on your operating system). In this case, spatial uniqueness cannot be guaranteed. Nevertheless, a collision should have very low probability.

Currently, the MAC address of an interface is taken into account only on FreeBSD and Linux. On other operating systems, MariaDB uses a randomly generated 48-bit number.

Statements using the `UUID()` function are not [safe for replication](#).

The results are generated according to the "DCE 1.1:Remote Procedure Call" (Appendix A) CAE (Common Applications Environment) Specifications published by The Open Group in October 1997 ([Document Number C706](#)).

Examples

```
SELECT UUID();  
+-----+  
| UUID() |  
+-----+  
| cd41294a-afb0-11df-bc9b-00241dd75637 |  
+-----+
```

See Also

- [UUID_SHORT\(\)](#) - Return short (64 bit) Universal Unique Identifier
- [SYS_GUID](#) - UUID without the - character for Oracle compatibility
- [UUID data type](#)

ROW

MariaDB starting with [10.3.0](#)

The `ROW` data type was introduced in [MariaDB 10.3.0](#).

Syntax

```
ROW (<field name> <data type> [{, <field name> <data type>}... ])
```

Description

ROW is a data type for [stored procedure](#) variables.

Features

ROW fields as normal variables

ROW fields (members) act as normal variables, and are able to appear in all query parts where a stored procedure variable is allowed:

- Assignment is using the := operator and the [SET](#) command:

```
a.x:= 10;  
a.x:= b.x;  
SET a.x= 10, a.y=20, a.z= b.z;
```

- Passing to functions and operators:

```
SELECT f1(rec.a), rec.a<10;
```

- Clauses (select list, WHERE, HAVING, LIMIT, etc...):

```
SELECT var.a, t1.b FROM t1 WHERE t1.b=var.b LIMIT var.c;
```

- INSERT values:

```
INSERT INTO t1 VALUES (rec.a, rec.b, rec.c);
```

- SELECT .. INTO targets

```
SELECT a,b INTO rec.a, rec.b FROM t1 WHERE t1.id=10;
```

- Dynamic SQL out parameters ([EXECUTE](#) and [EXECUTE IMMEDIATE](#))

```
EXECUTE IMMEDIATE 'CALL proc_with_out_param(?)' USING rec.a;
```

ROW type variables as FETCH targets

ROW type variables are allowed as [FETCH](#) targets:

```
FETCH cur INTO rec;
```

where `cur` is a CURSOR and `rec` is a ROW type stored procedure variable.

Note, currently an attempt to use `FETCH` for a ROW type variable returns this error:

```
ERROR 1328 (HY000): Incorrect number of FETCH variables
```

FETCH from a cursor `cur` into a ROW variable `rec` works as follows:

- The number of fields in `cur` must match the number of fields in `rec`. Otherwise, an error is reported.
- Assignment is done from left to right. The first cursor field is assigned to the first variable field, the second cursor field is assigned to the second variable field, etc.

Contents

- [Syntax](#)
- [Description](#)
- [Features](#)
 - [ROW fields as normal variables](#)
 - [ROW type variables as FETCH targets](#)
 - [ROW type variables as SELECT...INTO targets](#)
- [Features not implemented](#)
- [Examples](#)
 - [Declaring a ROW in a stored procedure](#)
 - [FETCH Examples](#)
 - [SELECT...INTO Examples](#)
- [See Also](#)

- Field names in `rec` are not important and can differ from field names in `cur`.

See [FETCH Examples](#) (below) for examples of using this with `sql_mode=ORACLE` and `sql_mode=DEFAULT`.

ROW type variables as SELECT...INTO targets

ROW type variables are allowed as `SELECT..INTO` targets with some differences depending on which `sql_mode` is in use.

- When using `sql_mode=ORACLE`, `table%ROWTYPE` and `cursor%ROWTYPE` variables can be used as `SELECT...INTO` targets.
- Using multiple `ROW` variables in the `SELECT..INTO` list will report an error.
- Using `ROW` variables with a different column count than in the `SELECT..INTO` list will report an error.

See [SELECT...INTO Examples](#) (below) for examples of using this with `sql_mode=ORACLE` and `sql_mode=DEFAULT`.

Features not implemented

The following features are planned, but not implemented yet:

- Returning a ROW type expression from a stored function (see [MDEV-12252](#)). This will need some grammar change to support field names after parentheses:

```
SELECT f1().x FROM DUAL;
```

- Returning a ROW type expression from a built-in hybrid type function, such as `CASE`, `IF`, etc.
- ROW of ROWs

Examples

Declaring a ROW in a stored procedure

```
DELIMITER $$  
CREATE PROCEDURE p1()  
BEGIN  
    DECLARE r ROW (c1 INT, c2 VARCHAR(10));  
    SET r.c1= 10;  
    SET r.c2= 'test';  
    INSERT INTO t1 VALUES (r.c1, r.c2);  
END;  
$$  
DELIMITER ;  
CALL p1();
```

FETCH Examples

A complete `FETCH` example for `sql_mode=ORACLE`:

```

DROP TABLE IF EXISTS t1;
CREATE TABLE t1 (a INT, b VARCHAR(32));
INSERT INTO t1 VALUES (10,'b10');
INSERT INTO t1 VALUES (20,'b20');
INSERT INTO t1 VALUES (30,'b30');

SET sql_mode=oracle;
DROP PROCEDURE IF EXISTS p1;
DELIMITER $$
CREATE PROCEDURE p1 AS
    rec ROW(a INT, b VARCHAR(32));
    CURSOR c IS SELECT a,b FROM t1;
BEGIN
    OPEN c;
    LOOP
        FETCH c INTO rec;
        EXIT WHEN c%NOTFOUND;
        SELECT ('rec=(' || rec.a || ',' || rec.b||')');
    END LOOP;
    CLOSE c;
END;
$$
DELIMITER ;
CALL p1();

```

A complete `FETCH` example for `sql_mode=DEFAULT` :

```

DROP TABLE IF EXISTS t1;
CREATE TABLE t1 (a INT, b VARCHAR(32));
INSERT INTO t1 VALUES (10,'b10');
INSERT INTO t1 VALUES (20,'b20');
INSERT INTO t1 VALUES (30,'b30');

SET sql_mode=DEFAULT;
DROP PROCEDURE IF EXISTS p1;
DELIMITER $$
CREATE PROCEDURE p1()
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE rec ROW(a INT, b VARCHAR(32));
    DECLARE c CURSOR FOR SELECT a,b FROM t1;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
    OPEN c;
read_loop:
    LOOP
        FETCH c INTO rec;
        IF done THEN
            LEAVE read_loop;
        END IF;
        SELECT CONCAT('rec=(',rec.a,',',rec.b,')');
    END LOOP;
    CLOSE c;
END;
$$
DELIMITER ;
CALL p1();

```

SELECT...INTO Examples

A `SELECT...INTO` example for `sql_mode=DEFAULT` :

```

SET sql_mode=DEFAULT;
DROP TABLE IF EXISTS t1;
DROP PROCEDURE IF EXISTS p1;
CREATE TABLE t1 (a INT, b VARCHAR(32));
INSERT INTO t1 VALUES (10,'b10');
DELIMITER $$ 
CREATE PROCEDURE p1()
BEGIN
    DECLARE rec1 ROW(a INT, b VARCHAR(32));
    SELECT * FROM t1 INTO rec1;
    SELECT rec1.a, rec1.b;
END;
$$
DELIMITER ;
CALL p1();

```

The above example returns:

rec1.a	rec1.b
10	b10

A SELECT...INTO example for `sql_mode=ORACLE` :

```

SET sql_mode=ORACLE;
DROP TABLE IF EXISTS t1;
DROP PROCEDURE IF EXISTS p1;
CREATE TABLE t1 (a INT, b VARCHAR(32));
INSERT INTO t1 VALUES (10,'b10');
DELIMITER $$ 
CREATE PROCEDURE p1 AS
    rec1 ROW(a INT, b VARCHAR(32));
BEGIN
    SELECT * FROM t1 INTO rec1;
    SELECT rec1.a, rec1.b;
END;
$$
DELIMITER ;
CALL p1();

```

The above example returns:

rec1.a	rec1.b
10	b10

An example for `sql_mode=ORACLE` using `table%ROWTYPE` variables as `SELECT..INTO` targets:

```

SET sql_mode=ORACLE;
DROP TABLE IF EXISTS t1;
DROP PROCEDURE IF EXISTS p1;
CREATE TABLE t1 (a INT, b VARCHAR(32));
INSERT INTO t1 VALUES (10,'b10');
DELIMITER $$ 
CREATE PROCEDURE p1 AS
    rec1 t1%ROWTYPE;
BEGIN
    SELECT * FROM t1 INTO rec1;
    SELECT rec1.a, rec1.b;
END;
$$
DELIMITER ;
CALL p1();

```

The above example returns:

rec1.a	rec1.b
10	b10

An example for `sql_mode=ORACLE` using `cursor%ROWTYPE` variables as `SELECT..INTO` targets:

```

SET sql_mode=ORACLE;
DROP TABLE IF EXISTS t1;
DROP PROCEDURE IF EXISTS p1;
CREATE TABLE t1 (a INT, b VARCHAR(32));
INSERT INTO t1 VALUES (10,'b10');
DELIMITER $$ 
CREATE PROCEDURE p1 AS
    CURSOR cur1 IS SELECT * FROM t1;
    rec1 cur1%ROWTYPE;
BEGIN
    SELECT * FROM t1 INTO rec1;
    SELECT rec1.a, rec1.b;
END;
$$
DELIMITER ;
CALL p1();

```

The above example returns:

rec1.a	rec1.b
10	b10

See Also

- [STORED PROCEDURES](#)
- [DECLARE Variable](#)

SET Data Type

Syntax

```
SET('value1','value2',...) [CHARACTER SET charset_name] [COLLATE collation_name]
```

Description

A set. A string object that can have zero or more values, each of which must be chosen from the list of values 'value1', 'value2', ... A SET column can have a maximum of 64 members. SET values are represented internally as integers.

SET values cannot contain commas.

If a SET contains duplicate values, an error will be returned if [strict mode](#) is enabled, or a warning if strict mode is not enabled.

See Also

- [Character Sets and Collations](#)
- [Data Type Storage Requirements](#)

TIMESTAMP

Syntax

```
TIMESTAMP [(<microsecond precision>)]
```

Description

A timestamp in the format `YYYY-MM-DD HH:MM:SS.fffffff`.

The timestamp field is generally used to define at which moment in time a row was added or updated and by default will automatically be assigned the current datetime when a record is inserted or updated. The automatic properties only apply to the first TIMESTAMP in the record; subsequent TIMESTAMP columns will not be changed.

MariaDB starting with 10.1.2

MariaDB 10.1.2 introduced the `--mysql56-temporal-format` option, on by default, which allows MariaDB to store TIMESTAMPs using the same low-level format MySQL 5.6 uses.

For more information, see [Internal Format](#).

Contents

1. [Syntax](#)
2. [Description](#)
3. [Supported Values](#)
4. [Automatic Values](#)
5. [Time Zones](#)
6. [Limitations](#)
7. [SQL_MODE=MAXDB](#)
8. [Internal Format](#)
9. [Examples](#)
10. [See Also](#)

Supported Values

MariaDB stores values that use the `TIMESTAMP` data type as the number of seconds since '1970-01-01 00:00:00' ([UTC](#)). This means that the `TIMESTAMP` data type can hold values between '1970-01-01 00:00:01' ([UTC](#)) and '2038-01-19 03:14:07' ([UTC](#)).

MariaDB can also store `microseconds` with a precision between 0 and 6. If no microsecond precision is specified, then 0 is used by default.

Automatic Values

MariaDB has special behavior for the first column that uses the `TIMESTAMP` data type in a specific table. For the first column that uses the `TIMESTAMP` data type in a specific table, MariaDB automatically assigns the following properties to the column:

- `DEFAULT CURRENT_TIMESTAMP`
- `ON UPDATE CURRENT_TIMESTAMP`

This means that if the column is not explicitly assigned a value in an `INSERT` or `UPDATE` query, then MariaDB will automatically initialize the column's value with the current date and time.

This automatic initialization for `INSERT` and `UPDATE` queries can also be **explicitly enabled** for a column that uses the `TIMESTAMP` data type by specifying the `DEFAULT CURRENT_TIMESTAMP` and `ON UPDATE CURRENT_TIMESTAMP` clauses for the column. In these clauses, any synonym of `CURRENT_TIMESTAMP` is accepted, including `CURRENT_TIMESTAMP()`, `NOW()`, `LOCALTIME`, `LOCALTIME()`, `LOCALTIMESTAMP`, and `LOCALTIMESTAMP()`.

This automatic initialization for `INSERT` queries can also be **explicitly disabled** for a column that uses the `TIMESTAMP` data type by specifying a constant `DEFAULT` value. For example, `DEFAULT 0`.

This automatic initialization for `UPDATE` queries can also be **explicitly disabled** for a column that uses the `TIMESTAMP` data type by specifying a `DEFAULT` clause for the column, but no `ON UPDATE` clause. If a `DEFAULT` clause is explicitly specified for a column that uses the `TIMESTAMP` data

type, but an `ON UPDATE` clause is not specified for the column, then the timestamp value will not automatically change when an `UPDATE` statement is executed.

MariaDB also has special behavior if `NULL` is assigned to column that uses the `TIMESTAMP` data type. If the column is assigned the `NULL` value in an `INSERT` or `UPDATE` query, then MariaDB will automatically initialize the column's value with the current date and time. For details, see [NULL values in MariaDB](#).

This automatic initialization for `NULL` values can also be **explicitly disabled** for a column that uses the `TIMESTAMP` data type by specifying the `NULL` attribute for the column. In this case, if the column's value is set to `NULL`, then the column's value will actually be set to `NULL`.

Time Zones

If a column uses the `TIMESTAMP` data type, then any inserted values are converted from the session's time zone to [Coordinated Universal Time \(UTC\)](#) when stored, and converted back to the session's time zone when retrieved.

MariaDB does not currently store any time zone identifier with the value of the `TIMESTAMP` data type. See [MDEV-10018](#) for more information.

MariaDB does not currently support time zone literals that contain time zone identifiers. See [MDEV-11829](#) for more information.

Limitations

- Because the `TIMESTAMP` value is stored as Epoch Seconds, the timestamp value '1970-01-01 00:00:00' (UTC) is reserved since the second #0 is used to represent '0000-00-00 00:00:00'.
- In [MariaDB 5.5](#) and before there could only be one `TIMESTAMP` column per table that had `CURRENT_TIMESTAMP` defined as its default value. This limit has no longer applied since [MariaDB 10.0](#).

SQL_MODE=MAXDB

If the `SQL_MODE` is set to `MAXDB`, `TIMESTAMP` fields will be silently converted to `DATETIME`.

Internal Format

In [MariaDB 10.1.2](#) a new temporal format was introduced from MySQL 5.6 that alters how the `TIME`, `DATETIME` and `TIMESTAMP` columns operate at lower levels. These changes allow these temporal data types to have fractional parts and negative values. You can disable this feature using the `mysql156_temporal_format` system variable.

Tables that include `TIMESTAMP` values that were created on an older version of MariaDB or that were created while the `mysql156_temporal_format` system variable was disabled continue to store data using the older data type format.

In order to update table columns from the older format to the newer format, execute an `ALTER TABLE... MODIFY COLUMN` statement that changes the column to the *same* data type. This change may be needed if you want to export the table's tablespace and import it onto a server that has `mysql156_temporal_format=ON` set (see [MDEV-15225](#)).

For instance, if you have a `TIMESTAMP` column in your table:

```
SHOW VARIABLES LIKE 'mysql156_temporal_format';

+-----+-----+
| Variable_name      | Value   |
+-----+-----+
| mysql156_temporal_format | ON      |
+-----+-----+

ALTER TABLE example_table MODIFY ts_col TIMESTAMP;
```

When MariaDB executes the `ALTER TABLE` statement, it converts the data from the older temporal format to the newer one.

In the event that you have several tables and columns using temporal data types that you want to switch over to the new format, make sure the system variable is enabled, then perform a dump and restore using `mysqldump`. The columns using relevant temporal data types are restored using the new temporal format.

Starting from [MariaDB 10.5.1](#) columns with old temporal formats are marked with a `/* mariadb-5.3 */` comment in the output of `SHOW CREATE TABLE`, `SHOW COLUMNS`, `DESCRIBE` statements, as well as in the `COLUMN_TYPE` column of the `INFORMATION_SCHEMA.COLUMNS` Table.

```

SHOW CREATE TABLE mariadb5312_timestamp\G
*****
      1. row *****
Table: mariadb5312_timestamp
Create Table: CREATE TABLE `mariadb5312_timestamp` (
  `ts0` timestamp /* mariadb-5.3 */ NOT NULL DEFAULT current_timestamp() ON UPDATE current_timestamp(),
  `ts6` timestamp(6) /* mariadb-5.3 */ NOT NULL DEFAULT '0000-00-00 00:00:00.000000'
) ENGINE=MyISAM DEFAULT CHARSET=latin1

```

Note: Prior to MySQL 4.1 a different format for the `TIMESTAMP` datatype was used. This format is unsupported in MariaDB 5.1 and upwards.

Examples

```

CREATE TABLE t (id INT, ts TIMESTAMP);

DESC t;
+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default           | Extra          |
+-----+-----+-----+-----+
| id    | int(11)   | YES  |     | NULL              |                |
| ts    | timestamp  | NO   |     | CURRENT_TIMESTAMP | on update CURRENT_TIMESTAMP |
+-----+-----+-----+-----+

INSERT INTO t(id) VALUES (1),(2);

SELECT * FROM t;
+-----+
| id  | ts           |
+-----+
| 1   | 2013-07-22 12:50:05 |
| 2   | 2013-07-22 12:50:05 |
+-----+

INSERT INTO t VALUES (3,NULL),(4,'2001-07-22 12:12:12');

SELECT * FROM t;
+-----+
| id  | ts           |
+-----+
| 1   | 2013-07-22 12:50:05 |
| 2   | 2013-07-22 12:50:05 |
| 3   | 2013-07-22 12:51:56 |
| 4   | 2001-07-22 12:12:12 |
+-----+

```

Converting to Unix epoch:

```

SELECT ts, UNIX_TIMESTAMP(ts) FROM t;
+-----+-----+
| ts           | UNIX_TIMESTAMP(ts) |
+-----+-----+
| 2013-07-22 12:50:05 | 1374490205 |
| 2013-07-22 12:50:05 | 1374490205 |
| 2013-07-22 12:51:56 | 1374490316 |
| 2001-07-22 12:12:12 | 995796732 |
+-----+-----+

```

Update also changes the timestamp:

```
UPDATE t set id=5 WHERE id=1;
```

```
SELECT * FROM t;
+-----+
| id | ts           |
+-----+
| 5  | 2013-07-22 14:52:33 |
| 2  | 2013-07-22 12:50:05 |
| 3  | 2013-07-22 12:51:56 |
| 4  | 2001-07-22 12:12:12 |
+-----+
```

Default NULL:

```
CREATE TABLE t2 (id INT, ts TIMESTAMP NULL ON UPDATE CURRENT_TIMESTAMP);
```

```
INSERT INTO t(id) VALUES (1),(2);
```

```
SELECT * FROM t2;
```

```
INSERT INTO t2(id) VALUES (1),(2);
```

```
SELECT * FROM t2;
```

```
+-----+
| id | ts       |
+-----+
| 1  | NULL    |
| 2  | NULL    |
+-----+
```

```
UPDATE t2 SET id=3 WHERE id=1;
```

```
SELECT * FROM t2;
```

```
+-----+
| id | ts       |
+-----+
| 3  | 2013-07-22 15:32:22 |
| 2  | NULL    |
+-----+
```

Only the first timestamp is automatically inserted and updated:

```
CREATE TABLE t3 (id INT, ts1 TIMESTAMP, ts2 TIMESTAMP);
```

```
INSERT INTO t3(id) VALUES (1),(2);
```

```
SELECT * FROM t3;
```

```
+-----+-----+
| id | ts1           | ts2           |
+-----+-----+
| 1  | 2013-07-22 15:35:07 | 0000-00-00 00:00:00 |
| 2  | 2013-07-22 15:35:07 | 0000-00-00 00:00:00 |
+-----+-----+
```

```
DESC t3;
```

```
+-----+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default          | Extra   |
+-----+-----+-----+-----+-----+
| id    | int(11)   | YES  |     | NULL             | |
| ts1   | timestamp  | NO   |     | CURRENT_TIMESTAMP | on update CURRENT_TIMESTAMP |
| ts2   | timestamp  | NO   |     | 0000-00-00 00:00:00 |
+-----+-----+-----+-----+-----+
```

Explicitly setting a timestamp with the `CURRENT_TIMESTAMP` function:

```

INSERT INTO t3(id,ts2) VALUES (3,CURRENT_TIMESTAMP());

SELECT * FROM t3;
+-----+-----+
| id   | ts1           | ts2          |
+-----+-----+
| 1    | 2013-07-22 15:35:07 | 0000-00-00 00:00:00 |
| 2    | 2013-07-22 15:35:07 | 0000-00-00 00:00:00 |
| 3    | 2013-07-22 15:38:52 | 2013-07-22 15:38:52 |
+-----+-----+

```

Specifying the timestamp as NOT NULL:

```

CREATE TABLE t4 (id INT, ts TIMESTAMP NOT NULL);

INSERT INTO t4(id) VALUES (1);
SELECT SLEEP(1);
INSERT INTO t4(id,ts) VALUES (2,NULL);

SELECT * FROM t4;

```

See Also

- [Data Type Storage Requirements](#)

TINYBLOB

Syntax

TINYBLOB

Description

A [BLOB](#) column with a maximum length of 255 ($2^8 - 1$) bytes. Each TINYBLOB value is stored using a one-byte length prefix that indicates the number of bytes in the value.

See Also

- [BLOB](#)
- [BLOB and TEXT Data Types](#)
- [Data Type Storage Requirements](#)

TINYTEXT

Syntax

TINYTEXT [CHARACTER SET charset_name] [COLLATE collation_name]

Description

A [TEXT](#) column with a maximum length of 255 ($2^8 - 1$) characters. The effective maximum length is less if the value contains multi-byte characters. Each TINYTEXT value is stored using a one-byte length prefix that indicates the number of bytes in the value.

See Also

- [TEXT](#)

- BLOB and TEXT Data Types
- Data Type Storage Requirements

VALUES / VALUE

Syntax

MariaDB starting with 10.3.3

```
VALUE(col_name)
```

MariaDB until 10.3.2

```
VALUES(col_name)
```

Description

In an `INSERT ... ON DUPLICATE KEY UPDATE` statement, you can use the `VALUES(col_name)` function in the `UPDATE` clause to refer to column values from the `INSERT` portion of the statement. In other words, `VALUES(col_name)` in the `UPDATE` clause refers to the value of `col_name` that would be inserted, had no duplicate-key conflict occurred. This function is especially useful in multiple-row inserts.

The `VALUES()` function is meaningful only in `INSERT ... ON DUPLICATE KEY UPDATE` statements and returns `NULL` otherwise.

In MariaDB 10.3.3 this function was renamed to `VALUE()`, because it's incompatible with the standard Table Value Constructors syntax, implemented in MariaDB 10.3.3.

The `VALUES()` function can still be used even from MariaDB 10.3.3, but only in `INSERT ... ON DUPLICATE KEY UPDATE` statements; it's a syntax error otherwise.

Examples

MariaDB starting with 10.3.3

```
INSERT INTO t (a,b,c) VALUES (1,2,3),(4,5,6)
ON DUPLICATE KEY UPDATE c=VALUE(a)+VALUE(b);
```

MariaDB until 10.3.2

```
INSERT INTO t (a,b,c) VALUES (1,2,3),(4,5,6)
ON DUPLICATE KEY UPDATE c=VALUES(a)+VALUES(b);
```

Syntax

OR, ||

Description

Logical OR. When both operands are non-NULL, the result is 1 if any operand is non-zero, and 0 otherwise. With a NULL operand, the result is 1 if the other operand is non-zero, and NULL otherwise. If both operands are NULL, the result is NULL.

For this operator, [short-circuit evaluation](#) can be used.

Note that, if the `PIPES_AS_CONCAT SQL_MODE` is set, `||` is used as a string concatenation operator. This means that `a || b` is the same as `CONCAT(a,b)`. See [CONCAT\(\)](#) for details.

Contents

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

Oracle Mode

MariaDB starting with 10.3

In [Oracle mode from MariaDB 10.3](#), `||` ignores `NULL`.

Examples

```
SELECT 1 || 1;
+-----+
| 1 || 1 |
+-----+
|      1 |
+-----+  
  
SELECT 1 || 0;
+-----+
| 1 || 0 |
+-----+
|      1 |
+-----+  
  
SELECT 0 || 0;
+-----+
| 0 || 0 |
+-----+
|      0 |
+-----+  
  
SELECT 0 || NULL;
+-----+
| 0 || NULL |
+-----+
|      NULL |
+-----+  
  
SELECT 1 || NULL;
+-----+
| 1 || NULL |
+-----+
|      1 |
+-----+
```

In [Oracle mode](#), from MariaDB 10.3:

```
SELECT 0 || NULL;
+-----+
| 0 || NULL |
+-----+
|      0 |
+-----+
```

See Also

- [Oracle mode from MariaDB 10.3](#)

XOR

Syntax

XOR

Description

XOR stands for eXclusive OR. Returns NULL if either operand is NULL. For non-NULL operands, evaluates to 1 if an odd number of operands is non-zero, otherwise 0 is returned.

Contents

- [Syntax](#)
- [Description](#)
- [Examples](#)

Examples

```
SELECT 1 XOR 1;
+-----+
| 1 XOR 1 |
+-----+
|      0   |
+-----+

SELECT 1 XOR 0;
+-----+
| 1 XOR 0 |
+-----+
|      1   |
+-----+

SELECT 1 XOR NULL;
+-----+
| 1 XOR NULL |
+-----+
|      NULL   |
+-----+
```

In the following example, the right `1 XOR 1` is evaluated first, and returns `0`. Then, `1 XOR 0` is evaluated, and `1` is returned.

```
SELECT 1 XOR 1 XOR 1;
+-----+
| 1 XOR 1 XOR 1 |
+-----+
|      1         |
+-----+
```

Assignment Operator (=)

Syntax

`identifier = expr`

Description

The equal sign is used as both an assignment operator in certain contexts, and as a [comparison operator](#). When used as assignment operator, the value on the right is assigned to the variable (or column, in some contexts) on the left.

Since its use can be ambiguous, unlike the [:= assignment operator](#), the `=` assignment operator cannot be used in all contexts, and is only valid as part of a `SET` statement, or the `SET` clause of an `UPDATE` statement

This operator works with both [user-defined variables](#) and [local variables](#).

Contents

- [Syntax](#)
- [Description](#)
- [Examples](#)

Examples

```
UPDATE table_name SET x = 2 WHERE x > 100;
```

```
SET @x = 1, @y := 2;
```

Stored Aggregate Functions

MariaDB starting with [10.3.3](#)

The ability to create stored aggregate functions was added in [MariaDB 10.3.3](#).

[Aggregate functions](#) are functions that are computed over a sequence of rows and return one result for the sequence of rows.

Creating a custom aggregate function is done using the [CREATE FUNCTION](#) statement with two main differences:

- The addition of the AGGREGATE keyword, so `CREATE AGGREGATE FUNCTION`
- The `FETCH GROUP NEXT ROW` instruction inside the loop
- Oracle PL/SQL compatibility using SQL/PL is provided

Contents

1. [Standard Syntax](#)
 1. [Using SQL/PL](#)
2. [Examples](#)
 1. [SQL/PL Example](#)
3. [See Also](#)

Standard Syntax

```
CREATE AGGREGATE FUNCTION function_name (parameters) RETURNS return_type
BEGIN
    All types of declarations
    DECLARE CONTINUE HANDLER FOR NOT FOUND RETURN return_val;
    LOOP
        FETCH GROUP NEXT ROW; // fetches next row from table
        other instructions
    END LOOP;
END
```

Stored aggregate functions were a [2016 Google Summer of Code](#) project by Varun Gupta.

Using SQL/PL

```
SET sql_mode=Oracle;
DELIMITER //

CREATE AGGREGATE FUNCTION function_name (parameters) RETURN return_type
    declarations
BEGIN
    LOOP
        FETCH GROUP NEXT ROW; -- fetches next row from table
        -- other instructions

    END LOOP;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN return_val;
END //
DELIMITER ;
```

Examples

First a simplified example:

```

CREATE TABLE marks(stud_id INT, grade_count INT);

INSERT INTO marks VALUES (1,6), (2,4), (3,7), (4,5), (5,8);

SELECT * FROM marks;
+-----+
| stud_id | grade_count |
+-----+
|      1 |          6 |
|      2 |          4 |
|      3 |          7 |
|      4 |          5 |
|      5 |          8 |
+-----+

DELIMITER //
CREATE AGGREGATE FUNCTION IF NOT EXISTS aggregate_count(x INT) RETURNS INT
BEGIN
    DECLARE count_students INT DEFAULT 0;
    DECLARE CONTINUE HANDLER FOR NOT FOUND
    RETURN count_students;
    LOOP
        FETCH GROUP NEXT ROW;
        IF x THEN
            SET count_students = count_students+1;
        END IF;
    END LOOP;
END //
DELIMITER ;

```

A non-trivial example that cannot easily be rewritten using existing functions:

```

DELIMITER //
CREATE AGGREGATE FUNCTION medi_int(x INT) RETURNS DOUBLE
BEGIN
    DECLARE CONTINUE HANDLER FOR NOT FOUND
    BEGIN
        DECLARE res DOUBLE;
        DECLARE cnt INT DEFAULT (SELECT COUNT(*) FROM tt);
        DECLARE lim INT DEFAULT (cnt-1) DIV 2;
        IF cnt % 2 = 0 THEN
            SET res = (SELECT AVG(a) FROM (SELECT a FROM tt ORDER BY a LIMIT lim,2) ttt);
        ELSE
            SET res = (SELECT a FROM tt ORDER BY a LIMIT lim,1);
        END IF;
        DROP TEMPORARY TABLE tt;
        RETURN res;
    END;
    CREATE TEMPORARY TABLE tt (a INT);
    LOOP
        FETCH GROUP NEXT ROW;
        INSERT INTO tt VALUES (x);
    END LOOP;
END //
DELIMITER ;

```

SQL/PL Example

This uses the same marks table as created above.

```

SET sql_mode=Oracle;
DELIMITER //

CREATE AGGREGATE FUNCTION aggregate_count(x INT) RETURN INT AS count_students INT DEFAULT 0;
BEGIN
LOOP
  FETCH GROUP NEXT ROW;
  IF x THEN
    SET count_students := count_students+1;
  END IF;
END LOOP;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RETURN count_students;
END aggregate_count //
DELIMITER ;

```

SELECT aggregate_count(stud_id) **FROM** marks;

See Also

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

AVG

Syntax

```
AVG([DISTINCT] expr)
```

Description

Returns the average value of expr. The DISTINCT option can be used to return the average of the distinct values of expr. NULL values are ignored. It is an [aggregate function](#), and so can be used with the [GROUP BY](#) clause.

AVG() returns NULL if there were no matching rows.

From [MariaDB 10.2.0](#), AVG() can be used as a [window function](#).

Contents

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

Examples

```

CREATE TABLE sales (sales_value INT);

INSERT INTO sales VALUES(10),(20),(20),(40);

SELECT AVG(sales_value) FROM sales;
+-----+
| AVG(sales_value) |
+-----+
|      22.5000 |
+-----+

SELECT AVG(DISTINCT(sales_value)) FROM sales;
+-----+
| AVG(DISTINCT(sales_value)) |
+-----+
|          23.3333 |
+-----+

```

Commonly, AVG() is used with a [GROUP BY](#) clause:

```

CREATE TABLE student (name CHAR(10), test CHAR(10), score TINYINT);

INSERT INTO student VALUES
('Chun', 'SQL', 75), ('Chun', 'Tuning', 73),
('Esben', 'SQL', 43), ('Esben', 'Tuning', 31),
('Kaolin', 'SQL', 56), ('Kaolin', 'Tuning', 88),
('Tatiana', 'SQL', 87), ('Tatiana', 'Tuning', 83);

SELECT name, AVG(score) FROM student GROUP BY name;
+-----+
| name | AVG(score) |
+-----+
| Chun |    74.0000 |
| Esben |   37.0000 |
| Kaolin |   72.0000 |
| Tatiana |  85.0000 |
+-----+

```

Be careful to avoid this common mistake, not grouping correctly and returning mismatched data:

```

SELECT name,test,AVG(score) FROM student;
+-----+
| name | test | MIN(score) |
+-----+
| Chun | SQL |       31 |
+-----+

```

As a [window function](#):

```

CREATE TABLE student_test (name CHAR(10), test CHAR(10), score TINYINT);

INSERT INTO student_test VALUES
    ('Chun', 'SQL', 75), ('Chun', 'Tuning', 73),
    ('Esben', 'SQL', 43), ('Esben', 'Tuning', 31),
    ('Kaolin', 'SQL', 56), ('Kaolin', 'Tuning', 88),
    ('Tatiana', 'SQL', 87), ('Tatiana', 'Tuning', 83);

SELECT name, test, score, AVG(score) OVER (PARTITION BY test)
    AS average_by_test FROM student_test;
+-----+-----+-----+-----+
| name | test | score | average_by_test |
+-----+-----+-----+-----+
| Chun | SQL | 75 | 65.2500 |
| Chun | Tuning | 73 | 68.7500 |
| Esben | SQL | 43 | 65.2500 |
| Esben | Tuning | 31 | 68.7500 |
| Kaolin | SQL | 56 | 65.2500 |
| Kaolin | Tuning | 88 | 68.7500 |
| Tatiana | SQL | 87 | 65.2500 |
| Tatiana | Tuning | 83 | 68.7500 |
+-----+-----+-----+-----+

```

See Also

- [MAX](#) (maximum)
- [MIN](#) (minimum)
- [SUM](#) (sum total)

BIT_AND

Syntax

```
BIT_AND(expr) [over_clause]
```

Description

Returns the bitwise AND of all bits in *expr*. The calculation is performed with 64-bit ([BIGINT](#)) precision. It is an [aggregate function](#), and so can be used with the [GROUP BY](#) clause.

If no rows match, `BIT_AND` will return a value with all bits set to 1. NULL values have no effect on the result unless all results are NULL, which is treated as no match.

From [MariaDB 10.2.0](#), `BIT_AND` can be used as a [window function](#) with the addition of the *over_clause*.

Examples

```

CREATE TABLE vals (x INT);

INSERT INTO vals VALUES(111),(110),(100);

SELECT BIT_AND(x), BIT_OR(x), BIT_XOR(x) FROM vals;
+-----+-----+-----+
| BIT_AND(x) | BIT_OR(x) | BIT_XOR(x) |
+-----+-----+-----+
| 100 | 111 | 101 |
+-----+-----+-----+

```

As an [aggregate function](#):

```

CREATE TABLE vals2 (category VARCHAR(1), x INT);

INSERT INTO vals2 VALUES
('a',111),('a',110),('a',100),
('b','000'),('b',001),('b',011);

SELECT category, BIT_AND(x), BIT_OR(x), BIT_XOR(x)
  FROM vals2 GROUP BY category;
+-----+-----+-----+
| category | BIT_AND(x) | BIT_OR(x) | BIT_XOR(x) |
+-----+-----+-----+
| a        |      100 |      111 |      101 |
| b        |        0 |       11 |       10 |
+-----+-----+-----+

```

No match:

```

SELECT BIT_AND(NULL);
+-----+
| BIT_AND(NULL) |
+-----+
| 18446744073709551615 |
+-----+

```

See Also

- [BIT_OR](#)
- [BIT_XOR](#)

BIT_OR

Syntax

```
BIT_OR(expr) [over_clause]
```

Description

Returns the bitwise OR of all bits in `expr`. The calculation is performed with 64-bit (`BIGINT`) precision. It is an [aggregate function](#), and so can be used with the `GROUP BY` clause.

If no rows match, `BIT_OR` will return a value with all bits set to `0`. NULL values have no effect on the result unless all results are NULL, which is treated as no match.

From [MariaDB 10.2.0](#), `BIT_OR` can be used as a [window function](#) with the addition of the `over_clause`.

Contents

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

Examples

```

CREATE TABLE vals (x INT);

INSERT INTO vals VALUES(111),(110),(100);

SELECT BIT_AND(x), BIT_OR(x), BIT_XOR(x) FROM vals;
+-----+-----+-----+
| BIT_AND(x) | BIT_OR(x) | BIT_XOR(x) |
+-----+-----+-----+
|      100 |      111 |      101 |
+-----+-----+-----+

```

As an [aggregate function](#):

```

CREATE TABLE vals2 (category VARCHAR(1), x INT);

INSERT INTO vals2 VALUES
('a',111),('a',110),('a',100),
('b','000'),('b',001),('b',011);

SELECT category, BIT_AND(x), BIT_OR(x), BIT_XOR(x)
  FROM vals2 GROUP BY category;
+-----+-----+-----+
| category | BIT_AND(x) | BIT_OR(x) | BIT_XOR(x) |
+-----+-----+-----+
| a        |      100 |      111 |      101 |
| b        |        0 |       11 |       10 |
+-----+-----+-----+

```

No match:

```

SELECT BIT_OR(NULL);
+-----+
| BIT_OR(NULL) |
+-----+
|          0 |
+-----+

```

See Also

- [BIT_AND](#)
- [BIT_XOR](#)

CLOSE

Syntax

```
CLOSE cursor_name
```

Description

This statement closes a previously [opened](#) cursor. The cursor must have been previously opened or else an error occurs.

If not closed explicitly, a cursor is closed at the end of the compound statement in which it was declared.

See [Cursor Overview](#) for an example.

Contents

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

See Also

- [Cursor Overview](#)
- [DECLARE CURSOR](#)
- [OPEN cursor_name](#)
- [FETCH cursor_name](#)
- [Cursors in Oracle mode](#)

DECLARE CURSOR

Syntax

<= MariaDB 10.2

```
DECLARE cursor_name CURSOR FOR select_statement
```

From MariaDB 10.3

```
DECLARE cursor_name CURSOR [(cursor_formal_parameter[, ...])] FOR select_statement  
cursor_formal_parameter:  
    name type [collate clause]
```

From MariaDB 10.8

```
DECLARE cursor_name CURSOR [(cursor_formal_parameter[, ...])] FOR select_statement  
cursor_formal_parameter:  
    [IN] name type [collate clause]
```

Description

This statement declares a [cursor](#). Multiple cursors may be declared in a [stored program](#), but each cursor in a given block must have a unique name.

`select_statement` is not executed until the [OPEN](#) statement is executed. It is important to remember this if the query produces an error, or calls functions which have side effects.

A `SELECT` associated to a cursor can use variables, but the query itself cannot be a variable, and cannot be dynamically composed. The `SELECT` statement cannot have an `INTO` clause.

Cursors must be declared before [HANDLERS](#), but after local variables and [CONDITIONS](#).

Contents

1. [Syntax](#)
2. [Description](#)
 1. [Parameters](#)
 2. [IN](#)
 3. [See Also](#)

Parameters

MariaDB starting with 10.3.0

From [MariaDB 10.3.0](#), cursors can have parameters. This is a non-standard SQL extension. Cursor parameters can appear in any part of the `DECLARE CURSOR select_statement` where a stored procedure variable is allowed (select list, WHERE, HAVING, LIMIT etc).

IN

MariaDB starting with 10.8.0

From [MariaDB 10.8.0](#) preview release, the `IN` qualifier is supported in the `cursor_formal_parameter` part of the syntax.

See [Cursor Overview](#) for an example.

See Also

- [Cursor Overview](#)
- [OPEN cursor_name](#)
- [FETCH cursor_name](#)
- [CLOSE cursor_name](#)
- [Cursors in Oracle mode](#)

FETCH

Syntax

```
FETCH cursor_name INTO var_name [, var_name] ...
```

Description

This statement fetches the next row (if a row exists) using the specified [open cursor](#), and advances the cursor pointer.

`var_name` can be a [local variable](#), but *not* a [user-defined variable](#).

If no more rows are available, a `No Data` condition occurs with `SQLSTATE` value `02000`. To detect this condition, you can set up a handler for it (or for a `NOT FOUND` condition).

See [Cursor Overview](#) for an example.

Contents

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

See Also

- [Cursor Overview](#)
- [DECLARE CURSOR](#)
- [OPEN cursor_name](#)
- [CLOSE cursor_name](#)
- [Cursors in Oracle mode](#)

GOTO

MariaDB starting with [10.3](#)

The `GOTO` statement was introduced in [MariaDB 10.3](#) for Oracle compatibility.

Syntax

```
GOTO label
```

Description

The `GOTO` statement causes the code to jump to the specified label, and continue operating from there. It is only accepted when in [Oracle mode](#).

Contents

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

Example

```

SET sql_mode=ORACLE;

DELIMITER //

CREATE OR REPLACE PROCEDURE p1 AS

BEGIN

    SELECT 1;
    GOTO label;
    SELECT 2;
    <<label>>
    SELECT 3;

END;

//


DELIMITER

call p1();
+---+
| 1 |
+---+
| 1 |
+---+
1 row in set (0.000 sec)

+---+
| 3 |
+---+
| 3 |
+---+
1 row in set (0.000 sec)

```

IF

Syntax

```

IF search_condition THEN statement_list
    [ELSEIF search_condition THEN statement_list] ...
    [ELSE statement_list]
END IF;

```

Description

`IF` implements a basic conditional construct. If the `search_condition` evaluates to true, the corresponding SQL statement list is executed. If no `search_condition` matches, the statement list in the `ELSE` clause is executed. Each `statement_list` consists of one or more statements.

See Also

- The [IF\(\) function](#), which differs from the `IF` statement described above.
- [Changes in Oracle mode from MariaDB 10.3](#)

BIT_XOR

Syntax

```
BIT_XOR(expr) [over_clause]
```

Description

Returns the bitwise XOR of all bits in `expr`. The calculation is performed with 64-bit (`BIGINT`) precision. It is an [aggregate function](#), and so can be used with the `GROUP BY` clause.

If no rows match, `BIT_XOR` will return a value with all bits set to `0`. NULL values have no effect on the result unless all results are NULL, which is treated as no match.

From [MariaDB 10.2.0](#), `BIT_XOR` can be used as a [window function](#) with the addition of the `over_clause`.

Examples

```
CREATE TABLE vals (x INT);

INSERT INTO vals VALUES(111),(110),(100);

SELECT BIT_AND(x), BIT_OR(x), BIT_XOR(x) FROM vals;
+-----+-----+-----+
| BIT_AND(x) | BIT_OR(x) | BIT_XOR(x) |
+-----+-----+-----+
|      100 |      111 |      101 |
+-----+-----+-----+
```

As an [aggregate function](#):

```
CREATE TABLE vals2 (category VARCHAR(1), x INT);

INSERT INTO vals2 VALUES
('a',111),('a',110),('a',100),
('b','000'),('b',001),('b',011);

SELECT category, BIT_AND(x), BIT_OR(x), BIT_XOR(x)
  FROM vals2 GROUP BY category;
+-----+-----+-----+
| category | BIT_AND(x) | BIT_OR(x) | BIT_XOR(x) |
+-----+-----+-----+
| a        |      100 |      111 |      101 |
| b        |        0 |       11 |       10 |
+-----+-----+-----+
```

No match:

```
SELECT BIT_XOR(NULL);
+-----+
| BIT_XOR(NULL) |
+-----+
|      0 |
+-----+
```

See Also

- [BIT_AND](#)
- [BIT_OR](#)

COUNT

Syntax

```
COUNT(expr)
```

Description

Contents

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

Returns a count of the number of non-NULL values of expr in the rows retrieved by a [SELECT](#) statement. The result is a [BIGINT](#) value. It is an [aggregate function](#), and so can be used with the [GROUP BY](#) clause.

`COUNT(*)` counts the total number of rows in a table.

`COUNT()` returns 0 if there were no matching rows.

From [MariaDB 10.2.0](#), `COUNT()` can be used as a [window function](#).

Contents

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

Examples

```
CREATE TABLE student (name CHAR(10), test CHAR(10), score TINYINT);

INSERT INTO student VALUES
('Chun', 'SQL', 75), ('Chun', 'Tuning', 73),
('Esben', 'SQL', 43), ('Esben', 'Tuning', 31),
('Kaolin', 'SQL', 56), ('Kaolin', 'Tuning', 88),
('Tatiana', 'SQL', 87), ('Tatiana', 'Tuning', 83);

SELECT COUNT(*) FROM student;
+-----+
| COUNT(*) |
+-----+
|      8   |
+-----+
```

`COUNT(DISTINCT)` example:

```
SELECT COUNT(DISTINCT name) FROM student;
+-----+
| COUNT(DISTINCT name) |
+-----+
|          4          |
+-----+
```

As a [window function](#)

```
CREATE OR REPLACE TABLE student_test (name CHAR(10), test CHAR(10), score TINYINT);

INSERT INTO student_test VALUES
('Chun', 'SQL', 75), ('Chun', 'Tuning', 73),
('Esben', 'SQL', 43), ('Esben', 'Tuning', 31),
('Kaolin', 'SQL', 56), ('Kaolin', 'Tuning', 88),
('Tatiana', 'SQL', 87);

SELECT name, test, score, COUNT(score) OVER (PARTITION BY name)
    AS tests_written FROM student_test;
+-----+-----+-----+
| name | test | score | tests_written |
+-----+-----+-----+
| Chun | SQL  | 75  |          2  |
| Chun | Tuning | 73  |          2  |
| Esben | SQL  | 43  |          2  |
| Esben | Tuning | 31  |          2  |
| Kaolin | SQL  | 56  |          2  |
| Kaolin | Tuning | 88  |          2  |
| Tatiana | SQL  | 87  |          1  |
+-----+-----+-----+
```

See Also

- [SELECT](#)
- [COUNT DISTINCT](#)
- [Window Functions](#)

MAX

Syntax

```
MAX([DISTINCT] expr)
```

Description

Returns the largest, or maximum, value of `expr`. `MAX()` can also take a string argument in which case it returns the maximum string value. The `DISTINCT` keyword can be used to find the maximum of the distinct values of `expr`, however, this produces the same result as omitting `DISTINCT`.

Note that `SET` and `ENUM` fields are currently compared by their string value rather than their relative position in the set, so `MAX()` may produce a different highest result than `ORDER BY DESC`.

It is an [aggregate function](#), and so can be used with the [GROUP BY](#) clause.

From [MariaDB 10.2.2](#), `MAX()` can be used as a [window function](#).

`MAX()` returns `NULL` if there were no matching rows.

Contents

- [Syntax](#)
- [Description](#)
- [Examples](#)
- [See Also](#)

Examples

```
CREATE TABLE student (name CHAR(10), test CHAR(10), score TINYINT);

INSERT INTO student VALUES
('Chun', 'SQL', 75), ('Chun', 'Tuning', 73),
('Esben', 'SQL', 43), ('Esben', 'Tuning', 31),
('Kaolin', 'SQL', 56), ('Kaolin', 'Tuning', 88),
('Tatiana', 'SQL', 87), ('Tatiana', 'Tuning', 83);

SELECT name, MAX(score) FROM student GROUP BY name;
+-----+-----+
| name | MAX(score) |
+-----+-----+
| Chun |      75 |
| Esben |     43 |
| Kaolin |    88 |
| Tatiana |   87 |
+-----+-----+
```

MAX string:

```
SELECT MAX(name) FROM student;
+-----+
| MAX(name) |
+-----+
| Tatiana |
+-----+
```

Be careful to avoid this common mistake, not grouping correctly and returning mismatched data:

```
SELECT name,test,MAX(SCORE) FROM student;
+-----+-----+
| name | test | MAX(SCORE) |
+-----+-----+
| Chun | SQL |      88 |
+-----+-----+
```

Difference between `ORDER BY DESC` and `MAX()`:

```

CREATE TABLE student2(name CHAR(10),grade ENUM('b','c','a'));

INSERT INTO student2 VALUES('Chun','b'),('Esben','c'),('Kaolin','a');

SELECT MAX(grade) FROM student2;
+-----+
| MAX(grade) |
+-----+
| c          |
+-----+

SELECT grade FROM student2 ORDER BY grade DESC LIMIT 1;
+-----+
| grade   |
+-----+
| a       |
+-----+

```

As a [window function](#):

```

CREATE OR REPLACE TABLE student_test (name CHAR(10), test CHAR(10), score TINYINT);
INSERT INTO student_test VALUES
    ('Chun', 'SQL', 75), ('Chun', 'Tuning', 73),
    ('Esben', 'SQL', 43), ('Esben', 'Tuning', 31),
    ('Kaolin', 'SQL', 56), ('Kaolin', 'Tuning', 88),
    ('Tatiana', 'SQL', 87);

SELECT name, test, score, MAX(score)
    OVER (PARTITION BY name) AS highest_score FROM student_test;
+-----+-----+-----+-----+
| name | test | score | highest_score |
+-----+-----+-----+-----+
| Chun | SQL | 75 | 75 |
| Chun | Tuning | 73 | 75 |
| Esben | SQL | 43 | 43 |
| Esben | Tuning | 31 | 43 |
| Kaolin | SQL | 56 | 88 |
| Kaolin | Tuning | 88 | 88 |
| Tatiana | SQL | 87 | 87 |
+-----+-----+-----+-----+

```

See Also

- [AVG](#) (average)
- [MIN](#) (minimum)
- [SUM](#) (sum total)
- [MIN/MAX optimization](#) used by the optimizer
- [GREATEST\(\)](#) returns the largest value from a list

ITERATE

Syntax

```
ITERATE label
```

`ITERATE` can appear only within `LOOP`, `REPEAT`, and `WHILE` statements. `ITERATE` means "do the loop again", and uses the statement's `label` to determine which statements to repeat. The label must be in the same stored program, not in a caller procedure.

If you try to use `ITERATE` with a non-existing label, or if the label is associated to a construct which is not a loop, the following error will be produced:

```
ERROR 1308 (42000): ITERATE with no matching label: <label_name>
```

Below is an example of how `ITERATE` might be used:

```

CREATE PROCEDURE doiterate(p1 INT)
BEGIN
    label1: LOOP
        SET p1 = p1 + 1;
        IF p1 < 10 THEN ITERATE label1; END IF;
        LEAVE label1;
    END LOOP label1;
    SET @x = p1;
END

```

See Also

- [LEAVE](#) - Exits a loop (or any labeled code block)

LOOP

Syntax

```
[begin_label:] LOOP
    statement_list
END LOOP [end_label]
```

Description

`LOOP` implements a simple loop construct, enabling repeated execution of the statement list, which consists of one or more statements, each terminated by a semicolon (i.e., `;`) statement delimiter. The statements within the loop are repeated until the loop is exited; usually this is accomplished with a `LEAVE` statement.

A `LOOP` statement can be [labeled](#). `end_label` cannot be given unless `begin_label` also is present. If both are present, they must be the same.

See [Delimiters](#) in the `mysql` client for more on delimiter usage in the client.

Contents

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

See Also

- [LOOP in Oracle mode](#)
- [ITERATE](#)
- [LEAVE](#)
- [FOR Loops](#)

MIN

Syntax

```
MIN([DISTINCT] expr)
```

Description

Returns the minimum value of `expr`. `MIN()` may take a string argument, in which case it returns the minimum string value. The `DISTINCT` keyword can be used to find the minimum of the distinct values of `expr`, however, this produces the same result as omitting `DISTINCT`.

Note that `SET` and `ENUM` fields are currently compared by their string value rather than their relative position in the set, so `MIN()` may produce a different lowest result than `ORDER BY ASC`.

It is an [aggregate function](#), and so can be used with the `GROUP BY` clause.

From [MariaDB 10.2.2](#), `MIN()` can be used as a [window function](#).

`MIN()` returns `NULL` if there were no matching rows.

Contents

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

Examples

```
CREATE TABLE student (name CHAR(10), test CHAR(10), score TINYINT);

INSERT INTO student VALUES
('Chun', 'SQL', 75), ('Chun', 'Tuning', 73),
('Esben', 'SQL', 43), ('Esben', 'Tuning', 31),
('Kaolin', 'SQL', 56), ('Kaolin', 'Tuning', 88),
('Tatiana', 'SQL', 87), ('Tatiana', 'Tuning', 83);

SELECT name, MIN(score) FROM student GROUP BY name;
+-----+
| name | MIN(score) |
+-----+
| Chun |      73 |
| Esben |     31 |
| Kaolin |    56 |
| Tatiana |   83 |
+-----+
```

MIN() with a string:

```
SELECT MIN(name) FROM student;
+-----+
| MIN(name) |
+-----+
| Chun      |
+-----+
```

Be careful to avoid this common mistake, not grouping correctly and returning mismatched data:

```
SELECT name,test,MIN(score) FROM student;
+-----+
| name | test | MIN(score) |
+-----+
| Chun | SQL |      31 |
+-----+
```

Difference between ORDER BY ASC and MIN():

```
CREATE TABLE student2(name CHAR(10),grade ENUM('b','c','a'));

INSERT INTO student2 VALUES('Chun','b'),('Esben','c'),('Kaolin','a');

SELECT MIN(grade) FROM student2;
+-----+
| MIN(grade) |
+-----+
| a          |
+-----+

SELECT grade FROM student2 ORDER BY grade ASC LIMIT 1;
+-----+
| grade |
+-----+
| b     |
+-----+
```

As a window function:

```

CREATE OR REPLACE TABLE student_test (name CHAR(10), test CHAR(10), score TINYINT);
INSERT INTO student_test VALUES
    ('Chun', 'SQL', 75), ('Chun', 'Tuning', 73),
    ('Esben', 'SQL', 43), ('Esben', 'Tuning', 31),
    ('Kaolin', 'SQL', 56), ('Kaolin', 'Tuning', 88),
    ('Tatiana', 'SQL', 87);

```

```

SELECT name, test, score, MIN(score)
    OVER (PARTITION BY name) AS lowest_score FROM student_test;
+-----+-----+-----+-----+
| name | test | score | lowest_score |
+-----+-----+-----+-----+
| Chun | SQL | 75 | 73 |
| Chun | Tuning | 73 | 73 |
| Esben | SQL | 43 | 31 |
| Esben | Tuning | 31 | 31 |
| Kaolin | SQL | 56 | 56 |
| Kaolin | Tuning | 88 | 56 |
| Tatiana | SQL | 87 | 87 |
+-----+-----+-----+-----+

```

See Also

- [AVG](#) (average)
- [MAX](#) (maximum)
- [SUM](#) (sum total)
- [MIN/MAX optimization](#) used by the optimizer
- [LEAST\(\)](#) returns the smallest value from a list.

STD

Syntax

```
STD(expr)
```

Description

Returns the population standard deviation of *expr*. This is an extension to standard SQL. The standard SQL function [STDDEV_POP\(\)](#) can be used instead.

It is an [aggregate function](#), and so can be used with the [GROUP BY](#) clause.

From [MariaDB 10.2.2](#), STD() can be used as a [window function](#).

This function returns `NULL` if there were no matching rows.

Examples

As an [aggregate function](#):

```

CREATE OR REPLACE TABLE stats (category VARCHAR(2), x INT);

INSERT INTO stats VALUES
('a',1),('a',2),('a',3),
('b',11),('b',12),('b',20),('b',30),('b',60);

SELECT category, STDDEV_POP(x), STDDEV_SAMP(x), VAR_POP(x)
  FROM stats GROUP BY category;
+-----+-----+-----+
| category | STDDEV_POP(x) | STDDEV_SAMP(x) | VAR_POP(x) |
+-----+-----+-----+
| a        |     0.8165 |      1.0000 |    0.6667 |
| b        |   18.0400 |   20.1693 | 325.4400 |
+-----+-----+-----+

```

As a [window function](#):

```

CREATE OR REPLACE TABLE student_test (name CHAR(10), test CHAR(10), score TINYINT);

INSERT INTO student_test VALUES
('Chun', 'SQL', 75), ('Chun', 'Tuning', 73),
('Esben', 'SQL', 43), ('Esben', 'Tuning', 31),
('Kaolin', 'SQL', 56), ('Kaolin', 'Tuning', 88),
('Tatiana', 'SQL', 87);

SELECT name, test, score, STDDEV_POP(score)
  OVER (PARTITION BY test) AS stddev_results FROM student_test;
+-----+-----+-----+-----+
| name   | test   | score | stddev_results |
+-----+-----+-----+-----+
| Chun   | SQL   | 75   | 16.9466 |
| Chun   | Tuning | 73   | 24.1247 |
| Esben  | SQL   | 43   | 16.9466 |
| Esben  | Tuning | 31   | 24.1247 |
| Kaolin | SQL   | 56   | 16.9466 |
| Kaolin | Tuning | 88   | 24.1247 |
| Tatiana| SQL   | 87   | 16.9466 |
+-----+-----+-----+-----+

```

See Also

- [STDDEV_POP](#) (equivalent, standard SQL)
- [STDDEV](#) (equivalent, Oracle-compatible non-standard SQL)
- [VAR_POP](#) (variance)
- [STDDEV_SAMP](#) (sample standard deviation)

STDDEV

Syntax

`STDDEV(expr)`

Description

Returns the population standard deviation of `expr`. This function is provided for compatibility with Oracle. The standard SQL function `STDDEV_POP()` can be used instead.

It is an [aggregate function](#), and so can be used with the `GROUP BY` clause.

From [MariaDB 10.2.2](#), `STDDEV()` can be used as a [window function](#).

This function returns `NULL` if there were no matching rows.

Contents

- [Syntax](#)
- [Description](#)
- [Examples](#)
- [See Also](#)

Examples

As an [aggregate function](#):

```
CREATE OR REPLACE TABLE stats (category VARCHAR(2), x INT);

INSERT INTO stats VALUES
  ('a',1),('a',2),('a',3),
  ('b',11),('b',12),('b',20),('b',30),('b',60);

SELECT category, STDDEV_POP(x), STDDEV_SAMP(x), VAR_POP(x)
  FROM stats GROUP BY category;
+-----+-----+-----+
| category | STDDEV_POP(x) | STDDEV_SAMP(x) | VAR_POP(x) |
+-----+-----+-----+
| a        |      0.8165 |      1.0000 |     0.6667 |
| b        |    18.0400 |    20.1693 | 325.4400 |
+-----+-----+-----+
```

As a [window function](#):

```
CREATE OR REPLACE TABLE student_test (name CHAR(10), test CHAR(10), score TINYINT);

INSERT INTO student_test VALUES
  ('Chun', 'SQL', 75), ('Chun', 'Tuning', 73),
  ('Esben', 'SQL', 43), ('Esben', 'Tuning', 31),
  ('Kaolin', 'SQL', 56), ('Kaolin', 'Tuning', 88),
  ('Tatiana', 'SQL', 87);

SELECT name, test, score, STDDEV_POP(score)
  OVER (PARTITION BY test) AS stddev_results
  FROM student_test;
+-----+-----+-----+-----+
| name   | test   | score | stddev_results |
+-----+-----+-----+-----+
| Chun   | SQL    | 75   |    16.9466 |
| Chun   | Tuning | 73   |    24.1247 |
| Esben  | SQL    | 43   |    16.9466 |
| Esben  | Tuning | 31   |    24.1247 |
| Kaolin | SQL    | 56   |    16.9466 |
| Kaolin | Tuning | 88   |    24.1247 |
| Tatiana| SQL    | 87   |    16.9466 |
+-----+-----+-----+-----+
```

See Also

- [STDDEV_POP](#) (equivalent, standard SQL)
- [STD](#) (equivalent, non-standard SQL)
- [VAR_POP](#) (variance)
- [STDDEV_SAMP](#) (sample standard deviation)

OPEN

Syntax

<= MariaDB 10.2

```
OPEN cursor_name
```

From MariaDB 10.3

```
OPEN cursor_name [expression[,...]];
```

Description

This statement opens a [cursor](#) which was previously declared with [DECLARE CURSOR](#).

The query associated to the [DECLARE CURSOR](#) is executed when [OPEN](#) is executed. It is important to remember this if the query produces an error, or calls functions which have side effects.

This is necessary in order to [FETCH](#) rows from a cursor.

See [Cursor Overview](#) for an example.

See Also

- [Cursor Overview](#)
- [DECLARE CURSOR](#)
- [FETCH cursor_name](#)
- [CLOSE cursor_name](#)
- [Cursors in Oracle mode](#)

RETURN

Syntax

```
RETURN expr
```

The `RETURN` statement terminates execution of a [stored function](#) and returns the value `expr` to the function caller. There must be at least one `RETURN` statement in a stored function. If the function has multiple exit points, all exit points must have a `RETURN`.

This statement is not used in [stored procedures](#), [triggers](#), or [events](#). [LEAVE](#) can be used instead.

The following example shows that `RETURN` can return the result of a [scalar subquery](#):

```
CREATE FUNCTION users_count() RETURNS BOOL
  READS SQL DATA
BEGIN
  RETURN (SELECT COUNT(DISTINCT User) FROM mysql.user);
END;
```

STDDEV_POP

Syntax

```
STDDEV_POP(expr)
```

Description

Returns the population standard deviation of `expr` (the square root of [VAR_POP\(\)](#)). You can also use [STD\(\)](#) or [STDDEV\(\)](#), which are equivalent but not standard SQL.

It is an [aggregate function](#), and so can be used with the [GROUP BY](#) clause.

From [MariaDB 10.2.2](#), `STDDEV_POP()` can be used as a [window function](#).

`STDDEV_POP()` returns `NULL` if there were no matching rows.

Examples

As an [aggregate function](#):

Contents

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

Contents

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

```

CREATE OR REPLACE TABLE stats (category VARCHAR(2), x INT);

INSERT INTO stats VALUES
  ('a',1),('a',2),('a',3),
  ('b',11),('b',12),('b',20),('b',30),('b',60);

SELECT category, STDDEV_POP(x), STDDEV_SAMP(x), VAR_POP(x)
  FROM stats GROUP BY category;
+-----+-----+-----+
| category | STDDEV_POP(x) | STDDEV_SAMP(x) | VAR_POP(x) |
+-----+-----+-----+
| a         |     0.8165 |      1.0000 |    0.6667 |
| b         |   18.0400 |   20.1693 | 325.4400 |
+-----+-----+-----+

```

As a [window function](#):

```

CREATE OR REPLACE TABLE student_test (name CHAR(10), test CHAR(10), score TINYINT);

INSERT INTO student_test VALUES
  ('Chun', 'SQL', 75), ('Chun', 'Tuning', 73),
  ('Esben', 'SQL', 43), ('Esben', 'Tuning', 31),
  ('Kaolin', 'SQL', 56), ('Kaolin', 'Tuning', 88),
  ('Tatiana', 'SQL', 87);

SELECT name, test, score, STDDEV_POP(score)
  OVER (PARTITION BY test) AS stddev_results FROM student_test;
+-----+-----+-----+-----+
| name   | test   | score | stddev_results |
+-----+-----+-----+-----+
| Chun   | SQL    | 75   | 16.9466 |
| Chun   | Tuning | 73   | 24.1247 |
| Esben  | SQL    | 43   | 16.9466 |
| Esben  | Tuning | 31   | 24.1247 |
| Kaolin | SQL    | 56   | 16.9466 |
| Kaolin | Tuning | 88   | 24.1247 |
| Tatiana| SQL    | 87   | 16.9466 |
+-----+-----+-----+-----+

```

See Also

- [STD](#) (equivalent, non-standard SQL)
- [STDDEV](#) (equivalent, Oracle-compatible non-standard SQL)
- [VAR_POP](#) (variance)
- [STDDEV_SAMP](#) (sample standard deviation)

SUM

Syntax

```
SUM([DISTINCT] expr)
```

Description

Returns the sum of *expr*. If the return set has no rows, `SUM()` returns `NULL`. The `DISTINCT` keyword can be used to sum only the distinct values of *expr*.

From [MariaDB 10.2.0](#), `SUM()` can be used as a [window function](#), although not with the `DISTINCT` specifier.

Contents

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

Examples

```

CREATE TABLE sales (sales_value INT);
INSERT INTO sales VALUES(10),(20),(20),(40);

SELECT SUM(sales_value) FROM sales;
+-----+
| SUM(sales_value) |
+-----+
|      90 |
+-----+

SELECT SUM(DISTINCT(sales_value)) FROM sales;
+-----+
| SUM(DISTINCT(sales_value)) |
+-----+
|        70 |
+-----+

```

Commonly, SUM is used with a [GROUP BY](#) clause:

```

CREATE TABLE sales (name CHAR(10), month CHAR(10), units INT);

INSERT INTO sales VALUES
('Chun', 'Jan', 75), ('Chun', 'Feb', 73),
('Esben', 'Jan', 43), ('Esben', 'Feb', 31),
('Kaolin', 'Jan', 56), ('Kaolin', 'Feb', 88),
('Tatiana', 'Jan', 87), ('Tatiana', 'Feb', 83);

SELECT name, SUM(units) FROM sales GROUP BY name;
+-----+
| name    | SUM(units) |
+-----+
| Chun    |      148 |
| Esben   |      74  |
| Kaolin  |      144 |
| Tatiana |      170 |
+-----+

```

The [GROUP BY](#) clause is required when using an aggregate function along with regular column data, otherwise the result will be a mismatch, as in the following common type of mistake:

```

SELECT name, SUM(units) FROM sales
;-----+
| name | SUM(units) |
+-----+
| Chun |      536 |
+-----+

```

As a [window function](#):

```

CREATE OR REPLACE TABLE student_test (name CHAR(10), test CHAR(10), score TINYINT);
INSERT INTO student_test VALUES
    ('Chun', 'SQL', 75), ('Chun', 'Tuning', 73),
    ('Esben', 'SQL', 43), ('Esben', 'Tuning', 31),
    ('Kaolin', 'SQL', 56), ('Kaolin', 'Tuning', 88),
    ('Tatiana', 'SQL', 87);

SELECT name, test, score, SUM(score) OVER (PARTITION BY name) AS total_score FROM student_test;
+-----+-----+-----+-----+
| name | test | score | total_score |
+-----+-----+-----+-----+
| Chun | SQL | 75 | 148 |
| Chun | Tuning | 73 | 148 |
| Esben | SQL | 43 | 74 |
| Esben | Tuning | 31 | 74 |
| Kaolin | SQL | 56 | 144 |
| Kaolin | Tuning | 88 | 144 |
| Tatiana | SQL | 87 | 87 |
+-----+-----+-----+-----+

```

See Also

- [AVG](#) (average)
- [MAX](#) (maximum)
- [MIN](#) (minimum)

VARIANCE

Syntax

`VARIANCE(expr)`

Description

Returns the population standard variance of `expr`. This is an extension to standard SQL. The standard SQL function [VAR_POP\(\)](#) can be used instead.

Variance is calculated by

- working out the mean for the set
- for each number, subtracting the mean and squaring the result
- calculate the average of the resulting differences

It is an [aggregate function](#), and so can be used with the [GROUP BY](#) clause.

From [MariaDB 10.2.2](#), `VARIANCE()` can be used as a [window function](#).

`VARIANCE()` returns `NULL` if there were no matching rows.

Examples

Contents

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

```

CREATE TABLE v(i tinyint);

INSERT INTO v VALUES(101),(99);

SELECT VARIANCE(i) FROM v;
+-----+
| VARIANCE(i) |
+-----+
| 1.0000 |
+-----+

INSERT INTO v VALUES(120),(80);

SELECT VARIANCE(i) FROM v;
+-----+
| VARIANCE(i) |
+-----+
| 200.5000 |
+-----+

```

As an [aggregate function](#):

```

CREATE OR REPLACE TABLE stats (category VARCHAR(2), x INT);

INSERT INTO stats VALUES
('a',1),('a',2),('a',3),
('b',11),('b',12),('b',20),('b',30),('b',60);

SELECT category, STDDEV_POP(x), STDDEV_SAMP(x), VAR_POP(x)
FROM stats GROUP BY category;
+-----+-----+-----+
| category | STDDEV_POP(x) | STDDEV_SAMP(x) | VAR_POP(x) |
+-----+-----+-----+
| a         |      0.8165 |       1.0000 |     0.6667 |
| b         |     18.0400 |     20.1693 | 325.4400 |
+-----+-----+-----+

```

As a [window function](#):

```

CREATE OR REPLACE TABLE student_test (name CHAR(10), test CHAR(10), score TINYINT);

INSERT INTO student_test VALUES
('Chun', 'SQL', 75), ('Chun', 'Tuning', 73),
('Esben', 'SQL', 43), ('Esben', 'Tuning', 31),
('Kaolin', 'SQL', 56), ('Kaolin', 'Tuning', 88),
('Tatiana', 'SQL', 87);

SELECT name, test, score, VAR_POP(score)
OVER (PARTITION BY test) AS variance_results FROM student_test;
+-----+-----+-----+-----+
| name   | test   | score | variance_results |
+-----+-----+-----+-----+
| Chun   | SQL    | 75   | 287.1875 |
| Chun   | Tuning | 73   | 582.0000 |
| Esben  | SQL    | 43   | 287.1875 |
| Esben  | Tuning | 31   | 582.0000 |
| Kaolin | SQL    | 56   | 287.1875 |
| Kaolin | Tuning | 88   | 582.0000 |
| Tatiana| SQL    | 87   | 287.1875 |
+-----+-----+-----+-----+

```

See Also

- [VAR_POP](#) (equivalent, standard SQL)
- [STDDEV_POP](#) (population standard deviation)
- [STDDEV_SAMP](#) (sample standard deviation)

WHILE

Syntax

```
[begin_label:] WHILE search_condition DO  
    statement_list  
END WHILE [end_label]
```

Description

The statement list within a `WHILE` statement is repeated as long as the `search_condition` is true. `statement_list` consists of one or more statements. If the loop must be executed at least once, `REPEAT ... LOOP` can be used instead.

A `WHILE` statement can be [labeled](#). `end_label` cannot be given unless `begin_label` also is present. If both are present, they must be the same.

Examples

```
CREATE PROCEDURE dowhile()  
BEGIN  
    DECLARE v1 INT DEFAULT 5;  
  
    WHILE v1 > 0 DO  
        ...  
        SET v1 = v1 - 1;  
    END WHILE;  
END
```

VAR_POP

Syntax

```
VAR_POP(expr)
```

Description

Returns the population standard variance of `expr`. It considers rows as the whole population, not as a sample, so it has the number of rows as the denominator. You can also use `VARIANCE()`, which is equivalent but is not standard SQL.

Variance is calculated by

- working out the mean for the set
- for each number, subtracting the mean and squaring the result
- calculate the average of the resulting differences

It is an [aggregate function](#), and so can be used with the `GROUP BY` clause.

From [MariaDB 10.2.2](#), `VAR_POP()` can be used as a [window function](#).

`VAR_POP()` returns `NULL` if there were no matching rows.

Contents

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

Examples

```

CREATE TABLE v(i tinyint);

INSERT INTO v VALUES(101),(99);

SELECT VAR_POP(i) FROM v;
+-----+
| VAR_POP(i) |
+-----+
| 1.0000 |
+-----+

INSERT INTO v VALUES(120),(80);

SELECT VAR_POP(i) FROM v;
+-----+
| VAR_POP(i) |
+-----+
| 200.5000 |
+-----+

```

As an [aggregate function](#):

```

CREATE OR REPLACE TABLE stats (category VARCHAR(2), x INT);

INSERT INTO stats VALUES
('a',1),('a',2),('a',3),
('b',11),('b',12),('b',20),('b',30),('b',60);

SELECT category, STDDEV_POP(x), STDDEV_SAMP(x), VAR_POP(x)
FROM stats GROUP BY category;
+-----+-----+-----+
| category | STDDEV_POP(x) | STDDEV_SAMP(x) | VAR_POP(x) |
+-----+-----+-----+
| a         |      0.8165 |      1.0000 |     0.6667 |
| b         |     18.0400 |     20.1693 | 325.4400 |
+-----+-----+-----+

```

As a [window function](#):

```

CREATE OR REPLACE TABLE student_test (name CHAR(10), test CHAR(10), score TINYINT);

INSERT INTO student_test VALUES
('Chun', 'SQL', 75), ('Chun', 'Tuning', 73),
('Esben', 'SQL', 43), ('Esben', 'Tuning', 31),
('Kaolin', 'SQL', 56), ('Kaolin', 'Tuning', 88),
('Tatiana', 'SQL', 87);

SELECT name, test, score, VAR_POP(score)
OVER (PARTITION BY test) AS variance_results FROM student_test;
+-----+-----+-----+
| name   | test   | score | variance_results |
+-----+-----+-----+
| Chun   | SQL    | 75   | 287.1875 |
| Esben  | SQL    | 43   | 287.1875 |
| Kaolin | SQL    | 56   | 287.1875 |
| Tatiana| SQL    | 87   | 287.1875 |
| Chun   | Tuning | 73   | 582.0000 |
| Esben  | Tuning | 31   | 582.0000 |
| Kaolin | Tuning | 88   | 582.0000 |
+-----+-----+-----+

```

See Also

- [VARIANCE](#) (equivalent, non-standard SQL)
- [STDDEV_POP](#) (population standard deviation)
- [STDDEV_SAMP](#) (sample standard deviation)

VAR_SAMP

Syntax

```
VAR_SAMP(expr)
```

Description

Returns the sample variance of *expr*. That is, the denominator is the number of rows minus one.

It is an [aggregate function](#), and so can be used with the `GROUP BY` clause.

From [MariaDB 10.2.2](#), `VAR_SAMP()` can be used as a [window function](#).

`VAR_SAMP()` returns `NULL` if there were no matching rows.

Contents

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

Examples

As an [aggregate function](#):

```
CREATE OR REPLACE TABLE stats (category VARCHAR(2), x INT);

INSERT INTO stats VALUES
    ('a',1),('a',2),('a',3),
    ('b',11),('b',12),('b',20),('b',30),('b',60);

SELECT category, STDDEV_POP(x), STDDEV_SAMP(x), VAR_POP(x)
    FROM stats GROUP BY category;
+-----+-----+-----+
| category | STDDEV_POP(x) | STDDEV_SAMP(x) | VAR_POP(x) |
+-----+-----+-----+
| a        |      0.8165 |      1.0000 |     0.6667 |
| b        |     18.0400 |     20.1693 |  325.4400 |
+-----+-----+-----+
```

As a [window function](#):

```
CREATE OR REPLACE TABLE student_test (name CHAR(10), test CHAR(10), score TINYINT);

INSERT INTO student_test VALUES
    ('Chun', 'SQL', 75), ('Chun', 'Tuning', 73),
    ('Esben', 'SQL', 43), ('Esben', 'Tuning', 31),
    ('Kaolin', 'SQL', 56), ('Kaolin', 'Tuning', 88),
    ('Tatiana', 'SQL', 87);

SELECT name, test, score, VAR_SAMP(score)
    OVER (PARTITION BY test) AS variance_results FROM student_test;
+-----+-----+-----+
| name   | test   | score | variance_results |
+-----+-----+-----+
| Chun   | SQL    | 75   |    382.9167 |
| Chun   | Tuning | 73   |    873.0000 |
| Esben  | SQL    | 43   |    382.9167 |
| Esben  | Tuning | 31   |    873.0000 |
| Kaolin | SQL    | 56   |    382.9167 |
| Kaolin | Tuning | 88   |    873.0000 |
| Tatiana| SQL    | 87   |    382.9167 |
+-----+-----+-----+
```

See Also

- [VAR_POP](#) (variance)
- [STDDEV_POP](#) (population standard deviation)

CHARSET

Syntax

```
CHARSET(str)
```

Description

Returns the [character set](#) of the string argument. If `str` is not a string, it is considered as a binary string (so the function returns 'binary'). This applies to `NULL`, too. The return value is a string in the utf8 [character set](#).

Examples

```
SELECT CHARSET('abc');
+-----+
| CHARSET('abc') |
+-----+
| latin1          |
+-----+

SELECT CHARSET(CONVERT('abc' USING utf8));
+-----+
| CHARSET(CONVERT('abc' USING utf8)) |
+-----+
| utf8            |
+-----+

SELECT CHARSET(USER());
+-----+
| CHARSET(USER()) |
+-----+
| utf8           |
+-----+
```

BUFFER

A synonym for [ST_BUFFER](#).

CONVEXHULL

A synonym for [ST_CONVEXHULL](#).

GEOMETRYCOLLECTION

Syntax

```
GeometryCollection(g1,g2,...)
```

Description

Constructs a [WKB](#) GeometryCollection. If any argument is not a well-formed WKB representation of a geometry, the return value is `NULL`.

Examples

```

CREATE TABLE gis_geometrycollection (g GEOMETRYCOLLECTION);
SHOW FIELDS FROM gis_geometrycollection;
INSERT INTO gis_geometrycollection VALUES
(GeomCollFromText('GEOMETRYCOLLECTION(POINT(0 0), LINESTRING(0 0,10 10))')),
(GeometryFromWKB(AsWKB(GeometryCollection(Point(44, 6), LineString(Point(3, 6), Point(7, 9))))),
(GeomFromText('GeometryCollection()')),
(GeomFromText('GeometryCollection EMPTY')));

```

MULTILINESTRING

Syntax

```
MultiLineString(ls1,ls2,...)
```

Description

Constructs a WKB MultiLineString value using [WKB LineString](#) arguments. If any argument is not a WKB LineString, the return value is `NULL`.

Example

```

CREATE TABLE gis_multi_line (g MULTILINESTRING);
INSERT INTO gis_multi_line VALUES
(MultiLineStringFromText('MULTILINESTRING((10 48,10 21,10 0),(16 0,16 23,16 48))'),
(MLineFromText('MULTILINESTRING((10 48,10 21,10 0))'),
(MLineFromWKB(AsWKB(MultiLineString(LineString(Point(1, 2), Point(3, 5)), LineString(Point(2, 5),Point(5, 8),Point(21, 7))))));

```

MULTIPOINT

Syntax

```
MultiPoint(pt1,pt2,...)
```

Description

Constructs a [WKB MultiPoint](#) value using WKB [Point](#) arguments. If any argument is not a WKB Point, the return value is `NULL`.

Examples

```

SET @g = ST_GEOFROMTEXT('MultiPoint( 1 1, 2 2, 5 3, 7 2, 9 3, 8 4, 6 6, 6 9, 4 9, 1 5 )');

CREATE TABLE gis_multi_point (g MULTIPOINT);
INSERT INTO gis_multi_point VALUES
(MultiPointFromText('MULTIPOINT(0 0,10 10,10 20,20 20)'),
(MPointFromText('MULTIPOINT(1 1,11 11,11 21,21 21)'),
(MPointFromWKB(AsWKB(MultiPoint(Point(3, 6), Point(4, 10))))));

```

COERCIBILITY

Syntax

```
COERCIBILITY(str)
```

Description

Returns the collation coercibility value of the string argument. Coercibility defines what will be converted to what in case of collation conflict, with an expression with higher coercibility being converted to the collation of an expression with lower coercibility.

Coercibility	Description	Example
0	Explicit	Value using a COLLATE clause
1	No collation	Concatenated strings using different collations
2	Implicit	Column value
3	Constant	USER() return value
4	Coercible	Literal string
5	Ignorable	NULL or derived from NULL

Examples

```
SELECT COERCIBILITY('abc' COLLATE latin1_swedish_ci);
+-----+
| COERCIBILITY('abc' COLLATE latin1_swedish_ci) |
+-----+
|          0 |
+-----+


SELECT COERCIBILITY(USER());
+-----+
| COERCIBILITY(USER()) |
+-----+
|          3 |
+-----+


SELECT COERCIBILITY('abc');
+-----+
| COERCIBILITY('abc') |
+-----+
|          4 |
+-----+
```

CURRENT_ROLE

Syntax

```
CURRENT_ROLE, CURRENT_ROLE()
```

Description

Returns the current [role](#) name. This determines your access privileges. The return value is a string in the utf8 [character set](#).

If there is no current role, NULL is returned.

The output of `SELECT CURRENT_ROLE` is equivalent to the contents of the [ENABLED_ROLES](#) Information Schema table.

[USER\(\)](#) returns the combination of user and host used to login. [CURRENT_USER\(\)](#) returns the account used to determine current connection's privileges.

Examples

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

SET ROLE staff;

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

CURRENT_USER

Syntax

```
CURRENT_USER, CURRENT_USER()
```

Description

Returns the user name and host name combination for the MariaDB account that the server used to authenticate the current client. This account determines your access privileges. The return value is a string in the utf8 [character set](#).

The value of `CURRENT_USER()` can differ from the value of `USER()`. `CURRENT_ROLE()` returns the current active role.

Examples

```
shell> mysql --user="anonymous"

select user(),current_user();
+-----+-----+
| user()      | current_user() |
+-----+-----+
| anonymous@localhost | @localhost |
+-----+-----+
```

When calling `CURRENT_USER()` in a stored procedure, it returns the owner of the stored procedure, as defined with `DEFINER`.

See Also

- [USER\(\)](#)
- [CREATE PROCEDURE](#)

DATABASE

Syntax

```
DATABASE()
```

Description

Returns the default (current) database name as a string in the utf8 [character set](#). If there is no default database, `DATABASE()` returns `NULL`. Within a [stored routine](#), the default database is the database that the routine is associated with, which is not necessarily the same as the database that is

the default in the calling context.

SCHEMA() is a synonym for DATABASE().

To select a default database, the [USE](#) statement can be run. Another way to set the default database is specifying its name at [mysql](#) command line client startup.

Examples

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

USE test;
Database changed

SELECT DATABASE();
+-----+
| DATABASE() |
+-----+
| test       |
+-----+
```

DEFAULT

Syntax

```
DEFAULT(col_name)
```

Description

Returns the default value for a table column. If the column has no default value (and is not `NULLABLE` - `NULLABLE` fields have a `NULL` default), an error is returned.

For integer columns using [AUTO_INCREMENT](#), `0` is returned.

When using `DEFAULT` as a value to set in an [INSERT](#) or [UPDATE](#) statement, you can use the bare keyword `DEFAULT` without the parentheses and argument to refer to the column in context. You can only use `DEFAULT` as a bare keyword if you are using it alone without a surrounding expression or function.

Examples

Select only non-default values for a column:

```
SELECT i FROM t WHERE i != DEFAULT(i);
```

Update values to be one greater than the default value:

```
UPDATE t SET i = DEFAULT(i)+1 WHERE i < 100;
```

When referring to the default value exactly in `UPDATE` or `INSERT`, you can omit the argument:

```
INSERT INTO t (i) VALUES (DEFAULT);
UPDATE t SET i = DEFAULT WHERE i < 100;
```

```
CREATE OR REPLACE TABLE t (
  i INT NOT NULL AUTO_INCREMENT,
  j TINYINT NOT NULL,
```

```

j INT DEFAULT 3,
l INT NOT NULL DEFAULT 4,
m INT,
PRIMARY KEY (i)
);

DESC t;
+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra           |
+-----+-----+-----+-----+
| i     | int(11)   | NO   | PRI  | NULL    | auto_increment |
| j     | int(11)   | NO   |       | NULL    |                 |
| k     | int(11)   | YES  |       | 3        |                 |
| l     | int(11)   | NO   |       | 4        |                 |
| m     | int(11)   | YES  |       | NULL    |                 |
+-----+-----+-----+-----+

INSERT INTO t (j) VALUES (1);
INSERT INTO t (j,m) VALUES (2,2);
INSERT INTO t (j,l,m) VALUES (3,3,3);

SELECT * FROM t;
+-----+-----+-----+
| i | j | k | l | m |
+-----+-----+-----+
| 1 | 1 | 3 | 4 | NULL |
| 2 | 2 | 3 | 4 | 2  |
| 3 | 3 | 3 | 3 | 3  |
+-----+-----+-----+


SELECT DEFAULT(i), DEFAULT(k), DEFAULT(l), DEFAULT(m) FROM t;
+-----+-----+-----+
| DEFAULT(i) | DEFAULT(k) | DEFAULT(l) | DEFAULT(m) |
+-----+-----+-----+
|          0 |         3 |         4 |      NULL |
|          0 |         3 |         4 |      NULL |
|          0 |         3 |         4 |      NULL |
+-----+-----+-----+


SELECT DEFAULT(i), DEFAULT(k), DEFAULT(l), DEFAULT(m), DEFAULT(j) FROM t;
ERROR 1364 (HY000): Field 'j' doesn't have a default value

SELECT * FROM t WHERE i = DEFAULT(i);
Empty set (0.001 sec)

SELECT * FROM t WHERE j = DEFAULT(j);
ERROR 1364 (HY000): Field 'j' doesn't have a default value

SELECT * FROM t WHERE k = DEFAULT(k);
+-----+-----+-----+
| i | j | k | l | m |
+-----+-----+-----+
| 1 | 1 | 3 | 4 | NULL |
| 2 | 2 | 3 | 4 | 2  |
| 3 | 3 | 3 | 3 | 3  |
+-----+-----+-----+


SELECT * FROM t WHERE l = DEFAULT(l);
+-----+-----+-----+
| i | j | k | l | m |
+-----+-----+-----+
| 1 | 1 | 3 | 4 | NULL |
| 2 | 2 | 3 | 4 | 2  |
+-----+-----+-----+


SELECT * FROM t WHERE m = DEFAULT(m);
Empty set (0.001 sec)

SELECT * FROM t WHERE m <= DEFAULT(m);
+-----+-----+-----+
| i | j | k | l | m |
+-----+-----+-----+

```

1	1	3	4	NULL	
---	---	---	---	------	--

See Also

- [CREATE TABLE DEFAULT Clause](#)

POINT

Syntax

```
Point(x,y)
```

Description

Constructs a [WKB](#) Point using the given coordinates.

Examples

```
SET @g = ST_GEOMFROMTEXT('Point(1 1)');

CREATE TABLE gis_point (g POINT);
INSERT INTO gis_point VALUES
  (PointFromText('POINT(10 10'))),
  (PointFromText('POINT(20 10'))),
  (PointFromText('POINT(20 20'))),
  (PointFromWKB(AsWKB(PointFromText('POINT(10 20)))));
```

PointOnSurface

A synonym for [ST_PointOnSurface](#).

ST_INTERSECTION

Syntax

```
ST_INTERSECTION(g1,g2)
```

Description

Returns a geometry that is the intersection, or shared portion, of geometry *g1* and geometry *g2*.

Examples

```
SET @g1 = ST_GEOMFROMTEXT('POINT(2 1)');
SET @g2 = ST_GEOMFROMTEXT('LINESTRING(2 1, 0 2)');
SELECT ASTEXT(ST_INTERSECTION(@g1,@g2));
+-----+
| ASTEXT(ST_INTERSECTION(@g1,@g2)) |
+-----+
| POINT(2 1)                         |
+-----+
```

ST_POINTONSURFACE

MariaDB starting with 10.1.2

ST_POINTONSURFACE() was introduced in [MariaDB 10.1.2](#)

Syntax

```
ST_PointOnSurface(g)
PointOnSurface(g)
```

Description

Given a geometry, returns a [POINT](#) guaranteed to intersect a surface. However, see [MDEV-7514](#).

ST_PointOnSurface() and PointOnSurface() are synonyms.

BINLOG

Syntax

```
BINLOG 'str'
```

Description

BINLOG is an internal-use statement. It is generated by the [mariadb-binlog/mysqlbinlog](#) program as the printable representation of certain events in [binary log](#) files. The 'str' value is a base 64-encoded string that the server decodes to determine the data change indicated by the corresponding event. This statement requires the [SUPER](#) privilege (<= MariaDB 10.5.1) or the [BINLOG REPLAY](#) privilege (>= MariaDB 10.5.2).

See also

- [MariaDB replication](#)

DECODE_HISTOGRAM

Syntax

```
DECODE_HISTOGRAM(hist_type,histogram)
```

Description

Returns a string of comma separated numeric values corresponding to a probability distribution represented by the histogram of type `hist_type` (`SINGLE_PREC_HB` or `DOUBLE_PREC_HB`). The `hist_type` and `histogram` would be commonly used from the [mysql.column_stats table](#).

See [Histogram Based Statistics](#) for details.

Examples

```

CREATE TABLE origin (
  i INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
  v INT UNSIGNED NOT NULL
);

INSERT INTO origin(v) VALUES
(1),(2),(3),(4),(5),(10),(20),
(30),(40),(50),(60),(70),(80),
(90),(100),(200),(400),(800);

SET histogram_size=10,histogram_type=SINGLE_PREC_HB;

ANALYZE TABLE origin PERSISTENT FOR ALL;
+-----+-----+-----+
| Table | Op      | Msg_type | Msg_text
+-----+-----+-----+
| test.origin | analyze | status    | Engine-independent statistics collected |
| test.origin | analyze | status    | OK
+-----+-----+-----+

SELECT db_name,table_name,column_name,hist_type,
hex(histogram),decode_histogram(hist_type,histogram)
FROM mysql.column_stats WHERE db_name='test' and table_name='origin';
+-----+-----+-----+-----+
| db_name | table_name | column_name | hist_type       | hex(histogram)           | decode_histogram(hist_type,histogram)
+-----+-----+-----+-----+
| test     | origin     | i          | SINGLE_PREC_HB | 0F2D3C5A7887A5C3D2F0 | 0.059,0.118,0.059,0.118,0.118,0.059,0.118,0.118,0.059
| test     | origin     | v          | SINGLE_PREC_HB | 000001060C0F161C1F7F | 0.000,0.000,0.004,0.020,0.024,0.012,0.027,0.024,0.012
+-----+-----+-----+-----+

SET histogram_size=20,histogram_type=DOUBLE_PREC_HB;

ANALYZE TABLE origin PERSISTENT FOR ALL;
+-----+-----+-----+
| Table | Op      | Msg_type | Msg_text
+-----+-----+-----+
| test.origin | analyze | status    | Engine-independent statistics collected |
| test.origin | analyze | status    | OK
+-----+-----+-----+

SELECT db_name,table_name,column_name,
hist_type,hex(histogram),decode_histogram(hist_type,histogram)
FROM mysql.column_stats WHERE db_name='test' and table_name='origin';
+-----+-----+-----+-----+
| db_name | table_name | column_name | hist_type       | hex(histogram)           | decode_histogram(hist_type,histogram)
+-----+-----+-----+-----+
| test     | origin     | i          | DOUBLE_PREC_HB | 0F0F2D2D3C3C5A5A78788787A5A5C3C3D2D2F0F0 | 0.05882,0.11765,0.05882,0.11765,0
| test     | origin     | v          | DOUBLE_PREC_HB | 5200F600480116067E0CB30F1B16831CB81FD67F | 0.00125,0.00250,0.00125,0.01877,0
+-----+-----+-----+-----+

```

FOUND ROWS

Syntax

FOUND ROWS()

Description

A `SELECT` statement may include a `LIMIT` clause to restrict the number of rows the server returns to the client. In some cases, it is desirable to know how many rows the statement would have returned without the `LIMIT`, but without running the statement again. To obtain this row count, include a `SQL_CALC_FOUND_ROWS` option in the `SELECT` statement, and then invoke `FOUND_ROWS()` afterwards.

You can also use `FOUND_ROWS()` to obtain the number of rows returned by a `SELECT` which does not contain a `LIMIT` clause. In this case you don't need to use the `SQL_CALC_FOUND_ROWS` option. This can be useful for example in a `stored procedure`.

Also, this function works with some other statements which return a resultset, including [SHOW](#), [DESC](#) and [HELP](#). For [DELETE ... RETURNING](#) you should use [ROW_COUNT\(\)](#). It also works as a [prepared statement](#), or after executing a prepared statement.

Statements which don't return any results don't affect FOUND_ROWS() - the previous value will still be returned.

Warning: When used after a [CALL](#) statement, this function returns the number of rows selected by the last query in the procedure, not by the whole procedure.

Statements using the FOUND_ROWS() function are not [safe for replication](#).

Examples

```
SHOW ENGINES;
+-----+-----+-----+
| Engine | Support | Comment           | Transactions | XA | Savepoints |
+-----+-----+-----+
| InnoDB | DEFAULT | Supports transactions, row-level locking, and foreign keys | YES        | YES | YES      |
...
| SPHINX | YES     | Sphinx storage engine          | NO         | NO  | NO       |
+-----+-----+-----+
11 rows in set (0.01 sec)

SELECT FOUND_ROWS();
+-----+
| FOUND_ROWS() |
+-----+
|      11      |
+-----+

SELECT SQL_CALC_FOUND_ROWS * FROM tbl_name WHERE id > 100 LIMIT 10;

SELECT FOUND_ROWS();
+-----+
| FOUND_ROWS() |
+-----+
|      23      |
+-----+
```

See Also

- [ROW_COUNT\(\)](#)

LOAD INDEX

Syntax

```
LOAD INDEX INTO CACHE
tbl_index_list [, tbl_index_list] ...

tbl_index_list:
tbl_name
[[INDEX|KEY] (index_name[, index_name] ...)]
[IGNORE LEAVES]
```

Description

The `LOAD INDEX INTO CACHE` statement preloads a table index into the key cache to which it has been assigned by an explicit `CACHE INDEX` statement, or into the default key cache otherwise. `LOAD INDEX INTO CACHE` is used only for [MyISAM](#) or [Aria](#) tables. Until [MariaDB 5.3](#), it was not supported for tables having user-defined partitioning, but this limitation was removed in [MariaDB 5.5](#).

The `IGNORE LEAVES` modifier causes only blocks for the nonleaf nodes of the index to be preloaded.

LAST_INSERT_ID

Syntax

```
LAST_INSERT_ID(), LAST_INSERT_ID(expr)
```

Description

`LAST_INSERT_ID()` (no arguments) returns the first automatically generated value successfully inserted for an `AUTO_INCREMENT` column as a result of the most recently executed `INSERT` statement. The value of `LAST_INSERT_ID()` remains unchanged if no rows are successfully inserted.

If one gives an argument to `LAST_INSERT_ID()`, then it will return the value of the expression and the next call to `LAST_INSERT_ID()` will return the same value. The value will also be sent to the client and can be accessed by the `mysql_insert_id` function.

For example, after inserting a row that generates an `AUTO_INCREMENT` value, you can get the value like this:

```
SELECT LAST_INSERT_ID();
+-----+
| LAST_INSERT_ID() |
+-----+
|         9 |
+-----+
```

You can also use `LAST_INSERT_ID()` to delete the last inserted row:

```
DELETE FROM product WHERE id = LAST_INSERT_ID();
```

If no rows were successfully inserted, `LAST_INSERT_ID()` returns 0.

The value of `LAST_INSERT_ID()` will be consistent across all versions if all rows in the `INSERT` or `UPDATE` statement were successful.

The currently executing statement does not affect the value of `LAST_INSERT_ID()`. Suppose that you generate an `AUTO_INCREMENT` value with one statement, and then refer to `LAST_INSERT_ID()` in a multiple-row `INSERT` statement that inserts rows into a table with its own `AUTO_INCREMENT` column. The value of `LAST_INSERT_ID()` will remain stable in the second statement; its value for the second and later rows is not affected by the earlier row insertions. (However, if you mix references to `LAST_INSERT_ID()` and `LAST_INSERT_ID(expr)`, the effect is undefined.)

If the previous statement returned an error, the value of `LAST_INSERT_ID()` is undefined. For transactional tables, if the statement is rolled back due to an error, the value of `LAST_INSERT_ID()` is left undefined. For manual `ROLLBACK`, the value of `LAST_INSERT_ID()` is not restored to that before the transaction; it remains as it was at the point of the `ROLLBACK`.

Within the body of a stored routine (procedure or function) or a trigger, the value of `LAST_INSERT_ID()` changes the same way as for statements executed outside the body of these kinds of objects. The effect of a stored routine or trigger upon the value of `LAST_INSERT_ID()` that is seen by following statements depends on the kind of routine:

- If a `stored procedure` executes statements that change the value of `LAST_INSERT_ID()`, the new value will be seen by statements that follow the procedure call.
- For `stored functions` and `triggers` that change the value, the value is restored when the function or trigger ends, so following statements will not see a changed value.

Examples

```
CREATE TABLE t (
    id INTEGER UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    f VARCHAR(1))
ENGINE = InnoDB;

INSERT INTO t(f) VALUES('a');

SELECT LAST_INSERT_ID();
+-----+
| LAST_INSERT_ID() |
+-----+
```

Contents

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

```

+-----+
|          1 |
+-----+

INSERT INTO t(f) VALUES('b');

INSERT INTO t(f) VALUES('c');

SELECT LAST_INSERT_ID();
+-----+
| LAST_INSERT_ID() |
+-----+
|          3 |
+-----+

INSERT INTO t(f) VALUES('d'),('e');

SELECT LAST_INSERT_ID();
+-----+
| LAST_INSERT_ID() |
+-----+
|          4 |
+-----+

SELECT * FROM t;
+-----+
| id | f    |
+-----+
| 1  | a    |
| 2  | b    |
| 3  | c    |
| 4  | d    |
| 5  | e    |
+-----+

SELECT LAST_INSERT_ID(12);
+-----+
| LAST_INSERT_ID(12) |
+-----+
|          12 |
+-----+

SELECT LAST_INSERT_ID();
+-----+
| LAST_INSERT_ID() |
+-----+
|          12 |
+-----+

INSERT INTO t(f) VALUES('f');

SELECT LAST_INSERT_ID();
+-----+
| LAST_INSERT_ID() |
+-----+
|          6 |
+-----+

SELECT * FROM t;
+-----+
| id | f    |
+-----+
| 1  | a    |
| 2  | b    |
| 3  | c    |
| 4  | d    |
| 5  | e    |
| 6  | f    |
+-----+

SELECT LAST_INSERT_ID(12);

```

```

+-----+
| LAST_INSERT_ID(12) |
+-----+
|          12 |
+-----+

INSERT INTO t(f) VALUES('g');

SELECT * FROM t;
+---+---+
| id | f   |
+---+---+
| 1  | a   |
| 2  | b   |
| 3  | c   |
| 4  | d   |
| 5  | e   |
| 6  | f   |
| 7  | g   |
+---+---+

```

See Also

- [mysql_insert_id](#)
- [AUTO_INCREMENT](#)
- [AUTO_INCREMENT handling in InnoDB](#)
- [Sequences](#) - an alternative to auto_increment available from MariaDB 10.3

PROCEDURE ANALYSE

Syntax

```
analyse([max_elements[,max_memory]])
```

Description

This procedure is defined in the sql/sql_analyse.cc file. It examines the result from a query and returns an analysis of the results that suggests optimal data types for each column. To obtain this analysis, append PROCEDURE ANALYSE to the end of a [SELECT](#) statement:

```
SELECT ... FROM ... WHERE ... PROCEDURE ANALYSE([max_elements,[max_memory]])
```

For example:

```
SELECT col1, col2 FROM table1 PROCEDURE ANALYSE(10, 2000);
```

The results show some statistics for the values returned by the query, and propose an optimal data type for the columns. This can be helpful for checking your existing tables, or after importing new data. You may need to try different settings for the arguments so that PROCEDURE ANALYSE() does not suggest the ENUM data type when it is not appropriate.

The arguments are optional and are used as follows:

- max_elements (default 256) is the maximum number of distinct values that analyse notices per column. This is used by analyse to check whether the optimal data type should be of type ENUM; if there are more than max_elements distinct values, then ENUM is not a suggested type.
- max_memory (default 8192) is the maximum amount of memory that analyse should allocate per column while trying to find all distinct values.

See Also

- [PROCEDURE](#)
- [SELECT](#)

SHOW FUNCTION CODE

Syntax

```
SHOW FUNCTION CODE func_name
```

Description

`SHOW FUNCTION CODE` shows a representation of the internal implementation of the stored function.

It is similar to `SHOW PROCEDURE CODE` but for [stored functions](#).

DO

Syntax

```
DO expr [, expr] ...
```

Description

`DO` executes the expressions but does not return any results. In most respects, `DO` is shorthand for `SELECT expr, ...`, but has the advantage that it is slightly faster when you do not care about the result.

`DO` is useful primarily with functions that have side effects, such as [RELEASE_LOCK\(\)](#).

LAST_VALUE

Syntax

```
LAST_VALUE(expr,[expr,...])
```

```
LAST_VALUE(expr) OVER (
    [ PARTITION BY partition_expression ]
    [ ORDER BY order_list ]
)
```

Description

`LAST_VALUE()` evaluates all expressions and returns the last.

This is useful together with [setting user variables to a value with @var:=expr](#), for example when you want to get data of rows updated/deleted without having to do two queries against the table.

Since [MariaDB 10.2.2](#), `LAST_VALUE` can be used as a [window function](#).

Returns `NULL` if no last value exists.

Contents

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

Examples

```

CREATE TABLE t1 (a int, b int);
INSERT INTO t1 VALUES(1,10),(2,20);
DELETE FROM t1 WHERE a=1 AND last_value(@a:=a,@b:=b,1);
SELECT @a,@b;
+-----+
| @a   | @b   |
+-----+
|    1 |    10 |
+-----+

```

As a [window function](#):

```

CREATE TABLE t1 (
  pk int primary key,
  a int,
  b int,
  c char(10),
  d decimal(10, 3),
  e real
);

INSERT INTO t1 VALUES
( 1, 0, 1, 'one', 0.1, 0.001),
( 2, 0, 2, 'two', 0.2, 0.002),
( 3, 0, 3, 'three', 0.3, 0.003),
( 4, 1, 2, 'three', 0.4, 0.004),
( 5, 1, 1, 'two', 0.5, 0.005),
( 6, 1, 1, 'one', 0.6, 0.006),
( 7, 2, NULL, 'n_one', 0.5, 0.007),
( 8, 2, 1, 'n_two', NULL, 0.008),
( 9, 2, 2, NULL, 0.7, 0.009),
(10, 2, 0, 'n_four', 0.8, 0.010),
(11, 2, 10, NULL, 0.9, NULL);

SELECT pk, FIRST_VALUE(pk) OVER (ORDER BY pk) AS first_asc,
       LAST_VALUE(pk) OVER (ORDER BY pk) AS last_asc,
       FIRST_VALUE(pk) OVER (ORDER BY pk DESC) AS first_desc,
       LAST_VALUE(pk) OVER (ORDER BY pk DESC) AS last_desc
FROM t1
ORDER BY pk DESC;

```

```

+-----+-----+-----+-----+
| pk | first_asc | last_asc | first_desc | last_desc |
+-----+-----+-----+-----+
| 11 |      1 |     11 |      11 |      11 |
| 10 |      1 |     10 |      11 |      10 |
|  9 |      1 |      9 |      11 |      9 |
|  8 |      1 |      8 |      11 |      8 |
|  7 |      1 |      7 |      11 |      7 |
|  6 |      1 |      6 |      11 |      6 |
|  5 |      1 |      5 |      11 |      5 |
|  4 |      1 |      4 |      11 |      4 |
|  3 |      1 |      3 |      11 |      3 |
|  2 |      1 |      2 |      11 |      2 |
|  1 |      1 |      1 |      11 |      1 |
+-----+-----+-----+-----+

```

```

CREATE OR REPLACE TABLE t1 (i int);
INSERT INTO t1 VALUES (1),(2),(3),(4),(5),(6),(7),(8),(9),(10);

SELECT i,
       FIRST_VALUE(i) OVER (ORDER BY i ROWS BETWEEN CURRENT ROW and 1 FOLLOWING) AS f_1f,
       LAST_VALUE(i) OVER (ORDER BY i ROWS BETWEEN CURRENT ROW and 1 FOLLOWING) AS l_1f,
       FIRST_VALUE(i) OVER (ORDER BY i ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS f_1p1f,
       LAST_VALUE(i) OVER (ORDER BY i ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS f_1p1f,
       FIRST_VALUE(i) OVER (ORDER BY i ROWS BETWEEN 2 PRECEDING AND 1 PRECEDING) AS f_2p1p,
       LAST_VALUE(i) OVER (ORDER BY i ROWS BETWEEN 2 PRECEDING AND 1 PRECEDING) AS f_2p1p,
       FIRST_VALUE(i) OVER (ORDER BY i ROWS BETWEEN 1 FOLLOWING AND 2 FOLLOWING) AS f_1f2f,
       LAST_VALUE(i) OVER (ORDER BY i ROWS BETWEEN 1 FOLLOWING AND 2 FOLLOWING) AS f_1f2f
  FROM t1;

```

i	f_1f	l_1f	f_1p1f	f_1p1f	f_2p1p	f_2p1p	f_1f2f	f_1f2f
1	1	2	1	2	NULL	NULL	2	3
2	2	3	1	3	1	1	3	4
3	3	4	2	4	1	2	4	5
4	4	5	3	5	2	3	5	6
5	5	6	4	6	3	4	6	7
6	6	7	5	7	4	5	7	8
7	7	8	6	8	5	6	8	9
8	8	9	7	9	6	7	9	10
9	9	10	8	10	7	8	10	10
10	10	10	9	10	8	9	NULL	NULL

See Also

- [Setting a variable to a value](#)

ROW_COUNT

Syntax

```
ROW_COUNT()
```

Description

ROW_COUNT() returns the number of rows updated, inserted or deleted by the preceding statement. This is the same as the row count that the mysql client displays and the value from the [mysql_affected_rows\(\)](#) C API function.

Generally:

- For statements which return a result set (such as [SELECT](#), [SHOW](#), [DESC](#) or [HELP](#)), returns -1, even when the result set is empty. This is also true for administrative statements, such as [OPTIMIZE](#).
- For DML statements other than [SELECT](#) and for [ALTER TABLE](#), returns the number of affected rows.
- For DDL statements (including [TRUNCATE](#)) and for other statements which don't return any result set (such as [USE](#), [DO](#), [SIGNAL](#) or [DEALLOCATE PREPARE](#)), returns 0.

For [UPDATE](#), affected rows is by default the number of rows that were actually changed. If the [CLIENT_FOUND_ROWS](#) flag to [mysql_real_connect\(\)](#) is specified when connecting to mysqld, affected rows is instead the number of rows matched by the WHERE clause.

For [REPLACE](#), deleted rows are also counted. So, if REPLACE deletes a row and adds a new row, ROW_COUNT() returns 2.

For [INSERT ... ON DUPLICATE KEY](#), updated rows are counted twice. So, if INSERT adds a new rows and modifies another row, ROW_COUNT() returns 3.

ROW_COUNT() does not take into account rows that are not directly deleted/updated by the last statement. This means that rows deleted by foreign keys or triggers are not counted.

Warning: You can use ROW_COUNT() with prepared statements, but you need to call it after EXECUTE, not after [DEALLOCATE PREPARE](#), because the row count for allocate prepare is always 0.

Warning: When used after a `CALL` statement, this function returns the number of rows affected by the last statement in the procedure, not by the whole procedure.

Warning: After `INSERT DELAYED`, `ROW_COUNT()` returns the number of the rows you tried to insert, not the number of the successful writes.

This information can also be found in the [diagnostics area](#).

Statements using the `ROW_COUNT()` function are not [safe for replication](#).

Examples

```
CREATE TABLE t (A INT);

INSERT INTO t VALUES(1),(2),(3);

SELECT ROW_COUNT();
+-----+
| ROW_COUNT() |
+-----+
|      3 |
+-----+

DELETE FROM t WHERE A IN(1,2);

SELECT ROW_COUNT();
+-----+
| ROW_COUNT() |
+-----+
|      2 |
+-----+
```

Example with prepared statements:

```
SET @q = 'INSERT INTO t VALUES(1),(2),(3);';

PREPARE stmt FROM @q;

EXECUTE stmt;
Query OK, 3 rows affected (0.39 sec)
Records: 3  Duplicates: 0  Warnings: 0

SELECT ROW_COUNT();
+-----+
| ROW_COUNT() |
+-----+
|      3 |
+-----+
```

See Also

- [FOUND_ROWS\(\)](#)

DUAL

Description

You are allowed to specify `DUAL` as a dummy table name in situations where no tables are referenced, such as the following `SELECT` statement:

```
SELECT 1 + 1 FROM DUAL;
+-----+
| 1 + 1 |
+-----+
|      2 |
+-----+
```

`DUAL` is purely for the convenience of people who require that all `SELECT` statements should have `FROM` and possibly other clauses. MariaDB ignores the clauses. MariaDB does not require `FROM DUAL` if no tables are referenced.

`FROM DUAL` could be used when you only `SELECT` computed values, but require a `WHERE` clause, perhaps to test that a script correctly handles empty resultsets:

```
SELECT 1 FROM DUAL WHERE FALSE;
Empty set (0.00 sec)
```

See Also

- [SELECT](#)

FOR UPDATE

InnoDB supports row-level locking. Selected rows can be locked using `LOCK IN SHARE MODE` or `FOR UPDATE`. In both cases, a lock is acquired on the rows read by the query, and it will be released when the current transaction is committed.

The `FOR UPDATE` clause of `SELECT` applies only when `autocommit` is set to 0 or the `SELECT` is enclosed in a transaction. A lock is acquired on the rows, and other transactions are prevented from writing the rows, acquire locks, and from reading them (unless their isolation level is `READ UNCOMMITTED`).

If `autocommit` is set to 1, the `LOCK IN SHARE MODE` and `FOR UPDATE` clauses have no effect.

If the isolation level is set to `SERIALIZABLE`, all plain `SELECT` statements are converted to `SELECT ... LOCK IN SHARE MODE`.

Example

```
SELECT * FROM trans WHERE period=2001 FOR UPDATE;
```

See Also

- [SELECT](#)
- [LOCK IN SHARE MODE](#)
- [InnoDB Lock Modes](#)

USER

Syntax

```
USER()
```

Description

Returns the current MariaDB user name and host name, given when authenticating to MariaDB, as a string in the utf8 [character set](#).

Note that the value of `USER()` may differ from the value of `CURRENT_USER()`, which is the user used to authenticate the current client. `CURRENT_ROLE()` returns the current active role.

`SYSTEM_USER()` and `SESSION_USER` are synonyms for `USER()`.

Statements using the `USER()` function or one of its synonyms are not [safe for statement level replication](#).

Examples

```
shell> mysql --user="anonymous"

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

To select only the IP address, use [SUBSTRING_INDEX\(\)](#),

```
SELECT SUBSTRING_INDEX(USER(), '@', -1);
+-----+
| SUBSTRING_INDEX(USER(), '@', -1) |
+-----+
| 192.168.0.101 |
+-----+
```

See Also

- [CURRENT_USER\(\)](#)

VERSION

Syntax

```
VERSION()
```

Description

Returns a string that indicates the MariaDB server version. The string uses the utf8 [character set](#).

Examples

```
SELECT VERSION();
+-----+
| VERSION() |
+-----+
| 10.4.7-MariaDB |
+-----+
```

The `VERSION()` string may have one or more of the following suffixes:

Suffix	Description
-embedded	The server is an embedded server (<code>libmysqld</code>).
-log	General logging, slow logging or binary (replication) logging is enabled.
-debug	The server is compiled for debugging.
-valgrind	The server is compiled to be instrumented with valgrind.

Changing the Version String

Some old legacy code may break because they are parsing the `VERSION` string and expecting a MySQL string or a simple version string like Joomla til API17, see [MDEV-7780](#).

From [MariaDB 10.2](#), one can fool these applications by setting the version string from the command line or the `my.cnf` files with `--version=....`

Not Equal Operator: !=

Syntax

```
<>, !=
```

Description

Not equal operator. Evaluates both SQL expressions and returns 1 if they are not equal and 0 if they are equal, or `NULL` if either expression is `NULL`. If the expressions return different data types, (for instance, a number and a string), performs type conversion.

When used in row comparisons these two queries return the same results:

```
SELECT (t1.a, t1.b) != (t2.x, t2.y)
FROM t1 INNER JOIN t2;

SELECT (t1.a != t2.x) OR (t1.b != t2.y)
FROM t1 INNER JOIN t2;
```

Contents

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

Examples

```
SELECT '.01' <> '0.01';
+-----+
| '.01' <> '0.01' |
+-----+
|          1 |
+-----+

SELECT .01 <> '0.01';
+-----+
| .01 <> '0.01' |
+-----+
|          0 |
+-----+

SELECT 'zapp' <> 'zapp';
+-----+
| 'zapp' <> 'zapp' |
+-----+
|          1 |
+-----+
```

<=>

Syntax

```
<=>
```

Description

NULL-safe equal operator. It performs an equality comparison like the [= operator](#), but returns 1 rather than `NULL` if both operands are `NULL`, and 0 rather than `NULL` if one operand is `NULL`.

`a <=> b` is equivalent to `a = b OR (a IS NULL AND b IS NULL)`.

When used in row comparisons these two queries return the same results:

```

SELECT (t1.a, t1.b) <= (t2.x, t2.y)
FROM t1 INNER JOIN t2;

SELECT (t1.a <= t2.x) AND (t1.b <= t2.y)
FROM t1 INNER JOIN t2;

```

See also [NULL Values in MariaDB](#).

Examples

```

SELECT 1 <= 1, NULL <= NULL, 1 <= NULL;
+-----+
| 1 <= 1 | NULL <= NULL | 1 <= NULL |
+-----+
|     1   |          1   |      0    |
+-----+

SELECT 1 = 1, NULL = NULL, 1 = NULL;
+-----+
| 1 = 1 | NULL = NULL | 1 = NULL |
+-----+
|     1   |      NULL  |      NULL |
+-----+

```

=

Syntax

```
left_expr = right_expr
```

Description

Equal operator. Evaluates both SQL expressions and returns 1 if they are equal, 0 if they are not equal, or `NULL` if either expression is `NULL`. If the expressions return different data types (for example, a number and a string), a type conversion is performed.

When used in row comparisons these two queries are synonymous and return the same results:

Contents

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

```

SELECT (t1.a, t1.b) = (t2.x, t2.y) FROM t1 INNER JOIN t2;

SELECT (t1.a = t2.x) AND (t1.b = t2.y) FROM t1 INNER JOIN t2;

```

To perform a `NULL`-safe comparison, use the `<=` operator.

`=` can also be used as an [assignment operator](#).

Examples

```

SELECT 1 = 0;
+-----+
| 1 = 0 |
+-----+
|      0 |
+-----+

SELECT '0' = 0;
+-----+
| '0' = 0 |
+-----+
|      1 |
+-----+

SELECT '0.0' = 0;
+-----+
| '0.0' = 0 |
+-----+
|      1 |
+-----+

SELECT '0.01' = 0;
+-----+
| '0.01' = 0 |
+-----+
|      0 |
+-----+

SELECT '.01' = 0.01;
+-----+
| '.01' = 0.01 |
+-----+
|      1 |
+-----+

SELECT (5 * 2) = CONCAT('1', '0');
+-----+
| (5 * 2) = CONCAT('1', '0') |
+-----+
|      1 |
+-----+

SELECT 1 = NULL;
+-----+
| 1 = NULL |
+-----+
|      NULL |
+-----+

SELECT NULL = NULL;
+-----+
| NULL = NULL |
+-----+
|      NULL |
+-----+

```

HANDLER Commands

Syntax

```

HANDLER tbl_name OPEN [ [AS] alias]
HANDLER tbl_name READ index_name { = | >= | <= | < } (value1,value2,...)
    [ WHERE where_condition ] [LIMIT ... ]
HANDLER tbl_name READ index_name { FIRST | NEXT | PREV | LAST }
    [ WHERE where_condition ] [LIMIT ... ]
HANDLER tbl_name READ { FIRST | NEXT }
    [ WHERE where_condition ] [LIMIT ... ]
HANDLER tbl_name CLOSE

```

Description

The `HANDLER` statement provides direct access to table storage engine interfaces for key lookups and key or table scans. It is available for at least [Aria](#), [Memory](#), [MyISAM](#) and [InnoDB](#) tables (and should work with most 'normal' storage engines, but not with system tables, [MERGE](#) or [views](#)).

`HANDLER ... OPEN` opens a table, allowing it to be accessible to subsequent `HANDLER ... READ` statements. The table can either be opened using an alias (which must then be used by `HANDLER ... READ`, or a table name.

The table object is only closed when `HANDLER ... CLOSE` is called by the session, and is not shared by other sessions.

[Prepared statements](#) work with `HANDLER READ`, which gives a much higher performance (50% speedup) as there is no parsing and all data is transformed in binary (without conversions to text, as with the normal protocol).

The `HANDLER` command does not work with [partitioned tables](#).

Key Lookup

A key lookup is started with:

```
HANDLER tbl_name READ index_name { = | >= | <= | < } (value,value) [LIMIT...]
```

The values stands for the value of each of the key columns. For most key types (except for HASH keys in MEMORY storage engine) you can use a prefix subset of it's columns.

If you are using `LIMIT`, then in case of `>=` or `>` then there is an implicit `NEXT` implied, while if you are using `<=` or `<` then there is an implicit `PREV` implied.

After the initial read, you can use

```
HANDLER tbl_name READ index_name NEXT [ LIMIT ... ]
or
HANDLER tbl_name READ index_name PREV [ LIMIT ... ]
```

to scan the rows in key order.

Note that the row order is not defined for keys with duplicated values and will vary from engine to engine.

Key Scans

You can scan a table in key order by doing:

```
HANDLER tbl_name READ index_name FIRST [ LIMIT ... ]
HANDLER tbl_name READ index_name NEXT [ LIMIT ... ]
```

or, if the handler supports backwards key scans (most do):

```
HANDLER tbl_name READ index_name LAST [ LIMIT ... ]
HANDLER tbl_name READ index_name PREV [ LIMIT ... ]
```

Contents

1. [Syntax](#)
2. [Description](#)
3. [Key Lookup](#)
4. [Key Scans](#)
5. [Table Scans](#)
6. [Limitations](#)
 1. [Finding 'Old Rows'](#)
 2. [Invisible Columns](#)
 3. [System-Versioned Tables](#)
 4. [Other Limitations](#)
7. [Error Codes](#)
8. [See Also](#)

Table Scans

You can scan a table in row order by doing:

```
HANDLER tbl_name READ FIRST [ LIMIT ... ]
HANDLER tbl_name READ NEXT [ LIMIT ... ]
```

Limitations

As this is a direct interface to the storage engine, some limitations may apply for what you can do and what happens if the table changes. Here follows some of the common limitations:

Finding 'Old' Rows'

HANDLER READ is not transaction safe, consistent or atomic. It's ok for the storage engine to returns rows that existed when you started the scan but that were later deleted. This can happen as the storage engine may cache rows as part of the scan from a previous read.

You may also find rows committed since the scan originally started.

Invisible Columns

HANDLER ... READ also reads the data of [invisible-columns](#).

System-Versioned Tables

HANDLER ... READ reads everything from [system-versioned tables](#), and so includes `row_start` and `row_end` fields, as well as all rows that have since been deleted or changed, including when history partitions are used.

Other Limitations

- If you do an [ALTER TABLE](#), all your HANDLER's for that table are automatically closed.
- If you do an ALTER TABLE for a table that is used by some other connection with HANDLER, the ALTER TABLE will wait for the HANDLER to be closed.
- For HASH keys, you must use all key parts when searching for a row.
- For HASH keys, you can't do a key scan of all values. You can only find all rows with the same key value.
- While each HANDLER READ command is atomic, if you do a scan in many steps, then some engines may give you error 1020 if the table changed between the commands. Please refer to the [specific engine handler page](#) if this happens.

Error Codes

- Error 1031 (ER_ILLEGAL_HA) Table storage engine for 't1' doesn't have this option
 - If you get this for HANDLER OPEN it means the storage engine doesn't support HANDLER calls.
 - If you get this for HANDLER READ it means you are trying to use an incomplete HASH key.
- Error 1020 (ER_CHECKREAD) Record has changed since last read in table '...'
 - This means that the table changed between two reads and the handler can't handle this case for the given scan.

See Also

- [What is MariaDB 5.3](#)

LOCK IN SHARE MODE

InnoDB supports row-level locking. Selected rows can be locked using `LOCK IN SHARE MODE` or `FOR UPDATE`. In both cases, a lock is acquired on the rows read by the query, and it will be released when the current transaction is committed.

When `LOCK IN SHARE MODE` is specified in a [SELECT](#) statement, MariaDB will wait until all transactions that have modified the rows are committed. Then, a write lock is acquired. All transactions can read the rows, but if they want to modify them, they have to wait until your transaction is committed.

If `autocommit` is set to 1, the `LOCK IN SHARE MODE` and `FOR UPDATE` clauses have no effect.

See Also

- [SELECT](#)
- [FOR UPDATE](#)
- [InnoDB Lock Modes](#)

PROCEDURE

The `PROCEDURE` clause of [SELECT](#) passes the whole result set to a Procedure which will process it. These Procedures are not [Stored Procedures](#), and can only be written in the C language, so it is necessary to recompile the server.

Currently, the only available procedure is [ANALYSE](#), which examines the resultset and suggests the optimal datatypes for each column. It is defined in the `sql/sql_analyse.cc` file, and can be used as an example to create more Procedures.

This clause cannot be used in a [view's](#) definition.

See Also

- [SELECT](#)
- [Stored Procedures](#)

>

Syntax

>

Description

Greater than operator. Evaluates both SQL expressions and returns 1 if the left value is greater than the right value and 0 if it is not, or `NULL` if either expression is `NULL`. If the expressions return different data types, (for instance, a number and a string), performs type conversion.

Contents

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

When used in row comparisons these two queries return the same results:

```
SELECT (t1.a, t1.b) > (t2.x, t2.y)
FROM t1 INNER JOIN t2;

SELECT (t1.a > t2.x) OR ((t1.a = t2.x) AND (t1.b > t2.y))
FROM t1 INNER JOIN t2;
```

Examples

```
SELECT 2 > 2;
+-----+
| 2 > 2 |
+-----+
|      0 |
+-----+

SELECT 'b' > 'a';
+-----+
| 'b' > 'a' |
+-----+
|      1 |
+-----+
```

>=

Syntax

Description

Greater than or equal operator. Evaluates both SQL expressions and returns 1 if the left value is greater than or equal to the right value and 0 if it is not, or `NULL` if either expression is `NULL`. If the expressions return different data types, (for instance, a number and a string), performs type conversion.

When used in row comparisons these two queries return the same results:

```
SELECT (t1.a, t1.b) >= (t2.x, t2.y)
FROM t1 INNER JOIN t2;

SELECT (t1.a > t2.x) OR ((t1.a = t2.x) AND (t1.b >= t2.y))
FROM t1 INNER JOIN t2;
```

Contents

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

Examples

```
SELECT 2 >= 2;
+-----+
| 2 >= 2 |
+-----+
|      1 |
+-----+

SELECT 'A' >= 'a';
+-----+
| 'A' >= 'a' |
+-----+
|          1 |
+-----+
```

COALESCE

Syntax

```
COALESCE(value,...)
```

Description

Returns the first non-`NULL` value in the list, or `NULL` if there are no non-`NULL` values. At least one parameter must be passed.

The function is useful when substituting a default value for null values when displaying data.

See also [NULL Values in MariaDB](#).

Examples

```
SELECT COALESCE(NULL,1);
+-----+
| COALESCE(NULL,1) |
+-----+
|          1 |
+-----+
```

```
SELECT COALESCE(NULL,NULL,NULL);
+-----+
| COALESCE(NULL,NULL,NULL) |
+-----+
|          NULL          |
+-----+
```

When two arguments are given, COALESCE() is the same as [IFNULL\(\)](#):

```
SET @a=NULL, @b=1;

SELECT COALESCE(@a, @b), IFNULL(@a, @b);
+-----+-----+
| COALESCE(@a, @b) | IFNULL(@a, @b) |
+-----+-----+
|          1          |          1          |
+-----+
```

Hex type confusion:

```
CREATE TABLE t1 (a INT, b VARCHAR(10));
INSERT INTO t1 VALUES (0x31, 0x61),(COALESCE(0x31), COALESCE(0x61));

SELECT * FROM t1;
+-----+
| a      | b      |
+-----+
| 49    | a      |
| 1     | a      |
+-----+
```

The reason for the differing results above is that when 0x31 is inserted directly to the column, it's treated as a number (see [Hexadecimal Literals](#)), while when 0x31 is passed to COALESCE(), it's treated as a string, because:

- HEX values have a string data type by default.
- COALESCE() has the same data type as the argument.

Substituting zero for NULL (in this case when the aggregate function returns NULL after finding no rows):

```
SELECT SUM(score) FROM student;
+-----+
| SUM(score) |
+-----+
|      NULL   |
+-----+

SELECT COALESCE(SUM(score),0) FROM student;
+-----+
| COALESCE(SUM(score),0) |
+-----+
|          0           |
+-----+
```

See also

- [NULL values](#)
- [IS NULL operator](#)
- [IS NOT NULL operator](#)
- [IFNULL function](#)
- [NULLIF function](#)
- [CONNECT data types](#)

IN

Syntax

```
expr IN (value,...)
```

Description

Returns 1 if *expr* is equal to any of the values in the IN list, else returns 0. If all values are constants, they are evaluated according to the type of *expr* and sorted. The search for the item then is done using a binary search. This means IN is very quick if the IN value list consists entirely of constants. Otherwise, type conversion takes place according to the rules described at [Type Conversion](#), but applied to all the arguments.

If *expr* is NULL, IN always returns NULL. If at least one of the values in the list is NULL, and one of the comparisons is true, the result is 1. If at least one of the values in the list is NULL and none of the comparisons is true, the result is NULL.

Examples

```
SELECT 2 IN (0,3,5,7);
+-----+
| 2 IN (0,3,5,7) |
+-----+
|          0 |
+-----+
```

```
SELECT 'wefwf' IN ('wee','wefwf','weg');
+-----+
| 'wefwf' IN ('wee','wefwf','weg') |
+-----+
|                      1 |
+-----+
```

Type conversion:

```
SELECT 1 IN ('1', '2', '3');
+-----+
| 1 IN ('1', '2', '3') |
+-----+
|          1 |
+-----+
```

```
SELECT NULL IN (1, 2, 3);
+-----+
| NULL IN (1, 2, 3) |
+-----+
|          NULL |
+-----+  
  
MariaDB [(none)]> SELECT 1 IN (1, 2, NULL);
+-----+
| 1 IN (1, 2, NULL) |
+-----+
|          1 |
+-----+  
  
MariaDB [(none)]> SELECT 5 IN (1, 2, NULL);
+-----+
| 5 IN (1, 2, NULL) |
+-----+
|          NULL |
+-----+
```

CONTAINS

Syntax

```
Contains(g1,g2)
```

Description

Returns 1 or 0 to indicate whether a geometry g_1 completely contains geometry g_2 . `CONTAINS()` is based on the original MySQL implementation and uses object bounding rectangles, while `ST_CONTAINS()` uses object shapes.

This tests the opposite relationship to [Within\(\)](#).

DISJOINT

Syntax

```
Disjoint(g1,g2)
```

Description

Returns 1 or 0 to indicate whether g_1 is spatially disjoint from (does not intersect) g_2 .

`DISJOINT()` tests the opposite relationship to [INTERSECTS\(\)](#).

`DISJOINT()` is based on the original MySQL implementation and uses object bounding rectangles, while `ST_DISJOINT()` uses object shapes.

EQUALS

Syntax

```
Equals(g1,g2)
```

From [MariaDB 10.2.3](#):

```
MBREQUALS(g1,g2)
```

Description

Returns 1 or 0 to indicate whether g_1 is spatially equal to g_2 .

`EQUALS()` is based on the original MySQL implementation and uses object bounding rectangles, while `ST_EQUALS()` uses object shapes.

From [MariaDB 10.2.3](#), `MBREQUALS` is a synonym for `Equals`.

INTERSECTS

Syntax

```
INTERSECTS(g1,g2)
```

Description

Returns 1 or 0 to indicate whether geometry g_1 spatially intersects geometry g_2 .

`INTERSECTS()` is based on the original MySQL implementation and uses object bounding rectangles, while `ST_INTERSECTS()` uses object shapes.

`INTERSECTS()` tests the opposite relationship to [DISJOINT\(\)](#).

OVERLAPS

Syntax

```
OVERLAPS(g1,g2)
```

Description

Returns 1 or 0 to indicate whether g1 spatially overlaps g2 . The term spatially overlaps is used if two geometries intersect and their intersection results in a geometry of the same dimension but not equal to either of the given geometries.

OVERLAPS() is based on the original MySQL implementation and uses object bounding rectangles, while [ST_OVERLAPS\(\)](#) uses object shapes.

IS

Syntax

```
IS boolean_value
```

Description

Tests a value against a boolean value, where boolean_value can be TRUE, FALSE, or UNKNOWN.

There is an important difference between using IS TRUE or comparing a value with TRUE using = . When using = , only 1 equals to TRUE. But when using IS TRUE, all values which are logically true (like a number > 1) return TRUE.

Examples

```
SELECT 1 IS TRUE, 0 IS FALSE, NULL IS UNKNOWN;
+-----+-----+-----+
| 1 IS TRUE | 0 IS FALSE | NULL IS UNKNOWN |
+-----+-----+-----+
|      1 |         1 |           1 |
+-----+-----+-----+
```

Difference between = and IS TRUE :

```
SELECT 2 = TRUE, 2 IS TRUE;
+-----+-----+
| 2 = TRUE | 2 IS TRUE |
+-----+-----+
|      0 |         1 |
+-----+-----+
```

See Also

- [Boolean Literals](#)
- [BOOLEAN Data Type](#)

IS NOT

Syntax

```
IS NOT boolean_value
```

Description

Tests a value against a boolean value, where boolean_value can be TRUE, FALSE, or UNKNOWN.

Examples

```
SELECT 1 IS NOT UNKNOWN, 0 IS NOT UNKNOWN, NULL IS NOT UNKNOWN;
+-----+-----+-----+
| 1 IS NOT UNKNOWN | 0 IS NOT UNKNOWN | NULL IS NOT UNKNOWN |
+-----+-----+-----+
|           1 |           1 |          0 |
+-----+-----+-----+
```

```
SELECT NULL IS NOT TRUE, NULL IS NOT FALSE;
+-----+-----+
| NULL IS NOT TRUE | NULL IS NOT FALSE |
+-----+-----+
|           1 |           1 |
+-----+-----+
```

IS NULL

Syntax

```
IS NULL
```

Description

Tests whether a value is NULL. See also [NULL Values in MariaDB](#).

Examples

```
SELECT 1 IS NULL, 0 IS NULL, NULL IS NULL;
+-----+-----+-----+
| 1 IS NULL | 0 IS NULL | NULL IS NULL |
+-----+-----+-----+
|          0 |          0 |          1 |
+-----+-----+-----+
```

Compatibility

Some ODBC applications use the syntax `auto_increment_field IS NOT NULL` to find the latest row that was inserted with an autogenerated key value. If your applications need this, you can set the [sql_auto_is_null](#) variable to 1.

```
SET @@sql_auto_is_null=1;
CREATE TABLE t1 (auto_increment_column INT NOT NULL AUTO_INCREMENT PRIMARY KEY);
INSERT INTO t1 VALUES (NULL);
SELECT * FROM t1 WHERE auto_increment_column IS NULL;

+-----+
| auto_increment_column |
+-----+
|           1 |
+-----+
```

See also

- [NULL values](#)
- [IS NOT NULL operator](#)
- [COALESCE function](#)
- [IFNULL function](#)
- [NULLIF function](#)
- [CONNECT data types](#)

LEAST

Syntax

```
LEAST(value1,value2,...)
```

Description

With two or more arguments, returns the smallest (minimum-valued) argument. The arguments are compared using the following rules:

- If the return value is used in an INTEGER context or all arguments are integer-valued, they are compared as integers.
- If the return value is used in a REAL context or all arguments are real-valued, they are compared as reals.
- If any argument is a case-sensitive string, the arguments are compared as case-sensitive strings.
- In all other cases, the arguments are compared as case-insensitive strings.

`LEAST()` returns NULL if any argument is NULL.

Examples

```
SELECT LEAST(2,0);
+-----+
| LEAST(2,0) |
+-----+
|      0      |
+-----+
```

```
SELECT LEAST(34.0,3.0,5.0,767.0);
+-----+
| LEAST(34.0,3.0,5.0,767.0) |
+-----+
|          3.0          |
+-----+
```

```
SELECT LEAST('B','A','C');
+-----+
| LEAST('B','A','C') |
+-----+
| A                 |
+-----+
```

NOT BETWEEN

Syntax

```
expr NOT BETWEEN min AND max
```

Description

This is the same as NOT (`expr BETWEEN min AND max`).

Note that the meaning of the alternative form NOT expr BETWEEN min AND max is affected by the HIGH_NOT_PRECEDENCE SQL_MODE flag.

Examples

```
SELECT 1 NOT BETWEEN 2 AND 3;
+-----+
| 1 NOT BETWEEN 2 AND 3 |
+-----+
|           1 |
+-----+
```

```
SELECT 'b' NOT BETWEEN 'a' AND 'c';
+-----+
| 'b' NOT BETWEEN 'a' AND 'c' |
+-----+
|          0 |
+-----+
```

NULL:

```
SELECT 1 NOT BETWEEN 1 AND NULL;
+-----+
| 1 NOT BETWEEN 1 AND NULL |
+-----+
|           NULL |
+-----+
```

NOT IN

Syntax

```
expr NOT IN (value,...)
```

Description

This is the same as NOT (expr IN (value,...)).

Examples

```
SELECT 2 NOT IN (0,3,5,7);
+-----+
| 2 NOT IN (0,3,5,7) |
+-----+
|           1 |
+-----+
```

```
SELECT 'wefwf' NOT IN ('wee','wefwf','weg');
+-----+
| 'wefwf' NOT IN ('wee','wefwf','weg') |
+-----+
|          0 |
+-----+
```

```
SELECT 1 NOT IN ('1', '2', '3');
+-----+
| 1 NOT IN ('1', '2', '3') |
+-----+
|          0 |
+-----+
```

NULL:

```
SELECT NULL NOT IN (1, 2, 3);
+-----+
| NULL NOT IN (1, 2, 3) |
+-----+
|          NULL |
+-----+
```

```
SELECT 1 NOT IN (1, 2, NULL);
+-----+
| 1 NOT IN (1, 2, NULL) |
+-----+
|          0 |
+-----+
```

```
SELECT 5 NOT IN (1, 2, NULL);
+-----+
| 5 NOT IN (1, 2, NULL) |
+-----+
|          NULL |
+-----+
```

ST_DIFFERENCE

Syntax

```
ST_DIFFERENCE(g1,g2)
```

Description

Returns a geometry representing the point set difference of the given geometry values.

Example

```
SET @g1 = POINT(10,10), @g2 = POINT(20,20);

SELECT ST_AsText(ST_Difference(@g1, @g2));
+-----+
| ST_AsText(ST_Difference(@g1, @g2)) |
+-----+
| POINT(10 10)                         |
+-----+
```

ST_DISTANCE

Syntax

```
ST_DISTANCE(g1,g2)
```

Description

Returns the distance between two geometries, or null if not given valid inputs.

Example

```
SELECT ST_Distance(POINT(1,2),POINT(2,2));
+-----+
| ST_Distance(POINT(1,2),POINT(2,2)) |
+-----+
|           1 |
+-----+
```

ST_LENGTH

Syntax

```
ST_LENGTH(ls)
```

Description

Returns as a double-precision number the length of the [LineString](#) value `ls` in its associated spatial reference.

Examples

```
SET @ls = 'LineString(1 1,2 2,3 3)';
SELECT ST_LENGTH(ST_GeomFromText(@ls));
+-----+
| ST_LENGTH(ST_GeomFromText(@ls)) |
+-----+
|           2.82842712474619 |
+-----+
```

Parentheses

Parentheses are sometimes called precedence operators - this means that they can be used to change the other [operator's precedence](#) in an expression. The expressions that are written between parentheses are computed before the expressions that are written outside. Parentheses must always contain an expression (that is, they cannot be empty), and can be nested.

For example, the following expressions could return different results:

- NOT a OR b
- NOT (a OR b)

In the first case, `NOT` applies to `a`, so if `a` is `FALSE` or `b` is `TRUE`, the expression returns `TRUE`. In the second case, `NOT` applies to the result of `a OR b`, so if at least one of `a` or `b` is `TRUE`, the expression is `TRUE`.

When the precedence of operators is not intuitive, you can use parentheses to make it immediately clear for whoever reads the statement.

The precedence of the `NOT` operator can also be affected by the `HIGH_NOT_PRECEDENCE SQL_MODE` flag.

Other uses

Parentheses must always be used to enclose [subqueries](#).

Parentheses can also be used in a `JOIN` statement between multiple tables to determine which tables must be joined first.

Also, parentheses are used to enclose the list of parameters to be passed to built-in functions, user-defined functions and stored routines. However, when no parameter is passed to a stored procedure, parentheses are optional. For builtin functions and user-defined functions, spaces are not

allowed between the function name and the open parenthesis, unless the `IGNORE_SPACE SQL_MODE` is set. For stored routines (and for functions if `IGNORE_SPACE` is set) spaces are allowed before the open parenthesis, including tab characters and new line characters.

Syntax errors

If there are more open parentheses than closed parentheses, the error usually looks like this:

```
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near '' a t line 1
```

Note the empty string.

If there are more closed parentheses than open parentheses, the error usually looks like this:

```
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near ')' at line 1
```

Note the quoted closed parenthesis.

ST_OVERLAPS

Syntax

```
ST_OVERLAPS(g1,g2)
```

Description

Returns `1` or `0` to indicate whether geometry `g1` spatially overlaps geometry `g2`.

The term spatially overlaps is used if two geometries intersect and their intersection results in a geometry of the same dimension but not equal to either of the given geometries.

`ST_OVERLAPS()` uses object shapes, while [OVERLAPS\(\)](#), based on the original MySQL implementation, uses object bounding rectangles.

TOUCHES

Syntax

```
Touches(g1,g2)
```

Description

Returns `1` or `0` to indicate whether `g1` spatially touches `g2`. Two geometries spatially touch if the interiors of the geometries do not intersect, but the boundary of one of the geometries intersects either the boundary or the interior of the other.

`TOUCHES()` is based on the original MySQL implementation and uses object bounding rectangles, while [ST_TOUCHES\(\)](#) uses object shapes.

CURRENT_DATE

Syntax

```
CURRENT_DATE, CURRENT_DATE()
```

Description

`CURRENT_DATE` and `CURRENT_DATE()` are synonyms for [CURDATE\(\)](#).

CURRENT_TIME

Syntax

```
CURRENT_TIME  
CURRENT_TIME([precision])
```

Description

`CURRENT_TIME` and `CURRENT_TIME()` are synonyms for [CURTIME\(\)](#).

See Also

- [Microseconds in MariaDB](#)

CURRENT_TIMESTAMP

Syntax

```
CURRENT_TIMESTAMP  
CURRENT_TIMESTAMP([precision])
```

Description

`CURRENT_TIMESTAMP` and `CURRENT_TIMESTAMP()` are synonyms for [NOW\(\)](#).

See Also

- [Microseconds in MariaDB](#)
- The [TIMESTAMP](#) data type

CURTIME

Syntax

```
CURTIME([precision])
```

Description

Returns the current time as a value in 'HH:MM:SS' or HHMMSS.ududu format, depending on whether the function is used in a string or numeric context. The value is expressed in the current [time zone](#).

The optional *precision* determines the microsecond precision. See [Microseconds in MariaDB](#).

Examples

```
SELECT CURTIME();
+-----+
| CURTIME() |
+-----+
| 12:45:39 |
+-----+

SELECT CURTIME() + 0;
+-----+
| CURTIME() + 0 |
+-----+
| 124545.000000 |
+-----+
```

With precision:

```
SELECT CURTIME(2);
+-----+
| CURTIME(2) |
+-----+
| 09:49:08.09 |
+-----+
```

See Also

- [Microseconds in MariaDB](#)

DATE FUNCTION

Syntax

```
DATE(expr)
```

Description

Extracts the date part of the date or datetime expression expr.

Examples

```
SELECT DATE('2013-07-18 12:21:32');
+-----+
| DATE('2013-07-18 12:21:32') |
+-----+
| 2013-07-18 |
+-----+
```

Error Handling

Until [MariaDB 5.5.32](#), some versions of MariaDB returned `0000-00-00` when passed an invalid date. From 5.5.32, NULL is returned.

DAY

Syntax

```
DAY(date)
```

Description

DAY() is a synonym for DAYOFMONTH().

DAYOFYEAR

Syntax

```
DAYOFYEAR(date)
```

Description

Returns the day of the year for date, in the range 1 to 366.

Examples

```
SELECT DAYOFYEAR('2018-02-16');
+-----+
| DAYOFYEAR('2018-02-16') |
+-----+
| 47 |
+-----+
```

FROM_DAYS

Syntax

```
FROM_DAYS(N)
```

Description

Given a day number N, returns a DATE value. The day count is based on the number of days from the start of the standard calendar (0000-00-00).

The function is not designed for use with dates before the advent of the Gregorian calendar in October 1582. Results will not be reliable since it doesn't account for the lost days when the calendar changed from the Julian calendar.

This is the converse of the [TO_DAYS\(\)](#) function.

Examples

```
SELECT FROM_DAYS(730669);
+-----+
| FROM_DAYS(730669) |
+-----+
| 2000-07-03 |
+-----+
```

LOCALTIME

Syntax

```
LOCALTIME
LOCALTIME([precision])
```

Description

`LOCALTIME` and `LOCALTIME()` are synonyms for `NOW()`.

See Also

- [Microseconds in MariaDB](#)

LOCALTIMESTAMP

Syntax

```
LOCALTIMESTAMP  
LOCALTIMESTAMP([precision])
```

Description

`LOCALTIMESTAMP` and `LOCALTIMESTAMP()` are synonyms for `NOW()`.

See Also

- [Microseconds in MariaDB](#)

MINUTE

Syntax

```
MINUTE(time)
```

Description

Returns the minute for `time`, in the range 0 to 59.

Examples

```
SELECT MINUTE('2013-08-03 11:04:03');  
+-----+  
| MINUTE('2013-08-03 11:04:03') |  
+-----+  
| 4 |  
+-----+  
  
SELECT MINUTE ('23:12:50');  
+-----+  
| MINUTE ('23:12:50') |  
+-----+  
| 12 |  
+-----+
```

CREATE FUNCTION UDF

Syntax

```
CREATE [OR REPLACE] [AGGREGATE] FUNCTION [IF NOT EXISTS] function_name
    RETURNS {STRING|INTEGER|REAL|DECIMAL}
    SONAME shared_library_name
```

Description

A user-defined function (UDF) is a way to extend MariaDB with a new function that works like a native (built-in) MariaDB function such as [ABS\(\)](#) or [CONCAT\(\)](#).

`function_name` is the name that should be used in SQL statements to invoke the function.

To create a function, you must have the [INSERT privilege](#) for the mysql database. This is necessary because `CREATE FUNCTION` adds a row to the [mysql.func system table](#) that records the function's name, type, and shared library name. If you do not have this table, you should run the [mysql_upgrade](#) command to create it.

UDFs need to be written in C, C++ or another language that uses C calling conventions, MariaDB needs to have been dynamically compiled, and your operating system must support dynamic loading.

For an example, see `sql/udf_example.cc` in the source tree. For a collection of existing UDFs see <http://www.mysqludf.org/>.

Statements making use of user-defined functions are not [safe for replication](#).

For creating a stored function as opposed to a user-defined function, see [CREATE FUNCTION](#).

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

RETURNS

The `RETURNS` clause indicates the type of the function's return value, and can be one of [STRING](#), [INTEGER](#), [REAL](#) or [DECIMAL](#). [DECIMAL](#) functions currently return string values and should be written like [STRING](#) functions.

shared_library_name

`shared_library_name` is the basename of the shared object file that contains the code that implements the function. The file must be located in the plugin directory. This directory is given by the value of the `plugin_dir` system variable. Note that before MariaDB/MySQL 5.1, the shared object could be located in any directory that was searched by your system's dynamic linker.

AGGREGATE

Aggregate functions are summary functions such as [SUM\(\)](#) and [AVG\(\)](#).

MariaDB starting with [10.4](#)

Aggregate UDF functions can be used as [window functions](#).

OR REPLACE

MariaDB starting with [10.1.3](#)

The `OR REPLACE` clause was added in [MariaDB 10.1.3](#)

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

```
DROP FUNCTION IF EXISTS function_name;
CREATE FUNCTION name ...;
```

IF NOT EXISTS

MariaDB starting with [10.1.3](#)

The `IF NOT EXISTS` clause was added in [MariaDB 10.1.3](#)

Contents

1. [Syntax](#)
2. [Description](#)
 1. [RETURNS](#)
 2. [shared_library_name](#)
 3. [AGGREGATE](#)
 4. [OR REPLACE](#)
 5. [IF NOT EXISTS](#)
 6. [Upgrading a UDF](#)
 7. [Examples](#)
3. [See Also](#)

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

Upgrading a UDF

To upgrade the UDF's shared library, first run a [DROP FUNCTION](#) statement, then upgrade the shared library and finally run the CREATE FUNCTION statement. If you upgrade without following this process, you may crash the server.

Examples

```
CREATE FUNCTION jsoncontains_path RETURNS integer SONAME 'ha_connect.so';
Query OK, 0 rows affected (0.00 sec)
```

OR REPLACE and IF NOT EXISTS:

```
CREATE FUNCTION jsoncontains_path RETURNS integer SONAME 'ha_connect.so';
ERROR 1125 (HY000): Function 'jsoncontains_path' already exists

CREATE OR REPLACE FUNCTION jsoncontains_path RETURNS integer SONAME 'ha_connect.so';
Query OK, 0 rows affected (0.00 sec)

CREATE FUNCTION IF NOT EXISTS jsoncontains_path RETURNS integer SONAME 'ha_connect.so';
Query OK, 0 rows affected, 1 warning (0.00 sec)

SHOW WARNINGS;
+-----+-----+
| Level | Code | Message |
+-----+-----+
| Note  | 1125 | Function 'jsoncontains_path' already exists |
+-----+-----+
```

See Also

- [Identifier Names](#)
- [DROP FUNCTION](#)
- [CREATE FUNCTION](#)

MONTH

Syntax

```
MONTH(date)
```

Description

Returns the month for `date` in the range 1 to 12 for January to December, or 0 for dates such as '0000-00-00' or '2008-00-00' that have a zero month part.

Examples

```
SELECT MONTH('2019-01-03');
+-----+
| MONTH('2019-01-03') |
+-----+
|            1 |
+-----+

SELECT MONTH('2019-00-03');
+-----+
| MONTH('2019-00-03') |
+-----+
|            0 |
+-----+
```

MONTHNAME

Syntax

```
MONTHNAME(date)
```

Description

Returns the full name of the month for date. The language used for the name is controlled by the value of the `lc_time_names` system variable. See [server locale](#) for more on the supported locales.

Examples

```
SELECT MONTHNAME('2019-02-03');
+-----+
| MONTHNAME('2019-02-03') |
+-----+
| February                |
+-----+
```

Changing the locale:

```
SET lc_time_names = 'fr_CA';

SELECT MONTHNAME('2019-05-21');
+-----+
| MONTHNAME('2019-05-21') |
+-----+
| mai                     |
+-----+
```

QUARTER

Syntax

```
QUARTER(date)
```

Description

Returns the quarter of the year for `date`, in the range 1 to 4. Returns 0 if month contains a zero value, or `NULL` if the given value is not otherwise a valid date (zero values are accepted).

Examples

```
SELECT QUARTER('2008-04-01');
+-----+
| QUARTER('2008-04-01') |
+-----+
|          2 |
+-----+

SELECT QUARTER('2019-00-01');
+-----+
| QUARTER('2019-00-01') |
+-----+
|          0 |
+-----+
```

SECOND

Syntax

```
SECOND(time)
```

Description

Returns the second for a given `time` (which can include [microseconds](#)), in the range 0 to 59, or `NULL` if not given a valid time value.

Examples

```
SELECT SECOND('10:05:03');
+-----+
| SECOND('10:05:03') |
+-----+
|          3 |
+-----+

SELECT SECOND('10:05:01.999999');
+-----+
| SECOND('10:05:01.999999') |
+-----+
|          1 |
+-----+
```

DROP FUNCTION UDF

Syntax

```
DROP FUNCTION [IF EXISTS] function_name
```

Description

This statement drops the [user-defined function](#) (UDF) named `function_name`.

To drop a function, you must have the [DELETE privilege](#) for the `mysql` database. This is because `DROP FUNCTION` removes the row from the `mysql.func` system table that records the function's name, type and shared library name.

For dropping a stored function, see [DROP FUNCTION](#).

Contents

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

Upgrading a UDF

To upgrade the UDF's shared library, first run a **DROP FUNCTION** statement, then upgrade the shared library and finally run the **CREATE FUNCTION** statement. If you upgrade without following this process, you may crash the server.

Examples

```
DROP FUNCTION jsoncontains_path;
```

IF EXISTS:

```
DROP FUNCTION jsoncontains_path;
ERROR 1305 (42000): FUNCTION test.jsoncontains_path does not exist
```

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

```
SHOW WARNINGS;
+-----+-----+
| Level | Code | Message |
+-----+-----+
| Note  | 1305 | FUNCTION test.jsoncontains_path does not exist |
+-----+-----+
```

Creating User-Defined Functions

User-defined functions allow MariaDB to be extended with a new function that works like a native (built-in) MariaDB function such as **ABS()** or **CONCAT()**. There are alternative ways to add a new function: writing a native function (which requires modifying and compiling the server source code), or writing a stored function.

Statements making use of user-defined functions are not safe for replication.

Functions are written in C or C++, and to make use of them, the operating system must support dynamic loading.

Each new SQL function requires corresponding functions written in C/C++. In the list below, at least the main function - **x()** - and one other, are required. **x** should be replaced by the name of the function you are creating.

All functions need to be thread-safe, so not global or static variables that change can be allocated. Memory is allocated in **x_init()** and freed in **x_deinit()**.

Contents

1. [Simple Functions](#)
 1. [x\(\)](#)
 2. [x_init\(\)](#)
 3. [x_deinit\(\)](#)
 4. [Description](#)
2. [Aggregate Functions](#)
 1. [x_clear\(\)](#)
 2. [x_add\(\)](#)
 3. [x_remove\(\)](#)
 4. [Description](#)
3. [Examples](#)
4. [See Also](#)

Simple Functions

x()

Required for all UDFs; this is where the results are calculated.

C/C++ type SQL type

char *	STRING
long long	INTEGER
double	REAL

DECIMAL functions return string values, and so should be written accordingly. It is not possible to create ROW functions.

x_init()

Initialization function for x(). Can be used for the following:

- Check the number of arguments to X() (the SQL equivalent).
- Verify the argument types, or to force arguments to be of a particular type after the function is called.

- Specify whether the result can be NULL.
- Specify the maximum result length.
- For REAL functions, specify the maximum number of decimals for the result.
- Allocate any required memory.

x_deinit()

De-initialization function for x(). Used to de-allocate memory that was allocated in x_init().

Description

Each time the SQL function X() is called:

- MariaDB will first call the C/C++ initialization function, x_init(), assuming it exists. All setup will be performed, and if it returns an error, the SQL statement is aborted and no further functions are called.
- If there is no x_init() function, or it has been called and did not return an error, x() is then called once per row.
- After all rows have finished processing, x_deinit() is called, if present, to clean up by de-allocating any memory that was allocated in x_init().
- See [User-defined Functions Calling Sequences](#) for more details on the functions.

Aggregate Functions

The following functions are required for aggregate functions, such as [AVG\(\)](#) and [SUM\(\)](#). When using [CREATE FUNCTION](#), the [AGGREGATE](#) keyword is required.

x_clear()

Used to reset the current aggregate, but without inserting the argument as the initial aggregate value for the new group.

x_add()

Used to add the argument to the current aggregate.

x_remove()

Starting from [MariaDB 10.4](#), improves the support of [window functions](#) (so it is not obligatory to add it) and should remove the argument from the current aggregate.

Description

Each time the aggregate SQL function X() is called:

- MariaDB will first call the C/C++ initialization function, x_init(), assuming it exists. All setup will be performed, and if it returns an error, the SQL statement is aborted and no further functions are called.
- If there is no x_init() function, or it has been called and did not return an error, x() is then called once per row.
- After all rows have finished processing, x_deinit() is called, if present, to clean up by de-allocating any memory that was allocated in x_init().
- MariaDB will first call the C/C++ initialization function, x_init(), assuming it exists. All setup will be performed, and if it returns an error, the SQL statement is aborted and no further functions are called.
- The table is sorted according to the [GROUP BY](#) expression.
- x_clear() is called for the first row of each new group.
- x_add() is called once per row for each row in the same group.
- x() is called when the group changes, or after the last row, to get the aggregate result.
- The latter three steps are repeated until all rows have been processed.
- After all rows have finished processing, x_deinit() is called, if present, to clean up by de-allocating any memory that was allocated in x_init().

Examples

For an example, see `sql/udf_example.cc` in the source tree. For a collection of existing UDFs see <https://github.com/mysqludf>.

See Also

- [Stored Functions](#)
- [Stored Aggregate Functions](#)
- [User-defined Functions Calling Sequences](#)

TIME Function

Syntax

```
TIME(expr)
```

Description

Extracts the time part of the time or datetime expression `expr` and returns it as a string.

Examples

```
SELECT TIME('2003-12-31 01:02:03');
+-----+
| TIME('2003-12-31 01:02:03') |
+-----+
| 01:02:03 |
+-----+

SELECT TIME('2003-12-31 01:02:03.000123');
+-----+
| TIME('2003-12-31 01:02:03.000123') |
+-----+
| 01:02:03.000123 |
+-----+
```

TIME_FORMAT

Syntax

```
TIME_FORMAT(time,format)
```

Description

This is used like the [DATE_FORMAT\(\)](#) function, but the format string may contain format specifiers only for hours, minutes, and seconds. Other specifiers produce a NULL value or 0.

Examples

```
SELECT TIME_FORMAT('100:00:00', '%H %k %h %I %l');
+-----+
| TIME_FORMAT('100:00:00', '%H %k %h %I %l') |
+-----+
| 100 100 04 04 4 |
+-----+
```

User-Defined Functions Calling Sequences

The functions described in [Creating User-defined Functions](#) are expanded on this page. They are declared as follows:

Simple Functions

x()

If x() returns an integer, it is declared as follows:

```
long long x(UDF_INIT *initid, UDF_ARGS *args,
            char *is_null, char *error);
```

If x() returns a string (DECIMAL functions also return string values), it is declared as follows:

```
char *x(UDF_INIT *initid, UDF_ARGS *args,
        char *result, unsigned long *length,
        char *is_null, char *error);
```

If x() returns a real, it is declared as follows:

```
double x(UDF_INIT *initid, UDF_ARGS *args,
          char *is_null, char *error);
```

x_init()

```
my_bool x_init(UDF_INIT *initid, UDF_ARGS *args, char *message);
```

x_deinit()

```
void x_deinit(UDF_INIT *initid);
```

Description

initid is a parameter passed to all three functions that points to a *UDF_INIT* structure, used for communicating information between the functions. Its structure members are:

- my_bool *maybe_null*
 - *maybe_null* should be set to 1 if x_init can return a NULL value. Defaults to 1 if any arguments are declared *maybe_null*.
- unsigned int *decimals*
 - Number of decimals after the decimal point. The default, if an explicit number of decimals is passed in the arguments to the main function, is the maximum number of decimals, so if 9.5, 9.55 and 9.555 are passed to the function, the default would be three (based on 9.555, the maximum). If there are no explicit number of decimals, the default is set to 31, or one more than the maximum for the DOUBLE, FLOAT and DECIMAL types. This default can be changed in the function to suit the actual calculation.
- unsigned int *max_length*
 - Maximum length of the result. For integers, the default is 21. For strings, the length of the longest argument. For reals, the default is 13 plus the number of decimals indicated by *initid->decimals*. The length includes any signs or decimal points. Can also be set to 65KB or 16MB in order to return a BLOB. The memory remains unallocated, but this is used to decide on the data type to use if the data needs to be temporarily stored.
- char **ptr*
 - A pointer for use as required by the function. Commonly, *initid->ptr* is used to communicate allocated memory, with x_init() allocating the memory and assigning it to this pointer, x() using it, and x_deinit() de-allocating it.
- my_bool *const_item*
 - Should be set to 1 in x_init() if x() always returns the same value, otherwise 0.

Aggregate Functions

x_clear()

x_clear() is a required function for aggregate functions, and is declared as follows:

```
void x_clear(UDF_INIT *initid, char *is_null, char *error);
```

It is called when the summary results need to be reset, that is at the beginning of each new group, but also to reset the values when there were no matching rows.

Contents

1. Simple Functions

1. x()
2. x_init()
3. x_deinit()
4. Description

2. Aggregate Functions

1. x_clear()
2. x_reset()
3. x_add()
4. x_remove()

`is_null` is set to point to `CHAR(0)` before calling `x_clear()`.

In the case of an error, you can store the value to which the error argument points (a single-byte variable, not a string string buffer) in the variable.

`x_reset()`

`x_reset()` is declared as follows:

```
void x_reset(UDF_INIT *initid, UDF_ARGS *args,
             char *is_null, char *error);
```

It is called on finding the first row in a new group. Should reset the summary variables, and then use `UDF_ARGS` as the first value in the group's internal summary value. The function is not required if the UDF interface uses `x_clear()`.

`x_add()`

`x_add()` is declared as follows:

```
void x_add(UDF_INIT *initid, UDF_ARGS *args,
            char *is_null, char *error);
```

It is called for all rows belonging to the same group, and should be used to add the value in `UDF_ARGS` to the internal summary variable.

`x_remove()`

`x_remove()` was added in [MariaDB 10.4](#) and is declared as follows (same as `x_add()`):

```
void x_remove(UDF_INIT* initid, UDF_ARGS* args,
              char* is_null, char *error );
```

It adds more efficient support of aggregate UDFs as [window functions](#). `x_remove()` should "subtract" the row (reverse `x_add()`). In [MariaDB 10.4](#) aggregate UDFs will work as WINDOW functions without `x_remove()` but it will not be so efficient.

If `x_remove()` supported (defined) detected automatically.

User-Defined Functions Security

The MariaDB server imposes a number of limitations on [user-defined functions](#) for security purposes.

- The `INSERT` privilege for the `mysql` database is required to run `CREATE FUNCTION`, as a record will be added to the `mysql.func-table`.
- The `DELETE` privilege for the `mysql` database is required to run `DROP FUNCTION` as the corresponding record will be removed from the `mysql.func-table`.
- UDF object files can only be placed in the plugin directory, as specified by the value of the `plugin_dir` system variable.
- At least one symbol, beyond the required `x()` - corresponding to an SQL function `X()` - is required. These can be `x_init()`, `x_deinit()`, `xxx_reset()`, `x_clear()` and `x_add()` functions (see [Creating User-defined Functions](#)). The `allow-suspicious-udfs` `mysqld` option (by default unset) provides a workaround, permitting only one symbol to be used. This is not recommended, as it opens the possibility of loading shared objects that are not legitimate user-defined functions.

TIME_TO_SEC

Syntax

```
TIME_TO_SEC(time)
```

Description

Returns the time argument, converted to seconds.

The value returned by `TIME_TO_SEC` is of type `DOUBLE`. Before [MariaDB 5.3](#) (and MySQL 5.6), the type was `INT`. The returned value preserves microseconds of the argument. See also [Microseconds in MariaDB](#).

Examples

```
SELECT TIME_TO_SEC('22:23:00');
+-----+
| TIME_TO_SEC('22:23:00') |
+-----+
|          80580 |
+-----+
```

```
SELECT TIME_TO_SEC('00:39:38');
+-----+
| TIME_TO_SEC('00:39:38') |
+-----+
|          2378 |
+-----+
```

```
SELECT TIME_TO_SEC('09:12:55.2355');
+-----+
| TIME_TO_SEC('09:12:55.2355') |
+-----+
|          33175.2355 |
+-----+
1 row in set (0.000 sec)
```

UTC_DATE

Syntax

```
UTC_DATE, UTC_DATE()
```

Description

Returns the current [UTC date](#) as a value in 'YYYY-MM-DD' or YYYYMMDD format, depending on whether the function is used in a string or numeric context.

Examples

```
SELECT UTC_DATE(), UTC_DATE() + 0;
+-----+
| UTC_DATE() | UTC_DATE() + 0 |
+-----+
| 2010-03-27 |      20100327 |
+-----+
```

UTC_TIME

Syntax

```
UTC_TIME
UTC_TIME([precision])
```

Description

Returns the current [UTC time](#) as a value in 'HH:MM:SS' or HHMMSS.ududu format, depending on whether the function is used in a string or numeric context.

The optional *precision* determines the microsecond precision. See [Microseconds in MariaDB](#).

Examples

```
SELECT UTC_TIME(), UTC_TIME() + 0;
+-----+-----+
| UTC_TIME() | UTC_TIME() + 0 |
+-----+-----+
| 17:32:34   | 173234.000000 |
+-----+-----+
```

With precision:

```
SELECT UTC_TIME(5);
+-----+
| UTC_TIME(5) |
+-----+
| 07:52:50.78369 |
+-----+
```

See Also

- [Microseconds in MariaDB](#)

mysql.func Table

The `mysql.func` table stores information about user-defined functions (UDFs) created with the `CREATE FUNCTION UDF` statement.

MariaDB starting with 10.4

In MariaDB 10.4 and later, this table uses the [Aria](#) storage engine.

MariaDB until 10.3

In MariaDB 10.3 and before, this table uses the [MyISAM](#) storage engine.

The `mysql.func` table contains the following fields:

Field	Type	Null	Key	Default	Description
name	char(64)	NO	PRI		UDF name
ret	tinyint(1)	NO		0	
dl	char(128)	NO			Shared library name
type	enum('function','aggregate')	NO		NULL	Type, either <code>function</code> or <code>aggregate</code> . Aggregate functions are summary functions such as <code>SUM()</code> and <code>AVG()</code> .

Example

```
SELECT * FROM mysql.func;
+-----+-----+-----+-----+
| name           | ret  | dl    | type   |
+-----+-----+-----+-----+
| spider_direct_sql | 2 | ha_spider.so | function |
| spider_bg_direct_sql | 2 | ha_spider.so | aggregate |
| spider_ping_table | 2 | ha_spider.so | function |
| spider_copy_tables | 2 | ha_spider.so | function |
| spider_flush_table_mon_cache | 2 | ha_spider.so | function |
+-----+-----+-----+-----+
```

AUTO_INCREMENT

Description

The `AUTO_INCREMENT` attribute can be used to generate a unique identity for new rows. When you insert a new record to the table (or upon adding an `AUTO_INCREMENT` attribute with the `ALTER TABLE` statement), and the `auto_increment` field is `NULL` or `DEFAULT` (in the case of an `INSERT`), the value will automatically be incremented. This also applies to 0, unless the `NO_AUTO_VALUE_ON_ZERO SQL_MODE` is enabled.

`AUTO_INCREMENT` columns start from 1 by default. The automatically generated value can never be lower than 0.

Each table can have only one `AUTO_INCREMENT` column. It must be defined as a key (not necessarily the `PRIMARY KEY` or `UNIQUE` key). In some storage engines (including the default `InnoDB`), if the key consists of multiple columns, the `AUTO_INCREMENT` column must be the first column. Storage engines that permit the column to be placed elsewhere are `Aria`, `MyISAM`, `MERGE`, `Spider`, `TokuDB`, `BLACKHOLE`, `FederatedX` and `Federated`.

```
CREATE TABLE animals (
    id MEDIUMINT NOT NULL AUTO_INCREMENT,
    name CHAR(30) NOT NULL,
    PRIMARY KEY (id)
);

INSERT INTO animals (name) VALUES
    ('dog'),('cat'),('penguin'),
    ('fox'),('whale'),('ostrich');
```

```
SELECT * FROM animals;
```

id	name
1	dog
2	cat
3	penguin
4	fox
5	whale
6	ostrich

`SERIAL` is an alias for `BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE`.

```
CREATE TABLE t (id SERIAL, c CHAR(1)) ENGINE=InnoDB;

SHOW CREATE TABLE t \G
***** 1. row *****
Table: t
Create Table: CREATE TABLE `t` (
  `id` bigint(20) unsigned NOT NULL AUTO_INCREMENT,
  `c` char(1) DEFAULT NULL,
  UNIQUE KEY `id` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

Setting or Changing the Auto_Increment Value

You can use an `ALTER TABLE` statement to assign a new value to the `auto_increment` table option, or set the `insert_id` server system variable to change the next `AUTO_INCREMENT` value inserted by the current session.

`LAST_INSERT_ID()` can be used to see the last `AUTO_INCREMENT` value inserted by the current session.

Contents

1. [Description](#)
2. [Setting or Changing the Auto_Increment Value](#)
3. [InnoDB](#)
4. [Setting Explicit Values](#)
5. [Missing Values](#)
6. [Replication](#)
7. [CHECK Constraints, DEFAULT Values and Virtual Columns](#)
8. [Generating Auto_Increment Values When Adding the Attribute](#)
9. [See Also](#)

```

ALTER TABLE animals AUTO_INCREMENT=8;

INSERT INTO animals (name) VALUES ('aardvark');

SELECT * FROM animals;
+----+-----+
| id | name   |
+----+-----+
| 1  | dog    |
| 2  | cat    |
| 3  | penguin |
| 4  | fox    |
| 5  | whale   |
| 6  | ostrich |
| 8  | aardvark|
+----+-----+

SET insert_id=12;

INSERT INTO animals (name) VALUES ('gorilla');

SELECT * FROM animals;
+----+-----+
| id | name   |
+----+-----+
| 1  | dog    |
| 2  | cat    |
| 3  | penguin |
| 4  | fox    |
| 5  | whale   |
| 6  | ostrich |
| 8  | aardvark|
| 12 | gorilla|
+----+-----+

```

InnoDB

Until [MariaDB 10.2.3](#), InnoDB used an auto-increment counter that is stored in memory. When the server restarts, the counter is re-initialized to the highest value used in the table, which cancels the effects of any `AUTO_INCREMENT = N` option in the table statements.

From [MariaDB 10.2.4](#), this restriction has been lifted and `AUTO_INCREMENT` is persistent.

See also [AUTO_INCREMENT Handling in InnoDB](#).

Setting Explicit Values

It is possible to specify a value for an `AUTO_INCREMENT` column. If the key is primary or unique, the value must not already exist in the key.

If the new value is higher than the current maximum value, the `AUTO_INCREMENT` value is updated, so the next value will be higher. If the new value is lower than the current maximum value, the `AUTO_INCREMENT` value remains unchanged.

The following example demonstrates these behaviors:

```

CREATE TABLE t (id INTEGER UNSIGNED AUTO_INCREMENT PRIMARY KEY) ENGINE = InnoDB;

INSERT INTO t VALUES (NULL);
SELECT id FROM t;
+----+
| id |
+----+
| 1 |
+----+

INSERT INTO t VALUES (10); -- higher value
SELECT id FROM t;
+----+
| id |
+----+
| 1 |
| 10 |
+----+

INSERT INTO t VALUES (2); -- lower value
INSERT INTO t VALUES (NULL); -- auto value
SELECT id FROM t;
+----+
| id |
+----+
| 1 |
| 2 |
| 10 |
| 11 |
+----+

```

The [ARCHIVE](#) storage engine does not allow to insert a value that is lower than the current maximum.

Missing Values

An AUTO_INCREMENT column normally has missing values. This happens because if a row is deleted, or an AUTO_INCREMENT value is explicitly updated, old values are never re-used. The REPLACE statement also deletes a row, and its value is wasted. With InnoDB, values can be reserved by a transaction; but if the transaction fails (for example, because of a ROLLBACK) the reserved value will be lost.

Thus AUTO_INCREMENT values can be used to sort results in a chronological order, but not to create a numeric sequence.

Replication

To make master-master or Galera safe to use AUTO_INCREMENT one should use the system variables [auto_increment_increment](#) and [auto_increment_offset](#) to generate unique values for each server.

CHECK Constraints, DEFAULT Values and Virtual Columns

MariaDB starting with [10.2.6](#)

From [MariaDB 10.2.6](#) auto_increment columns are no longer permitted in [CHECK constraints](#), [DEFAULT value expressions](#) and [virtual columns](#). They were permitted in earlier versions, but did not work correctly. See [MDEV-11117](#).

Generating Auto_Increment Values When Adding the Attribute

```

CREATE OR REPLACE TABLE t1 (a INT);
INSERT t1 VALUES (0),(0),(0);
ALTER TABLE t1 MODIFY a INT NOT NULL AUTO_INCREMENT PRIMARY KEY;
SELECT * FROM t1;
+---+
| a |
+---+
| 1 |
| 2 |
| 3 |
+---+

```

```

CREATE OR REPLACE TABLE t1 (a INT);
INSERT t1 VALUES (5),(0),(8),(0);
ALTER TABLE t1 MODIFY a INT NOT NULL AUTO_INCREMENT PRIMARY KEY;
SELECT * FROM t1;
+---+
| a |
+---+
| 5 |
| 6 |
| 8 |
| 9 |
+---+

```

If the [NO_AUTO_VALUE_ON_ZERO SQL_MODE](#) is set, zero values will not be automatically incremented:

```

SET SQL_MODE='no_auto_value_on_zero';
CREATE OR REPLACE TABLE t1 (a INT);
INSERT t1 VALUES (3), (0);
ALTER TABLE t1 MODIFY a INT NOT NULL AUTO_INCREMENT PRIMARY KEY;
SELECT * FROM t1;
+---+
| a |
+---+
| 0 |
| 3 |
+---+

```

See Also

- [Getting Started with Indexes](#)
- [Sequences](#) - an alternative to auto_increment available from [MariaDB 10.3](#)
- [AUTO_INCREMENT FAQ](#)
- [LAST_INSERT_ID\(\)](#)
- [AUTO_INCREMENT handling in InnoDB](#)
- [BLACKHOLE and AUTO_INCREMENT](#)
- [UUID_SHORT\(\)](#) - Generate unique ids
- [Generating Identifiers – from AUTO_INCREMENT to Sequence \(percona.com\)](#)

AsBinary

A synonym for [ST_AsBinary\(\)](#).

AsWKB

A synonym for [ST_AsBinary\(\)](#).

MLineFromWKB

Syntax

```
MLineFromWKB(wkb[,srid])
MultiLineStringFromWKB(wkb[,srid])
```

Description

Constructs a [MULTILINESTRING](#) value using its [WKB](#) representation and [SRID](#).

`MLineFromWKB()` and `MultiLineStringFromWKB()` are synonyms.

Examples

```
SET @g = ST_AsBinary(MLineFromText('MULTILINESTRING((10 48,10 21,10 0),(16 0,16 23,16 48))'));
SELECT ST_AsText(MLineFromWKB(@g));
+-----+
| ST_AsText(MLineFromWKB(@g)) |
+-----+
| MULTILINESTRING((10 48,10 21,10 0),(16 0,16 23,16 48)) |
+-----+
```

BIGINT

Syntax

```
BIGINT[(M)] [SIGNED | UNSIGNED | ZEROFILL]
```

Description

A large integer. The signed range is -9223372036854775808 to 9223372036854775807 . The unsigned range is 0 to 18446744073709551615 .

If a column has been set to ZEROFILL, all values will be prepended by zeros so that the BIGINT value contains a number of M digits.

Note: If the ZEROFILL attribute has been specified, the column will automatically become UNSIGNED.

For more details on the attributes, see [Numeric Data Type Overview](#).

SERIAL is an alias for:

```
BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE
```

INT8 is a synonym for BIGINT .

Examples

```
CREATE TABLE bigints (a BIGINT,b BIGINT UNSIGNED,c BIGINT ZEROFILL);
```

With `strict_mode` set, the default from [MariaDB 10.2.4](#):

```
INSERT INTO bigints VALUES (-10,-10,-10);
ERROR 1264 (22003): Out of range value for column 'b' at row 1

INSERT INTO bigints VALUES (-10,10,-10);
ERROR 1264 (22003): Out of range value for column 'c' at row 1

INSERT INTO bigints VALUES (-10,10,10);

INSERT INTO bigints VALUES (9223372036854775808,9223372036854775808,9223372036854775808);
ERROR 1264 (22003): Out of range value for column 'a' at row 1

INSERT INTO bigints VALUES (9223372036854775807,9223372036854775808,9223372036854775808);

SELECT * FROM bigints;
+-----+-----+-----+
| a    | b    | c    |
+-----+-----+-----+
|      -10 |          10 | 00000000000000000000000000000010 |
| 9223372036854775807 | 9223372036854775808 | 09223372036854775808 |
+-----+-----+-----+
```

With `strict_mode` unset, the default until MariaDB 10.2.3:

```
INSERT INTO bigints VALUES (-10,-10,-10);
Query OK, 1 row affected, 2 warnings (0.08 sec)
Warning (Code 1264): Out of range value for column 'b' at row 1
Warning (Code 1264): Out of range value for column 'c' at row 1

INSERT INTO bigints VALUES (-10,10,-10);
Query OK, 1 row affected, 1 warning (0.08 sec)
Warning (Code 1264): Out of range value for column 'c' at row 1

INSERT INTO bigints VALUES (-10,10,10);

INSERT INTO bigints VALUES (9223372036854775808,9223372036854775808,9223372036854775808);
Query OK, 1 row affected, 1 warning (0.07 sec)
Warning (Code 1264): Out of range value for column 'a' at row 1

INSERT INTO bigints VALUES (9223372036854775807,9223372036854775808,9223372036854775808);

SELECT * FROM bigints;
+-----+-----+-----+
| a      | b      | c      |
+-----+-----+-----+
|          -10 |          0 | 000000000000000000000000 |
|          -10 |         10 | 000000000000000000000000 |
|          -10 |         10 | 000000000000000000000010 |
| 9223372036854775807 | 9223372036854775808 | 09223372036854775808 |
| 9223372036854775807 | 9223372036854775808 | 09223372036854775808 |
+-----+-----+-----+
```

See Also

- Numeric Data Type Overview
 - TINYINT
 - SMALLINT
 - MEDIUMINT
 - INTEGER

BINARY

This page describes the **BINARY** data type. For details about the operator, see [Binary Operator](#).

Syntax

Description

The `BINARY` type is similar to the `CHAR` type, but stores binary byte strings rather than non-binary character strings. `M` represents the column length in bytes.

It contains no character set, and comparison and sorting are based on the numeric value of the bytes.

If the maximum length is exceeded, and `SQL strict mode` is not enabled , the extra characters will be dropped with a warning. If strict mode is enabled, an error will occur.

`BINARY` values are right-padded with `0x00` (the zero byte) to the specified length when inserted. The padding is *not* removed on select, so this needs to be taken into account when sorting and comparing, where all bytes are significant. The zero byte, `0x00` is less than a space for comparison purposes.

Examples

Inserting too many characters, first with strict mode off, then with it on:

```
CREATE TABLE bins (a BINARY(10));

INSERT INTO bins VALUES('12345678901');
Query OK, 1 row affected, 1 warning (0.04 sec)

SELECT * FROM bins;
+-----+
| a    |
+-----+
| 1234567890 |
+-----+

SET sql_mode='STRICT_ALL_TABLES';

INSERT INTO bins VALUES('12345678901');
ERROR 1406 (22001): Data too long for column 'a' at row 1
```

Sorting is performed with the byte value:

```
TRUNCATE bins;

INSERT INTO bins VALUES('A'),('B'),('a'),('b');

SELECT * FROM bins ORDER BY a;
+-----+
| a    |
+-----+
| A    |
| B    |
| a    |
| b    |
+-----+
```

Using `CAST` to sort as a `CHAR` instead:

```
SELECT * FROM bins ORDER BY CAST(a AS CHAR);
+-----+
| a    |
+-----+
| a    |
| A    |
| b    |
| B    |
+-----+
```

Contents

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

The field is a BINARY(10), so padding of two '\0's are inserted, causing comparisons that don't take this into account to fail:

```
TRUNCATE bins;

INSERT INTO bins VALUES('12345678');

SELECT a = '12345678', a = '12345678\0\0' FROM bins;
+-----+-----+
| a = '12345678' | a = '12345678\0\0' |
+-----+-----+
|          0 |           1 |
+-----+-----+
```

See Also

- [CHAR](#)
- [Data Type Storage Requirements](#)

BIT

Syntax

```
BIT[(M)]
```

Description

A bit-field type. M indicates the number of bits per value, from 1 to 64. The default is 1 if M is omitted.

Bit values can be inserted with b'value' notation, where value is the bit value in 0's and 1's.

Bit fields are automatically zero-padded from the left to the full length of the bit, so for example in a BIT(4) field, '10' is equivalent to '0010'.

Bits are returned as binary, so to display them, either add 0, or use a function such as [HEX](#), [OCT](#) or [BIN](#) to convert them.

Examples

```
CREATE TABLE b ( b1 BIT(8) );
```

With [strict_mode](#) set, the default from [MariaDB 10.2.4](#):

```
INSERT INTO b VALUES (b'11111111');

INSERT INTO b VALUES (b'01010101');

INSERT INTO b VALUES (b'111111111111');
ERROR 1406 (22001): Data too long for column 'b1' at row 1

SELECT b1+0, HEX(b1), OCT(b1), BIN(b1) FROM b;
+-----+-----+-----+
| b1+0 | HEX(b1) | OCT(b1) | BIN(b1) |
+-----+-----+-----+
|   255 |   FF    |   377   | 11111111 |
|    85  |    55   |   125   | 01010101 |
+-----+-----+-----+
```

With [strict_mode](#) unset, the default until [MariaDB 10.2.3](#):

```

INSERT INTO b VALUES (b'11111111'),(b'01010101'),(b'111111111111');
Query OK, 3 rows affected, 1 warning (0.10 sec)
Records: 3 Duplicates: 0 Warnings: 1

SHOW WARNINGS;
+-----+
| Level | Code | Message |
+-----+
| Warning | 1264 | Out of range value for column 'b1' at row 3 |
+-----+

SELECT b1+0, HEX(b1), OCT(b1), BIN(b1) FROM b;
+-----+
| b1+0 | HEX(b1) | OCT(b1) | BIN(b1) |
+-----+
| 255 | FF      | 377    | 11111111 |
| 85  | 55      | 125    | 1010101  |
| 255 | FF      | 377    | 11111111 |
+-----+

```

MPointFromWKB

Syntax

```

MPointFromWKB(wkb[,srid])
MultiPointFromWKB(wkb[,srid])

```

Description

Constructs a [MULTIPOINT](#) value using its [WKB](#) representation and [SRID](#).

`MPointFromWKB()` and `MultiPointFromWKB()` are synonyms.

Examples

```

SET @g = ST_AsBinary(MPointFromText('MultiPoint( 1 1, 2 2, 5 3, 7 2, 9 3, 8 4, 6 6, 6 9, 4 9, 1 5 )'));

SELECT ST_AsText(MPointFromWKB(@g));
+-----+
| ST_AsText(MPointFromWKB(@g)) |
+-----+
| MULTIPOINT(1 1,2 2,5 3,7 2,9 3,8 4,6 6,6 9,4 9,1 5) |
+-----+

```

GeomCollFromWKB

A synonym for [ST_GeomCollFromWKB](#).

GeometryCollectionFromWKB

A synonym for [ST_GeomCollFromWKB](#).

GeometryFromWKB

A synonym for [ST_GeomFromWKB](#).

GeomFromWKB

A synonym for [ST_GeomFromWKB](#).

LineFromWKB

A synonym for [ST_LineFromWKB](#).

LineStringFromWKB

A synonym for [ST_LineFromWKB](#).

BLOB

Syntax

`BLOB[(M)]`

Description

A `BLOB` column with a maximum length of 65,535 ($2^{16} - 1$) bytes. Each `BLOB` value is stored using a two-byte length prefix that indicates the number of bytes in the value.

An optional length `M` can be given for this type. If this is done, MariaDB creates the column as the smallest `BLOB` type large enough to hold values `M` bytes long.

`BLOBS` can also be used to store [dynamic columns](#).

Before [MariaDB 10.2.1](#), `BLOB` and `TEXT` columns could not be assigned a `DEFAULT` value. This restriction was lifted in [MariaDB 10.2.1](#).

Contents

1. [Syntax](#)
2. [Description](#)
 1. [Indexing](#)
 2. [Oracle Mode](#)
3. [See Also](#)

Indexing

MariaDB starting with [10.4](#)

From [MariaDB 10.4](#), it is possible to set a [unique index](#) on a column that uses the `BLOB` data type. In previous releases this was not possible, as the index would only guarantee the uniqueness of a fixed number of characters.

Oracle Mode

MariaDB starting with [10.3](#)

In [Oracle mode from MariaDB 10.3](#), `BLOB` is a synonym for `LONGBLOB`.

See Also

- [BLOB and TEXT Data Types](#)
- [Data Type Storage Requirements](#)
- [Oracle mode from MariaDB 10.3](#)

BOOLEAN

Syntax

`BOOL, BOOLEAN`

Description

These types are synonyms for [TINYINT\(1\)](#). A value of zero is considered false. Non-zero values are considered true.

However, the values `TRUE` and `FALSE` are merely aliases for 1 and 0. See [Boolean Literals](#), as well as the `IS` operator for testing values against a boolean.

Examples

```
CREATE TABLE boo (i BOOLEAN);

DESC boo;
+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+
| i     | tinyint(1) | YES  |      | NULL    |       |
+-----+-----+-----+-----+
```

```
SELECT IF(0, 'true', 'false');
+-----+
| IF(0, 'true', 'false') |
+-----+
| false                  |
+-----+

SELECT IF(1, 'true', 'false');
+-----+
| IF(1, 'true', 'false') |
+-----+
| true                   |
+-----+

SELECT IF(2, 'true', 'false');
+-----+
| IF(2, 'true', 'false') |
+-----+
| true                   |
+-----+
```

TRUE and FALSE as aliases for 1 and 0:

```
SELECT IF(0 = FALSE, 'true', 'false');

+-----+
| IF(0 = FALSE, 'true', 'false') |
+-----+
| true                         |
+-----+

SELECT IF(1 = TRUE, 'true', 'false');

+-----+
| IF(1 = TRUE, 'true', 'false') |
+-----+
| true                         |
+-----+

SELECT IF(2 = TRUE, 'true', 'false');

+-----+
| IF(2 = TRUE, 'true', 'false') |
+-----+
| false                        |
+-----+

SELECT IF(2 = FALSE, 'true', 'false');

+-----+
| IF(2 = FALSE, 'true', 'false') |
+-----+
| false                        |
+-----+
```

The last two statements display the results shown because 2 is equal to neither 1 nor 0.

See Also

- Boolean Literals
- IS operator

CHAR

This article covers the CHAR data type. See [CHAR Function](#) for the function.

Syntax

```
[NATIONAL] CHAR[(M)] [CHARACTER SET charset_name] [COLLATE collation_name]
```

Description

A fixed-length string that is always right-padded with spaces to the specified length when stored. M represents the column length in characters. The range of M is 0 to 255. If M is omitted, the length is 1.

CHAR(0) columns can contain 2 values: an empty string or NULL. Such columns cannot be part of an index. The [CONNECT](#) storage engine does not support CHAR(0).

Note: Trailing spaces are removed when CHAR values are retrieved unless the PAD_CHAR_TO_FULL_LENGTH SQL mode is enabled.

Before MariaDB 10.2, all collations were of type PADSPACE, meaning that CHAR (as well as VARCHAR and TEXT) values are compared without regard for trailing spaces. This does not apply to the LIKE pattern-matching operator, which takes into account trailing spaces.

If a unique index consists of a column where trailing pad characters are stripped or ignored, inserts into that column where values differ only by the number of trailing pad characters will result in a duplicate-key error.

Examples

Trailing spaces:

```
CREATE TABLE strtest (c CHAR(10));
INSERT INTO strtest VALUES('Maria   ');

SELECT c='Maria',c='Maria'  ' FROM strtest;
+-----+
| c='Maria' | c='Maria'  |
+-----+
|      1    |          1  |
+-----+

SELECT c LIKE 'Maria',c LIKE 'Maria'  ' FROM strtest;
+-----+
| c LIKE 'Maria' | c LIKE 'Maria'  |
+-----+
|      1        |          0  |
+-----+
```

NO PAD Collations

MariaDB starting with 10.2

NO PAD collations regard trailing spaces as normal characters. You can get a list of all NO PAD collations by querying the [Information Schema Collations table](#), for example:

Contents

1. [Syntax](#)
2. [Description](#)
3. [Examples](#)
4. [NO PAD Collations](#)
5. [See Also](#)

```
SELECT collation_name FROM information_schema.collations
  WHERE collation_name LIKE "%nopad%";
+-----+-----+
| collation_name | |
+-----+-----+
| big5_chinese_nopad_ci | |
| big5_nopad_bin | |
...|
```

See Also

- [CHAR Function](#)
- [VARCHAR](#)
- [BINARY](#)
- [Data Type Storage Requirements](#)

MultiLineStringFromWKB

A synonym for [MLineFromWKB\(\)](#).

MultiPointFromWKB

A synonym for [MPointFromWKB](#).

MultiPolygonFromWKB

Synonym for [MPolyFromWKB](#).

PointFromWKB

A synonym for [ST_PointFromWKB](#).

PolyFromWKB

A synonym for [ST_PolyFromWKB](#).

PolygonFromWKB

A synonym for [ST_PolyFromWKB](#).

ST_AsBinary

Syntax

```
ST_AsBinary(g)
AsBinary(g)
ST_AsWKB(g)
AsWKB(g)
```

Description

Converts a value in internal geometry format to its [WKB](#) representation and returns the binary result.

`ST_AsBinary()` , `AsBinary()` , `ST_AsWKB()` and `AsWKB()` are synonyms,

Examples

```

SET @poly = ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))';
SELECT ST_AsBinary(@poly);

SELECT ST_AsText(ST_GeomFromWKB(ST_AsWKB(@poly)));
+-----+
| ST_AsText(ST_GeomFromWKB(ST_AsWKB(@poly))) |
+-----+
| POLYGON((0 0,0 1,1 1,1 0,0 0))           |
+-----+

```

ST_AsWKB

A synonym for [ST_AsBinary\(\)](#).

ST_GeometryCollectionFromWKB

A synonym for [ST_GeomCollFromWKB](#).

ST_GeometryFromWKB

A synonym for [ST_GeomFromWKB](#).

DATE

Syntax

DATE

Contents

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

Description

A date. The supported range is '1000-01-01' to '9999-12-31'. MariaDB displays DATE values in 'YYYY-MM-DD' format, but can be assigned dates in looser formats, including strings or numbers, as long as they make sense. These include a short year, YY-MM-DD, no delimiters, YYMMDD, or any other acceptable delimiter, for example YYYY/MM/DD. For details, see [date and time literals](#).

'0000-00-00' is a permitted special value (zero-date), unless the [NO_ZERO_DATE SQL_MODE](#) is used. Also, individual components of a date can be set to 0 (for example: '2015-00-12'), unless the [NO_ZERO_IN_DATE SQL_MODE](#) is used. In many cases, the result of an expression involving a zero-date, or a date with zero-parts, is NULL. If the [ALLOW_INVALID_DATES SQL_MODE](#) is enabled, if the day part is in the range between 1 and 31, the date does not produce any error, even for months that have less than 31 days.

Oracle Mode

MariaDB starting with 10.3

In Oracle mode from MariaDB 10.3, DATE with a time portion is a synonym for DATETIME. See also [mariadb_schema](#).

Examples

```

CREATE TABLE t1 (d DATE);

INSERT INTO t1 VALUES ("2010-01-12"), ("2011-2-28"), ('120314'),('13*04*21');

SELECT * FROM t1;
+-----+
| d      |
+-----+
| 2010-01-12 |
| 2011-02-28 |
| 2012-03-14 |
| 2013-04-21 |
+-----+

```

See Also

- [mariadb_schema](#) data type qualifier

DATETIME

Syntax

```
DATETIME [(microsecond precision)]
```

Description

A date and time combination.

MariaDB displays `DATETIME` values in '`YYYY-MM-DD HH:MM:SS.fffffff`' format, but allows assignment of values to `DATETIME` columns using either strings or numbers. For details, see [date and time literals](#).

`DATETIME` columns also accept `CURRENT_TIMESTAMP` as the default value.

MariaDB 10.1.2 introduced the `--mysql56-temporal-format` option, on by default, which allows MariaDB to store DATETIMES using the same low-level format MySQL 5.6 uses. For more information, see [Internal Format](#), below.

For storage requirements, see [Data Type Storage Requirements](#).

Supported Values

MariaDB stores values that use the `DATETIME` data type in a format that supports values between `1000-01-01 00:00:00.000000` and `9999-12-31 23:59:59.999999`.

MariaDB can also store `microseconds` with a precision between 0 and 6. If no microsecond precision is specified, then 0 is used by default.

MariaDB also supports '`0000-00-00`' as a special zero-date value, unless `NO_ZERO_DATE` is specified in the `SQL_MODE`. Similarly, individual components of a date can be set to `0` (for example: '`2015-00-12`'), unless `NO_ZERO_DATE` is specified in the `SQL_MODE`. In many cases, the result of an expression involving a zero-date, or a date with zero-parts, is `NULL`. If the `ALLOW_INVALID_DATES` `SQL_MODE` is enabled, if the day part is in the range between 1 and 31, the date does not produce any error, even for months that have less than 31 days.

Time Zones

If a column uses the `DATETIME` data type, then any inserted values are stored as-is, so no automatic time zone conversions are performed.

MariaDB also does not currently support time zone literals that contain time zone identifiers. See [MDEV-11829](#) for more information.

MariaDB validates `DATETIME` literals against the session's time zone. For example, if a specific time range never occurred in a specific time zone due to daylight savings time, then `DATETIME` values within that range would be invalid for that time zone.

For example, daylight savings time started on March 10, 2019 in the US, so the time range between 02:00:00 and 02:59:59 is invalid for that day in US time zones:

```
SET time_zone = 'America/New_York';
Query OK, 0 rows affected (0.000 sec)

INSERT INTO timestamp_test VALUES ('2019-03-10 02:55:05');
ERROR 1292 (22007): Incorrect datetime value: '2019-03-10 02:55:05' for column `db1`.`timestamp_test`.`timestamp_test` at row 1
```

But that same time range is fine in other time zones, such as [Coordinated Universal Time \(UTC\)](#). For example:

```
SET time_zone = 'UTC';
Query OK, 0 rows affected (0.000 sec)

INSERT INTO timestamp_test VALUES ('2019-03-10 02:55:05');
Query OK, 1 row affected (0.002 sec)
```

Oracle Mode

Contents

1. [Syntax](#)
2. [Description](#)
3. [Supported Values](#)
4. [Time Zones](#)
5. [Oracle Mode](#)
6. [Internal Format](#)
7. [Examples](#)
8. [See Also](#)

MariaDB starting with 10.3

In Oracle mode from MariaDB 10.3, DATE with a time portion is a synonym for DATETIME . See also mariadb_schema.

Internal Format

In MariaDB 10.1.2 a new temporal format was introduced from MySQL 5.6 that alters how the TIME , DATETIME and TIMESTAMP columns operate at lower levels. These changes allow these temporal data types to have fractional parts and negative values. You can disable this feature using the mysql56_temporal_format system variable.

Tables that include TIMESTAMP values that were created on an older version of MariaDB or that were created while the mysql56_temporal_format system variable was disabled continue to store data using the older data type format.

In order to update table columns from the older format to the newer format, execute an ALTER TABLE... MODIFY COLUMN statement that changes the column to the *same* data type. This change may be needed if you want to export the table's tablespace and import it onto a server that has mysql56_temporal_format=ON set (see MDEV-15225).

For instance, if you have a DATETIME column in your table:

```
SHOW VARIABLES LIKE 'mysql56_temporal_format';
```

```
+-----+-----+
| Variable_name      | Value   |
+-----+-----+
| mysql56_temporal_format | ON    |
+-----+-----+
```

```
ALTER TABLE example_table MODIFY ts_col DATETIME;
```

When MariaDB executes the ALTER TABLE statement, it converts the data from the older temporal format to the newer one.

In the event that you have several tables and columns using temporal data types that you want to switch over to the new format, make sure the system variable is enabled, then perform a dump and restore using mysqldump . The columns using relevant temporal data types are restored using the new temporal format.

Starting from MariaDB 10.5.1 columns with old temporal formats are marked with a /* mariadb-5.3 */ comment in the output of SHOW CREATE TABLE , SHOW COLUMNS , DESCRIBE statements, as well as in the COLUMN_TYPE column of the INFORMATION_SCHEMA.COLUMNS Table.

```
SHOW CREATE TABLE mariadb5312_datetime\G
***** 1. row *****
Table: mariadb5312_datetime
Create Table: CREATE TABLE `mariadb5312_datetime` (
  `dt0` datetime /* mariadb-5.3 */ DEFAULT NULL,
  `dt6` datetime(6) /* mariadb-5.3 */ DEFAULT NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1
```

Examples

```
CREATE TABLE t1 (d DATETIME);
```

```
INSERT INTO t1 VALUES ("2011-03-11"), ("2012-04-19 13:08:22"),
("2013-07-18 13:44:22.123456");
```

```
SELECT * FROM t1;
+-----+
| d      |
+-----+
| 2011-03-11 00:00:00 |
| 2012-04-19 13:08:22 |
| 2013-07-18 13:44:22 |
+-----+
```

```

CREATE TABLE t2 (d DATETIME(6));

INSERT INTO t2 VALUES ("2011-03-11"), ("2012-04-19 13:08:22"),
("2013-07-18 13:44:22.123456");

SELECT * FROM t2;
+-----+
| d      |
+-----+
| 2011-03-11 00:00:00.000000 |
| 2012-04-19 13:08:22.000000 |
| 2013-07-18 13:44:22.123456 |
+-----++
```

Strings used in datetime context are automatically converted to datetime(6). If you want to have a datetime without seconds, you should use [CONVERT\(..,datetime\)](#).

```

SELECT CONVERT('2007-11-30 10:30:19',datetime);
+-----+
| CONVERT('2007-11-30 10:30:19',datetime) |
+-----+
| 2007-11-30 10:30:19                   |
+-----+


SELECT CONVERT('2007-11-30 10:30:19',datetime(6));
+-----+
| CONVERT('2007-11-30 10:30:19',datetime(6)) |
+-----+
| 2007-11-30 10:30:19.000000               |
+-----+
```

See Also

- [Data Type Storage Requirements](#)
- [CONVERT\(\)](#)
- [Oracle mode from MariaDB 10.3](#)
- [mariadb_schema data type qualifier](#)

ST_GeomFromWKB

Syntax

```

ST_GeomFromWKB(wkb[,srid])
ST_GeometryFromWKB(wkb[,srid])
GeomFromWKB(wkb[,srid])
GeometryFromWKB(wkb[,srid])
```

Description

Constructs a geometry value of any type using its [WKB](#) representation and SRID.

`ST_GeomFromWKB()` , `ST_GeometryFromWKB()` , `GeomFromWKB()` and `GeometryFromWKB()` are synonyms.

Examples

```

SET @g = ST_AsBinary(ST_LineFromText('LINESTRING(0 4, 4 6)');

SELECT ST_AsText(ST_GeomFromWKB(@g));
+-----+
| ST_AsText(ST_GeomFromWKB(@g)) |
+-----+
| LINESTRING(0 4,4 6)           |
+-----+

```

ST_LineFromWKB

Syntax

```

ST_LineFromWKB(wkb[,srid])
LineFromWKB(wkb[,srid])
ST_LineStringFromWKB(wkb[,srid])
LineStringFromWKB(wkb[,srid])

```

Description

Constructs a LINESTRING value using its [WKB](#) representation and SRID.

`ST_LineFromWKB()` , `LineFromWKB()` , `ST_LineStringFromWKB()` , and `LineStringFromWKB()` are synonyms.

Examples

```

SET @g = ST_AsBinary(ST_LineFromText('LineString(0 4,4 6)');

SELECT ST_AsText(ST_LineFromWKB(@g)) AS l;
+-----+
| l           |
+-----+
| LINESTRING(0 4,4 6) |
+-----+

```

DECIMAL

Syntax

```
DECIMAL[(M[,D])] [SIGNED | UNSIGNED | ZEROFILL]
```

Description

A packed "exact" fixed-point number. `M` is the total number of digits (the precision) and `D` is the number of digits after the decimal point (the scale).

- The decimal point and (for negative numbers) the "-" sign are not counted in `M`.
- If `D` is `0`, values have no decimal point or fractional part and on [INSERT](#) the value will be rounded to the nearest `DECIMAL`.
- The maximum number of digits (`M`) for `DECIMAL` is 65.
- The maximum number of supported decimals (`D`) is 30 before [Mariadb 10.2.1](#) and 38 afterwards.
- If `D` is omitted, the default is `0`. If `M` is omitted, the default is `10`.

`UNSIGNED` , if specified, disallows negative values.

`ZEROFILL` , if specified, pads the number with zeros, up to the total number of digits specified by `M`.

All basic calculations (+, -, *, /) with `DECIMAL` columns are done with a precision of 65 digits.

Contents

- [Syntax](#)
- [Description](#)
- [Examples](#)
- [See Also](#)

For more details on the attributes, see [Numeric Data Type Overview](#).

`DEC`, `NUMERIC` and `FIXED` are synonyms, as well as `NUMBER` in [Oracle mode from MariaDB 10.3](#).

Examples

```
CREATE TABLE t1 (d DECIMAL UNSIGNED ZEROFILL);

INSERT INTO t1 VALUES (1),(2),(3),(4.0),(5.2),(5.7);
Query OK, 6 rows affected, 2 warnings (0.16 sec)
Records: 6 Duplicates: 0 Warnings: 2

Note (Code 1265): Data truncated for column 'd' at row 5
Note (Code 1265): Data truncated for column 'd' at row 6

SELECT * FROM t1;
+-----+
| d    |
+-----+
| 0000000001 |
| 0000000002 |
| 0000000003 |
| 0000000004 |
| 0000000005 |
| 0000000006 |
+-----+
```

With `strict_mode` set, the default from [MariaDB 10.2.4](#):

```
INSERT INTO t1 VALUES (-7);
ERROR 1264 (22003): Out of range value for column 'd' at row 1
```

With `strict_mode` unset, the default until [MariaDB 10.2.3](#):

```
INSERT INTO t1 VALUES (-7);
Query OK, 1 row affected, 1 warning (0.02 sec)
Warning (Code 1264): Out of range value for column 'd' at row 1

SELECT * FROM t1;
+-----+
| d    |
+-----+
| 0000000001 |
| 0000000002 |
| 0000000003 |
| 0000000004 |
| 0000000005 |
| 0000000006 |
| 0000000000 |
+-----+
```

See Also

- [Numeric Data Type Overview](#)
- [Oracle mode from MariaDB 10.3](#)

ENUM

Syntax

```
ENUM('value1','value2',...) [CHARACTER SET charset_name] [COLLATE collation_name]
```

Description

An enumeration. A string object that can have only one value, chosen from the list of values 'value1', 'value2', ..., NULL or the special " error value. In theory, an ENUM column can have a maximum of 65,535 distinct values; in practice, the real maximum depends on many factors. ENUM values are represented internally as integers.

Trailing spaces are automatically stripped from ENUM values on table creation.

ENUMs require relatively little storage space compared to strings, either one or two bytes depending on the number of enumeration values.

NULL and empty values

An ENUM can also contain NULL and empty values. If the ENUM column is declared to permit NULL values, NULL becomes a valid value, as well as the default value (see below). If strict SQL Mode is not enabled, and an invalid value is inserted into an ENUM, a special empty string, with an index value of zero (see Numeric index, below), is inserted, with a warning. This may be confusing, because the empty string is also a possible value, and the only difference if that is this case its index is not 0. Inserting will fail with an error if strict mode is active.

If a DEFAULT clause is missing, the default value will be:

- NULL if the column is nullable;
- otherwise, the first value in the enumeration.

Numeric index

ENUM values are indexed numerically in the order they are defined, and sorting will be performed in this numeric order. We suggest not using ENUM to store numerals, as there is little to no storage space benefit, and it is easy to confuse the enum integer with the enum numeral value by leaving out the quotes.

An ENUM defined as ENUM('apple','orange','pear') would have the following index values:

Index	Value
NULL	NULL
0	"
1	'apple'
2	'orange'
3	'pear'

Examples

Contents

1. [Syntax](#)
2. [Description](#)
 1. [NULL and empty values](#)
 2. [Numeric index](#)
3. [Examples](#)
4. [See Also](#)

```

CREATE TABLE fruits (
    id INT NOT NULL auto_increment PRIMARY KEY,
    fruit ENUM('apple','orange','pear'),
    bushels INT);

DESCRIBE fruits;
+-----+-----+-----+-----+
| Field | Type            | Null | Key | Default | Extra          |
+-----+-----+-----+-----+
| id    | int(11)         | NO   | PRI | NULL    | auto_increment |
| fruit | enum('apple','orange','pear') | YES  |     | NULL    |                |
| bushels | int(11)        | YES  |     | NULL    |                |
+-----+-----+-----+-----+

INSERT INTO fruits
(fruit,bushels) VALUES
('pear',20),
('apple',100),
('orange',25);

INSERT INTO fruits
(fruit,bushels) VALUES
('avocado',10);
ERROR 1265 (01000): Data truncated for column 'fruit' at row 1

SELECT * FROM fruits;
+-----+-----+
| id | fruit | bushels |
+-----+-----+
| 1  | pear  |      20 |
| 2  | apple  |     100 |
| 3  | orange |      25 |
+-----+-----+

```

Selecting by numeric index:

```

SELECT * FROM fruits WHERE fruit=2;
+-----+-----+
| id | fruit | bushels |
+-----+-----+
| 3  | orange |      25 |
+-----+-----+

```

Sorting is according to the index value:

```

CREATE TABLE enums (a ENUM('2','1'));

INSERT INTO enums VALUES ('1'),('2');

SELECT * FROM enums ORDER BY a ASC;
+-----+
| a   |
+-----+
| 2   |
| 1   |
+-----+

```

It's easy to get confused between returning the enum integer with the stored value, so we don't suggest using ENUM to store numerals. The first example returns the 1st indexed field ('2' has an index value of 1, as it's defined first), while the second example returns the string value '1'.

```
SELECT * FROM enums WHERE a=1;
+-----+
| a    |
+-----+
| 2    |
+-----+

SELECT * FROM enums WHERE a='1';
+-----+
| a    |
+-----+
| 1    |
+-----+
```

See Also

- [Data Type Storage Requirements](#)

ST_LineStringFromWKB

A synonym for [ST_LineFromWKB](#).

ST_PointFromWKB

Syntax

```
ST_PointFromWKB(wkb[,srid])
PointFromWKB(wkb[,srid])
```

Description

Constructs a [POINT](#) value using its [WKB](#) representation and [SRID](#).

`ST_PointFromWKB()` and `PointFromWKB()` are synonyms.

Examples

```
SET @g = ST_AsBinary(ST_PointFromText('POINT(0 4)'));

SELECT ST_AsText(ST_PointFromWKB(@g)) AS p;
+-----+
| p      |
+-----+
| POINT(0 4) |
+-----+
```

ST_PolyFromWKB

Syntax

```
ST_PolyFromWKB(wkb[,srid])
ST_PolygonFromWKB(wkb[,srid])
PolyFromWKB(wkb[,srid])
PolygonFromWKB(wkb[,srid])
```

Description

Constructs a POLYGON value using its WKB representation and SRID.

ST_PolyFromWKB() , ST_PolygonFromWKB() , PolyFromWKB() and PolygonFromWKB() are synonyms.

Examples

```
SET @g = ST_AsBinary(ST_PolyFromText('POLYGON((1 1,1 5,4 9,6 9,9 3,7 2,1 1))'));

SELECT ST_AsText(ST_PolyFromWKB(@g)) AS p;
+-----+
| p           |
+-----+
| POLYGON((1 1,1 5,4 9,6 9,9 3,7 2,1 1)) |
+-----+
```

ST_PolygonFromWKB

A synonym for [ST_PolyFromWKB](#).

DOUBLE

Syntax

```
DOUBLE[(M,D)] [SIGNED | UNSIGNED | ZEROFILL]
DOUBLE PRECISION[(M,D)] [SIGNED | UNSIGNED | ZEROFILL]
REAL[(M,D)] [SIGNED | UNSIGNED | ZEROFILL]
```

Description

A normal-size (double-precision) floating-point number (see [FLOAT](#) for a single-precision floating-point number).

Allowable values are:

- -1.7976931348623157E+308 to -2.2250738585072014E-308
- 0
- 2.2250738585072014E-308 to 1.7976931348623157E+308

These are the theoretical limits, based on the IEEE standard. The actual range might be slightly smaller depending on your hardware or operating system.

M is the total number of digits and D is the number of digits following the decimal point. If M and D are omitted, values are stored to the limits allowed by the hardware. A double-precision floating-point number is accurate to approximately 15 decimal places.

UNSIGNED , if specified, disallows negative values.

ZEROFILL , if specified, pads the number with zeros, up to the total number of digits specified by M .

REAL and DOUBLE PRECISION are synonyms, unless the REAL_AS_FLOAT SQL mode is enabled, in which case REAL is a synonym for [FLOAT](#) rather than DOUBLE.

See [Floating Point Accuracy](#) for issues when using floating-point numbers.

For more details on the attributes, see [Numeric Data Type Overview](#).

Contents

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

Examples

```

CREATE TABLE t1 (d DOUBLE(5,0) zerofill);

INSERT INTO t1 VALUES (1),(2),(3),(4);

SELECT * FROM t1;
+---+
| d   |
+---+
| 00001 |
| 00002 |
| 00003 |
| 00004 |
+---+

```

FLOAT

Syntax

```
FLOAT[(M,D)] [SIGNED | UNSIGNED | ZEROFILL]
```

Description

A small (single-precision) floating-point number (see [DOUBLE](#) for a regular-size floating point number). Allowable values are:

- -3.402823466E+38 to -1.175494351E-38
- 0
- 1.175494351E-38 to 3.402823466E+38.

These are the theoretical limits, based on the IEEE standard. The actual range might be slightly smaller depending on your hardware or operating system.

M is the total number of digits and D is the number of digits following the decimal point. If M and D are omitted, values are stored to the limits allowed by the hardware. A single-precision floating-point number is accurate to approximately 7 decimal places.

[UNSIGNED](#), if specified, disallows negative values.

Using FLOAT might give you some unexpected problems because all calculations in MariaDB are done with double precision. See [Floating Point Accuracy](#).

For more details on the attributes, see [Numeric Data Type Overview](#).

JSON Data Type

MariaDB starting with [10.2.7](#)

The JSON alias was added in [MariaDB 10.2.7](#). This was done to make it possible to use JSON columns in [statement based replication](#) from MySQL to MariaDB and to make it possible for MariaDB to read [mysqldumps](#) from MySQL.

JSON is an alias for [LONGTEXT](#) introduced for compatibility reasons with MySQL's JSON data type. MariaDB implements this as a LONGTEXT rather, as the JSON data type contradicts the SQL standard, and MariaDB's benchmarks indicate that performance is at least equivalent.

In order to ensure that a valid json document is inserted, the [JSON_VALID](#) function can be used as a [CHECK constraint](#). This constraint is automatically included for types using the JSON alias from [MariaDB 10.4.3](#).

Examples

Contents

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

Contents

1. [Examples](#)
2. [Replicating JSON Data Between MySQL and MariaDB](#)
3. [Converting a MySQL TABLE with JSON Fields to MariaDB](#)
4. [Differences Between MySQL JSON Strings and MariaDB JSON Strings](#)
5. [See Also](#)

```

CREATE TABLE t (j JSON);

DESC t;
+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+
| j     | longtext | YES  |      | NULL    |       |
+-----+-----+-----+-----+

```

With validation:

```

CREATE TABLE t2 (
  j JSON
  CHECK (JSON_VALID(j))
);

INSERT INTO t2 VALUES ('invalid');
ERROR 4025 (23000): CONSTRAINT `j` failed for `test`.`t2`

INSERT INTO t2 VALUES ('{"id": 1, "name": "Monty"}');
Query OK, 1 row affected (0.13 sec)

```

Replicating JSON Data Between MySQL and MariaDB

The JSON type in MySQL stores the JSON object in a compact form, not as [LONGTEXT](#) as in MariaDB. This means that row based replication will not work for JSON types from MySQL to MariaDB.

There are a few different ways to solve this:

- Use statement based replication.
- Change the JSON column to type TEXT in MySQL
- If you must use row-based replication and cannot change the MySQL master from JSON to TEXT, you can try to introduce an intermediate MySQL slave and change the column type from JSON to TEXT on it. Then you replicate from this intermediate slave to MariaDB.

Converting a MySQL TABLE with JSON Fields to MariaDB

MariaDB can't directly access MySQL's JSON format.

There are a few different ways to move the table to MariaDB:

- Change the JSON column to type TEXT in MySQL. After this, MariaDB can directly use the table without any need for a dump and restore.
- [Use mariadb-dump/mysqldump to copy the table.](#)

Differences Between MySQL JSON Strings and MariaDB JSON Strings

- In MySQL, JSON is an object and is [compared according to json values](#). In MariaDB JSON strings are normal strings and compared as strings. One exception is when using [JSON_EXTRACT\(\)](#) in which case strings are unescaped before comparison.

See Also

- [JSON Functions](#)
- [CONNECT JSON Table Type](#)
- [MDEV-9144](#)

BOUNDARY

A synonym for [ST_BOUNDARY](#).

DIMENSION

A synonym for [ST_DIMENSION](#).

ENVELOPE

A synonym for [ST_ENVELOPE](#).

GeometryN

A synonym for [ST_GeometryN](#).

GeometryType

A synonym for [ST_GeometryType](#).

IsClosed

A synonym for [ST_IsClosed](#).

IsEmpty

A synonym for [ST_IsEmpty](#).

IsRing

A synonym for [ST_IsRing](#).

IsSimple

A synonym for [ST_IsSimple](#).

NumGeometries

A synonym for [ST_NumGeometries](#).

SRID

A synonym for [ST_SRID](#).

ST_DIMENSION

Syntax

```
ST_Dimension(g)  
Dimension(g)
```

Description

Returns the inherent dimension of the geometry value g . The result can be

Dimension	Definition
-1	empty geometry
0	geometry with no length or area
1	geometry with no area but nonzero length
2	geometry with nonzero area

`ST_Dimension()` and `Dimension()` are synonyms.

Examples

```

SELECT Dimension(GeomFromText('LineString(1 1,2 2)'));
+-----+
| Dimension(GeomFromText('LineString(1 1,2 2)')) |
+-----+
| 1 |
+-----+

```

Geometry Types

Description

MariaDB provides a standard way of creating spatial columns for geometry types, for example, with `CREATE TABLE` or `ALTER TABLE`. Currently, spatial columns are supported for MyISAM, InnoDB, NDB, and ARCHIVE tables. See also [SPATIAL INDEX](#).

The basic geometry type is `GEOMETRY`. But the type can be more specific. The following types are supported:

Geometry Types
POINT
LINESTRING
POLYGON
MULTIPOINT
MULTILINESTRING
MULTIPOLYGON
GEOMETRYCOLLECTION
GEOMETRY

Contents

1. [Description](#)
2. [Examples](#)
 1. [POINT](#)
 2. [LINESTRING](#)
 3. [POLYGON](#)
 4. [MULTIPOINT](#)
 5. [MULTILINESTRING](#)
 6. [MULTIPOLYGON](#)
 7. [GEOMETRYCOLLECTION](#)
 8. [GEOMETRY](#)

Examples

Note: For clarity, only one type is listed per table in the examples below, but a table row can contain multiple types. For example:

```
CREATE TABLE object (shapeA POLYGON, shapeB LINESTRING);
```

POINT

```

CREATE TABLE gis_point (g POINT);
SHOW FIELDS FROM gis_point;
INSERT INTO gis_point VALUES
    (PointFromText('POINT(10 10)'), 
     (PointFromText('POINT(20 10)'), 
      (PointFromText('POINT(20 20)'), 
       (PointFromWKB(AsWKB(PointFromText('POINT(10 20))))));

```

LINESTRING

```

CREATE TABLE gis_line (g LINESTRING);
SHOW FIELDS FROM gis_line;
INSERT INTO gis_line VALUES
    (LineFromText('LINESTRING(0 0,0 10,10 0)'), 
     (LineStringFromText('LINESTRING(10 10,20 10,20 20,10 20,10 10)'), 
      (LineStringFromWKB(AsWKB(LineString(Point(10, 10), Point(40, 10))))));

```

POLYGON

```
CREATE TABLE gis_polygon (g POLYGON);
SHOW FIELDS FROM gis_polygon;
INSERT INTO gis_polygon VALUES
    (PolygonFromText('POLYGON((10 10,20 10,20 20,10 20,10 10))')),
    (PolyFromText('POLYGON((0 0,50 0,50 50,0 50,0 0), (10 10,20 10,20 20,10 20,10 10))')),
    (PolyFromWKB(AsWKB(Polygon(LineString(Point(0, 0), Point(30, 0), Point(30, 30), Point(0, 0))))));
```

MULTIPOINT

```
CREATE TABLE gis_multi_point (g MULTIPOINT);
SHOW FIELDS FROM gis_multi_point;
INSERT INTO gis_multi_point VALUES
    (MultiPointFromText('MULTIPOINT(0 0,10 10,10 20,20 20)')),
    (MPointFromText('MULTIPOINT(1 1,11 11,11 21,21 21)')),
    (MPointFromWKB(AsWKB(MultiPoint(Point(3, 6), Point(4, 10)))));
```

MULTILINESTRING

```
CREATE TABLE gis_multi_line (g MULTILINESTRING);
SHOW FIELDS FROM gis_multi_line;
INSERT INTO gis_multi_line VALUES
    (MultiLineStringFromText('MULTILINESTRING((10 48,10 21,10 0),(16 0,16 23,16 48))'),
    (MLineFromText('MULTILINESTRING((10 48,10 21,10 0))')),
    (MLineFromWKB(AsWKB(MultiLineString(LineString(Point(1, 2), Point(3, 5)), LineString(Point(2, 5), Point(5, 8), Point(21, 7))))));
```

MULTIPOLYGON

```
CREATE TABLE gis_multi_polygon (g MULTIPOLYGON);
SHOW FIELDS FROM gis_multi_polygon;
INSERT INTO gis_multi_polygon VALUES
    (MultiPolygonFromText('MULTIPOLYGON(((28 26,28 0,84 0,84 42,28 26),(52 18,66 23,73 9,48 6,52 18),((59 18,67 18,67 13,59 13,59 18),(59 18,67 18,67 13,59 13,59 18))),((59 18,67 18,67 13,59 13,59 18),(59 18,67 18,67 13,59 13,59 18)))'),
    (MPolyFromText('MULTIPOLYGON(((28 26,28 0,84 0,84 42,28 26),(52 18,66 23,73 9,48 6,52 18),((59 18,67 18,67 13,59 13,59 18))),((59 18,67 18,67 13,59 13,59 18)))'),
    (MPolyFromWKB(AsWKB(MultiPolygon(Polygon(LineString(Point(0, 3), Point(3, 3), Point(3, 0), Point(0, 3)))))));
```

GEOMETRYCOLLECTION

```
CREATE TABLE gis_geometrycollection (g GEOMETRYCOLLECTION);
SHOW FIELDS FROM gis_geometrycollection;
INSERT INTO gis_geometrycollection VALUES
    (GeomCollFromText('GEOMETRYCOLLECTION(POINT(0 0), LINESTRING(0 0,10 10))'),
    (GeometryFromWKB(AsWKB(GeometryCollection(Point(44, 6), LineString(Point(3, 6), Point(7, 9)))))),
    (GeomFromText('GeometryCollection()')),
    (GeomFromText('GeometryCollection EMPTY')));
```

GEOMETRY

```
CREATE TABLE gis_geometry (g GEOMETRY);
SHOW FIELDS FROM gis_geometry;
INSERT INTO gis_geometry SELECT * FROM gis_point;
INSERT INTO gis_geometry SELECT * FROM gis_line;
INSERT INTO gis_geometry SELECT * FROM gis_polygon;
INSERT INTO gis_geometry SELECT * FROM gis_multi_point;
INSERT INTO gis_geometry SELECT * FROM gis_multi_line;
INSERT INTO gis_geometry SELECT * FROM gis_multi_polygon;
INSERT INTO gis_geometry SELECT * FROM gis_geometrycollection;
```

LONGTEXT

Syntax

```
LONGTEXT [CHARACTER SET charset_name] [COLLATE collation_name]
```

Description

A [TEXT](#) column with a maximum length of 4,294,967,295 or 4GB ($2^{32} - 1$) characters. The effective maximum length is less if the value contains multi-byte characters. The effective maximum length of LONGTEXT columns also depends on the configured maximum packet size in the client/server protocol and available memory. Each LONGTEXT value is stored using a four-byte length prefix that indicates the number of bytes in the value.

From [MariaDB 10.2.7](#), JSON is an alias for LONGTEXT. See [JSON Data Type](#) for details.

Contents

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

Oracle Mode

MariaDB starting with [10.3](#)

In [Oracle mode from MariaDB 10.3](#), CLOB is a synonym for LONGTEXT .

See Also

- [TEXT](#)
- [BLOB and TEXT Data Types](#)
- [Data Type Storage Requirements](#)
- [JSON Data Type](#)
- [Oracle mode from MariaDB 10.3](#)

MEDIUMINT

Syntax

```
MEDIUMINT[(M)] [SIGNED | UNSIGNED | ZEROFILL]
```

Description

A medium-sized integer. The signed range is -8388608 to 8388607. The unsigned range is 0 to 16777215.

ZEROFILL pads the integer with zeroes and assumes UNSIGNED (even if UNSIGNED is not specified).

[INT3](#) is a synonym for MEDIUMINT .

For details on the attributes, see [Numeric Data Type Overview](#).

Examples

```

CREATE TABLE mediumints (a MEDIUMINT,b MEDIUMINT UNSIGNED,c MEDIUMINT ZEROFILL);

DESCRIBE mediumints;
+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+
| a     | mediumint(9) | YES  |     | NULL    |       |
| b     | mediumint(8) unsigned | YES  |     | NULL    |       |
| c     | mediumint(8) unsigned zerofill | YES  |     | NULL    |       |
+-----+-----+-----+-----+

```

With `strict_mode` set, the default from MariaDB 10.2.4:

```

INSERT INTO mediumints VALUES (-10,-10,-10);
ERROR 1264 (22003): Out of range value for column 'b' at row 1

INSERT INTO mediumints VALUES (-10,10,-10);
ERROR 1264 (22003): Out of range value for column 'c' at row 1

INSERT INTO mediumints VALUES (-10,10,10);

INSERT INTO mediumints VALUES (8388608,8388608,8388608);
ERROR 1264 (22003): Out of range value for column 'a' at row 1

INSERT INTO mediumints VALUES (8388607,8388608,8388608);

SELECT * FROM mediumints;
+-----+-----+-----+
| a     | b     | c     |
+-----+-----+-----+
| -10   | 10   | 00000010 |
| 8388607 | 8388608 | 08388608 |
+-----+-----+-----+

```

With `strict_mode` unset, the default until MariaDB 10.2.3:

```

INSERT INTO mediumints VALUES (-10,-10,-10);
Query OK, 1 row affected, 2 warnings (0.05 sec)
Warning (Code 1264): Out of range value for column 'b' at row 1
Warning (Code 1264): Out of range value for column 'c' at row 1

INSERT INTO mediumints VALUES (-10,10,-10);
Query OK, 1 row affected, 1 warning (0.08 sec)
Warning (Code 1264): Out of range value for column 'c' at row 1

INSERT INTO mediumints VALUES (-10,10,10);

INSERT INTO mediumints VALUES (8388608,8388608,8388608);
Query OK, 1 row affected, 1 warning (0.05 sec)
Warning (Code 1264): Out of range value for column 'a' at row 1

INSERT INTO mediumints VALUES (8388607,8388608,8388608);

SELECT * FROM mediumints;
+-----+-----+-----+
| a     | b     | c     |
+-----+-----+-----+
| -10   | 0   | 00000000 |
| -10   | 0   | 00000000 |
| -10   | 10  | 00000000 |
| -10   | 10  | 00000010 |
| 8388607 | 8388608 | 08388608 |
| 8388607 | 8388608 | 08388608 |
+-----+-----+-----+

```

See Also

- [Numeric Data Type Overview](#)
- [TINYINT](#)
- [SMALLINT](#)
- [INTEGER](#)
- [BIGINT](#)

ST_GEOOMETRYN

Syntax

```
ST_GeometryN(gc,N)
GeometryN(gc,N)
```

Description

Returns the N-th geometry in the GeometryCollection *gc*. Geometries are numbered beginning with 1.

`ST_GeometryN()` and `GeometryN()` are synonyms.

Example

```
SET @gc = 'GeometryCollection(Point(1 1),LineString(12 14, 9 11))';

SELECT AsText(GeometryN(GeomFromText(@gc),1));
+-----+
| AsText(GeometryN(GeomFromText(@gc),1)) |
+-----+
| POINT(1 1)                                |
+-----+
```

Numeric Data Type Overview

There are a number of numeric data types:

- [TINYINT](#)
- [BOOLEAN](#) - Synonym for TINYINT(1)
- [INT1](#) - Synonym for TINYINT
- [SMALLINT](#)
- [INT2](#) - Synonym for SMALLINT
- [MEDIUMINT](#)
- [INT3](#) - Synonym for MEDIUMINT
- [INT](#), [INTEGER](#)
- [INT4](#) - Synonym for INT
- [BIGINT](#)
- [INT8](#) - Synonym for BIGINT
- [DECIMAL](#), [DEC](#), [NUMERIC](#), [FIXED](#)
- [FLOAT](#)
- [DOUBLE](#), [DOUBLE PRECISION](#), [REAL](#)
- [BIT](#)

See the specific articles for detailed information on each.

SIGNED, UNSIGNED and ZEROFILL

Most numeric types can be defined as `SIGNED`, `UNSIGNED` or `ZEROFILL`, for example:

```
TINYINT[(M)] [SIGNED | UNSIGNED | ZEROFILL]
```

If `SIGNED`, or no attribute, is specified, a portion of the numeric type will be reserved for the sign (plus or minus). For example, a TINYINT SIGNED can range from -128 to 127.

Contents

1. [SIGNED, UNSIGNED and ZEROFILL](#)
 1. [Examples](#)
2. [Range](#)
 1. [Examples](#)
3. [Auto_increment](#)

If `UNSIGNED` is specified, no portion of the numeric type is reserved for the sign, so for integer types range can be larger. For example, a `TINYINT UNSIGNED` can range from 0 to 255. Floating point and fixed-point types also can be `UNSIGNED`, but this only prevents negative values from being stored and doesn't alter the range.

If `ZEROFILL` is specified, 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.

Note that although the preferred syntax indicates that the attributes are exclusive, more than one attribute can be specified.

Until [MariaDB 10.2.7 \(MDEV-8659\)](#), any combination of the attributes could be used in any order, with duplicates. In this case:

- the presence of `ZEROFILL` makes the column `UNSIGNED ZEROFILL`.
- the presence of `UNSIGNED` makes the column `UNSIGNED`.

From [MariaDB 10.2.8](#), only the following combinations are supported:

- `SIGNED`
- `UNSIGNED`
- `ZEROFILL`
- `UNSIGNED ZEROFILL`
- `ZEROFILL UNSIGNED`

The latter two should be replaced with simply `ZEROFILL`, but are still accepted by the parser.

Examples

```
CREATE TABLE zf (
    i1 TINYINT SIGNED,
    i2 TINYINT UNSIGNED,
    i3 TINYINT ZEROFILL
);

INSERT INTO zf VALUES (2,2,2);

SELECT * FROM zf;
+---+---+---+
| i1 | i2 | i3 |
+---+---+---+
|   2 |    2 |  002 |
+---+---+---+
```

Range

When attempting to add a value that is out of the valid range for the numeric type, MariaDB will react depending on the `strict SQL_MODE` setting.

If `strict_mode` has been set (the default from [MariaDB 10.2.4](#)), MariaDB will return an error.

If `strict_mode` has not been set (the default until [MariaDB 10.2.3](#)), MariaDB will adjust the number to fit in the field, returning a warning.

Examples

With `strict_mode` set:

```
SHOW VARIABLES LIKE 'sql_mode';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| sql_mode      | STRICT_TRANS_TABLES,ERROR_FOR_DIVISION_BY_ZERO,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION |
+-----+-----+

CREATE TABLE ranges (i1 TINYINT, i2 SMALLINT, i3 TINYINT UNSIGNED);

INSERT INTO ranges VALUES (257,257,257);
ERROR 1264 (22003): Out of range value for column 'i1' at row 1

SELECT * FROM ranges;
Empty set (0.10 sec)
```

With strict_mode unset:

```
SHOW VARIABLES LIKE 'sql_mode%';
+-----+
| Variable_name | Value |
+-----+
| sql_mode      |       |
+-----+

CREATE TABLE ranges (i1 TINYINT, i2 SMALLINT, i3 TINYINT UNSIGNED);

INSERT INTO ranges VALUES (257,257,257);
Query OK, 1 row affected, 2 warnings (0.00 sec)

SHOW WARNINGS;
+-----+
| Level | Code | Message |
+-----+
| Warning | 1264 | Out of range value for column 'i1' at row 1 |
| Warning | 1264 | Out of range value for column 'i3' at row 1 |
+-----+
2 rows in set (0.00 sec)

SELECT * FROM ranges;
+---+---+---+
| i1 | i2 | i3 |
+---+---+---+
| 127 | 257 | 255 |
+---+---+---+
```

Auto_increment

The `AUTO_INCREMENT` attribute can be used to generate a unique identity for new rows. For more details, see [auto_increment](#).

SMALLINT

Syntax

```
SMALLINT[(M)] [SIGNED | UNSIGNED | ZEROFILL]
```

Description

A small `integer`. The signed range is -32768 to 32767. The unsigned range is 0 to 65535.

If a column has been set to `ZEROFILL`, all values will be prepended by zeros so that the `SMALLINT` value contains a number of M digits.

Note: If the `ZEROFILL` attribute has been specified, the column will automatically become `UNSIGNED`.

`INT2` is a synonym for `SMALLINT`.

For more details on the attributes, see [Numeric Data Type Overview](#).

Examples

```
CREATE TABLE smallints (a SMALLINT,b SMALLINT UNSIGNED,c SMALLINT ZEROFILL);
```

With `strict_mode` set, the default from [MariaDB 10.2.4](#):

```

INSERT INTO smallints VALUES (-10,-10,-10);
ERROR 1264 (22003): Out of range value for column 'b' at row 1

INSERT INTO smallints VALUES (-10,10,-10);
ERROR 1264 (22003): Out of range value for column 'c' at row 1

INSERT INTO smallints VALUES (-10,10,10);

INSERT INTO smallints VALUES (32768,32768,32768);
ERROR 1264 (22003): Out of range value for column 'a' at row 1

INSERT INTO smallints VALUES (32767,32768,32768);

SELECT * FROM smallints;
+-----+-----+-----+
| a    | b    | c    |
+-----+-----+-----+
| -10  | 10  | 00010 |
| 32767 | 32768 | 32768 |
+-----+-----+-----+

```

With `strict_mode` unset, the default until MariaDB 10.2.3:

```

INSERT INTO smallints VALUES (-10,-10,-10);
Query OK, 1 row affected, 2 warnings (0.09 sec)
Warning (Code 1264): Out of range value for column 'b' at row 1
Warning (Code 1264): Out of range value for column 'c' at row 1

INSERT INTO smallints VALUES (-10,10,-10);
Query OK, 1 row affected, 1 warning (0.08 sec)
Warning (Code 1264): Out of range value for column 'c' at row 1

INSERT INTO smallints VALUES (-10,10,10);

INSERT INTO smallints VALUES (32768,32768,32768);
Query OK, 1 row affected, 1 warning (0.04 sec)
Warning (Code 1264): Out of range value for column 'a' at row 1

INSERT INTO smallints VALUES (32767,32768,32768);

SELECT * FROM smallints;
+-----+-----+-----+
| a    | b    | c    |
+-----+-----+-----+
| -10  | 0   | 00000 |
| -10  | 10  | 00000 |
| -10  | 10  | 00010 |
| 32767 | 32768 | 32768 |
| 32767 | 32768 | 32768 |
+-----+-----+-----+

```

See Also

- [Numeric Data Type Overview](#)
- [TINYINT](#)
- [MEDIUMINT](#)
- [INTEGER](#)
- [BIGINT](#)

ST_GEOOMETRYTYPE

Syntax

```
ST_GeometryType(g)
GeometryType(g)
```

Description

Returns as a string the name of the geometry type of which the geometry instance `g` is a member. The name corresponds to one of the instantiable Geometry subclasses.

`ST_GeometryType()` and `GeometryType()` are synonyms.

Examples

```
SELECT GeometryType(GeomFromText('POINT(1 1)'));
+-----+
| GeometryType(GeomFromText('POINT(1 1)')) |
+-----+
| POINT                                |
+-----+
```

ST_IsEmpty

Syntax

```
ST_IsEmpty(g)
IsEmpty(g)
```

Description

`IsEmpty` is a function defined by the OpenGIS specification, but is not fully implemented by MariaDB or MySQL.

Since MariaDB and MySQL do not support GIS EMPTY values such as POINT EMPTY, as implemented it simply returns `1` if the geometry value `g` is invalid, `0` if it is valid, and `NULL` if the argument is `NULL`.

`ST_IsEmpty()` and `IsEmpty()` are synonyms.

ST_IsRing

MariaDB starting with 10.1.2

The ST_IsRing function was introduced in [MariaDB 10.1.2](#).

Syntax

```
ST_IsRing(g)
IsRing(g)
```

Description

Returns true if a given `LINESTRING` is a ring, that is, both `ST_IsClosed` and `ST_IsSimple`. A simple curve does not pass through the same point more than once. However, see [MDEV-7510](#).

`St_IsRing()` and `IsRing()` are synonyms.

String Literals

Strings are sequences of characters and are enclosed with quotes.

The syntax is:

```
[_charset_name]'string' [COLLATE collation_name]
```

For example:

```
'The MariaDB Foundation'  
_utf8 'Foundation' COLLATE utf8_unicode_ci;
```

Strings can either be enclosed in single quotes or in double quotes (the same character must be used to both open and close the string).

The ANSI SQL-standard does not permit double quotes for enclosing strings, and although MariaDB does by default, if the MariaDB server has enabled the [ANSI_QUOTES_SQL_MODE](#), double quotes will be treated as being used for [identifiers](#) instead of strings.

Strings that are next to each other are automatically concatenated. For example:

```
'The ' 'MariaDB ' 'Foundation'
```

and

```
'The MariaDB Foundation'
```

are equivalent.

The \ (backslash character) is used to escape characters (unless the [SQL_MODE](#) hasn't been set to [NO_BACKSLASH_ESCAPES](#)). For example:

```
'MariaDB's new features'
```

is not a valid string because of the single quote in the middle of the string, which is treated as if it closes the string, but is actually meant as part of the string, an apostrophe. The backslash character helps in situations like this:

```
'MariaDB\'s new features'
```

is now a valid string, and if displayed, will appear without the backslash.

```
SELECT 'MariaDB\'s new features';  
+-----+  
| MariaDB's new features |  
+-----+  
| MariaDB's new features |  
+-----+
```

Another way to escape the quoting character is repeating it twice:

```
SELECT 'I''m here', """Double""";  
+-----+-----+  
| I'm here | "Double" |  
+-----+-----+  
| I'm here | "Double" |  
+-----+-----+
```

Escape Sequences

There are other escape sequences also. Here is a full list:

Escape sequence	Character
\0	ASCII NUL (0x00).
\'	Single quote ("").
\"	Double quote ("").
\b	Backspace.

\n	Newline, or linefeed,.
\r	Carriage return.
\t	Tab.
\z	ASCII 26 (Control+Z). See note following the table.
\\	Backslash ("\\").
\%	"%" character. See note following the table.
_	A "_" character. See note following the table.

Escaping the % and _ characters can be necessary when using the [LIKE](#) operator, which treats them as special characters.

The ASCII 26 character (\z) needs to be escaped when included in a batch file which needs to be executed in Windows. The reason is that ASCII 26, in Windows, is the end of file (EOF).

Backslash (\), if not used as an escape character, must always be escaped. When followed by a character that is not in the above table, backslashes will simply be ignored.

TEXT

Syntax

```
TEXT[(M)] [CHARACTER SET charset_name] [COLLATE collation_name]
```

Description

A TEXT column with a maximum length of 65,535 ($2^{16} - 1$) characters. The effective maximum length is less if the value contains multi-byte characters. Each TEXT value is stored using a two-byte length prefix that indicates the number of bytes in the value. If you need a bigger storage, consider using [MEDIUMTEXT](#) instead.

An optional length M can be given for this type. If this is done, MariaDB creates the column as the smallest TEXT type large enough to hold values M characters long.

Before [MariaDB 10.2](#), all MariaDB collations were of type PADSPACE, meaning that TEXT (as well as [VARCHAR](#) and [CHAR](#) values) are compared without regard for trailing spaces. This does not apply to the [LIKE](#) pattern-matching operator, which takes into account trailing spaces.

Before [MariaDB 10.2.1](#), BLOB and TEXT columns could not be assigned a [DEFAULT](#) value. This restriction was lifted in [MariaDB 10.2.1](#).

Examples

Trailing spaces:

```
CREATE TABLE strtest (d TEXT(10));
INSERT INTO strtest VALUES('Maria   ');

SELECT d='Maria',d='Maria   ' FROM strtest;
+-----+-----+
| d='Maria' | d='Maria   ' |
+-----+-----+
|      1 |          1 |
+-----+-----+

SELECT d LIKE 'Maria',d LIKE 'Maria   ' FROM strtest;
+-----+-----+
| d LIKE 'Maria' | d LIKE 'Maria   ' |
+-----+-----+
|          0 |          1 |
+-----+-----+
```

Contents

1. [Syntax](#)
2. [Description](#)
3. [Examples](#)
4. [Indexing](#)
5. [Difference between VARCHAR and TEXT](#)
 1. [For Storage Engine Developers](#)
6. [See Also](#)

Indexing

`TEXT` columns can only be indexed over a specified length. This means that they cannot be used as the [primary key](#) of a table norm until [MariaDB 10.4](#), can a [unique index](#) be created on them.

MariaDB starting with [10.4](#)

Starting with [MariaDB 10.4](#), a unique index can be created on a `TEXT` column.

Internally, this uses hash indexing to quickly check the values and if a hash collision is found, the actual stored values are compared in order to retain the uniqueness.

Difference between `VARCHAR` and `TEXT`

- `VARCHAR` columns can be fully indexed. `TEXT` columns can only be indexed over a specified length.
- Using `TEXT` or `BLOB` in a `SELECT` query that uses temporary tables for storing intermediate results will force the temporary table to be disk based (using the [Aria storage engine](#) instead of the [memory storage engine](#), which is a bit slower. This is not that bad as the [Aria storage engine](#) caches the rows in memory. To get the benefit of this, one should ensure that the `aria_pagecache_buffer_size` variable is big enough to hold most of the row and index data for temporary tables.

For Storage Engine Developers

- Internally the full length of the `VARCHAR` column is allocated inside each TABLE objects record[] structure. As there are three such buffers, each open table will allocate 3 times max-length-to-store-varchar bytes of memory.
- `TEXT` and `BLOB` columns are stored with a pointer (4 or 8 bytes) + a 1-4 bytes length. The `TEXT` data is only stored once. This means that internally `TEXT` uses less memory for each open table but instead has the additional overhead that each `TEXT` object needs to be allocated and freed for each row access (with some caching in between).

See Also

- [BLOB and TEXT Data Types](#)
- [MEDIUMTEXT](#)
- [Data Type Storage Requirements](#)

ST_IsSimple

Syntax

```
ST_IsSimple(g)
IsSimple(g)
```

Description

Returns true if the given Geometry has no anomalous geometric points, false if it does, or NULL if given a NULL value.

`ST_IsSimple()` and `IsSimple()` are synonyms.

Examples

A `POINT` is always simple.

```
SET @g = 'Point(1 2)';
SELECT ST_ISSIMPLE(GEOMFROMTEXT(@g));
+-----+
| ST_ISSIMPLE(GEOMFROMTEXT(@g)) |
+-----+
| 1 |
+-----+
```

ST_NUMGEOMETRIES

Syntax

```
ST_NumGeometries(gc)
NumGeometries(gc)
```

Description

Returns the number of geometries in the GeometryCollection `gc`.

`ST_NumGeometries()` and `NumGeometries()` are synonyms.

Example

```
SET @gc = 'GeometryCollection(Point(1 1),LineString(2 2, 3 3))';

SELECT NUMGEOMETRIES(GeomFromText(@gc));
+-----+
| NUMGEOMETRIES(GeomFromText(@gc)) |
+-----+
|                               2 |
+-----+
```

ST_RELATE

MariaDB starting with 10.1.2

The `ST_RELATE()` function was introduced in [MariaDB 10.1.2](#)

Syntax

```
ST_Relate(g1, g2, i)
```

Description

Returns true if Geometry `g1` is spatially related to Geometry `g2` by testing for intersections between the interior, boundary and exterior of the two geometries as specified by the values in intersection matrix pattern `i`.

ST_SRID

Syntax

```
ST_SRID(g)
SRID(g)
```

Description

Returns an integer indicating the Spatial Reference System ID for the geometry value `g`.

In MariaDB, the SRID value is just an integer associated with the geometry value. All calculations are done assuming Euclidean (planar) geometry.

`ST_SRID()` and `SRID()` are synonyms.

Examples

```
SELECT SRID(GeomFromText('LineString(1 1,2 2)',101));
+-----+
| SRID(GeomFromText('LineString(1 1,2 2)',101)) |
+-----+
|          101 |
+-----+
```

TIME

Syntax

```
TIME [(<microsecond precision>)]
```

Description

A time. The range is '-838:59:59.999999' to '838:59:59.999999'. Microsecond precision can be from 0-6; if not specified 0 is used. Microseconds have been available since [MariaDB 5.3](#).

MariaDB displays `TIME` values in '`HH:MM:SS.ssssss`' format, but allows assignment of times in looser formats, including '`D HH:MM:SS`', '`HH:MM:SS`', '`HH:MM`', '`D HH:MM`', '`D HH`', '`SS`', or '`HHMMSS`', as well as permitting dropping of any leading zeros when a delimiter is provided, for example '`3:9:10`'. For details, see [date and time literals](#).

MariaDB starting with [10.1.2](#)

[MariaDB 10.1.2](#) introduced the `--mysql56-temporal-format` option, on by default, which allows MariaDB to store `TIME`s using the same low-level format MySQL 5.6 uses.

Contents

1. [Syntax](#)
2. [Description](#)
 1. [Internal Format](#)
3. [Examples](#)
4. [See also](#)

Internal Format

In [MariaDB 10.1.2](#) a new temporal format was introduced from MySQL 5.6 that alters how the `TIME`, `DATETIME` and `TIMESTAMP` columns operate at lower levels. These changes allow these temporal data types to have fractional parts and negative values. You can disable this feature using the `mysql56_temporal_format` system variable.

Tables that include `TIMESTAMP` values that were created on an older version of MariaDB or that were created while the `mysql56_temporal_format` system variable was disabled continue to store data using the older data type format.

In order to update table columns from the older format to the newer format, execute an `ALTER TABLE... MODIFY COLUMN` statement that changes the column to the *same* data type. This change may be needed if you want to export the table's tablespace and import it onto a server that has `mysql56_temporal_format=ON` set (see [MDEV-15225](#)).

For instance, if you have a `TIME` column in your table:

```
SHOW VARIABLES LIKE 'mysql56_temporal_format';

+-----+-----+
| Variable_name      | Value   |
+-----+-----+
| mysql56_temporal_format | ON     |
+-----+-----+

ALTER TABLE example_table MODIFY ts_col TIME;
```

When MariaDB executes the `ALTER TABLE` statement, it converts the data from the older temporal format to the newer one.

In the event that you have several tables and columns using temporal data types that you want to switch over to the new format, make sure the system variable is enabled, then perform a dump and restore using `mysqldump`. The columns using relevant temporal data types are restored using the new temporal format.

Starting from [MariaDB 10.5.1](#) columns with old temporal formats are marked with a `/* mariadb-5.3 */` comment in the output of `SHOW CREATE TABLE`, `SHOW COLUMNS`, `DESCRIBE` statements, as well as in the `COLUMN_TYPE` column of the `INFORMATION_SCHEMA.COLUMNS` Table.

```

SHOW CREATE TABLE mariadb5312_time\G
***** 1. row *****
Table: mariadb5312_time
Create Table: CREATE TABLE `mariadb5312_time` (
  `t0` time /* mariadb-5.3 */ DEFAULT NULL,
  `t6` time(6) /* mariadb-5.3 */ DEFAULT NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1

```

Note, columns with the current format are not marked with a comment.

Examples

```

INSERT INTO time VALUES ('90:00:00'), ('800:00:00'), (800), (22), (151413), ('9:6:3'), ('12 09');

SELECT * FROM time;
+-----+
| t      |
+-----+
| 90:00:00 |
| 800:00:00 |
| 00:08:00 |
| 00:00:22 |
| 15:14:13 |
| 09:06:03 |
| 297:00:00 |
+-----+

```

See also

- [Data Type Storage Requirements](#)

TINYINT

Syntax

```
TINYINT[(M)] [SIGNED | UNSIGNED | ZEROFILL]
```

Description

A very small [integer](#). The signed range is -128 to 127. The unsigned range is 0 to 255. For details on the attributes, see [Numeric Data Type Overview](#).

`INT1` is a synonym for `TINYINT`. `BOOL` and `BOOLEAN` are synonyms for `TINYINT(1)`.

Contents

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

Examples

```
CREATE TABLE tinyints (a TINYINT,b TINYINT UNSIGNED,c TINYINT ZEROFILL);
```

With `strict_mode` set, the default from [MariaDB 10.2.4](#):

```

INSERT INTO tinyints VALUES (-10,-10,-10);
ERROR 1264 (22003): Out of range value for column 'b' at row 1

INSERT INTO tinyints VALUES (-10,10,-10);
ERROR 1264 (22003): Out of range value for column 'c' at row 1

INSERT INTO tinyints VALUES (-10,10,10);

SELECT * FROM tinyints;
+-----+-----+
| a    | b    | c    |
+-----+-----+
| -10  |  10  |  010 |
+-----+-----+

INSERT INTO tinyints VALUES (128,128,128);
ERROR 1264 (22003): Out of range value for column 'a' at row 1

INSERT INTO tinyints VALUES (127,128,128);

SELECT * FROM tinyints;
+-----+-----+
| a    | b    | c    |
+-----+-----+
| -10  |  10  |  010 |
| 127  | 128  | 128 |
+-----+-----+

```

With `strict_mode` unset, the default until MariaDB 10.2.3:

```

INSERT INTO tinyints VALUES (-10,-10,-10);
Query OK, 1 row affected, 2 warnings (0.08 sec)
Warning (Code 1264): Out of range value for column 'b' at row 1
Warning (Code 1264): Out of range value for column 'c' at row 1

INSERT INTO tinyints VALUES (-10,10,-10);
Query OK, 1 row affected, 1 warning (0.11 sec)
Warning (Code 1264): Out of range value for column 'c' at row 1

INSERT INTO tinyints VALUES (-10,10,10);

SELECT * FROM tinyints;
+-----+-----+
| a    | b    | c    |
+-----+-----+
| -10  |   0  |  000 |
| -10  |  10  |  000 |
| -10  |  10  |  010 |
+-----+-----+

INSERT INTO tinyints VALUES (128,128,128);
Query OK, 1 row affected, 1 warning (0.19 sec)
Warning (Code 1264): Out of range value for column 'a' at row 1

INSERT INTO tinyints VALUES (127,128,128);

SELECT * FROM tinyints;
+-----+-----+
| a    | b    | c    |
+-----+-----+
| -10  |   0  |  000 |
| -10  |  10  |  000 |
| -10  |  10  |  010 |
| 127  | 128  | 128 |
| 127  | 128  | 128 |
+-----+-----+

```

See Also

- Numeric Data Type Overview
- SMALLINT
- MEDIUMINT
- INTEGER
- BIGINT
- BOOLEAN

VARBINARY

Syntax

`VARBINARY(M)`

Description

The VARBINARY type is similar to the [VARCHAR](#) type, but stores binary byte strings rather than non-binary character strings. `M` represents the maximum column length in bytes.

It contains no [character set](#), and comparison and sorting are based on the numeric value of the bytes.

If the maximum length is exceeded, and [SQL strict mode](#) is not enabled , the extra characters will be dropped with a warning. If strict mode is enabled, an error will occur.

Unlike [BINARY](#) values, VARBINARYs are not right-padded when inserting.

Contents

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

Oracle Mode

MariaDB starting with [10.3](#)

In [Oracle mode](#) from MariaDB 10.3, `RAW` is a synonym for `VARBINARY` .

Examples

Inserting too many characters, first with strict mode off, then with it on:

```
CREATE TABLE varbins (a VARBINARY(10));

INSERT INTO varbins VALUES('12345678901');
Query OK, 1 row affected, 1 warning (0.04 sec)

SELECT * FROM varbins;
+-----+
| a    |
+-----+
| 1234567890 |
+-----+

SET sql_mode='STRICT_ALL_TABLES';

INSERT INTO varbins VALUES('12345678901');
ERROR 1406 (22001): Data too long for column 'a' at row 1
```

Sorting is performed with the byte value:

```
TRUNCATE varbins;

INSERT INTO varbins VALUES('A'),('B'),('a'),('b');

SELECT * FROM varbins ORDER BY a;
+---+
| a   |
+---+
| A   |
| B   |
| a   |
| b   |
+---+
```

Using `CAST` to sort as a `CHAR` instead:

```
SELECT * FROM varbins ORDER BY CAST(a AS CHAR);
+---+
| a   |
+---+
| a   |
| A   |
| b   |
| B   |
+---+
```

See Also

- [VARCHAR](#)
- [Data Type Storage Requirements](#)
- [Oracle mode from MariaDB 10.3](#)

ASCII

Syntax

```
ASCII(str)
```

Description

Returns the numeric ASCII value of the leftmost character of the string argument. Returns `0` if the given string is empty and `NULL` if it is `NULL`.

`ASCII()` works for 8-bit characters.

Examples

```
SELECT ASCII(9);
+-----+
| ASCII(9) |
+-----+
|      57   |
+-----+

SELECT ASCII('9');
+-----+
| ASCII('9') |
+-----+
|      57   |
+-----+

SELECT ASCII('abc');
+-----+
| ASCII('abc') |
+-----+
|        97    |
+-----+
```

BIN

Syntax

```
BIN(N)
```

Description

Returns a string representation of the binary value of the given longlong (that is, `BIGINT`) number. This is equivalent to `CONV(N,10,2)`. The argument should be positive. If it is a `FLOAT`, it will be truncated. Returns `NULL` if the argument is `NULL`.

Examples

```
SELECT BIN(12);
+-----+
| BIN(12) |
+-----+
| 1100    |
+-----+
```

See Also

- [Binary literals](#)
- [CONV\(\)](#)
- [OCT\(\)](#)
- [HEX\(\)](#)

BIT_LENGTH

Syntax

```
BIT_LENGTH(str)
```

Description

Returns the length of the given string argument in bits. If the argument is not a string, it will be converted to string. If the argument is `NULL`, it returns

NULL .

Examples

```
SELECT BIT_LENGTH('text');
+-----+
| BIT_LENGTH('text') |
+-----+
|          32       |
+-----+
```

Contents

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

```
SELECT BIT_LENGTH('');
+-----+
| BIT_LENGTH('') |
+-----+
|          0       |
+-----+
```

Compatibility

PostgreSQL and Sybase support BIT_LENGTH().

VARCHAR

Syntax

```
[NATIONAL] VARCHAR(M) [CHARACTER SET charset_name] [COLLATE collation_name]
```

Description

A variable-length string. M represents the maximum column length in characters. The range of M is 0 to 65,532. The effective maximum length of a VARCHAR is subject to the maximum row size and the character set used. For example, utf8 characters can require up to three bytes per character, so a VARCHAR column that uses the utf8 character set can be declared to be a maximum of 21,844 characters.

Note: For the [ColumnStore](#) engine, M represents the maximum column length in bytes.

MariaDB stores VARCHAR values as a one-byte or two-byte length prefix plus data. The length prefix indicates the number of bytes in the value. A VARCHAR column uses one length byte if values require no more than 255 bytes, two length bytes if values may require more than 255 bytes.

MariaDB follows the standard SQL specification, and does not remove trailing spaces from VARCHAR values.

VARCHAR(0) columns can contain 2 values: an empty string or NULL. Such columns cannot be part of an index. The [CONNECT](#) storage engine does not support VARCHAR(0).

VARCHAR is shorthand for CHARACTER VARYING. NATIONAL VARCHAR is the standard SQL way to define that a VARCHAR column should use some predefined character set. MariaDB uses utf8 as this predefined character set, as does MySQL 4.1 and up. NVARCHAR is shorthand for NATIONAL VARCHAR.

Before [MariaDB 10.2](#), all MariaDB [collations](#) were of type `PADSPACE`, meaning that VARCHAR (as well as CHAR and TEXT values) are compared without regard for trailing spaces. This does not apply to the [LIKE](#) pattern-matching operator, which takes into account trailing spaces. From [MariaDB 10.2](#), a number of [NO PAD](#) collations are available.

If a unique index consists of a column where trailing pad characters are stripped or ignored, inserts into that column where values differ only by the number of trailing pad characters will result in a duplicate-key error.

Examples

The following are equivalent:

Contents

1. [Syntax](#)
2. [Description](#)
3. [Examples](#)
4. [Truncation](#)
5. [Difference Between VARCHAR and TEXT](#)
6. [Oracle Mode](#)
 - 1. [For Storage Engine Developers](#)
7. [See Also](#)

```
VARCHAR(30) CHARACTER SET utf8
NATIONAL VARCHAR(30)
NVARCHAR(30)
NCHAR VARCHAR(30)
NATIONAL CHARACTER VARYING(30)
NATIONAL CHAR VARYING(30)
```

Trailing spaces:

```
CREATE TABLE strtest (v VARCHAR(10));
INSERT INTO strtest VALUES('Maria   ');

SELECT v='Maria',v='Maria   ' FROM strtest;
+-----+-----+
| v='Maria' | v='Maria   ' |
+-----+-----+
|      1 |          1 |
+-----+-----+

SELECT v LIKE 'Maria',v LIKE 'Maria   ' FROM strtest;
+-----+-----+
| v LIKE 'Maria' | v LIKE 'Maria   ' |
+-----+-----+
|      0 |          1 |
+-----+-----+
```

Truncation

- Depending on whether or not `strict sql mode` is set, you will either get a warning or an error if you try to insert a string that is too long into a `VARCHAR` column. If the extra characters are spaces, the spaces that can't fit will be removed and you will always get a warning, regardless of the `sql mode` setting.

Difference Between `VARCHAR` and `TEXT`

- `VARCHAR` columns can be fully indexed. `TEXT` columns can only be indexed over a specified length.
- Using `TEXT` or `BLOB` in a `SELECT` query that uses temporary tables for storing intermediate results will force the temporary table to be disk based (using the `Aria storage engine` instead of the `memory storage engine`, which is a bit slower. This is not that bad as the `Aria storage engine` caches the rows in memory. To get the benefit of this, one should ensure that the `aria_pagecache_buffer_size` variable is big enough to hold most of the row and index data for temporary tables.

Oracle Mode

MariaDB starting with 10.3

In Oracle mode from MariaDB 10.3, `VARCHAR2` is a synonym.

For Storage Engine Developers

- Internally the full length of the `VARCHAR` column is allocated inside each `TABLE` objects `record[]` structure. As there are three such buffers, each open table will allocate 3 times max-length-to-store-varchar bytes of memory.
- `TEXT` and `BLOB` columns are stored with a pointer (4 or 8 bytes) + a 1-4 bytes length. The `TEXT` data is only stored once. This means that internally `TEXT` uses less memory for each open table but instead has the additional overhead that each `TEXT` object needs to be allocated and freed for each row access (with some caching in between).

See Also

- [VARBINARY](#)
- [TEXT](#)
- [CHAR](#)
- [Character Sets and Collations](#)

- Data Type Storage Requirements
- Oracle mode from MariaDB 10.3

YEAR Data Type

Syntax

```
YEAR[(4)]
```

Description

A year in two-digit or four-digit format. The default is four-digit format. Note that the two-digit format has been deprecated since [MariaDB 5.5.27](#).

In four-digit format, the allowable values are 1901 to 2155, and 0000. In two-digit format, the allowable values are 70 to 69, representing years from 1970 to 2069. MariaDB displays YEAR values in YYYY format, but allows you to assign values to YEAR columns using either strings or numbers.

Inserting numeric zero has a different result for YEAR(4) and YEAR(2). For YEAR(2), the value `00` reflects the year 2000. For YEAR(4), the value `0000` reflects the year zero. This only applies to numeric zero. String zero always reflects the year 2000.

Examples

Accepting a string or a number:

```
CREATE TABLE y(y YEAR);

INSERT INTO y VALUES (1990),('2012');

SELECT * FROM y;
+---+
| y |
+---+
| 1990 |
| 2012 |
+---+
```

With `strict_mode` set, the default from [MariaDB 10.2.4](#):

Out of range:

```
INSERT INTO y VALUES (1005),('3080');
ERROR 1264 (22003): Out of range value for column 'y' at row 1

INSERT INTO y VALUES ('2013-12-12');
ERROR 1265 (01000): Data truncated for column 'y' at row 1

SELECT * FROM y;
+---+
| y |
+---+
| 1990 |
| 2012 |
+---+
```

With `strict_mode` unset, the default until [MariaDB 10.2.3](#):

Out of range:

```

INSERT INTO y VALUES (1005),('3080');
Query OK, 2 rows affected, 2 warnings (0.05 sec)
Records: 2 Duplicates: 0 Warnings: 2

SHOW WARNINGS;
+-----+
| Level | Code | Message |
+-----+
| Warning | 1264 | Out of range value for column 'y' at row 1 |
| Warning | 1264 | Out of range value for column 'y' at row 2 |
+-----+

SELECT * FROM y;
+-----+
| y   |
+-----+
| 1990 |
| 2012 |
| 0000 |
| 0000 |
+-----+

```

Truncating:

```

INSERT INTO y VALUES ('2013-12-12');
Query OK, 1 row affected, 1 warning (0.05 sec)

SHOW WARNINGS;
+-----+
| Level | Code | Message |
+-----+
| Warning | 1265 | Data truncated for column 'y' at row 1 |
+-----+

SELECT * FROM y;
+-----+
| y   |
+-----+
| 1990 |
| 2012 |
| 0000 |
| 0000 |
| 2013 |
+-----+

```

Difference between YEAR(2) and YEAR(4), and string and numeric zero:

```

CREATE TABLE y2(y YEAR(4), y2 YEAR(2));
Query OK, 0 rows affected, 1 warning (0.40 sec)

Note (Code 1287): 'YEAR(2)' is deprecated and will be removed in a future release.
Please use YEAR(4) instead

INSERT INTO y2 VALUES(0,0),('0','0');

SELECT YEAR(y),YEAR(y2) FROM y2;
+-----+
| YEAR(y) | YEAR(y2) |
+-----+
|     0    |    2000   |
|  2000   |    2000   |
+-----+

```

See Also

- [YEAR\(\) function](#)

CHARACTER_LENGTH

Syntax

```
CHARACTER_LENGTH(str)
```

Description

CHARACTER_LENGTH() is a synonym for CHAR_LENGTH().

INSTR

Syntax

```
INSTR(str, substr)
```

Description

Returns the position of the first occurrence of substring *substr* in string *str*. This is the same as the two-argument form of LOCATE(), except that the order of the arguments is reversed.

INSTR() performs a case-insensitive search.

If any argument is NULL, returns NULL.

Contents

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

Examples

```
SELECT INSTR('foobarbar', 'bar');
+-----+
| INSTR('foobarbar', 'bar') |
+-----+
|          4 |
+-----+

SELECT INSTR('My', 'Maria');
+-----+
| INSTR('My', 'Maria') |
+-----+
|          0 |
+-----+
```

See Also

- [LOCATE\(\)](#) ; Returns the position of a string within a string
- [SUBSTRING_INDEX\(\)](#) ; Returns the substring from string before count occurrences of a delimiter

BEGIN END

Syntax

```
[begin_label:] BEGIN [NOT ATOMIC]
[statement_list]
END [end_label]
```

NOT ATOMIC is required when used outside of a stored procedure. Inside stored procedures or within an anonymous block, BEGIN alone starts a new anonymous block.

Description

`BEGIN ... END` syntax is used for writing compound statements. A compound statement can contain multiple statements, enclosed by the `BEGIN` and `END` keywords. `statement_list` represents a list of one or more statements, each terminated by a semicolon (i.e., `;`) statement delimiter.

`statement_list` is optional, which means that the empty compound statement (`BEGIN END`) is legal.

Note that `END` will perform a commit. If you are running in `autocommit` mode, every statement will be committed separately. If you are not running in `autocommit` mode, you must execute a `COMMIT` or `ROLLBACK` after `END` to get the database up to date.

Use of multiple statements requires that a client is able to send statement strings containing the `;` statement delimiter. This is handled in the `mysql` command-line client with the `DELIMITER` command. Changing the `;` end-of-statement delimiter (for example, to `//`) allows `;` to be used in a program body.

A compound statement within a `stored program` can be `labeled`. `end_label` cannot be given unless `begin_label` also is present. If both are present, they must be the same.

`BEGIN ... END` constructs can be nested. Each block can define its own variables, a `CONDITION`, a `HANDLER` and a `CURSOR`, which don't exist in the outer blocks. The most local declarations override the outer objects which use the same name (see example below).

The declarations order is the following:

- `DECLARE local_variables;`
- `DECLARE CONDITION s;`
- `DECLARE CURSOR s;`
- `DECLARE HANDLER s;`

Note that `DECLARE HANDLER` contains another `BEGIN ... END` construct.

Here is an example of a very simple, anonymous block:

```
BEGIN NOT ATOMIC
SET @a=1;
CREATE TABLE test.t1(a INT);
END|
```

Below is an example of nested blocks in a stored procedure:

```
CREATE PROCEDURE t( )
BEGIN
    DECLARE x TINYINT UNSIGNED DEFAULT 1;
    BEGIN
        DECLARE x CHAR(2) DEFAULT '02';
        DECLARE y TINYINT UNSIGNED DEFAULT 10;
        SELECT x, y;
    END;
    SELECT x;
END;
```

In this example, a `TINYINT` variable, `x` is declared in the outer block. But in the inner block `x` is re-declared as a `CHAR` and an `y` variable is declared. The inner `SELECT` shows the "new" value of `x`, and the value of `y`. But when `x` is selected in the outer block, the "old" value is returned. The final `SELECT` doesn't try to read `y`, because it doesn't exist in that context.

See Also

- [Using compound statements outside of stored programs](#)
- [Changes in Oracle mode from MariaDB 10.3](#)

CASE Statement

Syntax

Contents

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

```

CASE case_value
  WHEN when_value THEN statement_list
  [WHEN when_value THEN statement_list] ...
  [ELSE statement_list]
END CASE

```

Or:

```

CASE
  WHEN search_condition THEN statement_list
  [WHEN search_condition THEN statement_list] ...
  [ELSE statement_list]
END CASE

```

Description

The text on this page describes the `CASE` statement for [stored programs](#). See the [CASE OPERATOR](#) for details on the `CASE` operator outside of [stored programs](#).

The `CASE` statement for [stored programs](#) implements a complex conditional construct. If a `search_condition` evaluates to true, the corresponding SQL statement list is executed. If no search condition matches, the statement list in the `ELSE` clause is executed. Each `statement_list` consists of one or more statements.

The `CASE` statement cannot have an `ELSE NULL` clause, and it is terminated with `END CASE` instead of `END`. It implements a complex conditional construct. If a `search_condition` evaluates to true, the corresponding SQL statement list is executed. If no search condition matches, the statement list in the `ELSE` clause is executed. Each `statement_list` consists of one or more statements.

If no `when_value` or `search_condition` matches the value tested and the `CASE` statement contains no `ELSE` clause, a Case not found for `CASE` statement error results.

Each `statement_list` consists of one or more statements; an empty `statement_list` is not allowed. To handle situations where no value is matched by any `WHEN` clause, use an `ELSE` containing an empty `BEGIN ... END` block, as shown in this example:

```

DELIMITER |
CREATE PROCEDURE p()
BEGIN
  DECLARE v INT DEFAULT 1;
  CASE v
    WHEN 2 THEN SELECT v;
    WHEN 3 THEN SELECT 0;
    ELSE BEGIN END;
  END CASE;
END;
|

```

The indentation used here in the `ELSE` clause is for purposes of clarity only, and is not otherwise significant. See [Delimiters in the mysql client](#) for more on the use of the delimiter command.

Note: The syntax of the `CASE` statement used inside stored programs differs slightly from that of the SQL `CASE` expression described in [CASE OPERATOR](#). The `CASE` statement cannot have an `ELSE NULL` clause, and it is terminated with `END CASE` instead of `END`.

DECLARE CONDITION

Syntax

```

DECLARE condition_name CONDITION FOR condition_value

condition_value:
  SQLSTATE [VALUE] sqlstate_value
  | mysql_error_code

```

Description

The `DECLARE ... CONDITION` statement defines a named error condition. It specifies a condition that needs specific handling and associates a name with that condition. Later, the name can be used in a `DECLARE ... HANDLER`, `SIGNAL` or `RESIGNAL` statement (as long as the statement is located in the same `BEGIN ... END` block).

Conditions must be declared after `local variables`, but before `CURSORs` and `HANDLERS`.

A condition_value for `DECLARE ... CONDITION` can be an `SQLSTATE` value (a 5-character string literal) or a MySQL error code (a number). You should not use `SQLSTATE` value '00000' or MySQL error code 0, because those indicate success rather than an error condition. If you try, or if you specify an invalid `SQLSTATE` value, an error like this is produced:

```
ERROR 1407 (42000): Bad SQLSTATE: '00000'
```

For a list of `SQLSTATE` values and MariaDB error codes, see [MariaDB Error Codes](#).

LCASE

Syntax

```
LCASE(str)
```

Description

`LCASE()` is a synonym for [LOWER\(\)](#).

LEFT

Syntax

```
LEFT(str,len)
```

Description

Returns the leftmost `len` characters from the string `str`, or `NULL` if any argument is `NULL`.

Examples

```
SELECT LEFT('MariaDB', 5);
+-----+
| LEFT('MariaDB', 5) |
+-----+
| Maria           |
+-----+
```

LENGTHB

MariaDB starting with [10.3.1](#)

Introduced in [MariaDB 10.3.1](#) as part of the [Oracle compatibility enhancements](#).

Syntax

```
LENGTHB(str)
```

Description

`LENGTHB()` returns the length of the given string, in bytes. When [Oracle mode](#) is not set, this is a synonym for [LENGTH](#).

A multi-byte character counts as multiple bytes. This means that for a string containing five two-byte characters, `LENGTHB()` returns 10, whereas `CHAR_LENGTH()` returns 5.

If `str` is not a string value, it is converted into a string. If `str` is `NULL`, the function returns `NULL`.

Examples

When [Oracle mode](#) from MariaDB 10.3 is not set:

```
SELECT CHAR_LENGTH('π'), LENGTH('π'), LENGTHB('π'), OCTET_LENGTH('π');
+-----+-----+-----+
| CHAR_LENGTH('π') | LENGTH('π') | LENGTHB('π') | OCTET_LENGTH('π') |
+-----+-----+-----+
|           1 |         2 |         2 |          2 |
+-----+-----+-----+
```

In [Oracle mode](#) from MariaDB 10.3:

```
SELECT CHAR_LENGTH('π'), LENGTH('π'), LENGTHB('π'), OCTET_LENGTH('π');
+-----+-----+-----+
| CHAR_LENGTH('π') | LENGTH('π') | LENGTHB('π') | OCTET_LENGTH('π') |
+-----+-----+-----+
|           1 |         1 |         2 |          2 |
+-----+-----+-----+
```

See Also

- [CHAR_LENGTH\(\)](#)
- [LENGTH\(\)](#)
- [OCTET_LENGTH\(\)](#)

MID

Syntax

```
MID(str,pos,len)
```

Description

`MID(str,pos,len)` is a synonym for [SUBSTRING\(str,pos,len\)](#).

Examples

```
SELECT MID('abcd',4,1);
+-----+
| MID('abcd',4,1) |
+-----+
| d               |
+-----+

SELECT MID('abcd',2,2);
+-----+
| MID('abcd',2,2) |
+-----+
| bc              |
+-----+
```

A negative starting position:

```
SELECT MID('abcd', -2, 4);
+-----+
| MID('abcd', -2, 4) |
+-----+
| cd                |
+-----+
```

NOT LIKE

Syntax

```
expr NOT LIKE pat [ESCAPE 'escape_char']
```

Description

This is the same as [NOT \(expr LIKE pat \[ESCAPE 'escape_char'\]\)](#).

DECLARE HANDLER

Syntax

```
DECLARE handler_type HANDLER
    FOR condition_value [, condition_value] ...
    statement

handler_type:
    CONTINUE
    | EXIT
    | UNDO

condition_value:
    SQLSTATE [VALUE] sqlstate_value
    | condition_name
    | SQLWARNING
    | NOT FOUND
    | SQLEXCEPTION
    | mariadb_error_code
```

Description

The `DECLARE ... HANDLER` statement specifies handlers that each may deal with one or more conditions. If one of these conditions occurs, the specified statement is executed. `statement` can be a simple statement (for example, `SET var_name = value`), or it can be a compound statement written using [BEGIN](#) and [END](#).

Handlers must be declared after local variables, a `CONDITION` and a `CURSOR`.

For a `CONTINUE` handler, execution of the current program continues after execution of the handler statement. For an `EXIT` handler, execution terminates for the `BEGIN ... END` compound statement in which the handler is declared. (This is true even if the condition occurs in an inner block.) The `UNDO` handler type statement is not supported.

If a condition occurs for which no handler has been declared, the default action is `EXIT`.

A `condition_value` for `DECLARE ... HANDLER` can be any of the following values:

- An `SQLSTATE` value (a 5-character string literal) or a MariaDB error code (a number). You should not use `SQLSTATE` value '00000' or MariaDB error code 0, because those indicate success rather than an error condition. For a list of `SQLSTATE` values and MariaDB error codes, see [MariaDB Error Codes](#).
- A condition name previously specified with `DECLARE ... CONDITION`. It must be in the same stored program. See [DECLARE CONDITION](#).
- `SQLWARNING` is shorthand for the class of `SQLSTATE` values that begin with '01'.

- `NOT FOUND` is shorthand for the class of `SQLSTATE` values that begin with '02'. This is relevant only in the context of cursors and is used to control what happens when a cursor reaches the end of a data set. If no more rows are available, a `No Data` condition occurs with `SQLSTATE` value 02000. To detect this condition, you can set up a handler for it (or for a `NOT FOUND` condition). An example is shown in [Cursor Overview](#). This condition also occurs for `SELECT ... INTO var_list` statements that retrieve no rows.
- `SQLEXCEPTION` is shorthand for the class of `SQLSTATE` values that do not begin with '00', '01', or '02'.

When an error raises, in some cases it could be handled by multiple `HANDLER`s. For example, there may be a handler for 1050 error, a separate handler for the 42S01 `SQLSTATE`, and another separate handler for the `SQLEXCEPTION` class: in theory all occurrences of `HANDLER` may catch the 1050 error, but MariaDB chooses the `HANDLER` with the highest precedence. Here are the precedence rules:

- Handlers which refer to an error code have the highest precedence.
- Handlers which refer to a `SQLSTATE` come next.
- Handlers which refer to an error class have the lowest precedence.

In some cases, a statement could produce multiple errors. If this happens, in some cases multiple handlers could have the highest precedence. In such cases, the choice of the handler is indeterminate.

Note that if an error occurs within a `CONTINUE HANDLER` block, it can be handled by another `HANDLER`. However, a `HANDLER` which is already in the stack (that is, it has been called to handle an error and its execution didn't finish yet) cannot handle new errors—this prevents endless loops. For example, suppose that a stored procedure contains a `CONTINUE HANDLER` for `SQLWARNING` and another `CONTINUE HANDLER` for `NOT FOUND`. At some point, a `NOT FOUND` error occurs, and the execution enters the `NOT FOUND HANDLER`. But within that handler, a warning occurs, and the execution enters the `SQLWARNING HANDLER`. If another `NOT FOUND` error occurs, it cannot be handled again by the `NOT FOUND HANDLER`, because its execution is not finished.

When a `DECLARE HANDLER` block can handle more than one error condition, it may be useful to know which errors occurred. To do so, you can use the `GET DIAGNOSTICS` statement.

An error that is handled by a `DECLARE HANDLER` construct can be issued again using the `RESIGNAL` statement.

Below is an example using `DECLARE HANDLER`:

```
CREATE TABLE test.t (s1 INT, PRIMARY KEY (s1));

DELIMITER //

CREATE PROCEDURE handlerdemo( )
BEGIN
    DECLARE CONTINUE HANDLER FOR SQLSTATE '23000' SET @x2 = 1;
    SET @x = 1;
    INSERT INTO test.t VALUES (1);
    SET @x = 2;
    INSERT INTO test.t VALUES (1);
    SET @x = 3;
END;
//

DELIMITER ;

CALL handlerdemo();

SELECT @x;
+----+
| @x |
+----+
|   3 |
+----+
```

DECLARE Variable

Syntax

```
DECLARE var_name [, var_name] ... [[ROW] TYPE OF]] type [DEFAULT value]
```

Description

This statement is used to declare local variables within [stored programs](#). To provide a default value for the variable, include a `DEFAULT` clause. The

value can be specified as an expression (even subqueries are permitted); it need not be a constant. If the `DEFAULT` clause is missing, the initial value is `NULL`.

Local variables are treated like stored routine parameters with respect to data type and overflow checking. See [CREATE PROCEDURE](#).

Local variables must be declared before `CONDITION`s, `CURSOR`s and `HANDLER`s.

Local variable names are not case sensitive.

The scope of a local variable is within the `BEGIN ... END` block where it is declared. The variable can be referred to in blocks nested within the declaring block, except those blocks that declare a variable with the same name.

TYPE OF / ROW TYPE OF

Contents

1. [Syntax](#)
2. [Description](#)
 1. [TYPE OF / ROW TYPE OF](#)
 2. [Examples](#)
 3. [See Also](#)

MariaDB starting with [10.3](#)

`TYPE OF` and `ROW TYPE OF` anchored data types for stored routines were introduced in [MariaDB 10.3](#).

Anchored data types allow a data type to be defined based on another object, such as a table row, rather than specifically set in the declaration. If the anchor object changes, so will the anchored data type. This can lead to routines being easier to maintain, so that if the data type in the table is changed, it will automatically be changed in the routine as well.

Variables declared with `ROW TYPE OF` will have the same features as implicit `ROW` variables. It is not possible to use `ROW TYPE OF` variables in a `LIMIT` clause.

The real data type of `TYPE OF` and `ROW TYPE OF` `table_name` will become known at the very beginning of the stored routine call. [ALTER TABLE](#) or [DROP TABLE](#) statements performed inside the current routine on the tables that appear in anchors won't affect the data type of the anchored variables, even if the variable is declared after an [ALTER TABLE](#) or [DROP TABLE](#) statement.

The real data type of a `ROW TYPE OF` `cursor_name` variable will become known when execution enters into the block where the variable is declared. Data type instantiation will happen only once. In a cursor `ROW TYPE OF` variable that is declared inside a loop, its data type will become known on the very first iteration and won't change on further loop iterations.

The tables referenced in `TYPE OF` and `ROW TYPE OF` declarations will be checked for existence at the beginning of the stored routine call. [CREATE PROCEDURE](#) or [CREATE FUNCTION](#) will not check the referenced tables for existence.

Examples

`TYPE OF` and `ROW TYPE OF` from [MariaDB 10.3](#):

```
DECLARE tmp TYPE OF t1.a; -- Get the data type from the column {{a}} in the table {{t1}}
DECLARE rec1 ROW TYPE OF t1; -- Get the row data type from the table {{t1}}
DECLARE rec2 ROW TYPE OF cur1; -- Get the row data type from the cursor {{cur1}}
```

See Also

- [User-Defined variables](#)

FOR

MariaDB starting with [10.3](#)

FOR loops were introduced in [MariaDB 10.3](#).

Syntax

Integer range FOR loop:

```
[begin_label:]
FOR var_name IN [ REVERSE ] lower_bound .. upper_bound
DO statement_list
END FOR [ end_label ]
```

Explicit cursor FOR loop

```
[begin_label:]
FOR record_name IN cursor_name [ ( cursor_actual_parameter_list)]
DO statement_list
END FOR [ end_label ]
```

Explicit cursor FOR loop ([Oracle mode](#))

```
[begin_label:]
FOR record_name IN cursor_name [ ( cursor_actual_parameter_list)]
LOOP
    statement_list
END LOOP [ end_label ]
```

Implicit cursor FOR loop

```
[begin_label:]
FOR record_name IN ( select_statement )
DO statement_list
END FOR [ end_label ]
```

Description

FOR loops allow code to be executed a fixed number of times.

In an integer range FOR loop, MariaDB will compare the lower bound and upper bound values, and assign the lower bound value to a counter. If REVERSE is not specified, and the upper bound value is greater than or equal to the counter, the counter will be incremented and the statement will continue, after which the loop is entered again. If the upper bound value is greater than the counter, the loop will be exited.

If REVERSE is specified, the counter is decremented, and the upper bound value needs to be less than or equal for the loop to continue.

Contents

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

Examples

Integer range FOR loop:

```
CREATE TABLE t1 (a INT);

DELIMITER //

FOR i IN 1..3
DO
    INSERT INTO t1 VALUES (i);
END FOR;
//

DELIMITER ;

SELECT * FROM t1;
+---+
| a |
+---+
| 1 |
| 2 |
| 3 |
+---+
```

REVERSE integer range FOR loop:

```
CREATE OR REPLACE TABLE t1 (a INT);

DELIMITER //
FOR i IN REVERSE 4..12
DO
    INSERT INTO t1 VALUES (i);
END FOR;
//
```

Query OK, 9 rows affected (0.422 sec)

```
DELIMITER ;
```

```
SELECT * FROM t1;
```

a
12
11
10
9
8
7
6
5
4

Explicit cursor in [Oracle mode](#):

```

SET sql_mode=ORACLE;

CREATE OR REPLACE TABLE t1 (a INT, b VARCHAR(32));

INSERT INTO t1 VALUES (10,'b0');
INSERT INTO t1 VALUES (11,'b1');
INSERT INTO t1 VALUES (12,'b2');

DELIMITER //

CREATE OR REPLACE PROCEDURE p1(pa INT) AS
CURSOR cur(va INT) IS
    SELECT a, b FROM t1 WHERE a=va;
BEGIN
    FOR rec IN cur(pa)
    LOOP
        SELECT rec.a, rec.b;
    END LOOP;
END;
//

DELIMITER ;

CALL p1(10);
+-----+-----+
| rec.a | rec.b |
+-----+-----+
|    10 | b0    |
+-----+-----+

CALL p1(11);
+-----+-----+
| rec.a | rec.b |
+-----+-----+
|    11 | b1    |
+-----+-----+

CALL p1(12);
+-----+-----+
| rec.a | rec.b |
+-----+-----+
|    12 | b2    |
+-----+-----+

CALL p1(13);
Query OK, 0 rows affected (0.000 sec)

```

See Also

- [LOOP](#)

Labels

Syntax

```

label: <construct>
[label]

```

Labels are MariaDB [identifiers](#) which can be used to identify a `BEGIN ... END` construct or a loop. They have a maximum length of 16 characters and can be quoted with backticks (i.e., `).

Labels have a start part and an end part. The start part must precede the portion of code it refers to, must be followed by a colon (:) and can be on the same or different line. The end part is optional and adds nothing, but can make the code more readable. If used, the end part must precede the construct's delimiter (;). Constructs identified by a label can be nested. Each construct can be identified by only one label.

Labels need not be unique in the stored program they belong to. However, a label for an inner loop cannot be identical to a label for an outer loop. In

this case, the following error would be produced:

```
ERROR 1309 (42000): Redefining label <label_name>
```

[LEAVE](#) and [ITERATE](#) statements can be used to exit or repeat a portion of code identified by a label. They must be in the same [Stored Routine](#), [Trigger](#) or [Event](#) which contains the target label.

Below is an example using a simple label that is used to exit a [LOOP](#):

```
CREATE PROCEDURE `test_sp`()
BEGIN
    `my_label`:
    LOOP
        SELECT 'looping';
        LEAVE `my_label`;
    END LOOP;
    SELECT 'out of loop';
END;
```

The following label is used to exit a procedure, and has an end part:

```
CREATE PROCEDURE `test_sp`()
`my_label`:
BEGIN
    IF @var = 1 THEN
        LEAVE `my_label`;
    END IF;
    DO something();
END `my_label`;
```

LEAVE

Syntax

```
LEAVE label
```

This statement is used to exit the flow control construct that has the given [label](#). The label must be in the same stored program, not in a caller procedure. [LEAVE](#) can be used within [BEGIN ... END](#) or loop constructs ([LOOP](#), [REPEAT](#), [WHILE](#)). In [Stored Procedures](#), [Triggers](#) and [Events](#), [LEAVE](#) can refer to the outmost [BEGIN ... END](#) construct; in that case, the program exits the procedure. In [Stored Functions](#), [RETURN](#) can be used instead.

Note that [LEAVE](#) cannot be used to exit a [DECLARE HANDLER](#) block.

If you try to [LEAVE](#) a non-existing label, or if you try to [LEAVE](#) a [HANDLER](#) block, the following error will be produced:

```
ERROR 1308 (42000): LEAVE with no matching label: <label_name>
```

The following example uses [LEAVE](#) to exit the procedure if a condition is true:

```

CREATE PROCEDURE proc(IN p TINYINT)
CONTAINS SQL
`whole_proc`:
BEGIN
    SELECT 1;
    IF p < 1 THEN
        LEAVE `whole_proc`;
    END IF;
    SELECT 2;
END;

CALL proc(0);
+---+
| 1 |
+---+
| 1 |
+---+

```

See Also

- [ITERATE](#) - Repeats a loop execution

NOT REGEXP

Syntax

```
expr NOT REGEXP pat, expr NOT RLIKE pat
```

Description

This is the same as [NOT \(expr REGEXP pat\)](#).

OCTET_LENGTH

Syntax

```
OCTET_LENGTH(str)
```

Description

`OCTET_LENGTH()` returns the length of the given string, in octets (bytes). This is a synonym for [LENGTHB\(\)](#), and, when [Oracle mode from MariaDB 10.3](#) is not set, a synonym for [LENGTH\(\)](#).

A multi-byte character counts as multiple bytes. This means that for a string containing five two-byte characters, `OCTET_LENGTH()` returns 10, whereas [CHAR_LENGTH\(\)](#) returns 5.

If `str` is not a string value, it is converted into a string. If `str` is `NULL`, the function returns `NULL`.

Examples

When [Oracle mode](#) from MariaDB 10.3 is not set:

```

SELECT CHAR_LENGTH('π'), LENGTH('π'), LENGTHB('π'), OCTET_LENGTH('π');
+-----+-----+-----+
| CHAR_LENGTH('π') | LENGTH('π') | LENGTHB('π') | OCTET_LENGTH('π') |
+-----+-----+-----+
|          1 |         2 |         2 |          2 |
+-----+-----+-----+

```

In Oracle mode from MariaDB 10.3:

```
SELECT CHAR_LENGTH('π'), LENGTH('π'), LENGTHB('π'), OCTET_LENGTH('π');
+-----+-----+-----+
| CHAR_LENGTH('π') | LENGTH('π') | LENGTHB('π') | OCTET_LENGTH('π') |
+-----+-----+-----+
|           1 |         1 |         2 |         2 |
+-----+-----+-----+
```

See Also

- [CHAR_LENGTH\(\)](#)
- [LENGTH\(\)](#)
- [LENGTHB\(\)](#)
- [Oracle mode from MariaDB 10.3](#)

ORD

Syntax

```
ORD(str)
```

Description

If the leftmost character of the string `str` is a multi-byte character, returns the code for that character, calculated from the numeric values of its constituent bytes using this formula:

```
(1st byte code)
+ (2nd byte code x 256)
+ (3rd byte code x 256 x 256) ...
```

If the leftmost character is not a multi-byte character, `ORD()` returns the same value as the [ASCII\(\)](#) function.

Examples

```
SELECT ORD('2');
+-----+
| ORD('2') |
+-----+
|      50 |
+-----+
```

See Also

- [ASCII\(\)](#) - Return ASCII value of first character
- [CHAR\(\)](#) - Create a character from an integer value

POSITION

Syntax

```
POSITION(substr IN str)
```

Description

POSITION(substr IN str) is a synonym for [LOCATE\(substr,str\)](#).

It's part of ODBC 3.0.

QUOTE

Syntax

```
QUOTE(str)
```

Description

Quotes a string to produce a result that can be used as a properly escaped data value in an SQL statement. The string is returned enclosed by single quotes and with each instance of single quote (" ' "), backslash (" \ "), ASCII NUL , and Control-Z preceded by a backslash. If the argument is NULL , the return value is the word " NULL " without enclosing single quotes.

Examples

```
SELECT QUOTE("Don't!");
+-----+
| QUOTE("Don't!") |
+-----+
| 'Don\'t!' |
+-----+

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

REPEAT LOOP

Syntax

```
[begin_label:] REPEAT
    statement_list
UNTIL search_condition
END REPEAT [end_label]
```

The statement list within a REPEAT statement is repeated until the search_condition is true. Thus, a REPEAT always enters the loop at least once. statement_list consists of one or more statements, each terminated by a semicolon (i.e., ;) statement delimiter.

A REPEAT statement can be labeled. end_label cannot be given unless begin_label also is present. If both are present, they must be the same.

See [Delimiters](#) in the [mysql](#) client for more on client delimiter usage.

```

DELIMITER //

CREATE PROCEDURE dorepeat(p1 INT)
BEGIN
    SET @x = 0;
    REPEAT SET @x = @x + 1; UNTIL @x > p1 END REPEAT;
END
//

CALL dorepeat(1000)//

SELECT @x//
+-----+
| @x   |
+-----+
| 1001 |
+-----+

```

RESIGNAL

Syntax

```

RESIGNAL [error_condition]
[SET error_property
[, error_property] ...]

error_condition:
SQLSTATE [VALUE] 'sqlstate_value'
| condition_name

error_property:
error_property_name = <error_property_value>

error_property_name:
CLASS_ORIGIN
| SUBCLASS_ORIGIN
| MESSAGE_TEXT
| MYSQL_ERRNO
| CONSTRAINT_CATALOG
| CONSTRAINT_SCHEMA
| CONSTRAINT_NAME
| CATALOG_NAME
| SCHEMA_NAME
| TABLE_NAME
| COLUMN_NAME
| CURSOR_NAME

```

Description

The syntax of `RESIGNAL` and its semantics are very similar to `SIGNAL`. This statement can only be used within an error `HANDLER`. It produces an error, like `SIGNAL`. `RESIGNAL` clauses are the same as `SIGNAL`, except that they all are optional, even `SQLSTATE`. All the properties which are not specified in `RESIGNAL`, will be identical to the properties of the error that was received by the error `HANDLER`. For a description of the clauses, see [diagnostics area](#).

Note that `RESIGNAL` does not empty the diagnostics area: it just appends another error condition.

`RESIGNAL`, without any clauses, produces an error which is identical to the error that was received by `HANDLER`.

If used out of a `HANDLER` construct, `RESIGNAL` produces the following error:

```
ERROR 1645 (0K000): RESIGNAL when handler not active
```

Contents

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

In MariaDB 5.5, if a `HANDLER` contained a `CALL` to another procedure, that procedure could use `RESIGNAL`. Since MariaDB 10.0, trying to do this raises the above error.

For a list of SQLSTATE values and MariaDB error codes, see [MariaDB Error Codes](#).

The following procedure tries to query two tables which don't exist, producing a 1146 error in both cases. Those errors will trigger the `HANDLER`. The first time the error will be ignored and the client will not receive it, but the second time, the error is re-signaled, so the client will receive it.

```
CREATE PROCEDURE test_error( )
BEGIN
    DECLARE CONTINUE HANDLER
        FOR 1146
        BEGIN
            IF @hide_errors IS FALSE THEN
                RESIGNAL;
            END IF;
        END;
        SET @hide_errors = TRUE;
        SELECT 'Next error will be ignored' AS msg;
        SELECT `c` FROM `temptab_one`;
        SELECT 'Next error won''t be ignored' AS msg;
        SET @hide_errors = FALSE;
        SELECT `c` FROM `temptab_two`;
    END;

    CALL test_error();

+-----+
| msg           |
+-----+
| Next error will be ignored |
+-----+
+-----+
| msg           |
+-----+
| Next error won't be ignored |
+-----+
ERROR 1146 (42S02): Table 'test temptab_two' doesn't exist
```

The following procedure re-signals an error, modifying only the error message to clarify the cause of the problem.

```
CREATE PROCEDURE test_error()
BEGIN
    DECLARE CONTINUE HANDLER
        FOR 1146
        BEGIN
            RESIGNAL SET
            MESSAGE_TEXT = ``temptab` does not exist';
        END;
        SELECT `c` FROM `temptab`;
    END;

    CALL test_error();
ERROR 1146 (42S02): `temptab` does not exist
```

As explained above, this works on [MariaDB 5.5](#), but produces a 1645 error since 10.0.

```
CREATE PROCEDURE handle_error()
BEGIN
    RESIGNAL;
END;
CREATE PROCEDURE p()
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION CALL p();
    SIGNAL SQLSTATE '45000';
END;
```

See Also

- [Diagnostics Area](#)
- [SIGNAL](#)
- [HANDLER](#)
- [Stored Routines](#)
- [MariaDB Error Codes](#)

SELECT INTO

Syntax

```
SELECT col_name [, col_name] ...
    INTO var_name [, var_name] ...
    table_expr
```

Description

`SELECT ... INTO` enables selected columns to be stored directly into variables. No resultset is produced. The query should return a single row. If the query returns no rows, a warning with error code 1329 occurs (No data), and the variable values remain unchanged. If the query returns multiple rows, error 1172 occurs (Result consisted of more than one row). If it is possible that the statement may retrieve multiple rows, you can use `LIMIT 1` to limit the result set to a single row.

The `INTO` clause can also be specified at the end of the statement.

In the context of such statements that occur as part of events executed by the Event Scheduler, diagnostics messages (not only errors, but also warnings) are written to the error log, and, on Windows, to the application event log.

This statement can be used with both [local variables](#) and [user-defined variables](#).

For the complete syntax, see [SELECT](#).

Another way to set a variable's value is the [SET](#) statement.

`SELECT ... INTO` results are not stored in the [query cache](#) even if `SQL_CACHE` is specified.

Contents

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

Examples

```
SELECT id, data INTO @x,@y
FROM test.t1 LIMIT 1;
```

See Also

- [SELECT](#) - full `SELECT` syntax.
- [SELECT INTO OUTFILE](#) - formatting and writing the result to an external file.
- [SELECT INTO DUMPFILE](#) - binary-safe writing of the unformatted results to an external file.

REPEAT Function

Syntax

```
REPEAT(str,count)
```

Description

Returns a string consisting of the string `str` repeated `count` times. If `count` is less than 1, returns an empty string. Returns `NULL` if `str` or `count` are `NULL`.

Examples

```
SELECT QUOTE(REPEAT('MariaDB ',4));
+-----+
| QUOTE(REPEAT('MariaDB ',4)) |
+-----+
| 'MariaDB MariaDB MariaDB MariaDB ' |
+-----+
```

REPLACE Function

Syntax

```
REPLACE(str,from_str,to_str)
```

Description

Returns the string `str` with all occurrences of the string `from_str` replaced by the string `to_str`. `REPLACE()` performs a case-sensitive match when searching for `from_str`.

Examples

```
SELECT REPLACE('www.mariadb.org', 'w', 'Ww');
+-----+
| REPLACE('www.mariadb.org', 'w', 'Ww') |
+-----+
| WwWwWw.mariadb.org |
+-----+
```

REVERSE

Syntax

```
REVERSE(str)
```

Description

Returns the string `str` with the order of the characters reversed.

Examples

```
SELECT REVERSE('desserts');
+-----+
| REVERSE('desserts') |
+-----+
| stressed |
+-----+
```

RIGHT

Syntax

```
RIGHT(str,len)
```

Description

Returns the rightmost *Len* characters from the string *str*, or NULL if any argument is NULL.

Examples

```
SELECT RIGHT('MariaDB', 2);
+-----+
| RIGHT('MariaDB', 2) |
+-----+
| DB                  |
+-----+
```

SIGNAL

Syntax

```
SIGNAL error_condition
  [SET error_property
  [, error_property] ...]

error_condition:
  SQLSTATE [VALUE] 'sqlstate_value'
  | condition_name

error_property:
  error_property_name = <error_property_value>

error_property_name:
  CLASS_ORIGIN
  | SUBCLASS_ORIGIN
  | MESSAGE_TEXT
  | MYSQL_ERRNO
  | CONSTRAINT_CATALOG
  | CONSTRAINT_SCHEMA
  | CONSTRAINT_NAME
  | CATALOG_NAME
  | SCHEMA_NAME
  | TABLE_NAME
  | COLUMN_NAME
  | CURSOR_NAME
```

`SIGNAL` empties the [diagnostics area](#) and produces a custom error. This statement can be used anywhere, but is generally useful when used inside a [stored program](#). When the error is produced, it can be caught by a [HANDLER](#). If not, the current stored program, or the current statement, will terminate with the specified error.

Sometimes an error [HANDLER](#) just needs to [SIGNAL](#) the same error it received, optionally with some changes. Usually the [RESIGNAL](#) statement is the most convenient way to do this.

`error_condition` can be an `SQLSTATE` value or a named error condition defined via [DECLARE CONDITION](#). `SQLSTATE` must be a constant string consisting of five characters. These codes are standard to ODBC and ANSI SQL. For customized errors, the recommended `SQLSTATE` is '45000'. For a list of `SQLSTATE` values used by MariaDB, see the [MariaDB Error Codes](#) page. The `SQLSTATE` can be read via the API method `mysql_sqlstate()`.

To specify error properties user-defined variables and [local variables](#) can be used, as well as [character set conversions](#) (but you can't set a collation).

The error properties, their type and their default values are explained in the [diagnostics area](#) page.

Errors

If the `SQLSTATE` is not valid, the following error like this will be produced:

Contents

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

```
ERROR 1407 (42000): Bad SQLSTATE: '123456'
```

If a property is specified more than once, an error like this will be produced:

```
ERROR 1641 (42000): Duplicate condition information item 'MESSAGE_TEXT'
```

If you specify a condition name which is not declared, an error like this will be produced:

```
ERROR 1319 (42000): Undefined CONDITION: cond_name
```

If MYSQL_ERRNO is out of range, you will get an error like this:

```
ERROR 1231 (42000): Variable 'MYSQL_ERRNO' can't be set to the value of '0'
```

Examples

Here's what happens if `SIGNAL` is used in the client to generate errors:

```
SIGNAL SQLSTATE '01000';
Query OK, 0 rows affected, 1 warning (0.00 sec)

SHOW WARNINGS;

+-----+-----+
| Level | Code | Message           |
+-----+-----+
| Warning | 1642 | Unhandled user-defined warning condition |
+-----+
1 row in set (0.06 sec)

SIGNAL SQLSTATE '02000';
ERROR 1643 (02000): Unhandled user-defined not found condition
```

How to specify MYSQL_ERRNO and MESSAGE_TEXT properties:

```
SIGNAL SQLSTATE '45000' SET MYSQL_ERRNO=30001, MESSAGE_TEXT='Hello, world!';

ERROR 30001 (45000): Hello, world!
```

The following code shows how to use user variables, local variables and character set conversion with SIGNAL:

```
CREATE PROCEDURE test_error(x INT)
BEGIN
    DECLARE errno SMALLINT UNSIGNED DEFAULT 31001;
    SET @errormsg = 'Hello, world!';
    IF x = 1 THEN
        SIGNAL SQLSTATE '45000' SET
            MYSQL_ERRNO = errno,
            MESSAGE_TEXT = @errormsg;
    ELSE
        SIGNAL SQLSTATE '45000' SET
            MYSQL_ERRNO = _utf8'Hello, world!';
    END IF;
END;
```

How to use named error conditions:

```

CREATE PROCEDURE test_error(n INT)
BEGIN
    DECLARE `too_big` CONDITION FOR SQLSTATE '45000';
    IF n > 10 THEN
        SIGNAL `too_big`;
    END IF;
END;

```

In this example, we'll define a [HANDLER](#) for an error code. When the error occurs, we [SIGNAL](#) a more informative error which makes sense for our procedure:

```

CREATE PROCEDURE test_error()
BEGIN
    DECLARE EXIT HANDLER
    FOR 1146
    BEGIN
        SIGNAL SQLSTATE '45000' SET
        MESSAGE_TEXT = 'Temporary tables not found; did you call init() procedure?';
    END;
    -- this will produce a 1146 error
    SELECT `c` FROM `temptab`;
END;

```

See Also

- [Diagnostics Area](#)
- [RESIGNAL](#)
- [HANDLER](#)
- [Stored Routines](#)
- [MariaDB Error Codes](#)

LINESTRING

Syntax

```
LineString(pt1,pt2,...)
```

Description

Constructs a WKB LineString value from a number of WKB [Point](#) arguments. If any argument is not a WKB Point, the return value is `NULL`. If the number of [Point](#) arguments is less than two, the return value is `NULL`.

Examples

```

SET @ls = 'LineString(1 1,2 2,3 3)';

SELECT AsText(EndPoint(GeomFromText(@ls)));
+-----+
| AsText(EndPoint(GeomFromText(@ls))) |
+-----+
| POINT(3 3)                         |
+-----+

CREATE TABLE gis_line  (g LINESTRING);
INSERT INTO gis_line VALUES
  (LineFromText('LINESTRING(0 0,0 10,10 0)'),,
   (LineStringFromText('LINESTRING(10 10,20 10,20 20,10 20,10 10)')),
   (LineStringFromWKB(AsWKB(LineString(Point(10, 10), Point(40, 10))))));

```

SOUNDS LIKE

Syntax

```
expr1 SOUNDS LIKE expr2
```

Description

This is the same as `SOUNDEX(expr1) = SOUNDEX(expr2)`.

Example

```
SELECT givenname, surname FROM users WHERE givenname SOUNDS LIKE "robert";
+-----+-----+
| givenname | surname |
+-----+-----+
| Roberto   | Castro  |
+-----+-----+
```

SPACE

Syntax

```
SPACE(N)
```

Description

Returns a string consisting of `N` space characters. If `N` is NULL, returns NULL.

Examples

```
SELECT QUOTE(SPACE(6));
+-----+
| QUOTE(SPACE(6)) |
+-----+
| '               |
+-----+
```

STRCMP

Syntax

```
STRCMP(expr1,expr2)
```

Description

`STRCMP()` returns `0` if the strings are the same, `-1` if the first argument is smaller than the second according to the current sort order, and `1` if the strings are otherwise not the same. Returns `NULL` if either argument is `NULL`.

Examples

```

SELECT STRCMP('text', 'text2');
+-----+
| STRCMP('text', 'text2') |
+-----+
|          -1 |
+-----+

SELECT STRCMP('text2', 'text');
+-----+
| STRCMP('text2', 'text') |
+-----+
|          1 |
+-----+

SELECT STRCMP('text', 'text');
+-----+
| STRCMP('text', 'text') |
+-----+
|          0 |
+-----+

```

SUBSTR

Description

`SUBSTR()` is a synonym for `SUBSTRING()`.

UCASE

Syntax

```
UCASE(str)
```

Description

`UCASE()` is a synonym for `UPPER()`.

UPPER

Syntax

```
UPPER(str)
```

Description

Returns the string `str` with all characters changed to uppercase according to the current character set mapping. The default is latin1 (cp1252 West European).

```

SELECT UPPER(surname), givenname FROM users ORDER BY surname;
+-----+
| UPPER(surname) | givenname |
+-----+
| ABEL           | Jacinto   |
| CASTRO         | Robert     |
| COSTA          | Phestos   |
| MOSCHELLA      | Hippolytos|
+-----+

```

`UPPER()` is ineffective when applied to binary strings (`BINARY`, `VARBINARY`, `BLOB`). The description of `LOWER()` shows how to perform lowercase conversion of binary strings.

MULTIPOLYGON

Syntax

```
MultiPolygon(poly1,poly2,...)
```

Description

Constructs a WKB MultiPolygon value from a set of WKB [Polygon](#) arguments. If any argument is not a WKB Polygon, the return value is `NULL`.

Example

```
CREATE TABLE gis_multi_polygon (g MULTIPOLYGON);
INSERT INTO gis_multi_polygon VALUES
    (MultiPolygonFromText('MULTIPOLYGON(((28 26,28 0,84 0,84 42,28 26),(52 18,66 23,73 9,48 6,52 18)),((59 18,67 18,67 13,59 13,59 18),(52 18,66 23,73 9,48 6,52 18)),((59 18,67 18,67 13,59 13,59 18)))),(MPolyFromText('MULTIPOLYGON(((28 26,28 0,84 0,84 42,28 26),(52 18,66 23,73 9,48 6,52 18)),((59 18,67 18,67 13,59 13,59 18))))'),
    (MPolyFromWKB(AsWKB(MultiPolygon(Polygon(LineString(Point(0, 3), Point(3, 3), Point(3, 0), Point(0, 3)))))));
```

POLYGON

Syntax

```
Polygon(ls1,ls2,...)
```

Description

Constructs a WKB Polygon value from a number of [WKB LineString](#) arguments. If any argument does not represent the WKB of a LinearRing (that is, not a closed and simple LineString) the return value is `NULL`.

Note that according to the OpenGIS standard, a POLYGON should have exactly one ExteriorRing and all other rings should lie within that ExteriorRing and thus be the InteriorRings. Practically, however, some systems, including MariaDB's, permit polygons to have several 'ExteriorRings'. In the case of there being multiple, non-overlapping exterior rings [ST_NUMINTERIORRINGS\(\)](#) will return 1.

Examples

```
SET @g = ST_GeomFromText('POLYGON((1 1,1 5,4 9,6 9,9 3,7 2,1 1))';

CREATE TABLE gis_polygon (g POLYGON);
INSERT INTO gis_polygon VALUES
    (PolygonFromText('POLYGON((10 10,20 10,20 20,10 20,10 10))'),
    (PolyFromText('POLYGON((0 0,50 0,50 50,0 50,0 0), (10 10,20 10,20 20,10 20,10 10))')),
    (PolyFromWKB(AsWKB(Polygon(LineString(Point(0, 0), Point(30, 0), Point(30, 30), Point(0, 0))))));
```

Non-overlapping 'polygon':

```
SELECT ST_NumInteriorRings(ST_PolyFromText('POLYGON((0 0,10 0,10 10,0 10,0 0),
(-1 -1,-5 -1,-5,-1 -5,-1 -1))') AS NumInteriorRings;
+-----+
| NumInteriorRings |
+-----+
|          1      |
+-----+
```

ST_BUFFER

Syntax

```
ST_BUFFER(g1,r)
BUFFER(g1,r)
```

Description

Returns a geometry that represents all points whose distance from geometry *g1* is less than or equal to distance, or radius, *r*.

Uses for this function could include creating for example a new geometry representing a buffer zone around an island.

BUFFER() is a synonym.

Examples

Determining whether a point is within a buffer zone:

```
SET @g1 = ST_GEOFROMTEXT('POLYGON((10 10, 10 20, 20 20, 20 10, 10 10))');

SET @g2 = ST_GEOFROMTEXT('POINT(8 8)');

SELECT ST_WITHIN(@g2,ST_BUFFER(@g1,5));
+-----+
| ST_WITHIN(@g2,ST_BUFFER(@g1,5)) |
+-----+
|                                1 |
+-----+

SELECT ST_WITHIN(@g2,ST_BUFFER(@g1,1));
+-----+
| ST_WITHIN(@g2,ST_BUFFER(@g1,1)) |
+-----+
|                                0 |
+-----+
```

ST_CONVEXHULL

MariaDB starting with 10.1.2

ST_ConvexHull() was introduced in MariaDB 10.1.2

Syntax

```
ST_ConvexHull(g)
ConvexHull(g)
```

Description

Given a geometry, returns a geometry that is the minimum convex geometry enclosing all geometries within the set. Returns NULL if the geometry value is NULL or an empty value.

ST_ConvexHull() and ConvexHull() are synonyms.

Examples

The ConvexHull of a single point is simply the single point:

```
SET @g = ST_GEOFROMTEXT('Point(0 0)');
```

```
SELECT ST_ASTEXT(ST_CONVEXHULL(@g));
+-----+
| ST_ASTEXT(ST_CONVEXHULL(@g)) |
+-----+
| POINT(0 0) |
+-----+
```

```
SET @g = ST_GEOFROMTEXT('MultiPoint(0 0, 1 2, 2 3)');
```

```
SELECT ST_ASTEXT(ST_CONVEXHULL(@g));
+-----+
| ST_ASTEXT(ST_CONVEXHULL(@g)) |
+-----+
| POLYGON((0 0,1 2,2 3,0 0)) |
+-----+
```

```
SET @g = ST_GEOFROMTEXT('MultiPoint( 1 1, 2 2, 5 3, 7 2, 9 3, 8 4, 6 6, 6 9, 4 9, 1 5 )');
```

```
SELECT ST_ASTEXT(ST_CONVEXHULL(@g));
+-----+
| ST_ASTEXT(ST_CONVEXHULL(@g)) |
+-----+
| POLYGON((1 1,1 5,4 9,6 9,9 3,7 2,1 1)) |
+-----+
```

ST_SYMDIFFERENCE

Syntax

```
ST_SYMDIFFERENCE(g1,g2)
```

Description

Returns a geometry that represents the portions of geometry g_1 and geometry g_2 that don't intersect.

Examples

```
SET @g1 = ST_GEOFROMTEXT('LINESTRING(10 20, 10 40)');
```

```
SET @g2 = ST_GEOFROMTEXT('LINESTRING(10 15, 10 25)');
```

```
SELECT ASTEXT(ST_SYMDIFFERENCE(@g1,@g2));
+-----+
| ASTEXT(ST_SYMDIFFERENCE(@g1,@g2)) |
+-----+
| MULTILINESTRING((10 15,10 20),(10 25,10 40)) |
+-----+
```

```
SET @g2 = ST_GeoFromText('LINESTRING(10 20, 10 41)');
```

```
SELECT ASTEXT(ST_SYMDIFFERENCE(@g1,@g2));
+-----+
| ASTEXT(ST_SYMDIFFERENCE(@g1,@g2)) |
+-----+
| LINESTRING(10 40,10 41) |
+-----+
```

CREATE FUNCTION

Syntax

```
CREATE [OR REPLACE]
[DEFINER = {user | CURRENT_USER | role | CURRENT_ROLE }]
[AGGREGATE] FUNCTION [IF NOT EXISTS] func_name ([func_parameter[,...]])
RETURNS type
[characteristic ...]
RETURN func_body

func_parameter:
[ IN | OUT | INOUT | IN OUT ] param_name type

type:
Any valid MariaDB data type

characteristic:
LANGUAGE SQL
| [NOT] DETERMINISTIC
| { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
| SQL SECURITY { DEFINER | INVOKER }
| COMMENT 'string'

func_body:
Valid SQL procedure statement
```

Description

Use the `CREATE FUNCTION` statement to create a new [stored function](#). You must have the [CREATE ROUTINE](#) database privilege to use `CREATE FUNCTION`. A function takes any number of arguments and returns a value from the function body. The function body can be any valid SQL expression as you would use, for example, in any select expression. If you have the appropriate privileges, you can call the function exactly as you would any built-in function. See [Security](#) below for details on privileges.

You can also use a variant of the `CREATE FUNCTION` statement to install a user-defined function (UDF) defined by a plugin. See [CREATE FUNCTION \(UDF\)](#) for details.

You can use a `SELECT` statement for the function body by enclosing it in parentheses, exactly as you would to use a subselect for any other expression. The `SELECT` statement must return a single value. If more than one column is returned when the function is called, error 1241 results. If more than one row is returned when the function is called, error 1242 results. Use a `LIMIT` clause to ensure only one row is returned.

You can also replace the `RETURN` clause with a `BEGIN..END` compound statement. The compound statement must contain a `RETURN` statement. When the function is called, the `RETURN` statement immediately returns its result, and any statements after `RETURN` are effectively ignored.

By default, a function is associated with the current database. To associate the function explicitly with a given database, specify the fully-qualified name as `db_name.func_name` when you create it. If the function name is the same as the name of a built-in function, you must use the fully qualified name when you call it.

The parameter list enclosed within parentheses must always be present. If there are no parameters, an empty parameter list of () should be used. Parameter names are not case sensitive.

Each parameter can be declared to use any valid data type, except that the COLLATE attribute cannot be used.

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

IN | OUT | INOUT | IN OUT

Contents

1. [Syntax](#)
2. [Description](#)
 1. [IN | OUT | INOUT | IN OUT](#)
 2. [AGGREGATE](#)
 3. [RETURNS](#)
 4. [LANGUAGE SQL](#)
 5. [OR REPLACE](#)
 6. [IF NOT EXISTS](#)
 7. [\[NOT\] DETERMINISTIC](#)
 8. [MODIFIES SQL DATA](#)
 9. [READS SQL DATA](#)
 10. [CONTAINS SQL](#)
 11. [NO SQL](#)
 12. [Oracle Mode](#)
3. [Security](#)
4. [Character sets and collations](#)
5. [Examples](#)
6. [See Also](#)

The function parameter qualifiers for `IN`, `OUT`, `INOUT`, and `IN OUT` were added in a 10.8.0 preview release. Prior to 10.8.0 quantifiers were supported only in procedures.

`OUT`, `INOUT` and its equivalent `IN OUT`, are only valid if called from `SET` and not `SELECT`. These quantifiers are especially useful for creating functions with more than one return value. This allows functions to be more complex and nested.

```
DELIMITER $$  
CREATE FUNCTION add_func3(IN a INT, IN b INT, OUT c INT) RETURNS INT  
BEGIN  
    SET c = 100;  
    RETURN a + b;  
END;  
$$  
DELIMITER ;  
  
SET @a = 2;  
SET @b = 3;  
SET @c = 0;  
SET @res= add_func3(@a, @b, @c);  
  
SELECT add_func3(@a, @b, @c);  
ERROR 4186 (HY000): OUT or INOUT argument 3 for function add_func3 is not allowed here  
  
DELIMITER $$  
CREATE FUNCTION add_func4(IN a INT, IN b INT, d INT) RETURNS INT  
BEGIN  
    DECLARE c, res INT;  
    SET res = add_func3(a, b, c) + d;  
    if (c > 99) then  
        return 3;  
    else  
        return res;  
    end if;  
END;  
$$  
DELIMITER ;  
  
SELECT add_func4(1,2,3);  
+-----+  
| add_func4(1,2,3) |  
+-----+  
|            3 |  
+-----+
```

AGGREGATE

MariaDB starting with [10.3.3](#)

From [MariaDB 10.3.3](#), it is possible to create stored aggregate functions as well. See [Stored Aggregate Functions](#) for details.

RETURNS

The `RETURNS` clause specifies the return type of the function. `NULL` values are permitted with all return types.

What happens if the `RETURN` clause returns a value of a different type? It depends on the `SQL_MODE` in effect at the moment of the function creation.

If the `SQL_MODE` is strict (`STRICT_ALL_TABLES` or `STRICT_TRANS_TABLES` flags are specified), a 1366 error will be produced.

Otherwise, the value is coerced to the proper type. For example, if a function specifies an `ENUM` or `SET` value in the `RETURNS` clause, but the `RETURN` clause returns an integer, the value returned from the function is the string for the corresponding `ENUM` member or set of `SET` members.

MariaDB stores the `SQL_MODE` system variable setting that is in effect at the time a routine is created, and always executes the routine with this setting in force, regardless of the server SQL mode in effect when the routine is invoked.

LANGUAGE SQL

`LANGUAGE SQL` is a standard SQL clause, and it can be used in MariaDB for portability. However that clause has no meaning, because SQL is the only supported language for stored functions.

A function is deterministic if it can produce only one result for a given list of parameters. If the result may be affected by stored data, server variables, random numbers or any value that is not explicitly passed, then the function is not deterministic. Also, a function is non-deterministic if it uses non-deterministic functions like `NOW()` or `CURRENT_TIMESTAMP()`. The optimizer may choose a faster execution plan if it known that the function is deterministic. In such cases, you should declare the routine using the `DETERMINISTIC` keyword. If you want to explicitly state that the function is not deterministic (which is the default) you can use the `NOT DETERMINISTIC` keywords.

If you declare a non-deterministic function as `DETERMINISTIC`, you may get incorrect results. If you declare a deterministic function as `NOT DETERMINISTIC`, in some cases the queries will be slower.

OR REPLACE

MariaDB starting with 10.1.3

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

```
DROP FUNCTION IF EXISTS function_name;  
CREATE FUNCTION function_name ...;
```

with the exception that any existing [privileges](#) for the function are not dropped.

IF NOT EXISTS

MariaDB starting with 10.1.3

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

[NOT] DETERMINISTIC

The `[NOT] DETERMINISTIC` clause also affects [binary logging](#), because the `STATEMENT` format can not be used to store or replicate non-deterministic statements.

`CONTAINS SQL`, `NO SQL`, `READS SQL DATA`, and `MODIFIES SQL DATA` are informative clauses that tell the server what the function does. MariaDB does not check in any way whether the specified clause is correct. If none of these clauses are specified, `CONTAINS SQL` is used by default.

MODIFIES SQL DATA

`MODIFIES SQL DATA` means that the function contains statements that may modify data stored in databases. This happens if the function contains statements like `DELETE`, `UPDATE`, `INSERT`, `REPLACE` or DDL.

READS SQL DATA

`READS SQL DATA` means that the function reads data stored in databases, but does not modify any data. This happens if `SELECT` statements are used, but there no write operations are executed.

CONTAINS SQL

`CONTAINS SQL` means that the function contains at least one SQL statement, but it does not read or write any data stored in a database. Examples include `SET` or `DO`.

NO SQL

`NO SQL` means nothing, because MariaDB does not currently support any language other than SQL.

Oracle Mode

MariaDB starting with 10.3

From [MariaDB 10.3](#), a subset of Oracle's PL/SQL language has been supported in addition to the traditional SQL/PSM-based MariaDB syntax. See [Oracle mode from MariaDB 10.3](#) for details on changes when running Oracle mode.

Security

You must have the `EXECUTE` privilege on a function to call it. MariaDB automatically grants the `EXECUTE` and `ALTER ROUTINE` privileges to the account that called `CREATE FUNCTION`, even if the `DEFINER` clause was used.

Each function has an account associated as the definer. By default, the definer is the account that created the function. Use the `DEFINER` clause to specify a different account as the definer. You must have the `SUPER` privilege, or, from MariaDB 10.5.2, the `SET USER` privilege, to use the `DEFINER` clause. See [Account Names](#) for details on specifying accounts.

The `SQL SECURITY` clause specifies what privileges are used when a function is called. If `SQL SECURITY IS INVOKER`, the function body will be evaluated using the privileges of the user calling the function. If `SQL SECURITY IS DEFINER`, the function body is always evaluated using the privileges of the definer account. `DEFINER` is the default.

This allows you to create functions that grant limited access to certain data. For example, say you have a table that stores some employee information, and that you've granted `SELECT` privileges [only on certain columns](#) to the user account `roger`.

```
CREATE TABLE employees (name TINYTEXT, dept TINYTEXT, salary INT);
GRANT SELECT (name, dept) ON employees TO roger;
```

To allow the user the get the maximum salary for a department, define a function and grant the `EXECUTE` privilege:

```
CREATE FUNCTION max_salary (dept TINYTEXT) RETURNS INT RETURN
  (SELECT MAX(salary) FROM employees WHERE employees.dept = dept);
GRANT EXECUTE ON FUNCTION max_salary TO roger;
```

Since `SQL SECURITY` defaults to `DEFINER`, whenever the user `roger` calls this function, the subselect will execute with your privileges. As long as you have privileges to select the salary of each employee, the caller of the function will be able to get the maximum salary for each department without being able to see individual salaries.

Character sets and collations

Function return types can be declared to use any valid [character set and collation](#). If used, the `COLLATE` attribute needs to be preceded by a `CHARACTER SET` attribute.

If the character set and collation are not specifically set in the statement, the database defaults at the time of creation will be used. If the database defaults change at a later stage, the stored function character set/collation will not be changed at the same time; the stored function needs to be dropped and recreated to ensure the same character set/collation as the database is used.

Examples

The following example function takes a parameter, performs an operation using an SQL function, and returns the result.

```
CREATE FUNCTION hello (s CHAR(20))
  RETURNS CHAR(50) DETERMINISTIC
  RETURN CONCAT('Hello, ',s,'!');

SELECT hello('world');
+-----+
| hello('world') |
+-----+
| Hello, world! |
+-----+
```

You can use a compound statement in a function to manipulate data with statements like `INSERT` and `UPDATE`. The following example creates a counter function that uses a temporary table to store the current value. Because the compound statement contains statements terminated with semicolons, you have to first change the statement delimiter with the `DELIMITER` statement to allow the semicolon to be used in the function body. See [Delimiters in the mysql client](#) for more.

```

CREATE TEMPORARY TABLE counter (c INT);
INSERT INTO counter VALUES (0);
DELIMITER //
CREATE FUNCTION counter () RETURNS INT
BEGIN
    UPDATE counter SET c = c + 1;
    RETURN (SELECT c FROM counter LIMIT 1);
END //
DELIMITER ;

```

Character set and collation:

```

CREATE FUNCTION hello2 (s CHAR(20))
RETURNS CHAR(50) CHARACTER SET 'utf8' COLLATE 'utf8_bin' DETERMINISTIC
RETURN CONCAT('Hello, ',s,'!');

```

See Also

- [Identifier Names](#)
- [Stored Aggregate Functions](#)
- [CREATE FUNCTION \(UDF\)](#)
- [SHOW CREATE FUNCTION](#)
- [ALTER FUNCTION](#)
- [DROP FUNCTION](#)
- [SHOW FUNCTION STATUS](#)
- [Stored Routine Privileges](#)
- [Information Schema ROUTINES Table](#)

CREATE PROCEDURE

Syntax

```

CREATE
[OR REPLACE]
[DEFINER = { user | CURRENT_USER | role | CURRENT_ROLE }]
PROCEDURE sp_name ([proc_parameter[,...]])
[characteristic ...] routine_body

proc_parameter:
[ IN | OUT | INOUT ] param_name type

type:
Any valid MariaDB data type

characteristic:
LANGUAGE SQL
| [NOT] DETERMINISTIC
| { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
| SQL SECURITY { DEFINER | INVOKER }
| COMMENT 'string'

routine_body:
Valid SQL procedure statement

```

Description

Creates a [stored procedure](#). By default, a routine is associated with the default database. To associate the routine explicitly with a given database, specify the name as db_name.sp_name when you create it.

When the routine is invoked, an implicit USE db_name is performed (and undone when the routine terminates). This causes the routine to have the given default database while it executes. USE statements within stored routines are disallowed.

When a stored procedure has been created, you invoke it by using the `CALL` statement (see [CALL](#)).

To execute the `CREATE PROCEDURE` statement, it is necessary to have the `CREATE ROUTINE` privilege. By default MariaDB automatically grants the `ALTER ROUTINE` and `EXECUTE` privileges to the routine creator. See also [Stored Routine Privileges](#).

The `DEFINER` and `SQL SECURITY` clauses specify the security context to be used when checking access privileges at routine execution time, as described later. Requires the `SUPER` privilege, or, from [MariaDB 10.5.2](#), the `SET USER` privilege.

If the routine name is the same as the name of a built-in SQL function, you must use a space between the name and the following parenthesis when defining the routine, or a syntax error occurs. This is also true when you invoke the routine later. For this reason, we suggest that it is better to avoid re-using the names of existing SQL functions for your own stored routines.

The `IGNORE_SPACE` SQL mode applies to built-in functions, not to stored routines. It is always allowable to have spaces after a routine name, regardless of whether `IGNORE_SPACE` is enabled.

The parameter list enclosed within parentheses must always be present. If there are no parameters, an empty parameter list of `()` should be used. Parameter names are not case sensitive.

Each parameter can be declared to use any valid data type, except that the `COLLATE` attribute cannot be used.

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

IN/OUT/INOUT

Each parameter is an `IN` parameter by default. To specify otherwise for a parameter, use the keyword `OUT` or `INOUT` before the parameter name.

An `IN` parameter passes a value into a procedure. The procedure might modify the value, but the modification is not visible to the caller when the procedure returns. An `OUT` parameter passes a value from the procedure back to the caller. Its initial value is `NULL` within the procedure, and its value is visible to the caller when the procedure returns. An `INOUT` parameter is initialized by the caller, can be modified by the procedure, and any change made by the procedure is visible to the caller when the procedure returns.

For each `OUT` or `INOUT` parameter, pass a user-defined variable in the `CALL` statement that invokes the procedure so that you can obtain its value when the procedure returns. If you are calling the procedure from within another stored procedure or function, you can also pass a routine parameter or local routine variable as an `IN` or `INOUT` parameter.

DETERMINISTIC/NOT DETERMINISTIC

`DETERMINISTIC` and `NOT DETERMINISTIC` apply only to [functions](#). Specifying `DETERMINISTIC` or `NON-DETERMINISTIC` in procedures has no effect. The default value is `NOT DETERMINISTIC`. Functions are `DETERMINISTIC` when they always return the same value for the same input. For example, a `truncate` or `substring` function. Any function involving data, therefore, is always `NOT DETERMINISTIC`.

CONTAINS SQL/NO SQL/READS SQL DATA/MODIFIES SQL DATA

`CONTAINS SQL` , `NO SQL` , `READS SQL DATA` , and `MODIFIES SQL DATA` are informative clauses that tell the server what the function does. MariaDB does not check in any way whether the specified clause is correct. If none of these clauses are specified, `CONTAINS SQL` is used by default.

`MODIFIES SQL DATA` means that the function contains statements that may modify data stored in databases. This happens if the function contains statements like `DELETE`, `UPDATE`, `INSERT`, `REPLACE` or DDL.

`READS SQL DATA` means that the function reads data stored in databases, but does not modify any data. This happens if `SELECT` statements are used, but there no write operations are executed.

`CONTAINS SQL` means that the function contains at least one SQL statement, but it does not read or write any data stored in a database. Examples include `SET` or `DO`.

`NO SQL` means nothing, because MariaDB does not currently support any language other than SQL.

The `routine_body` consists of a valid SQL procedure statement. This can be a simple statement such as `SELECT` or `INSERT`, or it can be a compound statement written using `BEGIN` and `END`. Compound statements can contain declarations, loops, and other control structure statements. See [Programmatic and Compound Statements](#) for syntax details.

MariaDB allows routines to contain DDL statements, such as `CREATE` and `DROP`. MariaDB also allows `stored procedures` (but not `stored functions`) to contain SQL transaction statements such as `COMMIT`.

For additional information about statements that are not allowed in stored routines, see [Stored Routine Limitations](#).

Invoking stored procedure from within programs

Contents

1. [Syntax](#)
 2. [Description](#)
 1. [IN/OUT/INOUT](#)
 2. [DETERMINISTIC/NOT DETERMINISTIC](#)
 3. [CONTAINS SQL/NO SQL/READS SQL DATA/MODIFIES SQL DATA](#)
 4. [Invoking stored procedure from within programs](#)
 5. [OR REPLACE](#)
 6. [sql_mode](#)
 7. [Character Sets and Collations](#)
 8. [Oracle Mode](#)
3. [Examples](#)
 4. [See Also](#)

For information about invoking [stored procedures](#) from within programs written in a language that has a MariaDB/MySQL interface, see [CALL](#).

OR REPLACE

MariaDB starting with [10.1.3](#)

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

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

with the exception that any existing [privileges](#) for the procedure are not dropped.

sql_mode

MariaDB stores the `sql_mode` system variable setting that is in effect at the time a routine is created, and always executes the routine with this setting in force, regardless of the server `SQL mode` in effect when the routine is invoked.

Character Sets and Collations

Procedure parameters can be declared with any character set/collation. If the character set and collation are not specifically set, the database defaults at the time of creation will be used. If the database defaults change at a later stage, the stored procedure character set/collation will not be changed at the same time; the stored procedure needs to be dropped and recreated to ensure the same character set/collation as the database is used.

Oracle Mode

MariaDB starting with [10.3](#)

From [MariaDB 10.3](#), a subset of Oracle's PL/SQL language has been supported in addition to the traditional SQL/PSM-based MariaDB syntax. See [Oracle mode from MariaDB 10.3](#) for details on changes when running Oracle mode.

Examples

The following example shows a simple stored procedure that uses an `OUT` parameter. It uses the `DELIMITER` command to set a new delimiter for the duration of the process — see [Delimiters in the mysql client](#).

```
DELIMITER //

CREATE PROCEDURE simpleproc (OUT param1 INT)
BEGIN
    SELECT COUNT(*) INTO param1 FROM t;
END;
//

DELIMITER ;

CALL simpleproc(@a);

SELECT @a;
+----+
| @a |
+----+
|   1 |
+----+
```

Character set and collation:

```

DELIMITER //

CREATE PROCEDURE simpleproc2 (
    OUT param1 CHAR(10) CHARACTER SET 'utf8' COLLATE 'utf8_bin'
)
BEGIN
    SELECT CONCAT('a'),f1 INTO param1 FROM t;
END;
//
DELIMITER ;

```

CREATE OR REPLACE:

```

DELIMITER //

CREATE PROCEDURE simpleproc2 (
    OUT param1 CHAR(10) CHARACTER SET 'utf8' COLLATE 'utf8_bin'
)
BEGIN
    SELECT CONCAT('a'),f1 INTO param1 FROM t;
END;
//
ERROR 1304 (42000): PROCEDURE simpleproc2 already exists

DELIMITER ;

DELIMITER //

CREATE OR REPLACE PROCEDURE simpleproc2 (
    OUT param1 CHAR(10) CHARACTER SET 'utf8' COLLATE 'utf8_bin'
)
BEGIN
    SELECT CONCAT('a'),f1 INTO param1 FROM t;
END;
//
ERROR 1304 (42000): PROCEDURE simpleproc2 already exists

DELIMITER ;
Query OK, 0 rows affected (0.03 sec)

```

See Also

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

ST_UNION

Syntax

```
ST_UNION(g1,g2)
```

Description

Returns a geometry that is the union of the geometry *g1* and geometry *g2*.

Examples

```
SET @g1 = GEOMFROMTEXT('POINT (0 2)');
SET @g2 = GEOMFROMTEXT('POINT (2 0)');
SELECT ASTEXT(ST_UNION(@g1,@g2));
+-----+
| ASTEXT(ST_UNION(@g1,@g2)) |
+-----+
| MULTIPOINT(2 0,0 2)      |
+-----+
```

```
SET @g1 = GEOMFROMTEXT('POLYGON((0 0,0 3,3 3,3 0,0 0))');
SET @g2 = GEOMFROMTEXT('POLYGON((2 2,4 2,4 4,2 4,2 2))');
SELECT ASTEXT(ST_UNION(@g1,@g2));
+-----+
| ASTEXT(ST_UNION(@g1,@g2))           |
+-----+
| POLYGON((0 0,0 3,2 3,2 4,4 4,4 2,3 2,3 0,0 0)) |
+-----+
```

CACHE INDEX

Syntax

```
CACHE INDEX
tbl_index_list [, tbl_index_list] ...
IN key_cache_name

tbl_index_list:
tbl_name [[INDEX|KEY] (index_name[, index_name] ...)]
```

Description

The `CACHE INDEX` statement assigns table indexes to a specific key cache. It is used only for [MyISAM](#) tables.

A default key cache exists and cannot be destroyed. To create more key caches, the `key_buffer_size` server system variable.

The associations between tables indexes and key caches are lost on server restart. To recreate them automatically, it is necessary to configure caches in a [configuration file](#) and include some `CACHE INDEX` (and optionally `LOAD INDEX`) statements in the init file.

Examples

The following statement assigns indexes from the tables t1, t2, and t3 to the key cache named hot_cache:

```
CACHE INDEX t1, t2, t3 IN hot_cache;
+-----+-----+-----+
| Table | Op          | Msg_type | Msg_text |
+-----+-----+-----+
| test.t1 | assign_to_keycache | status    | OK       |
| test.t2 | assign_to_keycache | status    | OK       |
| test.t3 | assign_to_keycache | status    | OK       |
+-----+-----+-----+
```

Implementation (for MyISAM)

Normally `CACHE INDEX` should not take a long time to execute. Internally it's implemented the following way:

- Find the right key cache (under LOCK_global_system_variables)
- Open the table with a TL_READ_NO_INSERT lock.
- Flush the original key cache for the given file (under key cache lock)
- Flush the new key cache for the given file (safety)
- Move the file to the new key cache (under file share lock)

The only possible long operations are getting the locks for the table and flushing the original key cache, if there were many key blocks for the file in it.

We plan to also add CACHE INDEX for Aria tables if there is a need for this.

HELP Command

Syntax

```
HELP search_string
```

Description

The `HELP` command can be used in any MariaDB client, such as the `mysql` command-line client, to get basic syntax help and a short description for most commands and functions.

If you provide an argument to the `HELP` command, the `mysql` client uses it as a search string to access server-side help. The proper operation of this command requires that the help tables in the `mysql` database be initialized with help topic information.

If there is no match for the search string, the search fails. Use `HELP contents` to see a list of the help categories:

```
HELP contents
You asked for help about help category: "Contents"
For more information, type 'help <item>', where <item> is one of the following
categories:
  Account Management
  Administration
  Compound Statements
  Data Definition
  Data Manipulation
  Data Types
  Functions
  Functions and Modifiers for Use with GROUP BY
  Geographic Features
  Help Metadata
  Language Structure
  Plugins
  Procedures
  Sequences
  Table Maintenance
  Transactions
  User-Defined Functions
  Utility
```

If a search string matches multiple items, MariaDB shows a list of matching topics:

```
HELP drop
Many help items for your request exist.
To make a more specific request, please type 'help <item>',
where <item> is one of the following
topics:
ALTER TABLE
DROP DATABASE
DROP EVENT
DROP FUNCTION
DROP FUNCTION UDF
DROP INDEX
DROP PACKAGE
DROP PACKAGE BODY
DROP PROCEDURE
DROP ROLE
DROP SEQUENCE
DROP SERVER
DROP TABLE
DROP TRIGGER
DROP USER
DROP VIEW
```

Then you can enter a topic as the search string to see the help entry for that topic.

The help is provided with the MariaDB server and makes use of four help tables found in the `mysql` database: [help_relation](#), [help_topic](#), [help_category](#) and [help_keyword](#). These tables are populated by the `mysql_install_db` or `fill_help_table.sql` scripts which, until [MariaDB 10.4.7](#), contain data generated from an old version of MySQL.

CREATE TABLESPACE

The `CREATE 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.

KILL [CONNECTION | QUERY]

Syntax

```
KILL [HARD | SOFT] [CONNECTION | QUERY [ID] ] [thread_id | USER user_name | query_id]
```

Description

Each connection to `mysqld` runs in a separate thread. You can see which threads are running with the `SHOW PROCESSLIST` statement and kill a thread with the `KILL thread_id` statement. `KILL` allows the optional `CONNECTION` or `QUERY` modifier:

- `KILL CONNECTION` is the same as `KILL` with no modifier: It terminates the connection associated with the given thread or query id.
- `KILL QUERY` terminates the statement that the connection `thread_id` is currently executing, but leaves the connection itself intact.
- `KILL QUERY ID` (introduced in [MariaDB 10.0.5](#)) terminates the query by `query_id`, leaving the connection intact.

If a connection is terminated that has an active transaction, the transaction will be rolled back. If only a query is killed, the current transaction will stay active. See also [idle_transaction_timeout](#).

If you have the `PROCESS` privilege, you can see all threads. If you have the `SUPER` privilege, or, from [MariaDB 10.5.2](#), the `CONNECTION ADMIN` privilege, you can kill all threads and statements. Otherwise, you can see and kill only your own threads and statements.

Contents

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

Killing queries that repair or create indexes on MyISAM and Aria tables may result in corrupted tables. Use the `SOFT` option to avoid this!

The `HARD` option (default) kills a command as soon as possible. If you use `SOFT`, then critical operations that may leave a table in an inconsistent state will not be interrupted. Such operations include `REPAIR` and `INDEX` creation for `MyISAM` and `Aria` tables ([REPAIR TABLE](#), [OPTIMIZE TABLE](#)).

`KILL ... USER` `username` will kill all connections/queries for a given user. `USER` can be specified one of the following ways:

- `username` (Kill without regard to hostname)
- `username@hostname`
- `CURRENT_USER` or `CURRENT_USER()`

If you specify a thread id and that thread does not exist, you get the following error:

```
ERROR 1094 (HY000): Unknown thread id: <thread_id>
```

If you specify a query id that doesn't exist, you get the following error:

```
ERROR 1957 (HY000): Unknown query id: <query_id>
```

However, if you specify a user name, no error is issued for non-connected (or even non-existing) users. To check if the connection/query has been killed, you can use the `ROW_COUNT()` function.

A client whose connection is killed receives the following error:

```
ERROR 1317 (70100): Query execution was interrupted
```

To obtain a list of existing sessions, use the `SHOW PROCESSLIST` statement or query the `Information Schema PROCESSLIST` table.

Note: You cannot use `KILL` with the Embedded MySQL Server library because the embedded server merely runs inside the threads of the host application. It does not create any connection threads of its own.

Note: You can also use `mysqladmin kill` `thread_id [,thread_id...]` to kill connections. To get a list of running queries, use `mysqladmin processlist`. See [mysqladmin](#).

[Percona Toolkit](#) contains a program, `pt-kill` that can be used to automatically kill connections that match certain criteria. For example, it can be used to terminate idle connections, or connections that have been busy for more than 60 seconds.

See Also

- [Query limits and timeouts](#)
- [Aborting statements that exceed a certain time to execute](#)
- [idle_transaction_timeout](#)

RESET

Syntax

```
RESET reset_option [, reset_option] ...
```

Description

The `RESET` statement is used to clear the state of various server operations. You must have the `RELOAD privilege` to execute `RESET`.

`RESET` acts as a stronger version of the `FLUSH` statement.

The different `RESET` options are:

Option	Description
<code>SLAVE</code> [" <code>connection_name</code> "] [<code>ALL</code>]	Deletes all <code>relay logs</code> from the slave and reset the replication position in the master <code>binary log</code> .
<code>MASTER</code>	Deletes all old binary logs, makes the binary index file (<code>--log-bin-index</code>) empty and creates a new binary log file. This is useful when you want to reset the master to an initial state. If you want to just delete old, not used binary logs, you should use the <code>PURGE BINARY LOGS</code> command.
<code>QUERY CACHE</code>	Removes all queries from <code>the query cache</code> . See also <code>FLUSH QUERY CACHE</code> .

CREATE VIEW

Syntax

```
CREATE
[OR REPLACE]
[ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
[DEFINER = { user | CURRENT_USER | role | CURRENT_ROLE }]
[SQL SECURITY { DEFINER | INVOKER }]
VIEW [IF NOT EXISTS] view_name [(column_list)]
AS select_statement
[WITH [CASCADED | LOCAL] CHECK OPTION]
```

Description

The CREATE VIEW statement creates a new [view](#), or replaces an existing one if the OR REPLACE clause is given. If the view does not exist, CREATE OR REPLACE VIEW is the same as CREATE VIEW. If the view does exist, CREATE OR REPLACE VIEW is the same as [ALTER VIEW](#).

The select_statement is a [SELECT](#) statement that provides the definition of the view. (When you select from the view, you select in effect using the SELECT statement.) select_statement can select from base tables or other views.

The view definition is "frozen" at creation time, so changes to the underlying tables afterwards do not affect the view definition. For example, if a view is defined as SELECT * on a table, new columns added to the table later do not become part of the view. A [SHOW CREATE VIEW](#) shows that such queries are rewritten and column names are included in the view definition.

The view definition must be a query that does not return errors at view creation times. However, the base tables used by the views might be altered later and the query may not be valid anymore. In this case, querying the view will result in an error. [CHECK TABLE](#) helps in finding this kind of problems.

The [ALGORITHM clause](#) affects how MariaDB processes the view. The DEFINER and SQL SECURITY clauses specify the security context to be used when checking access privileges at view invocation time. The WITH CHECK OPTION clause can be given to constrain inserts or updates to rows in tables referenced by the view. These clauses are described later in this section.

The CREATE VIEW statement requires the CREATE VIEW privilege for the view, and some privilege for each column selected by the SELECT statement. For columns used elsewhere in the SELECT statement you must have the SELECT privilege. If the OR REPLACE clause is present, you must also have the DROP privilege for the view.

A view belongs to a database. By default, a new view is created in the default database. To create the view explicitly in a given database, specify the name as db_name.view_name when you create it.

```
CREATE VIEW test.v AS SELECT * FROM t;
```

Base tables and views share the same namespace within a database, so a database cannot contain a base table and a view that have the same name.

Views must have unique column names with no duplicates, just like base tables. By default, the names of the columns retrieved by the SELECT statement are used for the view column names. To define explicit names for the view columns, the optional column_list clause can be given as a list of comma-separated identifiers. The number of names in column_list must be the same as the number of columns retrieved by the SELECT statement.

MySQL until 5.1.28

Prior to MySQL 5.1.29, When you modify an existing view, the current view definition is backed up and saved. It is stored in that table's database directory, in a subdirectory named arc. The backup file for a view v is named v.frm-00001. If you alter the view again, the next backup is named v.frm-00002. The three latest view backup definitions are stored. Backed up view definitions are not preserved by [mysqldump](#), or any other such programs, but you can retain them using a file copy operation. However, they are not needed for anything but to provide you with a backup of your previous view definition. It is safe to remove these backup definitions, but only while mysqld is not running. If you delete the arc subdirectory or its files while mysqld is running, you will receive an error the next time you try to alter the view:

```
MariaDB [test]> ALTER VIEW v AS SELECT * FROM t;
ERROR 6 (HY000): Error on delete of '.\test\arc\v.frm-0004' (Errcode: 2)
```

Contents

1. [Syntax](#)
2. [Description](#)
 1. [WITH CHECK OPTION](#)
 2. [IF NOT EXISTS](#)
 3. [Atomic DDL](#)
3. [Examples](#)
4. [See Also](#)

Columns retrieved by the SELECT statement can be simple references to table columns. They can also be expressions that use functions, constant values, operators, and so forth.

Unqualified table or view names in the SELECT statement are interpreted with respect to the default database. A view can refer to tables or views in other databases by qualifying the table or view name with the proper database name.

A view can be created from many kinds of SELECT statements. It can refer to base tables or other views. It can use joins, UNION, and subqueries. The SELECT need not even refer to any tables. The following example defines a view that selects two columns from another table, as well as an expression calculated from those columns:

```
CREATE TABLE t (qty INT, price INT);

INSERT INTO t VALUES(3, 50);

CREATE VIEW v AS SELECT qty, price, qty*price AS value FROM t;

SELECT * FROM v;
+-----+-----+
| qty | price | value |
+-----+-----+
|    3 |     50 |    150 |
+-----+-----+
```

A view definition is subject to the following restrictions:

- The SELECT statement cannot contain a subquery in the FROM clause.
- The SELECT statement cannot refer to system or user variables.
- Within a stored program, the definition cannot refer to program parameters or local variables.
- The SELECT statement cannot refer to prepared statement parameters.
- Any table or view referred to in the definition must exist. However, after a view has been created, it is possible to drop a table or view that the definition refers to. In this case, use of the view results in an error. To check a view definition for problems of this kind, use the CHECK TABLE statement.
- The definition cannot refer to a TEMPORARY table, and you cannot create a TEMPORARY view.
- Any tables named in the view definition must exist at definition time.
- You cannot associate a trigger with a view.
- For valid identifiers to use as view names, see [Identifier Names](#).

ORDER BY is allowed in a view definition, but it is ignored if you select from a view using a statement that has its own ORDER BY.

For other options or clauses in the definition, they are added to the options or clauses of the statement that references the view, but the effect is undefined. For example, if a view definition includes a LIMIT clause, and you select from the view using a statement that has its own LIMIT clause, it is undefined which limit applies. This same principle applies to options such as ALL, DISTINCT, or SQL_SMALL_RESULT that follow the SELECT keyword, and to clauses such as INTO, FOR UPDATE, and LOCK IN SHARE MODE.

The PROCEDURE clause cannot be used in a view definition, and it cannot be used if a view is referenced in the FROM clause.

If you create a view and then change the query processing environment by changing system variables, that may affect the results that you get from the view:

```
CREATE VIEW v (mycol) AS SELECT 'abc';

SET sql_mode = '';

SELECT "mycol" FROM v;
+-----+
| mycol |
+-----+
| mycol |
+-----+

SET sql_mode = 'ANSI_QUOTES';

SELECT "mycol" FROM v;
+-----+
| mycol |
+-----+
| abc   |
+-----+
```

is executed that references the view. They were added in MySQL 5.1.2. The legal SQL SECURITY characteristic values are DEFINER and INVOKER. These indicate that the required privileges must be held by the user who defined or invoked the view, respectively. The default SQL SECURITY value is DEFINER.

If a user value is given for the DEFINER clause, it should be a MariaDB account in 'user_name'@'host_name' format (the same format used in the GRANT statement). The user_name and host_name values both are required. The definer can also be given as CURRENT_USER or CURRENT_USER(). The default DEFINER value is the user who executes the CREATE VIEW statement. This is the same as specifying DEFINER = CURRENT_USER explicitly.

If you specify the DEFINER clause, these rules determine the legal DEFINER user values:

- If you do not have the SUPER privilege, or, from MariaDB 10.5.2, the SET USER privilege, the only legal user value is your own account, either specified literally or by using CURRENT_USER. You cannot set the definer to some other account.
- If you have the SUPER privilege, or, from MariaDB 10.5.2, the SET USER privilege, you can specify any syntactically legal account name. If the account does not actually exist, a warning is generated.
- If the SQL SECURITY value is DEFINER but the definer account does not exist when the view is referenced, an error occurs.

Within a view definition, CURRENT_USER returns the view's DEFINER value by default. For views defined with the SQL SECURITY INVOKER characteristic, CURRENT_USER returns the account for the view's invoker. For information about user auditing within views, see <http://dev.mysql.com/doc/refman/5.1/en/account-activity-auditing.html>.

Within a stored routine that is defined with the SQL SECURITY DEFINER characteristic, CURRENT_USER returns the routine's DEFINER value. This also affects a view defined within such a program, if the view definition contains a DEFINER value of CURRENT_USER.

View privileges are checked like this:

- At view definition time, the view creator must have the privileges needed to use the top-level objects accessed by the view. For example, if the view definition refers to table columns, the creator must have privileges for the columns, as described previously. If the definition refers to a stored function, only the privileges needed to invoke the function can be checked. The privileges required when the function runs can be checked only as it executes: For different invocations of the function, different execution paths within the function might be taken.
- When a view is referenced, privileges for objects accessed by the view are checked against the privileges held by the view creator or invoker, depending on whether the SQL SECURITY characteristic is DEFINER or INVOKER, respectively.
- If reference to a view causes execution of a stored function, privilege checking for statements executed within the function depend on whether the function is defined with a SQL SECURITY characteristic of DEFINER or INVOKER. If the security characteristic is DEFINER, the function runs with the privileges of its creator. If the characteristic is INVOKER, the function runs with the privileges determined by the view's SQL SECURITY characteristic.

Example: A view might depend on a stored function, and that function might invoke other stored routines. For example, the following view invokes a stored function f():

```
CREATE VIEW v AS SELECT * FROM t WHERE t.id = f(t.name);
```

Suppose that f() contains a statement such as this:

```
IF name IS NULL then
  CALL p1();
ELSE
  CALL p2();
END IF;
```

The privileges required for executing statements within f() need to be checked when f() executes. This might mean that privileges are needed for p1() or p2(), depending on the execution path within f(). Those privileges must be checked at runtime, and the user who must possess the privileges is determined by the SQL SECURITY values of the view v and the function f().

The DEFINER and SQL SECURITY clauses for views are extensions to standard SQL. In standard SQL, views are handled using the rules for SQL SECURITY INVOKER.

If you invoke a view that was created before MySQL 5.1.2, it is treated as though it was created with a SQL SECURITY DEFINER clause and with a DEFINER value that is the same as your account. However, because the actual definer is unknown, MySQL issues a warning. To make the warning go away, it is sufficient to re-create the view so that the view definition includes a DEFINER clause.

The optional ALGORITHM clause is an extension to standard SQL. It affects how MariaDB processes the view. ALGORITHM takes three values: MERGE, TEMPTABLE, or UNDEFINED. The default algorithm is UNDEFINED if no ALGORITHM clause is present. See [View Algorithms](#) for more information.

Some views are updatable. That is, you can use them in statements such as UPDATE, DELETE, or INSERT to update the contents of the underlying table. For a view to be updatable, there must be a one-to-one relationship between the rows in the view and the rows in the underlying table. There are also certain other constructs that make a view non-updatable. See [Inserting and Updating with Views](#).

WITH CHECK OPTION

The WITH CHECK OPTION clause can be given for an updatable view to prevent inserts or updates to rows except those for which the WHERE clause in the select_statement is true.

In a WITH CHECK OPTION clause for an updatable view, the LOCAL and CASCADED keywords determine the scope of check testing when the view is defined in terms of another view. The LOCAL keyword restricts the CHECK OPTION only to the view being defined. CASCADED causes the checks for underlying views to be evaluated as well. When neither keyword is given, the default is CASCADED.

For more information about updatable views and the WITH CHECK OPTION clause, see [Inserting and Updating with Views](#).

IF NOT EXISTS

MariaDB starting with 10.1.3

The IF NOT EXISTS clause was added in MariaDB 10.1.3

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

Atomic DDL

MariaDB starting with 10.6.1

MariaDB 10.6.1 supports Atomic DDL and CREATE VIEW is atomic.

Examples

```
CREATE TABLE t (a INT, b INT) ENGINE = InnoDB;
INSERT INTO t VALUES (1,1), (2,2), (3,3);
CREATE VIEW v AS SELECT a, a*2 AS a2 FROM t;
SELECT * FROM v;
+---+---+
| a | a2 |
+---+---+
| 1 | 2 |
| 2 | 4 |
| 3 | 6 |
+---+---+
```

OR REPLACE and IF NOT EXISTS:

```
CREATE VIEW v AS SELECT a, a*2 AS a2 FROM t;
ERROR 1050 (42S01): Table 'v' already exists

CREATE OR REPLACE VIEW v AS SELECT a, a*2 AS a2 FROM t;
Query OK, 0 rows affected (0.04 sec)

CREATE VIEW IF NOT EXISTS v AS SELECT a, a*2 AS a2 FROM t;
Query OK, 0 rows affected, 1 warning (0.01 sec)

SHOW WARNINGS;
+-----+-----+
| Level | Code | Message           |
+-----+-----+
| Note  | 1050 | Table 'v' already exists |
+-----+-----+
```

See Also

- [Identifier Names](#)
- [ALTER VIEW](#)

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

DROP PACKAGE

MariaDB starting with [10.3.5](#)

Oracle-style packages were introduced in [MariaDB 10.3.5](#).

Syntax

```
DROP PACKAGE [IF EXISTS] [ db_name . ] package_name
```

Description

The `DROP PACKAGE` statement can be used when [Oracle SQL_MODE](#) is set.

The `DROP PACKAGE` statement drops a stored package entirely:

- Drops the package specification (earlier created using the [CREATE PACKAGE](#) statement).
- Drops the package implementation, if the implementation was already created using the [CREATE PACKAGE BODY](#) statement.

Contents

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

See Also

- [SHOW CREATE PACKAGE](#)
- [CREATE PACKAGE](#)
- [CREATE PACKAGE BODY](#)
- [DROP PACKAGE BODY](#)
- [Oracle SQL_MODE](#)

DROP PACKAGE BODY

MariaDB starting with [10.3.5](#)

Oracle-style packages were introduced in [MariaDB 10.3.5](#).

Syntax

```
DROP PACKAGE BODY [IF EXISTS] [ db_name . ] package_name
```

Description

The `DROP PACKAGE BODY` statement can be used when [Oracle SQL_MODE](#) is set.

The `DROP PACKAGE BODY` statement drops the package body (i.e the implementation), previously created using the [CREATE PACKAGE BODY](#) statement.

Note, `DROP PACKAGE BODY` drops only the package implementation, but does not drop the package specification. Use `DROP PACKAGE` to drop the package entirely (i.e. both implementation and specification).

Contents

1. [Syntax](#)
2. [Description](#)
3. [See also](#)

See also

- [CREATE PACKAGE](#)
- [SHOW CREATE PACKAGE](#)
- [DROP PACKAGE](#)
- [CREATE PACKAGE BODY](#)
- [SHOW CREATE PACKAGE BODY](#)

- Oracle SQL_MODE

DROP TABLESPACE

The `DROP 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.

LASTVAL

`LASTVAL` is a synonym for [PREVIOUS VALUE](#) for `sequence_name`.

NEXTVAL

`NEXTVAL` is a synonym for [NEXT VALUE](#) for `sequence_name`.

JSON_ARRAY

MariaDB starting with [10.2.3](#)

JSON functions were added in [MariaDB 10.2.3](#).

Syntax

```
JSON_ARRAY([value[, value2] ...])
```

Description

Returns a JSON array containing the listed values. The list can be empty.

Example

```
SELECT Json_Array(56, 3.1416, 'My name is "Foo"', NULL);
+-----+
| Json_Array(56, 3.1416, 'My name is "Foo"', NULL) |
+-----+
| [56, 3.1416, "My name is \"Foo\"", null]      |
+-----+
```

See also

- [JSON_MAKE_ARRAY](#), the CONNECT storage engine function

JSON_COMPACT

MariaDB starting with [10.2.4](#)

This function was added in [MariaDB 10.2.4](#).

Syntax

```
JSON_COMPACT(json_doc)
```

Description

Removes all unnecessary spaces so the json document is as short as possible.

Contents

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

Example

```
SET @j = '{ "A": 1, "B": [2, 3]}';  
  
SELECT JSON_COMPACT(@j), @j;  
+-----+  
| JSON_COMPACT(@j) | @j  
+-----+  
| {"A":1,"B":[2,3]} | { "A": 1, "B": [2, 3]} |  
+-----+
```

See Also

- [JSON video tutorial](#) covering JSON_COMPACT.

JSON_DETAILED

MariaDB starting with [10.2.4](#)

This function was added in [MariaDB 10.2.4](#).

Syntax

```
JSON_DETAILED(json_doc[, tab_size])
```

Description

Represents JSON in the most understandable way emphasizing nested structures.

Contents

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

Example

```
SET @j = '{ "A":1,"B": [2,3]}';  
  
SELECT @j;  
+-----+  
| @j  
+-----+  
| { "A":1,"B": [2,3]} |  
+-----+  
  
SELECT JSON_DETAILED(@j);  
+-----+  
| JSON_DETAILED(@j)  
+-----+  
| {  
    "A": 1,  
    "B":  
    [  
        2,  
        3  
    ]  
} |  
+-----+
```

See Also

- [JSON video tutorial](#) covering JSON_DETAILED.

JSON_LOOSE

MariaDB starting with [10.2.4](#)

This function was added in [MariaDB 10.2.4](#).

Syntax

```
JSON_LOOSE(json_doc)
```

Description

Adds spaces to a JSON document to make it look more readable.

Example

```
SET @j = '{ "A":1,"B":[2,3]}';  
  
SELECT JSON_LOOSE(@j), @j;  
+-----+-----+  
| JSON_LOOSE(@j) | @j |  
+-----+-----+  
| {"A": 1, "B": [2, 3]} | { "A":1,"B": [2,3]} |  
+-----+-----+
```

JSON_MERGE

MariaDB starting with [10.2.3](#)

JSON functions were added in [MariaDB 10.2.3](#).

Syntax

```
JSON_MERGE(json_doc, json_doc[, json_doc] ...)
```

Description

Merges the given JSON documents.

Returns the merged result, or NULL if any argument is NULL.

An error occurs if any of the arguments are not valid JSON documents.

`JSON_MERGE` has been deprecated since [MariaDB 10.2.25](#), [MariaDB 10.3.16](#) and [MariaDB 10.4.5](#). `JSON_MERGE_PATCH` is an RFC 7396-compliant replacement, and `JSON_MERGE_PRESERVE` is a synonym.

Example

```
SET @json1 = '[1, 2]';
SET @json2 = '[3, 4]';

SELECT JSON_MERGE(@json1,@json2);
+-----+
| JSON_MERGE(@json1,@json2) |
+-----+
| [1, 2, 3, 4] |
+-----+
```

See Also

- [JSON_MERGE_PATCH](#)
- [JSON_MERGE_PRESERVE](#)

JSON_OBJECT

MariaDB starting with [10.2.3](#)

JSON functions were added in [MariaDB 10.2.3](#).

Syntax

```
JSON_OBJECT([key, value[, key, value] ...])
```

Description

Returns a JSON object containing the given key/value pairs. The key/value list can be empty.

An error will occur if there are an odd number of arguments, or any key name is NULL.

Example

```
SELECT JSON_OBJECT("id", 1, "name", "Monty");
+-----+
| JSON_OBJECT("id", 1, "name", "Monty") |
+-----+
| {"id": 1, "name": "Monty"} |
+-----+
```

See also

- [JSON_MAKE_OBJECT](#), the CONNECT storage engine function

JSON_SET

MariaDB starting with [10.2.3](#)

JSON functions were added in [MariaDB 10.2.3](#).

Syntax

```
JSON_SET(json_doc, path, val[, path, val] ...)
```

Description

Updates or inserts data into a JSON document, returning the result, or NULL if any of the arguments are NULL or the optional path fails to find an object.

An error will occur if the JSON document is invalid, the path is invalid or if the path contains a * or **wildcard**.

JSON_SET can update or insert data, while [JSON_REPLACE](#) can only update, and [JSON_INSERT](#) only insert.

Examples

```
SELECT JSON_SET(Priv, '$.locked', 'true') FROM mysql.global_priv
```

CUME_DIST

MariaDB starting with 10.2

The CUME_DIST() function was first introduced with [window functions](#) in MariaDB 10.2.0.

Syntax

```
CUME_DIST() OVER (
    [ PARTITION BY partition_expression ]
    [ ORDER BY order_list ]
)
```

Contents

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

Description

CUME_DIST() is a [window function](#) that returns the cumulative distribution of a given row. The following formula is used to calculate the value:

```
(number of rows <= current row) / (total rows)
```

Examples

```
create table t1 (
    pk int primary key,
    a int,
    b int
);

insert into t1 values
( 1 , 0, 10),
( 2 , 0, 10),
( 3 , 1, 10),
( 4 , 1, 10),
( 8 , 2, 10),
( 5 , 2, 20),
( 6 , 2, 20),
( 7 , 2, 20),
( 9 , 4, 20),
(10 , 4, 20);

select pk, a, b,
    rank() over (order by a) as rank,
    percent_rank() over (order by a) as pct_rank,
    cume_dist() over (order by a) as cume_dist
from t1;
```

pk	a	b	rank	pct_rank	cume_dist
1	0	10	1	0.000000000	0.200000000
2	0	10	1	0.000000000	0.200000000
3	1	10	3	0.222222222	0.400000000

```

| 4 | 1 | 10 | 3 | 0.2222222222 | 0.4000000000 |
| 5 | 2 | 20 | 5 | 0.4444444444 | 0.8000000000 |
| 6 | 2 | 20 | 5 | 0.4444444444 | 0.8000000000 |
| 7 | 2 | 20 | 5 | 0.4444444444 | 0.8000000000 |
| 8 | 2 | 10 | 5 | 0.4444444444 | 0.8000000000 |
| 9 | 4 | 20 | 9 | 0.8888888889 | 1.0000000000 |
| 10 | 4 | 20 | 9 | 0.8888888889 | 1.0000000000 |
+-----+-----+-----+-----+

```

```

select pk, a, b,
       percent_rank() over (order by pk) as pct_rank,
       cume_dist() over (order by pk) as cume_dist
from t1 order by pk;
+-----+-----+-----+-----+

```

pk	a	b	pct_rank	cume_dist
1	0	10	0.0000000000	0.1000000000
2	0	10	0.1111111111	0.2000000000
3	1	10	0.2222222222	0.3000000000
4	1	10	0.3333333333	0.4000000000
5	2	20	0.4444444444	0.5000000000
6	2	20	0.5555555556	0.6000000000
7	2	20	0.6666666667	0.7000000000
8	2	10	0.7777777778	0.8000000000
9	4	20	0.8888888889	0.9000000000
10	4	20	1.0000000000	1.0000000000

```

select pk, a, b,
       percent_rank() over (partition by a order by a) as pct_rank,
       cume_dist() over (partition by a order by a) as cume_dist
from t1;
+-----+-----+-----+-----+

```

pk	a	b	pct_rank	cume_dist
1	0	10	0.0000000000	1.0000000000
2	0	10	0.0000000000	1.0000000000
3	1	10	0.0000000000	1.0000000000
4	1	10	0.0000000000	1.0000000000
5	2	20	0.0000000000	1.0000000000
6	2	20	0.0000000000	1.0000000000
7	2	20	0.0000000000	1.0000000000
8	2	10	0.0000000000	1.0000000000
9	4	20	0.0000000000	1.0000000000
10	4	20	0.0000000000	1.0000000000

See Also

- [PERCENT_RANK\(\)](#)

NTH_VALUE

MariaDB starting with [10.2.2](#)

The NTH_VALUE() function was first introduced with other window functions in [MariaDB 10.2](#).

Syntax

```

NTH_VALUE (expr[, num_row]) OVER (
  [ PARTITION BY partition_expression ]
  [ ORDER BY order_list ]
)

```

Description

The `NTH_VALUE` function returns the value evaluated at row number `num_row` of the window frame, starting from 1, or NULL if the row does not exist.

Contents

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

SPIDER_FLUSH_TABLE_MON_CACHE

Syntax

```
SPIDER_FLUSH_TABLE_MON_CACHE()
```

Description

A UDF installed with the [Spider Storage Engine](#), this function is used for refreshing monitoring server information. It returns a value of 1 .

Examples

```
SELECT SPIDER_FLUSH_TABLE_MON_CACHE();
+-----+
| SPIDER_FLUSH_TABLE_MON_CACHE() |
+-----+
| 1 |
+-----+
```

COLUMN_CHECK

Syntax

```
COLUMN_CHECK(dyncol_blob);
```

Description

Check if `dyncol_blob` is a valid packed dynamic columns blob. Return value of 1 means the blob is valid, return value of 0 means it is not.

Rationale: Normally, one works with valid dynamic column blobs. Functions like [COLUMN_CREATE](#), [COLUMN_ADD](#), [COLUMN_DELETE](#) always return valid dynamic column blobs. However, if a dynamic column blob is accidentally truncated, or transcoded from one character set to another, it will be corrupted. This function can be used to check if a value in a blob field is a valid dynamic column blob.

COLUMN_DELETE

Syntax

```
COLUMN_DELETE(dyncol_blob, column_nr, column_nr...);
COLUMN_DELETE(dyncol_blob, column_name, column_name...);
```

Description

Deletes a [dynamic column](#) with the specified name. Multiple names can be given. The return value is a dynamic column blob after the modification.

COLUMN_EXISTS

Syntax

```
COLUMN_EXISTS(dyncol_blob, column_nr);
COLUMN_EXISTS(dyncol_blob, column_name);
```

Description

Checks if a column with name `column_name` exists in `dyncol_blob`. If yes, return `1`, otherwise return `0`. See [dynamic columns](#) for more information.

COLUMN_LIST

Syntax

```
COLUMN_LIST(dyncol_blob);
```

Description

Returns a comma-separated list of column names. The names are quoted with backticks.

See [dynamic columns](#) for more information.

CALL

Syntax

```
CALL sp_name([parameter[,...]])
CALL sp_name[()]
```

Description

The `CALL` statement invokes a [stored procedure](#) that was defined previously with [CREATE PROCEDURE](#).

Stored procedure names can be specified as `database_name.procedure_name`. Procedure names and database names can be quoted with backticks `(`)`. This is necessary if they are reserved words, or contain special characters. See [identifier qualifiers](#) for details.

`CALL p()` and `CALL p` are equivalent.

If parentheses are used, any number of spaces, tab characters and newline characters are allowed between the procedure's name and the open parenthesis.

`CALL` can pass back values to its caller using parameters that are declared as `OUT` or `INOUT` parameters. If no value is assigned to an `OUT` parameter, `NULL` is assigned (and its former value is lost). To pass such values from another stored program you can use [user-defined variables](#), [local variables](#) or routine's parameters; in other contexts, you can only use user-defined variables.

`CALL` can also be executed as a prepared statement. Placeholders can be used for `IN` parameters in all versions of MariaDB; for `OUT` and `INOUT` parameters, placeholders can be used since [MariaDB 5.5](#).

When the procedure returns, a client program can also obtain the number of rows affected for the final statement executed within the routine: At the SQL level, call the `ROW_COUNT()` function; from the C API, call the `mysql_affected_rows()` function.

If the `CLIENT_MULTI_RESULTS` API flag is set, `CALL` can return any number of resultsets and the called stored procedure can execute prepared statements. If it is not set, at most one resultset can be returned and prepared statements cannot be used within procedures.

Concurrent Inserts

The [MyISAM](#) storage engine supports concurrent inserts. This feature allows `SELECT` statements to be executed during `INSERT` operations, reducing contention.

Whether concurrent inserts can be used or not depends on the value of the `concurrent_insert` server system variable:

- `NEVER` (0) disables concurrent inserts.
- `AUTO` (1) allows concurrent inserts only when the target table has no free blocks (no data in the middle of the table has been deleted after the

Contents

1. [Notes](#)
2. [See Also](#)

last [OPTIMIZE TABLE](#)). This is the default.

- **ALWAYS** (2) always enables concurrent inserts, in which case new rows are added at the end of a table if the table is being used by another thread.

If the [binary log](#) is used, [CREATE TABLE ... SELECT](#) and [INSERT ... SELECT](#) statements cannot use concurrent inserts. These statements acquire a read lock on the table, so concurrent inserts will need to wait. This way the log can be safely used to restore data.

Concurrent inserts are not used by replicas with the row based [replication](#) (see [binary log formats](#)).

If an [INSERT](#) statement contain the [HIGH_PRIORITY](#) clause, concurrent inserts cannot be used. [INSERT ... DELAYED](#) is usually unneeded if concurrent inserts are enabled.

[LOAD DATA INFILE](#) uses concurrent inserts if the [CONCURRENT](#) keyword is specified and [concurrent_insert](#) is not [NEVER](#). This makes the statement slower (even if no other sessions access the table) but reduces contention.

[LOCK TABLES](#) allows non-conflicting concurrent inserts if a [READ LOCAL](#) lock is used. Concurrent inserts are not allowed if the [LOCAL](#) keyword is omitted.

Notes

The decision to enable concurrent insert for a table is done when the table is opened. If you change the value of [concurrent_insert](#) it will only affect new opened tables. If you want it to work for also for tables in use or cached, you should do [FLUSH TABLES](#) after setting the variable.

See Also

- [INSERT](#)
- [INSERT DELAYED](#)
- [INSERT SELECT](#)
- [HIGH_PRIORITY](#) and [LOW_PRIORITY](#)
- [INSERT - Default & Duplicate Values](#)
- [INSERT IGNORE](#)
- [INSERT ON DUPLICATE KEY UPDATE](#)

EXCEPT

MariaDB starting with [10.3.0](#)

EXCEPT was introduced in [MariaDB 10.3.0](#).

The result of [EXCEPT](#) is all records of the left [SELECT](#) result set except records which are in right [SELECT](#) result set, i.e. it is subtraction of two result sets. From [MariaDB 10.6.1](#), [MINUS](#) is a synonym.

Syntax

```
SELECT ...
  (INTERSECT [ALL | DISTINCT] | EXCEPT [ALL | DISTINCT] | UNION [ALL | DISTINCT]) SELECT ...
  [(INTERSECT [ALL | DISTINCT] | EXCEPT [ALL | DISTINCT] | UNION [ALL | DISTINCT]) SELECT ...]
  [ORDER BY [column [, column ...]]]
  [LIMIT {[offset,] row_count | row_count OFFSET offset}]
```

Please note:

- Brackets for explicit operation precedence are not supported; use a subquery in the [FROM](#) clause as a workaround).

Description

MariaDB has supported [EXCEPT](#) and [INTERSECT](#) in addition to [UNION](#) since [MariaDB 10.3](#).

All behavior for naming columns, [ORDER BY](#) and [LIMIT](#) is the same as for [UNION](#).

[EXCEPT](#) implicitly supposes a [DISTINCT](#) operation.

The result of [EXCEPT](#) is all records of the left [SELECT](#) result except records which are in right [SELECT](#) result set, i.e. it is subtraction of two result

Contents

1. [Syntax](#)
 1. [Description](#)
 1. [Parentheses](#)
 2. [ALL/DISTINCT](#)
 2. [Examples](#)
 3. [See Also](#)

sets.

`EXCEPT` and `UNION` have the same operation precedence and `INTERSECT` has a higher precedence, unless running in Oracle mode, in which case all three have the same precedence.

MariaDB starting with 10.4.0

Parentheses

From MariaDB 10.4.0, parentheses can be used to specify precedence. Before this, a syntax error would be returned.

MariaDB starting with 10.5.0

ALL/DISTINCT

`EXCEPT ALL` and `EXCEPT DISTINCT` were introduced in MariaDB 10.5.0. The `ALL` operator leaves duplicates intact, while the `DISTINCT` operator removes duplicates. `DISTINCT` is the default behavior if neither operator is supplied, and the only behavior prior to MariaDB 10.5.

Examples

Show customers which are not employees:

```
(SELECT e_name AS name, email FROM customers)
EXCEPT
(SELECT c_name AS name, email FROM employees);
```

Difference between `UNION`, `EXCEPT` and `INTERSECT`. `INTERSECT ALL` and `EXCEPT ALL` are available from MariaDB 10.5.0.

```

CREATE TABLE seqs (i INT);
INSERT INTO seqs VALUES (1),(2),(2),(3),(3),(4),(5),(6);

SELECT i FROM seqs WHERE i <= 3 UNION SELECT i FROM seqs WHERE i>=3;
+----+
| i |
+----+
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
| 6 |
+----+

SELECT i FROM seqs WHERE i <= 3 UNION ALL SELECT i FROM seqs WHERE i>=3;
+----+
| i |
+----+
| 1 |
| 2 |
| 2 |
| 3 |
| 3 |
| 3 |
| 3 |
| 4 |
| 5 |
| 6 |
+----+

SELECT i FROM seqs WHERE i <= 3 EXCEPT SELECT i FROM seqs WHERE i>=3;
+----+
| i |
+----+
| 1 |
| 2 |
+----+

SELECT i FROM seqs WHERE i <= 3 EXCEPT ALL SELECT i FROM seqs WHERE i>=3;
+----+
| i |
+----+
| 1 |
| 2 |
| 2 |
+----+

SELECT i FROM seqs WHERE i <= 3 INTERSECT SELECT i FROM seqs WHERE i>=3;
+----+
| i |
+----+
| 3 |
+----+

SELECT i FROM seqs WHERE i <= 3 INTERSECT ALL SELECT i FROM seqs WHERE i>=3;
+----+
| i |
+----+
| 3 |
| 3 |
+----+

```

Parentheses for specifying precedence, from [MariaDB 10.4.0](#)

```

CREATE OR REPLACE TABLE t1 (a INT);
CREATE OR REPLACE TABLE t2 (b INT);
CREATE OR REPLACE TABLE t3 (c INT);

INSERT INTO t1 VALUES (1),(2),(3),(4);
INSERT INTO t2 VALUES (5),(6);
INSERT INTO t3 VALUES (1),(6);

((SELECT a FROM t1) UNION (SELECT b FROM t2)) EXCEPT (SELECT c FROM t3);
+---+
| a |
+---+
| 2 |
| 3 |
| 4 |
| 5 |
+---+

(SELECT a FROM t1) UNION ((SELECT b FROM t2) EXCEPT (SELECT c FROM t3));
+---+
| a |
+---+
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
+---+

```

See Also

- [UNION](#)
- [INTERSECT](#)
- [Get Set for Set Theory: UNION, INTERSECT and EXCEPT in SQL](#) (video tutorial)

HIGH_PRIORITY and LOW_PRIORITY

The [InnoDB](#) storage engine uses row-level locking to ensure data integrity. However some storage engines (such as [MEMORY](#), [MyISAM](#), [Aria](#) and [MERGE](#)) lock the whole table to prevent conflicts. These storage engines use two separate queues to remember pending statements; one is for [SELECTs](#) and the other one is for write statements ([INSERT](#), [DELETE](#), [UPDATE](#)). By default, the latter has a higher priority.

To give write operations a lower priority, the [low_priority_updates](#) server system variable can be set to `ON`. The option is available on both the global and session levels, and it can be set at startup or via the [SET](#) statement.

When too many table locks have been set by write statements, some pending [SELECTs](#) are executed. The maximum number of write locks that can be acquired before this happens is determined by the [max_write_lock_count](#) server system variable, which is dynamic.

If write statements have a higher priority (default), the priority of individual write statements ([INSERT](#), [REPLACE](#), [UPDATE](#), [DELETE](#)) can be changed via the [LOW_PRIORITY](#) attribute, and the priority of a [SELECT](#) statement can be raised via the [HIGH_PRIORITY](#) attribute. Also, [LOCK TABLES](#) supports a [LOW_PRIORITY](#) attribute for [WRITE](#) locks.

If read statements have a higher priority, the priority of an [INSERT](#) can be changed via the [HIGH_PRIORITY](#) attribute. However, the priority of other write statements cannot be raised individually.

The use of [LOW_PRIORITY](#) or [HIGH_PRIORITY](#) for an [INSERT](#) prevents [Concurrent Inserts](#) from being used.

See Also

- [INSERT](#)
- [INSERT DELAYED](#)
- [INSERT SELECT](#)
- [Concurrent Inserts](#)
- [INSERT - Default & Duplicate Values](#)
- [INSERT IGNORE](#)
- [INSERT ON DUPLICATE KEY UPDATE](#)

Contents

1. [See Also](#)

GROUP BY

Use the `GROUP BY` clause in a `SELECT` statement to group rows together that have the same value in one or more column, or the same computed value using expressions with any [functions and operators](#) except [grouping functions](#). When you use a `GROUP BY` clause, you will get a single result row for each group of rows that have the same value for the expression given in `GROUP BY`.

When grouping rows, grouping values are compared as if by the `=` operator. For string values, the `=` operator ignores trailing whitespace and may normalize characters and ignore case, depending on the [collation](#) in use.

You can use any of the grouping functions in your select expression. Their values will be calculated based on all the rows that have been grouped together for each result row. If you select a non-grouped column or a value computed from a non-grouped column, it is undefined which row the returned value is taken from. This is not permitted if the `ONLY_FULL_GROUP_BY SQL_MODE` is used.

You can use multiple expressions in the `GROUP BY` clause, separated by commas. Rows are grouped together if they match on each of the expressions.

You can also use a single integer as the grouping expression. If you use an integer n , the results will be grouped by the n th column in the select expression.

The `WHERE` clause is applied before the `GROUP BY` clause. It filters non-aggregated rows before the rows are grouped together. To filter grouped rows based on aggregate values, use the `HAVING` clause. The `HAVING` clause takes any expression and evaluates it as a boolean, just like the `WHERE` clause. You can use grouping functions in the `HAVING` clause. As with the select expression, if you reference non-grouped columns in the `HAVING` clause, the behavior is undefined.

By default, if a `GROUP BY` clause is present, the rows in the output will be sorted by the expressions used in the `GROUP BY`. You can also specify `ASC` or `DESC` (ascending, descending) after those expressions, like in `ORDER BY`. The default is `ASC`.

If you want the rows to be sorted by another field, you can add an explicit `ORDER BY`. If you don't want the result to be ordered, you can add `ORDER BY NULL`.

WITH ROLLUP

The `WITH ROLLUP` modifier adds extra rows to the resultset that represent super-aggregate summaries. For a full description with examples, see [SELECT WITH ROLLUP](#).

GROUP BY Examples

Consider the following table that records how many times each user has played and won a game:

```
CREATE TABLE plays (name VARCHAR(16), plays INT, wins INT);
INSERT INTO plays VALUES
("John", 20, 5),
("Robert", 22, 8),
("Wanda", 32, 8),
("Susan", 17, 3);
```

Get a list of win counts along with a count:

```
SELECT wins, COUNT(*) FROM plays GROUP BY wins;
+-----+-----+
| wins | COUNT(*) |
+-----+-----+
|   3 |      1 |
|   5 |      1 |
|   8 |      2 |
+-----+
3 rows in set (0.00 sec)
```

The `GROUP BY` expression can be a computed value, and can refer back to an identifier specified with `AS`. Get a list of win averages along with a count:

Contents

1. [WITH ROLLUP](#)
2. [GROUP BY Examples](#)
3. [See Also](#)

```

SELECT (wins / plays) AS winavg, COUNT(*) FROM plays GROUP BY winavg;
+-----+
| winavg | COUNT(*) |
+-----+
| 0.1765 |      1 |
| 0.2500 |      2 |
| 0.3636 |      1 |
+-----+
3 rows in set (0.00 sec)

```

You can use any [grouping function](#) in the select expression. For each win average as above, get a list of the average play count taken to get that average:

```

SELECT (wins / plays) AS winavg, AVG(plays) FROM plays
GROUP BY winavg;
+-----+
| winavg | AVG(plays) |
+-----+
| 0.1765 |    17.0000 |
| 0.2500 |    26.0000 |
| 0.3636 |    22.0000 |
+-----+
3 rows in set (0.00 sec)

```

You can filter on aggregate information using the `HAVING` clause. The `HAVING` clause is applied after `GROUP BY` and allows you to filter on aggregate data that is not available to the `WHERE` clause. Restrict the above example to results that involve an average number of plays over 20:

```

SELECT (wins / plays) AS winavg, AVG(plays) FROM plays
GROUP BY winavg HAVING AVG(plays) > 20;
+-----+
| winavg | AVG(plays) |
+-----+
| 0.2500 |    26.0000 |
| 0.3636 |    22.0000 |
+-----+
2 rows in set (0.00 sec)

```

See Also

- [SELECT](#)
- [Joins and Subqueries](#)
- [LIMIT](#)
- [ORDER BY](#)
- [Common Table Expressions](#)
- [SELECT WITH ROLLUP](#)
- [SELECT INTO OUTFILE](#)
- [SELECT INTO DUMPFILE](#)
- [FOR UPDATE](#)
- [LOCK IN SHARE MODE](#)
- [Optimizer Hints](#)

INSERT - Default & Duplicate Values

Default Values

If the `SQL_MODE` contains `STRICT_TRANS_TABLES` and you are [inserting](#) into a transactional table (like InnoDB), or if the `SQL_MODE` contains `STRICT_ALL_TABLES`, all `NOT NULL` columns which does not have a `DEFAULT` value (and is not `AUTO_INCREMENT`) must be explicitly referenced in `INSERT` statements. If not, an error like this is produced:

```
ERROR 1364 (HY000): Field 'col' doesn't have a default value
```

Contents

1. [Default Values](#)
2. [Duplicate Values](#)
3. [See Also](#)

In all other cases, if a `NOT NULL` column without a `DEFAULT` value is not referenced, an empty value will be inserted (for example, 0 for `INTEGER`

columns and " for CHAR columns). See [NULL Values in MariaDB:Inserting](#) for examples.

If a NOT NULL column having a DEFAULT value is not referenced, NULL will be inserted.

If a NULL column having a DEFAULT value is not referenced, its default value will be inserted. It is also possible to explicitly assign the default value using the DEFAULT keyword or the [DEFAULT\(\)](#) function.

If the DEFAULT keyword is used but the column does not have a DEFAULT value, an error like this is produced:

```
ERROR 1364 (HY000): Field 'col' doesn't have a default value
```

Duplicate Values

By default, if you try to insert a duplicate row and there is a UNIQUE index, INSERT stops and an error like this is produced:

```
ERROR 1062 (23000): Duplicate entry 'dup_value' for key 'col'
```

To handle duplicates you can use the [IGNORE](#) clause, [INSERT ON DUPLICATE KEY UPDATE](#) or the [REPLACE](#) statement. Note that the IGNORE and DELAYED options are ignored when you use [ON DUPLICATE KEY UPDATE](#).

See Also

- [INSERT](#)
- [INSERT DELAYED](#)
- [INSERT SELECT](#)
- [HIGH_PRIORITY and LOW_PRIORITY](#)
- [Concurrent Inserts](#)
- [INSERT IGNORE](#)
- [INSERT ON DUPLICATE KEY UPDATE](#)

INSERT

Syntax

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
[INTO] tbl_name [PARTITION (partition_list)] [(col,...)]
{VALUES | VALUE} ({expr | DEFAULT},...),(...),...
[ ON DUPLICATE KEY UPDATE
  col=expr
  [, col=expr] ... ] [RETURNING select_expr
  [, select_expr ...]]
```

Or:

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
[INTO] tbl_name [PARTITION (partition_list)]
SET col={expr | DEFAULT}, ...
[ ON DUPLICATE KEY UPDATE
  col=expr
  [, col=expr] ... ] [RETURNING select_expr
  [, select_expr ...]]
```

Or:

```
INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
[INTO] tbl_name [PARTITION (partition_list)] [(col,...)]
SELECT ...
[ ON DUPLICATE KEY UPDATE
  col=expr
  [, col=expr] ... ] [RETURNING select_expr
  [, select_expr ...]]
```

The `INSERT` statement is used to insert new rows into an existing table. The `INSERT ... VALUES` and `INSERT ... SELECT` forms of the statement insert rows based on explicitly specified values. The `INSERT ... SELECT` form inserts rows selected from another table or tables. `INSERT ... SELECT` is discussed further in the [INSERT ... SELECT](#) article.

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 `INSERT ... SELECT` to copy rows between different databases.

The `PARTITION` clause can be used in both the `INSERT` and the `SELECT` part. See [Partition Pruning and Selection](#) for details.

MariaDB starting with [10.5](#)

The `RETURNING` clause was introduced in [MariaDB 10.5](#).

Contents

1. [Syntax](#)
2. [INSERT DELAYED](#)
3. [HIGH PRIORITY and LOW PRIORITY](#)
4. [Defaults and Duplicate Values](#)
5. [INSERT IGNORE](#)
6. [INSERT ON DUPLICATE KEY UPDATE](#)
7. [Examples](#)
8. [INSERT ... RETURNING](#)
 1. [Examples](#)
9. [See Also](#)

The columns list is optional. It specifies which values are explicitly inserted, and in which order. If this clause is not specified, all values must be explicitly specified, in the same order they are listed in the table definition.

The list of value follow the `VALUES` or `VALUE` keyword (which are interchangeable, regardless how much values you want to insert), and is wrapped by parenthesis. The values must be listed in the same order as the columns list. It is possible to specify more than one list to insert more than one rows with a single statement. If many rows are inserted, this is a speed optimization.

For one-row statements, the `SET` clause may be more simple, because you don't need to remember the columns order. All values are specified in the form `col = expr`.

Values can also be specified in the form of a SQL expression or subquery. However, the subquery cannot access the same table that is named in the `INTO` clause.

If you use the `LOW_PRIORITY` keyword, execution of the `INSERT` is delayed until no other clients are reading from the table. If you use the `HIGH_PRIORITY` keyword, the statement has the same priority as `SELECT`s. This affects only storage engines that use only table-level locking (MyISAM, MEMORY, MERGE). However, if one of these keywords is specified, `concurrent inserts` cannot be used. See [HIGH_PRIORITY](#) and [LOW_PRIORITY](#) clauses for details.

INSERT DELAYED

For more details on the `DELAYED` option, see [INSERT DELAYED](#).

HIGH PRIORITY and LOW PRIORITY

See [HIGH_PRIORITY](#) and [LOW_PRIORITY](#).

Defaults and Duplicate Values

See [INSERT - Default & Duplicate Values](#) for details..

INSERT IGNORE

See [INSERT IGNORE](#).

INSERT ON DUPLICATE KEY UPDATE

See [INSERT ON DUPLICATE KEY UPDATE](#).

Examples

Specifying the column names:

```
INSERT INTO person (first_name, last_name) VALUES ('John', 'Doe');
```

Inserting more than 1 row at a time:

```
INSERT INTO tbl_name VALUES (1, "row 1"), (2, "row 2");
```

Using the `SET` clause:

```
INSERT INTO person SET first_name = 'John', last_name = 'Doe';
```

SELECTing from another table:

```
INSERT INTO contractor SELECT * FROM person WHERE status = 'c';
```

See [INSERT ON DUPLICATE KEY UPDATE](#) and [INSERT IGNORE](#) for further examples.

INSERT ... RETURNING

`INSERT ... RETURNING` returns a resultset of the inserted rows.

This returns the listed columns for all the rows that are inserted, 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 `INSERT` statement

```
INSERT INTO t2 VALUES (1,'Dog'),(2,'Lion'),(3,'Tiger'),(4,'Leopard')
RETURNING id2,id2+id2,id2&id2,id2||id2;
+-----+-----+-----+
| id2 | id2+id2 | id2&id2 | id2||id2 |
+-----+-----+-----+
| 1 | 2 | 3 | 4 |
| 2 | 4 | 6 | 8 |
| 3 | 6 | 3 | 4 |
| 4 | 8 | 4 | 1 |
+-----+-----+-----+
```

Using stored functions in `RETURNING`

```
DELIMITER |
CREATE FUNCTION f(arg INT) RETURNS INT
BEGIN
    RETURN (SELECT arg+arg);
END|
DELIMITER ;
PREPARE stmt FROM "INSERT INTO t1 SET id1=1, animal1='Bear' RETURNING f(id1), UPPER(animal1)";
EXECUTE stmt;
+-----+
| f(id1) | UPPER(animal1) |
+-----+
| 2 | BEAR |
+-----+
```

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 `INSERT...SELECT...RETURNING` if the table in the `RETURNING` clause is not the same as the `INSERT` table.

See Also

- [INSERT DELAYED](#)
- [INSERT SELECT](#)
- [REPLACE](#) Equivalent to DELETE + INSERT of conflicting row.
- [HIGH_PRIORITY](#) and [LOW_PRIORITY](#)
- Concurrent Inserts
- [INSERT - Default & Duplicate Values](#)
- [INSERT IGNORE](#)
- [INSERT ON DUPLICATE KEY UPDATE](#)
- How to quickly insert data into MariaDB

INSERT IGNORE

Ignoring Errors

Normally `INSERT` stops and rolls back when it encounters an error.

By using the `IGNORE` keyword all errors are converted to warnings, which will not stop inserts of additional rows.

The `IGNORE` and `DELAYED` options are ignored when you use [ON DUPLICATE KEY UPDATE](#).

Prior to MySQL and MariaDB 5.5.28, no warnings were issued for duplicate key errors when using `IGNORE`. You can get the old behavior if you set `OLD_MODE` to `NO_DUP_KEY_WARNINGS_WITH_IGNORE`.

Examples

```
CREATE TABLE t1 (x INT UNIQUE);

INSERT INTO t1 VALUES(1),(2);

INSERT INTO t1 VALUES(2),(3);
ERROR 1062 (23000): Duplicate entry '2' for key 'x'
SELECT * FROM t1;
+---+
| x |
+---+
| 1 |
| 2 |
+---+
2 rows in set (0.00 sec)

INSERT IGNORE INTO t1 VALUES(2),(3);
Query OK, 1 row affected, 1 warning (0.04 sec)

SHOW WARNINGS;
+-----+-----+-----+
| Level | Code | Message          |
+-----+-----+-----+
| Warning | 1062 | Duplicate entry '2' for key 'x' |
+-----+-----+-----+

SELECT * FROM t1;
+---+
| x |
+---+
| 1 |
| 2 |
| 3 |
+---+
```

Contents

1. [Ignoring Errors](#)
2. [Examples](#)
3. [See Also](#)

See [INSERT ON DUPLICATE KEY UPDATE](#) for further examples using that syntax.

See Also

- [INSERT](#)
- [INSERT DELAYED](#)

- [INSERT SELECT](#)
- [HIGH_PRIORITY and LOW_PRIORITY](#)
- [Concurrent Inserts](#)
- [INSERT - Default & Duplicate Values](#)
- [INSERT IGNORE](#)
- [INSERT ON DUPLICATE KEY UPDATE](#)

INSERT DELAYED

Syntax

```
INSERT DELAYED ...
```

Description

The `DELAYED` option for the `INSERT` statement is a MariaDB/MySQL extension to standard SQL that is very useful if you have clients that cannot or need not wait for the `INSERT` to complete. This is a common situation when you use MariaDB for logging and you also periodically run `SELECT` and `UPDATE` statements that take a long time to complete.

When a client uses `INSERT DELAYED`, it gets an okay from the server at once, and the row is queued to be inserted when the table is not in use by any other thread.

Another major benefit of using `INSERT DELAYED` is that inserts from many clients are bundled together and written in one block. This is much faster than performing many separate inserts.

Note that `INSERT DELAYED` is slower than a normal `INSERT` if the table is not otherwise in use. There is also the additional overhead for the server to handle a separate thread for each table for which there are delayed rows. This means that you should use `INSERT DELAYED` only when you are really sure that you need it.

The queued rows are held only in memory until they are inserted into the table. This means that if you terminate mysqld forcibly (for example, with `kill -9`) or if mysqld dies unexpectedly, any queued rows that have not been written to disk are lost.

The number of concurrent `INSERT DELAYED` threads is limited by the `max_delayed_threads` server system variables. If it is set to 0, `INSERT DELAYED` is disabled. The session value can be equal to the global value, or 0 to disable this statement for the current session. If this limit has been reached, the `DELAYED` clause will be silently ignored for subsequent statements (no error will be produced).

Limitations

There are some limitations on the use of `DELAYED`:

- `INSERT DELAYED` works only with [MyISAM](#), [MEMORY](#), [ARCHIVE](#), and [BLACKHOLE](#) tables. If you execute `INSERT DELAYED` with another storage engine, you will get an error like this: `ERROR 1616 (HY000): DELAYED option not supported for table 'tab_name'`
- For MyISAM tables, if there are no free blocks in the middle of the data file, concurrent `SELECT` and `INSERT` statements are supported. Under these circumstances, you very seldom need to use `INSERT DELAYED` with MyISAM.
- `INSERT DELAYED` should be used only for `INSERT` statements that specify value lists. The server ignores `DELAYED` for `INSERT ... SELECT` or `INSERT ... ON DUPLICATE KEY UPDATE` statements.
- Because the `INSERT DELAYED` statement returns immediately, before the rows are inserted, you cannot use `LAST_INSERT_ID()` to get the `AUTO_INCREMENT` value that the statement might generate.
- `DELAYED` rows are not visible to `SELECT` statements until they actually have been inserted.
- After `INSERT DELAYED`, `ROW_COUNT()` returns the number of the rows you tried to insert, not the number of the successful writes.
- `DELAYED` is ignored on slave replication servers, so that `INSERT DELAYED` is treated as a normal `INSERT` on slaves. This is because `DELAYED` could cause the slave to have different data than the master. `INSERT DELAYED` statements are not [safe for replication](#).
- Pending `INSERT DELAYED` statements are lost if a table is write locked and `ALTER TABLE` is used to modify the table structure.
- `INSERT DELAYED` is not supported for views. If you try, you will get an error like this: `ERROR 1347 (HY000): 'view_name' is not BASE TABLE`
- `INSERT DELAYED` is not supported for [partitioned tables](#).
- `INSERT DELAYED` is not supported within [stored programs](#).
- `INSERT DELAYED` does not work with [triggers](#).
- `INSERT DELAYED` does not work if there is a check constraint in place.
- `INSERT DELAYED` does not work if [skip-new](#) mode is active.

Contents

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

See Also

- [INSERT](#)
- [INSERT SELECT](#)
- [HIGH_PRIORITY and LOW_PRIORITY](#)
- [Concurrent Inserts](#)
- [INSERT - Default & Duplicate Values](#)
- [INSERT IGNORE](#)
- [INSERT ON DUPLICATE KEY UPDATE](#)

INSERT SELECT

Syntax

```
INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
      [INTO] tbl_name [(col_name,...)]
      SELECT ...
      [ ON DUPLICATE KEY UPDATE col_name=expr, ... ]
```

Description

With `INSERT ... SELECT`, you can quickly insert many rows into a table from one or more other tables. For example:

```
INSERT INTO tbl_temp2 (fld_id)
SELECT tbl_temp1.fld_order_id
FROM tbl_temp1 WHERE tbl_temp1.fld_order_id > 100;
```

Contents

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

`tbl_name` can also be specified in the form `db_name . tbl_name` (see [Identifier Qualifiers](#)). This allows to copy rows between different databases.

If the new table has a primary key or UNIQUE indexes, you can use `IGNORE` to handle duplicate key errors during the query. The newer values will not be inserted if an identical value already exists.

`REPLACE` can be used instead of `INSERT` to prevent duplicates on UNIQUE indexes by deleting old values. In that case, `ON DUPLICATE KEY UPDATE` cannot be used.

`INSERT ... SELECT` works for tables which already exist. To create a table for a given resultset, you can use `CREATE TABLE ... SELECT`.

See Also

- [INSERT](#)
- [INSERT DELAYED](#)
- [HIGH_PRIORITY and LOW_PRIORITY](#)
- [Concurrent Inserts](#)
- [INSERT - Default & Duplicate Values](#)
- [INSERT IGNORE](#)
- [INSERT ON DUPLICATE KEY UPDATE](#)

INTERSECT

MariaDB starting with 10.3.0

INTERSECT was introduced in [MariaDB 10.3.0](#).

The result of an intersect is the intersection of right and left `SELECT` results, i.e. only records that are present in both result sets will be included in the result of the operation.

Syntax

```

SELECT ...
  (INTERSECT [ALL | DISTINCT] | EXCEPT [ALL | DISTINCT] | UNION [ALL | DISTINCT]) SELECT ...
  [(INTERSECT [ALL | DISTINCT] | EXCEPT [ALL | DISTINCT] | UNION [ALL | DISTINCT]) SELECT ...]
  [ORDER BY [column [, column ...]]]
  [LIMIT {[offset,] row_count | row_count OFFSET offset}]

```

Description

MariaDB has supported `INTERSECT` (as well as `EXCEPT`) in addition to `UNION` since MariaDB 10.3.

All behavior for naming columns, `ORDER BY` and `LIMIT` is the same as for `UNION`.

`INTERSECT` implicitly supposes a `DISTINCT` operation.

The result of an intersect is the intersection of right and left `SELECT` results, i.e. only records that are present in both result sets will be included in the result of the operation.

`INTERSECT` has higher precedence than `UNION` and `EXCEPT` (unless running [running in Oracle mode](#), in which case all three have the same precedence). If possible it will be executed linearly but if not it will be translated to a subquery in the `FROM` clause:

```

(select a,b from t1)
union
(select c,d from t2)
intersect
(select e,f from t3)
union
(select 4,4);

```

will be translated to:

```

(select a,b from t1)
union
select c,d from
((select c,d from t2)
 intersect
 (select e,f from t3)) dummy_subselect
union
(select 4,4)

```

Contents

1. [Syntax](#)
 1. [Description](#)
 1. [Parentheses](#)
 2. [ALL/DISTINCT](#)
2. [Examples](#)
3. [See Also](#)

MariaDB starting with [10.4.0](#)

Parentheses

From [MariaDB 10.4.0](#), parentheses can be used to specify precedence. Before this, a syntax error would be returned.

MariaDB starting with [10.5.0](#)

ALL/DISTINCT

`INTERSECT ALL` and `INTERSECT DISTINCT` were introduced in [MariaDB 10.5.0](#). The `ALL` operator leaves duplicates intact, while the `DISTINCT` operator removes duplicates. `DISTINCT` is the default behavior if neither operator is supplied, and the only behavior prior to [MariaDB 10.5](#).

Examples

Show customers which are employees:

```

(SELECT e_name AS name, email FROM employees)
INTERSECT
(SELECT c_name AS name, email FROM customers);

```

Difference between UNION, EXCEPT and INTERSECT. INTERSECT ALL and EXCEPT ALL are available from MariaDB 10.5.0.

```
CREATE TABLE seqs (i INT);
INSERT INTO seqs VALUES (1),(2),(2),(3),(3),(4),(5),(6);

SELECT i FROM seqs WHERE i <= 3 UNION SELECT i FROM seqs WHERE i>=3;
+---+
| i |
+---+
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
| 6 |
+---+

SELECT i FROM seqs WHERE i <= 3 UNION ALL SELECT i FROM seqs WHERE i>=3;
+---+
| i |
+---+
| 1 |
| 2 |
| 2 |
| 3 |
| 3 |
| 3 |
| 3 |
| 4 |
| 5 |
| 6 |
+---+

SELECT i FROM seqs WHERE i <= 3 EXCEPT SELECT i FROM seqs WHERE i>=3;
+---+
| i |
+---+
| 1 |
| 2 |
+---+

SELECT i FROM seqs WHERE i <= 3 EXCEPT ALL SELECT i FROM seqs WHERE i>=3;
+---+
| i |
+---+
| 1 |
| 2 |
| 2 |
+---+

SELECT i FROM seqs WHERE i <= 3 INTERSECT SELECT i FROM seqs WHERE i>=3;
+---+
| i |
+---+
| 3 |
+---+

SELECT i FROM seqs WHERE i <= 3 INTERSECT ALL SELECT i FROM seqs WHERE i>=3;
+---+
| i |
+---+
| 3 |
| 3 |
+---+
```

Parentheses for specifying precedence, from MariaDB 10.4.0

```

CREATE OR REPLACE TABLE t1 (a INT);
CREATE OR REPLACE TABLE t2 (b INT);
CREATE OR REPLACE TABLE t3 (c INT);

INSERT INTO t1 VALUES (1),(2),(3),(4);
INSERT INTO t2 VALUES (5),(6);
INSERT INTO t3 VALUES (1),(6);

((SELECT a FROM t1) UNION (SELECT b FROM t2)) INTERSECT (SELECT c FROM t3);
+-----+
| a   |
+-----+
| 1   |
| 6   |
+-----+

(SELECT a FROM t1) UNION ((SELECT b FROM t2) INTERSECT (SELECT c FROM t3));
+-----+
| a   |
+-----+
| 1   |
| 2   |
| 3   |
| 4   |
| 6   |
+-----+

```

See Also

- [UNION](#)
- [EXCEPT](#)
- [Get Set for Set Theory: UNION, INTERSECT and EXCEPT in SQL \(video tutorial\)](#)

INSERT ON DUPLICATE KEY UPDATE

Syntax

```

INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
[INTO] tbl_name [PARTITION (partition_list)] [(col,...)]
{VALUES | VALUE} ({expr | DEFAULT},...),(...),...
[ ON DUPLICATE KEY UPDATE
  col=expr
  [, col=expr] ... ]

```

Or:

```

INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
[INTO] tbl_name [PARTITION (partition_list)]
SET col={expr | DEFAULT}, ...
[ ON DUPLICATE KEY UPDATE
  col=expr
  [, col=expr] ... ]

```

Or:

```

INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
[INTO] tbl_name [PARTITION (partition_list)] [(col,...)]
SELECT ...
[ ON DUPLICATE KEY UPDATE
  col=expr
  [, col=expr] ... ]

```

Description

`INSERT ... ON DUPLICATE KEY UPDATE` is a MariaDB/MySQL extension to the [INSERT](#) statement that, if it finds a duplicate unique or primary key, will instead perform an [UPDATE](#).

The row/s affected value is reported as 1 if a row is inserted, and 2 if a row is updated, unless the API's `CLIENT_FOUND_ROWS` flag is set.

If more than one unique index is matched, only the first is updated. It is not recommended to use this statement on tables with more than one unique index.

If the table has an `AUTO_INCREMENT` primary key and the statement inserts or updates a row, the [LAST_INSERT_ID\(\)](#) function returns its `AUTO_INCREMENT` value.

The [VALUES\(\)](#) function can only be used in a `ON DUPLICATE KEY UPDATE` clause and has no meaning in any other context. It returns the column values from the `INSERT` portion of the statement. This function is particularly useful for multi-rows inserts.

The `IGNORE` and `DELAYED` options are ignored when you use `ON DUPLICATE KEY UPDATE`.

MariaDB starting with 10.0

The PARTITION clause was introduced in [MariaDB 10.0](#). See [Partition Pruning and Selection](#) for details.

This statement activates INSERT and UPDATE triggers. See [Trigger Overview](#) for details.

See also a similar statement, [REPLACE](#).

Examples

```
CREATE TABLE ins_duplicate (id INT PRIMARY KEY, animal VARCHAR(30));
INSERT INTO ins_duplicate VALUES (1,'Aardvark'), (2,'Cheetah'), (3,'Zebra');
```

If there is no existing key, the statement runs as a regular `INSERT`:

```
INSERT INTO ins_duplicate VALUES (4,'Gorilla') ON DUPLICATE KEY UPDATE animal='Gorilla';
Query OK, 1 row affected (0.07 sec)
```

```
SELECT * FROM ins_duplicate;
+----+-----+
| id | animal |
+----+-----+
| 1 | Aardvark |
| 2 | Cheetah |
| 3 | Zebra |
| 4 | Gorilla |
+----+-----+
```

A regular `INSERT` with a primary key value of 1 will fail, due to the existing key:

```
INSERT INTO ins_duplicate VALUES (1,'Antelope');
ERROR 1062 (23000): Duplicate entry '1' for key 'PRIMARY'
```

However, we can use an `INSERT ON DUPLICATE KEY UPDATE` instead:

```
INSERT INTO ins_duplicate VALUES (1,'Antelope') ON DUPLICATE KEY UPDATE animal='Antelope';
Query OK, 2 rows affected (0.09 sec)
```

Note that there are two rows reported as affected, but this refers only to the `UPDATE`.

Contents

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

```
SELECT * FROM ins_duplicate;
```

id	animal
1	Antelope
2	Cheetah
3	Zebra
4	Gorilla

Adding a second unique column:

```
ALTER TABLE ins_duplicate ADD id2 INT;
UPDATE ins_duplicate SET id2=id+10;
ALTER TABLE ins_duplicate ADD UNIQUE KEY(id2);
```

Where two rows match the unique keys match, only the first is updated. This can be unsafe and is not recommended unless you are certain what you are doing. Note that the warning shown below appears in [MariaDB 5.5](#) and before, but has been removed in [MariaDB 10.0](#), as MariaDB now assumes that the keys are checked in order, as shown in [SHOW CREATE TABLE](#).

```
INSERT INTO ins_duplicate VALUES (2,'Lion',13) ON DUPLICATE KEY UPDATE animal='Lion';
Query OK, 2 rows affected, 1 warning (0.06 sec)
```

```
SHOW WARNINGS;
```

Level	Code	Message
Note	1592	Unsafe statement written to the binary log using statement format since BINLOG_FORMAT = STATEMENT. INSERT... ON DUP

```
SELECT * FROM ins_duplicate;
```

id	animal	id2
1	Antelope	11
2	Lion	12
3	Zebra	13
4	Gorilla	14

Although the third row with an id of 3 has an id2 of 13, which also matched, it was not updated.

Changing id to an auto_increment field. If a new row is added, the auto_increment is moved forward. If the row is updated, it remains the same.

```

ALTER TABLE `ins_duplicate` CHANGE `id` `id` INT( 11 ) NOT NULL AUTO_INCREMENT;
ALTER TABLE ins_duplicate DROP id2;
SELECT Auto_increment FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_NAME='ins_duplicate';
+-----+
| Auto_increment |
+-----+
|          5 |
+-----+

INSERT INTO ins_duplicate VALUES (2,'Leopard') ON DUPLICATE KEY UPDATE animal='Leopard';
Query OK, 2 rows affected (0.00 sec)

SELECT Auto_increment FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_NAME='ins_duplicate';
+-----+
| Auto_increment |
+-----+
|          5 |
+-----+

INSERT INTO ins_duplicate VALUES (5,'Wild Dog') ON DUPLICATE KEY UPDATE animal='Wild Dog';
Query OK, 1 row affected (0.09 sec)

SELECT * FROM ins_duplicate;
+---+-----+
| id | animal   |
+---+-----+
|  1 | Antelope |
|  2 | Leopard   |
|  3 | Zebra     |
|  4 | Gorilla   |
|  5 | Wild Dog  |
+---+-----+

SELECT Auto_increment FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_NAME='ins_duplicate';
+-----+
| Auto_increment |
+-----+
|          6 |
+-----+

```

Referring to column values from the INSERT portion of the statement:

```

INSERT INTO table (a,b,c) VALUES (1,2,3),(4,5,6)
    ON DUPLICATE KEY UPDATE c=VALUES(a)+VALUES(b);

```

See the [VALUES\(\)](#) function for more.

See Also

- [INSERT](#)
- [INSERT DELAYED](#)
- [INSERT SELECT](#)
- [HIGH_PRIORITY and LOW_PRIORITY](#)
- [Concurrent Inserts](#)
- [INSERT - Default & Duplicate Values](#)
- [INSERT IGNORE](#)
- [VALUES\(\)](#)

JOIN Syntax

Description

MariaDB supports the following `JOIN` syntaxes for the `table_references` part of `SELECT` statements and multiple-table `DELETE` and `UPDATE` statements:

```

table_references:
  table_reference [, table_reference] ...

table_reference:
  table_factor
  | join_table

table_factor:
  tbl_name [PARTITION (partition_list)]
    [query_system_time_period_specification] [[AS] alias] [index_hint_list]
  | table_subquery [query_system_time_period_specification] [AS] alias
  | ( table_references )
  | { ON table_reference LEFT OUTER JOIN table_reference
      ON conditional_expr }

join_table:
  table_reference [INNER | CROSS] JOIN table_factor [join_condition]
  | table_reference STRAIGHT_JOIN table_factor
  | table_reference STRAIGHT_JOIN table_factor ON conditional_expr
  | table_reference {LEFT|RIGHT} [OUTER] JOIN table_reference join_condition
  | table_reference NATURAL [{LEFT|RIGHT} [OUTER]] JOIN table_factor

join_condition:
  ON conditional_expr
  | USING (column_list)

query_system_time_period_specification:
  FOR SYSTEM_TIME AS OF point_in_time
  | FOR SYSTEM_TIME BETWEEN point_in_time AND point_in_time
  | FOR SYSTEM_TIME FROM point_in_time TO point_in_time
  | FOR SYSTEM_TIME ALL

point_in_time:
  [TIMESTAMP] expression
  | TRANSACTION expression

index_hint_list:
  index_hint [, index_hint] ...

index_hint:
  USE {INDEX|KEY}
    [{FOR {JOIN|ORDER BY|GROUP BY}] ([index_list])
  | IGNORE {INDEX|KEY}
    [{FOR {JOIN|ORDER BY|GROUP BY}] (index_list)
  | FORCE {INDEX|KEY}
    [{FOR {JOIN|ORDER BY|GROUP BY}] (index_list)

index_list:
  index_name [, index_name] ...

```

A table reference is also known as a join expression.

Each table can also be specified as `db_name . tabl_name`. This allows to write queries which involve multiple databases. See [Identifier Qualifiers](#) for syntax details.

The syntax of `table_factor` is extended in comparison with the SQL Standard. The latter accepts only `table_reference`, not a list of them inside a pair of parentheses.

This is a conservative extension if we consider each comma in a list of `table_reference` items as equivalent to an inner join. For example:

```

SELECT * FROM t1 LEFT JOIN (t2, t3, t4)
  ON (t2.a=t1.a AND t3.b=t1.b AND t4.c=t1.c)

```

is equivalent to:

```

SELECT * FROM t1 LEFT JOIN (t2 CROSS JOIN t3 CROSS JOIN t4)
  ON (t2.a=t1.a AND t3.b=t1.b AND t4.c=t1.c)

```

In MariaDB, `CROSS JOIN` is a syntactic equivalent to `INNER JOIN` (they can replace each other). In standard SQL, they are not equivalent. `INNER`

`JOIN` is used with an `ON` clause, `CROSS JOIN` is used otherwise.

In general, parentheses can be ignored in join expressions containing only inner join operations. MariaDB also supports nested joins (see <http://dev.mysql.com/doc/refman/5.1/en/nested-join-optimization.html>).

See [System-versioned tables](#) for more information about `FOR SYSTEM_TIME` syntax.

Index hints can be specified to affect how the MariaDB optimizer makes use of indexes. For more information, see [How to force query plans](#).

Examples

```
SELECT left_tbl.*  
  FROM left_tbl LEFT JOIN right_tbl ON left_tbl.id = right_tbl.id  
 WHERE right_tbl.id IS NULL;
```

LOAD XML

Syntax

```
LOAD XML [LOW_PRIORITY | CONCURRENT] [LOCAL] INFILE 'file_name'  
[REPLACE | IGNORE]  
INTO TABLE [db_name.]tbl_name  
[CHARACTER SET charset_name]  
[ROWS IDENTIFIED BY '<tagname>']  
[IGNORE number {LINES | ROWS}]  
[(column_or_user_var,...)]  
[SET col_name = expr,...]
```

Description

The `LOAD XML` statement reads data from an XML file into a table. The `file_name` must be given as a literal string. The `tagname` in the optional `ROWS IDENTIFIED BY` clause must also be given as a literal string, and must be surrounded by angle brackets (`<` and `>`).

`LOAD XML` acts as the complement of running the [mysql client](#) in XML output mode (that is, starting the client with the `--xml` option). To write data from a table to an XML file, use a command such as the following one from the system shell:

```
shell> mysql --xml -e 'SELECT * FROM mytable' > file.xml
```

To read the file back into a table, use `LOAD XML INFILE`. By default, the `<row>` element is considered to be the equivalent of a database table row; this can be changed using the `ROWS IDENTIFIED BY` clause.

This statement supports three different XML formats:

- Column names as attributes and column values as attribute values:

```
<row column1="value1" column2="value2" ...>
```

- Column names as tags and column values as the content of these tags:

```
<row>  
  <column1>value1</column1>  
  <column2>value2</column2>  
</row>
```

- Column names are the name attributes of `<field>` tags, and values are the contents of these tags:

```
<row>  
  <field name='column1'>value1</field>  
  <field name='column2'>value2</field>  
</row>
```

This is the format used by other tools, such as [mysqldump](#).

All 3 formats can be used in the same XML file; the import routine automatically detects the format for each row and interprets it correctly. Tags are matched based on the tag or attribute name and the column name.

The following clauses work essentially the same way for LOAD XML as they do for LOAD DATA:

- LOW_PRIORITY or CONCURRENT
- LOCAL
- REPLACE or IGNORE
- CHARACTER SET
- (column_or_user_var,...)
- SET

See [LOAD DATA](#) for more information about these clauses.

The IGNORE number LINES or IGNORE number ROWS clause causes the first number rows in the XML file to be skipped. It is analogous to the LOAD DATA statement's IGNORE ... LINES clause.

If the `LOW_PRIORITY` keyword is used, insertions are delayed until no other clients are reading from the table. The `CONCURRENT` keyword allows the use of [concurrent inserts](#). These clauses cannot be specified together.

This statement activates INSERT triggers.

See also

- The `CONNECT` storage engine has an [XML table type](#).

LIMIT

Description

Use the `LIMIT` clause to restrict the number of returned rows. When you use a single integer n with `LIMIT`, the first n rows will be returned. Use the `ORDER BY` clause to control which rows come first. You can also select a number of rows after an offset using either of the following:

```
LIMIT offset, row_count  
LIMIT row_count OFFSET offset
```

Contents

1. [Description](#)
 1. [Multi-Table Updates](#)
 2. [GROUP_CONCAT](#)
2. [Examples](#)
3. [See Also](#)

When you provide an offset m with a limit n , the first m rows will be ignored, and the following n rows will be returned.

Executing an `UPDATE` with the `LIMIT` clause is not safe for replication. `LIMIT 0` is an exception to this rule (see [MDEV-6170](#)).

There is a `LIMIT ROWS EXAMINED` optimization which provides the means to terminate the execution of `SELECT` statements which examine too many rows, and thus use too many resources. See [LIMIT ROWS EXAMINED](#).

Multi-Table Updates

MariaDB starting with 10.3.2

Until MariaDB 10.3.1, it was not possible to use `LIMIT` (or `ORDER BY`) in a multi-table `UPDATE` statement. This restriction was lifted in MariaDB 10.3.2.

GROUP_CONCAT

MariaDB starting with 10.3.2

Starting from MariaDB 10.3.3, it is possible to use `LIMIT` with `GROUP_CONCAT()`.

Examples

```
CREATE TABLE members (name VARCHAR(20));
```

```
INSERT INTO members VALUES('Jagdish'),('Kenny'),('Rokurou'),('Immaculada');
```

```
SELECT * FROM members;
```

name
Jagdish
Kenny
Rokurou
Immaculada

Select the first two names (no ordering specified):

```
SELECT * FROM members LIMIT 2;
```

name
Jagdish
Kenny

All the names in alphabetical order:

```
SELECT * FROM members ORDER BY name;
```

name
Immaculada
Jagdish
Kenny
Rokurou

The first two names, ordered alphabetically:

```
SELECT * FROM members ORDER BY name LIMIT 2;
```

name
Immaculada
Jagdish

The third name, ordered alphabetically (the first name would be offset zero, so the third is offset two):

```
SELECT * FROM members ORDER BY name LIMIT 2,1;
```

name
Kenny

From [MariaDB 10.3.2](#), `LIMIT` can be used in a multi-table update:

```

CREATE TABLE warehouse (product_id INT, qty INT);
INSERT INTO warehouse VALUES (1,100),(2,100),(3,100),(4,100);

CREATE TABLE store (product_id INT, qty INT);
INSERT INTO store VALUES (1,5),(2,5),(3,5),(4,5);

UPDATE warehouse,store SET warehouse.qty = warehouse.qty-2, store.qty = store.qty+2
WHERE (warehouse.product_id = store.product_id AND store.product_id >= 1)
ORDER BY store.product_id DESC LIMIT 2;

SELECT * FROM warehouse;
+-----+
| product_id | qty |
+-----+
|      1 | 100 |
|      2 | 100 |
|      3 |   98 |
|      4 |   98 |
+-----+

SELECT * FROM store;
+-----+
| product_id | qty |
+-----+
|      1 |    5 |
|      2 |    5 |
|      3 |    7 |
|      4 |    7 |
+-----+

```

From MariaDB 10.3.3, `LIMIT` can be used with `GROUP_CONCAT`, so, for example, given the following table:

```

CREATE TABLE d (dd DATE, cc INT);

INSERT INTO d VALUES ('2017-01-01',1);
INSERT INTO d VALUES ('2017-01-02',2);
INSERT INTO d VALUES ('2017-01-04',3);

```

the following query:

```

SELECT SUBSTRING_INDEX(GROUP_CONCAT(CONCAT_WS(":",dd,cc) ORDER BY cc DESC),":",1) FROM d;
+-----+
| SUBSTRING_INDEX(GROUP_CONCAT(CONCAT_WS(":",dd,cc) ORDER BY cc DESC),":",1) |
+-----+
| 2017-01-04:3 |
+-----+

```

can be more simply rewritten as:

```

SELECT GROUP_CONCAT(CONCAT_WS(":",dd,cc) ORDER BY cc DESC LIMIT 1) FROM d;
+-----+
| GROUP_CONCAT(CONCAT_WS(":",dd,cc) ORDER BY cc DESC LIMIT 1) |
+-----+
| 2017-01-04:3 |
+-----+

```

See Also

- [ROWNUM\(\) function](#)
- [SELECT](#)
- [UPDATE](#)
- [DELETE](#)
- [Joins and Subqueries](#)
- [ORDER BY](#)
- [GROUP BY](#)

- Common Table Expressions
- SELECT WITH ROLLUP
- SELECT INTO OUTFILE
- SELECT INTO DUMPFILE
- FOR UPDATE
- LOCK IN SHARE MODE
- Optimizer Hints
- SELECT ... OFFSET ... FETCH

Non-Recursive Common Table Expressions Overview

Common Table Expressions (CTEs) are a standard SQL feature, and are essentially temporary named result sets. There are two kinds of CTEs: Non-Recursive, which this article covers; and Recursive.

MariaDB starting with 10.2.1

Common table expressions were introduced in MariaDB 10.2.1.

Contents

1. Non-Recursive CTEs
 1. A CTE referencing Another CTE
 2. Multiple Uses of a CTE

Non-Recursive CTEs

The `WITH` keyword signifies a CTE. It is given a name, followed by a body (the main query) as follows:

CTEs are similar to derived tables. For example

```
WITH engineers AS
  ( SELECT * FROM employees
    WHERE dept = 'Engineering' )

SELECT * FROM engineers
WHERE ...
```

```
SELECT * FROM
  ( SELECT * FROM employees
    WHERE dept = 'Engineering' ) AS engineers
WHERE
...
```

A non-recursive CTE is basically a query-local `VIEW`. There are several advantages and caveats to them. The syntax is more readable than nested `FROM (SELECT ...)`. A CTE can refer to another and it can be referenced from multiple places.

A CTE referencing Another CTE

Using this format makes for a more readable SQL than a nested `FROM(SELECT ...)` clause. Below is an example of this:

```
WITH engineers AS (
  SELECT * FROM employees
  WHERE dept IN('Development','Support') ),
eu_engineers AS ( SELECT * FROM engineers WHERE country IN('NL',...) )
SELECT
...
FROM eu_engineers;
```

Multiple Uses of a CTE

This can be an 'anti-self join', for example:

```

WITH engineers AS (
SELECT * FROM employees
WHERE dept IN('Development','Support') )

SELECT * FROM engineers E1
WHERE NOT EXISTS
(SELECT 1 FROM engineers E2
 WHERE E2.country=E1.country
 AND E2.name <> E1.name );

```

Or, for year-over-year comparisons, for example:

```

WITH sales_product_year AS (
SELECT product, YEAR(ship_date) AS year,
SUM(price) AS total_amt
FROM item_sales
GROUP BY product, year )

SELECT *
FROM sales_product_year CUR,
sales_product_year PREV,
WHERE CUR.product=PREV.product
AND CUR.year=PREV.year + 1
AND CUR.total_amt > PREV.total_amt

```

Another use is to compare individuals against their group. Below is an example of how this might be executed:

```

WITH sales_product_year AS (
SELECT product,
YEAR(ship_date) AS year,
SUM(price) AS total_amt
FROM item_sales
GROUP BY product, year
)

SELECT *
FROM sales_product_year S1
WHERE
total_amt >
(SELECT 0.1 * SUM(total_amt)
FROM sales_product_year S2
WHERE S2.year = S1.year)

```

LOAD DATA INFILE

Syntax

```

LOAD DATA [LOW_PRIORITY | CONCURRENT] [LOCAL] INFILE 'file_name'
[REPLACE | IGNORE]
INTO TABLE tbl_name
[CHARACTER SET charset_name]
[{{FIELDS | COLUMNS}
[TERMINATED BY 'string']
[[OPTIONALLY] ENCLOSED BY 'char']
[ESCAPED BY 'char']
]
[LINES
[STARTING BY 'string']
[TERMINATED BY 'string']
]
[IGNORE number LINES]
[(col_name_or_user_var,...)]
[SET col_name = expr,...]

```

Description

LOAD DATA INFILE is unsafe for statement-based replication.

Reads rows from a text file into the designated table on the database at a very high speed. The file name must be given as a literal string.

Files are written to disk using the `SELECT INTO OUTFILE` statement. You can then read the files back into a table using the `LOAD DATA INFILE` statement. The `FIELDS` and `LINES` clauses are the same in both statements. These clauses are optional, but if both are specified then the `FIELDS` clause must precede `LINES`.

Executing this statement activates `INSERT` triggers.

One must have the `FILE` privilege to be able to execute `LOAD DATA`. This is to ensure the normal users will not attempt to read system files.

Note that MariaDB's `systemd` unit file restricts access to `/home`, `/root`, and `/run/user` by default. See [Configuring access to home directories](#).

Contents

1. [Syntax](#)
2. [Description](#)
 1. [LOAD DATA LOCAL INFILE](#)
 2. [REPLACE and IGNORE](#)
 3. [Character-sets](#)
 4. [Preprocessing Inputs](#)
 5. [Priority and Concurrency](#)
 6. [Progress Reporting](#)
 7. [Using mariadb-import/mysqlimport](#)
 8. [Indexing](#)
3. [Examples](#)
4. [See Also](#)

LOAD DATA LOCAL INFILE

When you execute the `LOAD DATA INFILE` statement, MariaDB Server attempts to read the input file from its own file system. In contrast, when you execute the `LOAD DATA LOCAL INFILE` statement, the client attempts to read the input file from its file system, and it sends the contents of the input file to the MariaDB Server. This allows you to load files from the client's local file system into the database.

In the event that you don't want to permit this operation (such as for security reasons), you can disable the `LOAD DATA LOCAL INFILE` statement on either the server or the client.

- The `LOAD DATA LOCAL INFILE` statement can be disabled on the server by setting the `local_infile` system variable to `0`.
- The `LOAD DATA LOCAL INFILE` statement can be disabled on the client. If you are using [MariaDB Connector/C](#), this can be done by unsetting the `CLIENT_LOCAL_FILES` capability flag with the `mysql_real_connect` function or by unsetting the `MYSQL_OPT_LOCAL_INFILE` option with `mysql_optionsv` function. If you are using a different client or client library, then see the documentation for your specific client or client library to determine how it handles the `LOAD DATA LOCAL INFILE` statement.

If the `LOAD DATA LOCAL INFILE` statement is disabled by either the server or the client and if the user attempts to execute it, then the server will cause the statement to fail with the following error message:

The used command is not allowed with this MariaDB version

Note that it is not entirely accurate to say that the MariaDB version does not support the command. It would be more accurate to say that the MariaDB configuration does not support the command. See [MDEV-20500](#) for more information.

From [MariaDB 10.5.2](#), the error message is more accurate:

The used command is not allowed because the MariaDB server or client has disabled the local infile capability

REPLACE and IGNORE

In cases where you load data from a file into a table that already contains data and has a `primary key`, you may encounter issues where the statement attempts to insert a row with a primary key that already exists. When this happens, the statement fails with Error 1064, protecting the data already on the table. In cases where you want MariaDB to overwrite duplicates, use the `REPLACE` keyword.

The `REPLACE` keyword works like the `REPLACE` statement. Here, the statement attempts to load the data from the file. If the row does not exist, it adds it to the table. If the row contains an existing Primary Key, it replaces the table data. That is, in the event of a conflict, it assumes the file contains the desired row.

This operation can cause a degradation in load speed by a factor of 20 or more if the part that has already been loaded is larger than the capacity of the `InnoDB Buffer Pool`. This happens because it causes a lot of turnaround in the buffer pool.

Use the `IGNORE` keyword when you want to skip any rows that contain a conflicting primary key. Here, the statement attempts to load the data from the file. If the row does not exist, it adds it to the table. If the row contains an existing primary key, it ignores the addition request and moves on to the next. That is, in the event of a conflict, it assumes the table contains the desired row.

Character-sets

When the statement opens the file, it attempts to read the contents using the default character-set, as defined by the `character_set_database` system variable.

In the cases where the file was written using a character-set other than the default, you can specify the character-set to use with the `CHARACTER SET` clause in the statement. It ignores character-sets specified by the `SET NAMES` statement and by the `character_set_client` system variable. Setting the `CHARACTER SET` clause to a value of `binary` indicates "no conversion."

The statement interprets all fields in the file as having the same character-set, regardless of the column data type. To properly interpret file contents, you must ensure that it was written with the correct character-set. If you write a data file with `mysqldump -T` or with the `SELECT INTO OUTFILE` statement with the `mysql` client, be sure to use the `--default-character-set` option, so that the output is written with the desired character-set.

When using mixed character sets, use the `CHARACTER SET` clause in both `SELECT INTO OUTFILE` and `LOAD DATA INFILE` to ensure that MariaDB correctly interprets the escape sequences.

The `character_set_filesystem` system variable controls the interpretation of the filename.

It is currently not possible to load data files that use the `ucs2` character set.

Preprocessing Inputs

`col_name_or_user_var` can be a column name, or a user variable. In the case of a variable, the `SET` statement can be used to preprocess the value before loading into the table.

Priority and Concurrency

In storage engines that perform table-level locking (`MyISAM`, `MEMORY` and `MERGE`), using the `LOW_PRIORITY` keyword, MariaDB delays insertions until no other clients are reading from the table. Alternatively, when using the `MyISAM` storage engine, you can use the `CONCURRENT` keyword to perform concurrent insertion.

The `LOW_PRIORITY` and `CONCURRENT` keywords are mutually exclusive. They cannot be used in the same statement.

Progress Reporting

The `LOAD DATA INFILE` statement supports `progress reporting`. You may find this useful when dealing with long-running operations. Using another client you can issue a `SHOW PROCESSLIST` query to check the progress of the data load.

Using mariadb-import/mysqlimport

MariaDB ships with a separate utility for loading data from files: `mariadb-import` (or `mysqlimport` before `MariaDB 10.5`). It operates by sending `LOAD DATA INFILE` statements to the server.

Using `mariadb-import/mysqlimport` you can compress the file using the `--compress` option, to get better performance over slow networks, providing both the client and server support the compressed protocol. Use the `--local` option to load from the local file system.

Indexing

In cases where the storage engine supports `ALTER TABLE... DISABLE KEYS` statements (`MyISAM` and `Aria`), the `LOAD DATA INFILE` statement automatically disables indexes during the execution.

Examples

You have a file with this content (note the the separator is ',', not tab, which is the default):

```
2,2
3,3
4,4
5,5
6,8
```

```

CREATE TABLE t1 (a int, b int, c int, d int);
LOAD DATA LOCAL INFILE
  '/tmp/loaddata7.dat' INTO TABLE t1 fields terminated by ',' (a,b) SET c=a+b;
SELECT * FROM t1;
+-----+-----+-----+
| a    | b    | c    |
+-----+-----+-----+
| 2    | 2    | 4    |
| 3    | 3    | 6    |
| 4    | 4    | 8    |
| 5    | 5    | 10   |
| 6    | 8    | 14   |
+-----+-----+-----+

```

Another example, given the following data (the separator is a tab):

```

1      a
2      b

```

The value of the first column is doubled before loading:

```

LOAD DATA INFILE '1d.txt' INTO TABLE ld (@i,v) SET i=@i*2;

SELECT * FROM ld;
+-----+-----+
| i    | v    |
+-----+-----+
| 2    | a    |
| 4    | b    |
+-----+-----+

```

See Also

- [How to quickly insert data into MariaDB](#)
- [Character Sets and Collations](#)
- [SELECT ... INTO OUTFILE](#)
- [mariadb-import/mysqlimport](#)

ORDER BY

Description

Use the `ORDER BY` clause to order a resultset, such as that are returned from a `SELECT` statement. You can specify just a column or use any expression with functions. If you are using the `GROUP BY` clause, you can use grouping functions in `ORDER BY`. Ordering is done after grouping.

You can use multiple ordering expressions, separated by commas. Rows will be sorted by the first expression, then by the second expression if they have the same value for the first, and so on.

You can use the keywords `ASC` and `DESC` after each ordering expression to force that ordering to be ascending or descending, respectively. Ordering is ascending by default.

You can also use a single integer as the ordering expression. If you use an integer *n*, the results will be ordered by the *n*th column in the select expression.

When string values are compared, they are compared as if by the `STRCMP` function. `STRCMP` ignores trailing whitespace and may normalize characters and ignore case, depending on the `collation` in use.

Duplicated entries in the `ORDER BY` clause are removed.

`ORDER BY` can also be used to order the activities of a `DELETE` or `UPDATE` statement (usually with the `LIMIT` clause).

Contents

- [Description](#)
- [Examples](#)
- [See Also](#)

MariaDB starting with 10.3.2

Until MariaDB 10.3.1, it was not possible to use `ORDER BY` (or `LIMIT`) in a multi-table `UPDATE` statement. This restriction was lifted in MariaDB 10.3.2.

MariaDB starting with 10.5

From MariaDB 10.5, MariaDB allows packed sort keys and values of non-sorted fields in the sort buffer. This can make filesort temporary files much smaller when VARCHAR, CHAR or BLOBS are used, notably speeding up some ORDER BY sorts.

Examples

```
CREATE TABLE seq (i INT, x VARCHAR(1));
INSERT INTO seq VALUES (1,'a'), (2,'b'), (3,'b'), (4,'f'), (5,'e');

SELECT * FROM seq ORDER BY i;
+-----+-----+
| i    | x    |
+-----+-----+
| 1    | a    |
| 2    | b    |
| 3    | b    |
| 4    | f    |
| 5    | e    |
+-----+-----+

SELECT * FROM seq ORDER BY i DESC;
+-----+-----+
| i    | x    |
+-----+-----+
| 5    | e    |
| 4    | f    |
| 3    | b    |
| 2    | b    |
| 1    | a    |
+-----+-----+

SELECT * FROM seq ORDER BY x,i;
+-----+-----+
| i    | x    |
+-----+-----+
| 1    | a    |
| 2    | b    |
| 3    | b    |
| 5    | e    |
| 4    | f    |
+-----+-----+
```

ORDER BY in an [UPDATE](#) statement, in conjunction with [LIMIT](#):

```
UPDATE seq SET x='z' WHERE x='b' ORDER BY i DESC LIMIT 1;

SELECT * FROM seq;
+-----+-----+
| i    | x    |
+-----+-----+
| 1    | a    |
| 2    | b    |
| 3    | z    |
| 4    | f    |
| 5    | e    |
+-----+-----+
```

From MariaDB 10.3.2, ORDER BY can be used in a multi-table update:

```

CREATE TABLE warehouse (product_id INT, qty INT);
INSERT INTO warehouse VALUES (1,100),(2,100),(3,100),(4,100);

CREATE TABLE store (product_id INT, qty INT);
INSERT INTO store VALUES (1,5),(2,5),(3,5),(4,5);

UPDATE warehouse,store SET warehouse.qty = warehouse.qty-2, store.qty = store.qty+2
WHERE (warehouse.product_id = store.product_id AND store.product_id >= 1)
ORDER BY store.product_id DESC LIMIT 2;

SELECT * FROM warehouse;
+-----+
| product_id | qty |
+-----+
|      1 | 100 |
|      2 | 100 |
|      3 |   98 |
|      4 |   98 |
+-----+

SELECT * FROM store;
+-----+
| product_id | qty |
+-----+
|      1 |    5 |
|      2 |    5 |
|      3 |    7 |
|      4 |    7 |
+-----+

```

See Also

- [Why is ORDER BY in a FROM subquery ignored?](#)
- [SELECT](#)
- [UPDATE](#)
- [DELETE](#)
- [Improvements to ORDER BY Optimization](#)
- [Joins and Subqueries](#)
- [LIMIT](#)
- [GROUP BY](#)
- [Common Table Expressions](#)
- [SELECT WITH ROLLUP](#)
- [SELECT INTO OUTFILE](#)
- [SELECT INTO DUMPFILE](#)
- [FOR UPDATE](#)
- [LOCK IN SHARE MODE](#)
- [Optimizer Hints](#)

Recursive Common Table Expressions Overview

MariaDB starting with [10.2.2](#)

Recursive Common Table Expressions have been supported since [MariaDB 10.2.2](#).

Common Table Expressions (CTEs) are a standard SQL feature, and are essentially temporary named result sets. CTEs first appeared in the SQL standard in 1999, and the first implementations began appearing in 2007.

There are two kinds of CTEs:

- [Non-recursive](#)
- Recursive, which this article covers.

SQL is generally poor at recursive structures.

CTEs permit a query to reference itself. A recursive CTE will repeatedly execute subsets of the data until it obtains the complete result set. This

makes it particularly useful for handing hierarchical or tree-structured data. [max_recursive_iterations](#) avoids infinite loops.

Contents

1. [Syntax example](#)
2. [Computation](#)
3. [Summary so far](#)
4. [CAST to avoid truncating data](#)
5. [Examples](#)
 1. [Transitive closure - determining bus destinations](#)
 2. [Computing paths - determining bus routes](#)
 3. [CAST to avoid data truncation](#)

Syntax example

WITH RECURSIVE signifies a recursive CTE. It is given a name, followed by a body (the main query) as follows:

Computation

Given the following structure:

First execute the anchor part of the query:

Next, execute the recursive part of the query:

Summary so far

```
with recursive R as (
  select anchor_data
  union [all]
  select recursive_part
    from R, ...
)
select ...
```

1. Compute anchor_data
2. Compute recursive_part to get the new data
3. if (new data is non-empty) goto 2;

CAST to avoid truncating data

As currently implemented by MariaDB and by the SQL Standard, data may be truncated if not correctly cast. It is necessary to **CAST** the column to the correct width if the CTE's recursive part produces wider values for a column than the CTE's nonrecursive part. Some other DBMS give an error in this situation, and MariaDB's behavior may change in future - see [MDEV-12325](#). See the examples below.

Examples

Transitive closure - determining bus destinations

Sample data:

```
CREATE TABLE bus_routes (origin varchar(50), dst varchar(50));
INSERT INTO bus_routes VALUES
  ('New York', 'Boston'),
  ('Boston', 'New York'),
  ('New York', 'Washington'),
  ('Washington', 'Boston'),
  ('Washington', 'Raleigh');
```

Now, we want to return the bus destinations with New York as the origin:

```

WITH RECURSIVE bus_dst AS (
    SELECT origin AS dst FROM bus_routes WHERE origin='New York'
UNION
    SELECT bus_routes.dst FROM bus_routes JOIN bus_dst ON bus_dst.dst= bus_routes.origin
)
SELECT * FROM bus_dst;
+-----+
| dst   |
+-----+
| New York |
| Boston  |
| Washington |
| Raleigh |
+-----+

```

The above example is computed as follows:

First, the anchor data is calculated:

- Starting from New York
- Boston and Washington are added

Next, the recursive part:

- Starting from Boston and then Washington
- Raleigh is added
- UNION excludes nodes that are already present.

Computing paths - determining bus routes

This time, we are trying to get bus routes such as “New York -> Washington -> Raleigh”.

Using the same sample data as the previous example:

```

WITH RECURSIVE paths (cur_path, cur_dest) AS (
    SELECT origin, origin FROM bus_routes WHERE origin='New York'
UNION
    SELECT CONCAT(paths.cur_path, ',', bus_routes.dst), bus_routes.dst
    FROM paths
    JOIN bus_routes
        ON paths.cur_dest = bus_routes.origin AND
        NOT FIND_IN_SET(bus_routes.dst, paths.cur_path)
)
SELECT * FROM paths;
+-----+-----+
| cur_path          | cur_dest   |
+-----+-----+
| New York          | New York   |
| New York,Boston   | Boston     |
| New York,Washington | Washington |
| New York,Washington,Boston | Boston     |
| New York,Washington,Raleigh | Raleigh   |
+-----+-----+

```

CAST to avoid data truncation

In the following example, data is truncated because the results are not specifically cast to a wide enough type:

```

WITH RECURSIVE tbl AS (
  SELECT NULL AS col
  UNION
  SELECT "THIS NEVER SHOWS UP" AS col FROM tbl
)
SELECT col FROM tbl
+-----+
| col |
+-----+
| NULL |
|      |
+-----+

```

Explicitly use [CAST](#) to overcome this:

```

WITH RECURSIVE tbl AS (
  SELECT CAST(NULL AS CHAR(50)) AS col
  UNION SELECT "THIS NEVER SHOWS UP" AS col FROM tbl
)
SELECT * FROM tbl;
+-----+
| col      |
+-----+
| NULL    |
| THIS NEVER SHOWS UP |
+-----+

```

SELECT INTO DUMPFILE

Syntax

```
SELECT ... INTO DUMPFILE 'file_path'
```

Contents

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

Description

`SELECT ... INTO DUMPFILE` is a [SELECT](#) clause which writes the resultset into a single unformatted row, without any separators, in a file. The results will not be returned to the client.

file_path can be an absolute path, or a relative path starting from the data directory. It can only be specified as a [string literal](#), not as a variable. However, the statement can be dynamically composed and executed as a prepared statement to work around this limitation.

This statement is binary-safe and so is particularly useful for writing [BLOB](#) values to file. It can be used, for example, to copy an image or an audio document from the database to a file. `SELECT ... INTO FILE` can be used to save a text file.

The file must not exist. It cannot be overwritten. A user needs the [FILE](#) privilege to run this statement. Also, MariaDB needs permission to write files in the specified location. If the [secure_file_priv](#) system variable is set to a non-empty directory name, the file can only be written to that directory.

Since [MariaDB 5.1](#), the [character_set_filesystem](#) system variable has controlled interpretation of file names that are given as literal strings.

Example

```

SELECT _utf8'Hello world!' INTO DUMPFILE '/tmp/world';

SELECT LOAD_FILE('/tmp/world') AS world;
+-----+
| world      |
+-----+
| Hello world! |
+-----+

```

See Also

- [SELECT](#)
- [LOAD_FILE\(\)](#)
- [SELECT INTO Variable](#)
- [SELECT INTO OUTFILE](#)

SELECT

Syntax

```

SELECT
    [ALL | DISTINCT | DISTINCTROW]
    [HIGH_PRIORITY]
    [STRAIGHT_JOIN]
    [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
    [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
    select_expr [, select_expr ...]
    [ FROM table_references
        [WHERE where_condition]
        [GROUP BY {col_name | expr | position} [ASC | DESC], ... [WITH ROLLUP]]
        [HAVING where_condition]
        [ORDER BY {col_name | expr | position} [ASC | DESC], ...]
        [LIMIT {[offset,] row_count | row_count OFFSET offset [ROWS EXAMINED rows_limit]} | 
         [OFFSET start { ROW | ROWS }]
         [FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } { ONLY | WITH TIES } ] ]
        procedure|[PROCEDURE procedure_name(argument_list)]
        [INTO OUTFILE 'file_name' [CHARACTER SET charset_name] [export_options] |
         INTO DUMPFILE 'file_name' | INTO var_name [, var_name] ]
        [FOR UPDATE lock_option | LOCK IN SHARE MODE lock_option]

    export_options:
        [{FIELDS | COLUMNS}
            [TERMINATED BY 'string']
            [[OPTIONALLY] ENCLOSED BY 'char']
            [ESCAPED BY 'char']
        ]
        [LINES
            [STARTING BY 'string']
            [TERMINATED BY 'string']
        ]
    ]

    lock_option:
        [WAIT n | NOWAIT | SKIP LOCKED]

```

Description

`SELECT` is used to retrieve rows selected from one or more tables, and can include [UNION](#) statements and [subqueries](#).

- Each `select_expr` expression indicates a column or data that you want to retrieve. You must have at least one select expression. See [Select Expressions](#) below.
- The `FROM` clause indicates the table or tables from which to retrieve rows. Use either a single table name or a `JOIN` expression. See [JOIN](#) for details. If no table is involved, `FROM DUAL` can be specified.
- Each table can also be specified as `db_name . tabl_name`. Each column can also be specified as `tbl_name . col_name` or even `db_name . tabl_name . col_name`. This allows one to write queries which involve multiple databases. See [Identifier Qualifiers](#) for syntax details.
- The `WHERE` clause, if given, indicates the condition or conditions that rows must satisfy to be selected. `where_condition` is an expression that evaluates to true for each row to be selected. The statement selects all rows if there is no `WHERE` clause.
 - In the `WHERE` clause, you can use any of the functions and operators that MariaDB supports, except for aggregate (summary) functions. See [Functions and Operators](#) and [Functions and Modifiers for use with GROUP BY](#) (aggregate).
- Use the `ORDER BY` clause to order the results.
- Use the `LIMIT` clause allows you to restrict the results to only a certain number of rows, optionally with an offset.

- Use the [GROUP BY](#) and [HAVING](#) clauses to group rows together when they have columns or computed values in common.

SELECT can also be used to retrieve rows computed without reference to any table.

Select Expressions

A SELECT statement must contain one or more select expressions, separated by commas. Each select expression can be one of the following:

- The name of a column.
- Any expression using [functions and operators](#).
- * to select all columns from all tables in the FROM clause.
- `tbl_name.*` to select all columns from just the table `tbl_name`.

When specifying a column, you can either use just the column name or qualify the column name with the name of the table using `tbl_name.col_name`. The qualified form is useful if you are joining multiple tables in the FROM clause. If you do not qualify the column names when selecting from multiple tables, MariaDB will try to find the column in each table. It is an error if that column name exists in multiple tables.

You can quote column names using backticks. If you are qualifying column names with table names, quote each part separately as ``tbl_name`.`col_name``.

If you use any [grouping functions](#) in any of the select expressions, all rows in your results will be implicitly grouped, as if you had used GROUP BY NULL .

DISTINCT

A query may produce some identical rows. By default, all rows are retrieved, even when their values are the same. To explicitly specify that you want to retrieve identical rows, use the ALL option. If you want duplicates to be removed from the resultset, use the DISTINCT option. DISTINCTROW is a synonym for DISTINCT . See also [COUNT DISTINCT](#) and [SELECT UNIQUE in Oracle mode](#).

INTO

The INTO clause is used to specify that the query results should be written to a file or variable.

- [SELECT INTO OUTFILE](#) - formatting and writing the result to an external file.
- [SELECT INTO DUMPFILE](#) - binary-safe writing of the unformatted results to an external file.
- [SELECT INTO Variable](#) - selecting and setting variables.

The reverse of SELECT INTO OUTFILE is [LOAD DATA](#).

LIMIT

Restricts the number of returned rows. See [LIMIT](#) and [LIMIT ROWS EXAMINED](#) for details.

LOCK IN SHARE MODE/FOR UPDATE

See [LOCK IN SHARE MODE](#) and [FOR UPDATE](#) for details on the respective locking clauses.

OFFSET ... FETCH

MariaDB starting with 10.6

See [SELECT ... OFFSET ... FETCH](#).

ORDER BY

Order a resultset. See [ORDER BY](#) for details.

PARTITION

Contents

1. [Syntax](#)
2. [Description](#)
 1. [Select Expressions](#)
 2. [DISTINCT](#)
 3. [INTO](#)
 4. [LIMIT](#)
 5. [LOCK IN SHARE MODE/FOR UPDATE](#)
 6. [OFFSET ... FETCH](#)
 7. [ORDER BY](#)
 8. [PARTITION](#)
 9. [PROCEDURE](#)
 10. [SKIP LOCKED](#)
 11. [SQL_CALC_FOUND_ROWS](#)
 12. [max_statement_time clause](#)
 13. [WAIT/NOWAIT](#)
3. [Examples](#)
4. [See Also](#)

Specifies to the optimizer which partitions are relevant for the query. Other partitions will not be read. See [Partition Pruning and Selection](#) for details.

PROCEDURE

Passes the whole result set to a C Procedure. See [PROCEDURE](#) and [PROCEDURE ANALYSE](#) (the only built-in procedure not requiring the server to be recompiled).

SKIP LOCKED

MariaDB starting with 10.6

The SKIP LOCKED clause was introduced in [MariaDB 10.6.0](#).

This causes those rows that couldn't be locked ([LOCK IN SHARE MODE](#) or [FOR UPDATE](#)) to be excluded from the result set. An explicit `NOWAIT` is implied here. This is only implemented on [InnoDB](#) tables and ignored otherwise.

SQL_CALC_FOUND_ROWS

When `SQL_CALC_FOUND_ROWS` is used, then MariaDB will calculate how many rows would have been in the result, if there would be no [LIMIT](#) clause. The result can be found by calling the function [FOUND_ROWS\(\)](#) in your next sql statement.

max_statement_time clause

By using `max_statement_time` in conjunction with [SET STATEMENT](#), it is possible to limit the execution time of individual queries. For example:

```
SET STATEMENT max_statement_time=100 FOR
    SELECT field1 FROM table_name ORDER BY field1;
```

WAIT/NOWAIT

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

Examples

```
SELECT f1,f2 FROM t1 WHERE (f3<=10) AND (f4='y');
```

See [Getting Data from MariaDB](#) (Beginner tutorial), or the various sub-articles, for more examples.

See Also

- [Getting Data from MariaDB](#) (Beginner tutorial)
- [Joins and Subqueries](#)
- [LIMIT](#)
- [ORDER BY](#)
- [GROUP BY](#)
- [Common Table Expressions](#)
- [SELECT WITH ROLLUP](#)
- [SELECT INTO OUTFILE](#)
- [SELECT INTO DUMPFILE](#)
- [FOR UPDATE](#)
- [LOCK IN SHARE MODE](#)
- [Optimizer Hints](#)
- [Oracle mode from MariaDB 10.3](#)

SELECT INTO OUTFILE

Syntax

```
SELECT ... INTO OUTFILE 'file_name'  
    [CHARACTER SET charset_name]  
    [export_options]  
  
export_options:  
    [{FIELDS | COLUMNS}  
        [TERMINATED BY 'string']  
        [[OPTIONALLY] ENCLOSED BY 'char']  
        [ESCAPED BY 'char']  
    ]  
    [LINES  
        [STARTING BY 'string']  
        [TERMINATED BY 'string']  
    ]
```

Description

`SELECT INTO OUTFILE` writes the resulting rows to a file, and allows the use of column and row terminators to specify a particular output format. The default is to terminate fields with tabs (`\t`) and lines with newlines (`\n`).

The file must not exist. It cannot be overwritten. A user needs the `FILE` privilege to run this statement. Also, MariaDB needs permission to write files in the specified location. If the `secure_file_priv` system variable is set to a non-empty directory name, the file can only be written to that directory.

The `LOAD DATA INFILE` statement complements `SELECT INTO OUTFILE`.

Character-sets

The `CHARACTER SET` clause specifies the `character set` in which the results are to be written. Without the clause, no conversion takes place (the binary character set). In this case, if there are multiple character sets, the output will contain these too, and may not easily be able to be reloaded.

In cases where you have two servers using different character-sets, using `SELECT INTO OUTFILE` to transfer data from one to the other can have unexpected results. To ensure that MariaDB correctly interprets the escape sequences, use the `CHARACTER SET` clause on both the `SELECT INTO OUTFILE` statement and the subsequent `LOAD DATA INFILE` statement.

Example

The following example produces a file in the CSV format:

```
SELECT customer_id, firstname, surname INTO OUTFILE '/exportdata/customers.txt'  
    FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '\"'  
    LINES TERMINATED BY '\n'  
    FROM customers;
```

See Also

- [SELECT](#)
- [LOAD_DATA\(\)](#) function
- [LOAD DATA INFILE](#)
- [SELECT INTO Variable](#)
- [SELECT INTO DUMPFILE](#)

SELECT WITH ROLLUP

Syntax

See [SELECT](#) for the full syntax.

Contents

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

Description

The `WITH ROLLUP` modifier adds extra rows to the resultset that represent super-aggregate summaries. The super-aggregated column is represented by a `NULL` value. Multiple aggregates over different columns will be added if there are multiple `GROUP BY` columns.

The `LIMIT` clause can be used at the same time, and is applied after the `WITH ROLLUP` rows have been added.

`WITH ROLLUP` cannot be used with `ORDER BY`. Some sorting is still possible by using `ASC` or `DESC` clauses with the `GROUP BY` column, although the super-aggregate rows will always be added last.

Examples

These examples use the following sample table

```
CREATE TABLE booksales (
    country VARCHAR(35), genre ENUM('fiction','non-fiction'), year YEAR, sales INT);

INSERT INTO booksales VALUES
    ('Senegal','fiction',2014,12234), ('Senegal','fiction',2015,15647),
    ('Senegal','non-fiction',2014,64980), ('Senegal','non-fiction',2015,78901),
    ('Paraguay','fiction',2014,87970), ('Paraguay','fiction',2015,76940),
    ('Paraguay','non-fiction',2014,8760), ('Paraguay','non-fiction',2015,9030);
```

The addition of the `WITH ROLLUP` modifier in this example adds an extra row that aggregates both years:

```
SELECT year, SUM(sales) FROM booksales GROUP BY year;
+-----+
| year | SUM(sales) |
+-----+
| 2014 | 173944 |
| 2015 | 180518 |
+-----+
2 rows in set (0.08 sec)

SELECT year, SUM(sales) FROM booksales GROUP BY year WITH ROLLUP;
+-----+
| year | SUM(sales) |
+-----+
| 2014 | 173944 |
| 2015 | 180518 |
| NULL | 354462 |
+-----+
```

In the following example, each time the genre, the year or the country change, another super-aggregate row is added:

Contents

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

```

SELECT country, year, genre, SUM(sales)
  FROM booksales GROUP BY country, year, genre;
+-----+-----+-----+-----+
| country | year | genre      | SUM(sales) |
+-----+-----+-----+-----+
| Paraguay | 2014 | fiction    | 87970      |
| Paraguay | 2014 | non-fiction | 8760       |
| Paraguay | 2015 | fiction    | 76940      |
| Paraguay | 2015 | non-fiction | 9030       |
| Senegal  | 2014 | fiction    | 12234      |
| Senegal  | 2014 | non-fiction | 64980      |
| Senegal  | 2015 | fiction    | 15647      |
| Senegal  | 2015 | non-fiction | 78901      |
+-----+-----+-----+-----+

```



```

SELECT country, year, genre, SUM(sales)
  FROM booksales GROUP BY country, year, genre WITH ROLLUP;
+-----+-----+-----+-----+
| country | year | genre      | SUM(sales) |
+-----+-----+-----+-----+
| Paraguay | 2014 | fiction    | 87970      |
| Paraguay | 2014 | non-fiction | 8760       |
| Paraguay | 2014 | NULL       | 96730      |
| Paraguay | 2015 | fiction    | 76940      |
| Paraguay | 2015 | non-fiction | 9030       |
| Paraguay | 2015 | NULL       | 85970      |
| Paraguay | NULL  | NULL       | 182700     |
| Senegal  | 2014 | fiction    | 12234      |
| Senegal  | 2014 | non-fiction | 64980      |
| Senegal  | 2014 | NULL       | 77214      |
| Senegal  | 2015 | fiction    | 15647      |
| Senegal  | 2015 | non-fiction | 78901      |
| Senegal  | 2015 | NULL       | 94548      |
| Senegal  | NULL  | NULL       | 171762     |
| NULL    | NULL  | NULL       | 354462     |
+-----+-----+-----+-----+

```

The LIMIT clause, applied after WITH ROLLUP:

```
SELECT country, year, genre, SUM(sales)
  FROM booksales GROUP BY country, year, genre WITH ROLLUP LIMIT 4;
```

country	year	genre	SUM(sales)
Paraguay	2014	fiction	87970
Paraguay	2014	non-fiction	8760
Paraguay	2014	NULL	96730
Paraguay	2015	fiction	76940

Sorting by year descending:

```

SELECT country, year, genre, SUM(sales)
  FROM booksales GROUP BY country, year DESC, genre WITH ROLLUP;
+-----+-----+-----+
| country | year | genre      | SUM(sales) |
+-----+-----+-----+
| Paraguay | 2015 | fiction    |    76940   |
| Paraguay | 2015 | non-fiction |    9030    |
| Paraguay | 2015 | NULL       |    85970   |
| Paraguay | 2014 | fiction    |    87970   |
| Paraguay | 2014 | non-fiction |    8760    |
| Paraguay | 2014 | NULL       |    96730   |
| Paraguay | NULL  | NULL       |   182700   |
| Senegal  | 2015 | fiction    |    15647   |
| Senegal  | 2015 | non-fiction |    78901   |
| Senegal  | 2015 | NULL       |    94548   |
| Senegal  | 2014 | fiction    |    12234   |
| Senegal  | 2014 | non-fiction |    64980   |
| Senegal  | 2014 | NULL       |    77214   |
| Senegal  | NULL  | NULL       |   171762   |
| NULL    | NULL  | NULL       |   354462   |
+-----+-----+-----+

```

See Also

- [SELECT](#)
- [Joins and Subqueries](#)
- [LIMIT](#)
- [ORDER BY](#)
- [GROUP BY](#)
- [Common Table Expressions](#)
- [SELECT INTO OUTFILE](#)
- [SELECT INTO DUMPFILE](#)
- [FOR UPDATE](#)
- [LOCK IN SHARE MODE](#)
- [Optimizer Hints](#)

UNION

`UNION` is used to combine the results from multiple `SELECT` statements into a single result set.

Syntax

```

SELECT ...
UNION [ALL | DISTINCT] SELECT ...
[UNION [ALL | DISTINCT] SELECT ...]
[ORDER BY [column [, column ...]]]
[LIMIT {[offset,] row_count | row_count OFFSET offset}]

```

Description

`UNION` is used to combine the results from multiple `SELECT` statements into a single result set.

The column names from the first `SELECT` statement are used as the column names for the results returned. Selected columns listed in corresponding positions of each `SELECT` statement should have the same data type. (For example, the first column selected by the first statement should have the same type as the first column selected by the other statements.)

If they don't, the type and length of the columns in the result take into account the values returned by all of the `SELECT`s, so there is no need for explicit casting. Note that currently this is not the case for [recursive CTEs](#) - see [MDEV-12325](#).

Table names can be specified as `db_name . tbl_name`. This permits writing `UNION`s which involve multiple databases. See [Identifier Qualifiers](#) for syntax details.

Contents

1. [Syntax](#)
2. [Description](#)
 1. [ALL/DISTINCT](#)
 2. [ORDER BY and LIMIT](#)
 3. [HIGH_PRIORITY](#)
 4. [SELECT ... INTO ...](#)
 5. [Parentheses](#)
3. [Examples](#)
4. [See Also](#)

UNION queries cannot be used with aggregate functions.

EXCEPT and UNION have the same operation precedence and INTERSECT has a higher precedence, unless running in Oracle mode, in which case all three have the same precedence.

ALL/DISTINCT

The ALL keyword causes duplicate rows to be preserved. The DISTINCT keyword (the default if the keyword is omitted) causes duplicate rows to be removed by the results.

UNION ALL and UNION DISTINCT can both be present in a query. In this case, UNION DISTINCT will override any UNION ALLs to its left.

MariaDB starting with 10.1.1

Until MariaDB 10.1.1, all UNION ALL statements required the server to create a temporary table. Since MariaDB 10.1.1, the server can in most cases execute UNION ALL without creating a temporary table, improving performance (see MDEV-334).

ORDER BY and LIMIT

Individual SELECTs can contain their own ORDER BY and LIMIT clauses. In this case, the individual queries need to be wrapped between parentheses. However, this does not affect the order of the UNION, so they only are useful to limit the record read by one SELECT.

The UNION can have global ORDER BY and LIMIT clauses, which affect the whole resultset. If the columns retrieved by individual SELECT statements have an alias (AS), the ORDER BY must use that alias, not the real column names.

HIGH_PRIORITY

Specifying a query as HIGH_PRIORITY will not work inside a UNION. If applied to the first SELECT, it will be ignored. Applying to a later SELECT results in a syntax error:

```
ERROR 1234 (42000): Incorrect usage/placement of 'HIGH_PRIORITY'
```

SELECT ... INTO ...

Individual SELECTs cannot be written INTO DUMPFILE or INTO OUTFILE. If the last SELECT statement specifies INTO DUMPFILE or INTO OUTFILE, the entire result of the UNION will be written. Placing the clause after any other SELECT will result in a syntax error.

If the result is a single row, SELECT ... INTO @var_name can also be used.

MariaDB starting with 10.4.0

Parentheses

From MariaDB 10.4.0, parentheses can be used to specify precedence. Before this, a syntax error would be returned.

Examples

UNION between tables having different column names:

```
(SELECT e_name AS name, email FROM employees)
UNION
(SELECT c_name AS name, email FROM customers);
```

Specifying the UNION's global order and limiting total rows:

```
(SELECT name, email FROM employees)
UNION
(SELECT name, email FROM customers)
ORDER BY name LIMIT 10;
```

Adding a constant row:

```
(SELECT 'John Doe' AS name, 'john.doe@example.net' AS email)
UNION
(SELECT name, email FROM customers);
```

Differing types:

```
SELECT CAST('x' AS CHAR(1)) UNION SELECT REPEAT('y',4);
+-----+
| CAST('x' AS CHAR(1)) |
+-----+
| x                      |
| yyyy                  |
+-----+
```

Returning the results in order of each individual SELECT by use of a sort column:

```
(SELECT 1 AS sort_column, e_name AS name, email FROM employees)
UNION
(SELECT 2, c_name AS name, email FROM customers) ORDER BY sort_column;
```

Difference between UNION, EXCEPT and INTERSECT. INTERSECT ALL and EXCEPT ALL are available from MariaDB 10.5.0.

```

CREATE TABLE seqs (i INT);
INSERT INTO seqs VALUES (1),(2),(2),(3),(3),(4),(5),(6);

SELECT i FROM seqs WHERE i <= 3 UNION SELECT i FROM seqs WHERE i>=3;
+----+
| i |
+----+
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
| 6 |
+----+

SELECT i FROM seqs WHERE i <= 3 UNION ALL SELECT i FROM seqs WHERE i>=3;
+----+
| i |
+----+
| 1 |
| 2 |
| 2 |
| 3 |
| 3 |
| 3 |
| 3 |
| 4 |
| 5 |
| 6 |
+----+

SELECT i FROM seqs WHERE i <= 3 EXCEPT SELECT i FROM seqs WHERE i>=3;
+----+
| i |
+----+
| 1 |
| 2 |
+----+

SELECT i FROM seqs WHERE i <= 3 EXCEPT ALL SELECT i FROM seqs WHERE i>=3;
+----+
| i |
+----+
| 1 |
| 2 |
| 2 |
+----+

SELECT i FROM seqs WHERE i <= 3 INTERSECT SELECT i FROM seqs WHERE i>=3;
+----+
| i |
+----+
| 3 |
+----+

SELECT i FROM seqs WHERE i <= 3 INTERSECT ALL SELECT i FROM seqs WHERE i>=3;
+----+
| i |
+----+
| 3 |
| 3 |
+----+

```

Parentheses for specifying precedence, from [MariaDB 10.4.0](#)

```

CREATE OR REPLACE TABLE t1 (a INT);
CREATE OR REPLACE TABLE t2 (b INT);
CREATE OR REPLACE TABLE t3 (c INT);

INSERT INTO t1 VALUES (1),(2),(3),(4);
INSERT INTO t2 VALUES (5),(6);
INSERT INTO t3 VALUES (1),(6);

((SELECT a FROM t1) UNION (SELECT b FROM t2)) INTERSECT (SELECT c FROM t3);
+----+
| a |
+----+
| 1 |
| 6 |
+----+

(SELECT a FROM t1) UNION ((SELECT b FROM t2) INTERSECT (SELECT c FROM t3));
+----+
| a |
+----+
| 1 |
| 2 |
| 3 |
| 4 |
| 6 |
+----+

```

See Also

- [SELECT](#)
- [EXCEPT](#)
- [INTERSECT](#)
- [Recursive Common Table Expressions Overview](#)
- [Get Set for Set Theory: UNION, INTERSECT and EXCEPT in SQL \(video tutorial\)](#)

WITH

MariaDB starting with [10.2.1](#)

Common Table Expressions were introduced in [MariaDB 10.2.1](#).

Syntax

```

WITH [RECURSIVE] table_reference [(columns_list)] AS (
    SELECT ...
)
[CYCLE cycle_column_list RESTRICT]
SELECT ...

```

Description

The `WITH` keyword signifies a [Common Table Expression](#) (CTE). It allows you to refer to a subquery expression many times in a query, as if having a temporary table that only exists for the duration of a query.

There are two kinds of CTEs:

- [Non-Recursive](#)
- [Recursive](#) (signified by the `RECURSIVE` keyword, supported since [MariaDB 10.2.2](#))

You can use `table_reference` as any normal table in the external `SELECT` part. You can also use `WITH` in subqueries, as well as with `EXPLAIN` and `SELECT`.

Poorly-formed recursive CTEs can in theory cause infinite loops. The `max_recursive_iterations` system variable limits the number of recursions.

Contents

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

CYCLE ... RESTRICT

MariaDB starting with 10.5.2

The CYCLE clause enables CTE cycle detection, avoiding excessive or infinite loops, MariaDB supports a relaxed, non-standard grammar.

The SQL Standard permits a CYCLE clause, as follows:

```
WITH RECURSIVE ... (
  ...
)
CYCLE <cycle column list>
SET <cycle mark column> TO <cycle mark value> DEFAULT <non-cycle mark value>
USING <path column>
```

where all clauses are mandatory.

MariaDB does not support this, but from 10.5.2 permits a non-standard relaxed grammar, as follows:

```
WITH RECURSIVE ... (
  ...
)
CYCLE <cycle column list> RESTRICT
```

With the use of CYCLE ... RESTRICT it makes no difference whether the CTE uses UNION ALL or UNION DISTINCT anymore. UNION ALL means "all rows, but without cycles", which is exactly what the CYCLE clause enables. And UNION DISTINCT means all rows should be different, which, again, is what will happen — as uniqueness is enforced over a subset of columns, complete rows will automatically all be different.

Examples

Below is an example with the WITH at the top level:

```
WITH t AS (SELECT a FROM t1 WHERE b >= 'c')
  SELECT * FROM t2, t WHERE t2.c = t.a;
```

The example below uses WITH in a subquery:

```
SELECT t1.a, t1.b FROM t1, t2
  WHERE t1.a > t2.c
    AND t2.c IN(WITH t AS (SELECT * FROM t1 WHERE t1.a < 5)
      SELECT t2.c FROM t2, t WHERE t2.c = t.a);
```

Below is an example of a Recursive CTE:

```
WITH RECURSIVE ancestors AS
( SELECT * FROM folks
  WHERE name='Alex'
  UNION
  SELECT f.*
    FROM folks AS f, ancestors AS a
    WHERE f.id = a.father OR f.id = a.mother )
SELECT * FROM ancestors;
```

Take the following structure, and data,

```

CREATE TABLE t1 (from_ int, to_ int);
INSERT INTO t1 VALUES (1,2), (1,100), (2,3), (3,4), (4,1);
SELECT * FROM t1;
+-----+-----+
| from_ | to_  |
+-----+-----+
|    1  |    2  |
|    1  |   100 |
|    2  |    3  |
|    3  |    4  |
|    4  |    1  |
+-----+-----+

```

Given the above, the following query would theoretically result in an infinite loop due to the last record in t1 (note that `max_recursive_iterations` is set to 10 for the purposes of this example, to avoid the excessive number of cycles):

```

SET max_recursive_iterations=10;

WITH RECURSIVE cte (depth, from_, to_) AS (
  SELECT 0,1,1 UNION DISTINCT SELECT depth+1, t1.from_, t1.to_
  FROM t1, cte WHERE t1.from_ = cte.to_
)
SELECT * FROM cte;
+-----+-----+-----+
| depth | from_ | to_  |
+-----+-----+-----+
|    0  |    1  |    1  |
|    1  |    1  |    2  |
|    1  |    1  |   100 |
|    2  |    2  |    3  |
|    3  |    3  |    4  |
|    4  |    4  |    1  |
|    5  |    1  |    2  |
|    5  |    1  |   100 |
|    6  |    2  |    3  |
|    7  |    3  |    4  |
|    8  |    4  |    1  |
|    9  |    1  |    2  |
|    9  |    1  |   100 |
|   10  |    2  |    3  |
+-----+-----+-----+

```

However, the CYCLE ... RESTRICT clause (from MariaDB 10.5.2) can overcome this:

```

WITH RECURSIVE cte (depth, from_, to_) AS (
  SELECT 0,1,1 UNION SELECT depth+1, t1.from_, t1.to_
  FROM t1, cte WHERE t1.from_ = cte.to_
)
CYCLE from_, to_ RESTRICT
SELECT * FROM cte;
+-----+-----+-----+
| depth | from_ | to_  |
+-----+-----+-----+
|    0  |    1  |    1  |
|    1  |    1  |    2  |
|    1  |    1  |   100 |
|    2  |    2  |    3  |
|    3  |    3  |    4  |
|    4  |    4  |    1  |
+-----+-----+-----+

```

See Also

- [Non-Recursive Common Table Expressions Overview](#)
- [Recursive Common Table Expressions Overview](#)

DESCRIBE

Syntax

```
{DESCRIBE | DESC} tbl_name [col_name | wild]
```

Description

`DESCRIBE` provides information about the columns in a table. It is a shortcut for `SHOW COLUMNS FROM`. These statements also display information for `views`.

`col_name` can be a column name, or a string containing the SQL "%" and "_" wildcard characters to obtain output only for the columns with names matching the string. There is no need to enclose the string within quotes unless it contains spaces or other special characters.

Contents

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

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

The description for `SHOW COLUMNS` provides more information about the output columns.

See Also

- [SHOW COLUMNS](#)
- [INFORMATION_SCHEMA.COLUMNS Table](#)
- [mysqlshow](#)

CROSSES

Syntax

```
Crosses(g1,g2)
```

Description

Returns 1 if `g1` spatially crosses `g2`. Returns `NULL` if `g1` is a `Polygon` or a `MultiPolygon`, or if `g2` is a `Point` or a `MultiPoint`. Otherwise, returns 0.

The term spatially crosses denotes a spatial relation between two given geometries that has the following properties:

- The two geometries intersect
- Their intersection results in a geometry that has a dimension that is one less than the maximum dimension of the two given geometries
- Their intersection is not equal to either of the two given geometries

`CROSSES()` is based on the original MySQL implementation, and uses object bounding rectangles, while `ST_CROSSES()` uses object shapes.

ST_CONTAINS

Syntax

```
ST_CONTAINS(g1,g2)
```

Description

Returns 1 or 0 to indicate whether a geometry g1 completely contains geometry g2 .

ST_CONTAINS() uses object shapes, while [CONTAINS\(\)](#), based on the original MySQL implementation, uses object bounding rectangles.

ST_CONTAINS tests the opposite relationship to [ST_WITHIN\(\)](#).

Examples

```
SET @g1 = ST_GEOFROMTEXT('POLYGON((175 150, 20 40, 50 60, 125 100, 175 150))');

SET @g2 = ST_GEOFROMTEXT('POINT(174 149)');

SELECT ST_CONTAINS(@g1,@g2);
+-----+
| ST_CONTAINS(@g1,@g2) |
+-----+
|          1 |
+-----+

SET @g2 = ST_GEOFROMTEXT('POINT(175 151)');

SELECT ST_CONTAINS(@g1,@g2);
+-----+
| ST_CONTAINS(@g1,@g2) |
+-----+
|          0 |
+-----+
```

ST_CROSSES

Syntax

```
ST_CROSSES(g1,g2)
```

Description

Returns 1 if geometry g1 spatially crosses geometry g2 . Returns NULL if g1 is a [Polygon](#) or a [MultiPolygon](#), or if g2 is a [Point](#) or a [MultiPoint](#). Otherwise, returns 0 .

The term spatially crosses denotes a spatial relation between two given geometries that has the following properties:

- The two geometries intersect
- Their intersection results in a geometry that has a dimension that is one less than the maximum dimension of the two given geometries
- Their intersection is not equal to either of the two given geometries

ST_CROSSES() uses object shapes, while [CROSSES\(\)](#), based on the original MySQL implementation, uses object bounding rectangles.

Examples

```

SET @g1 = ST_GEOFROMTEXT('LINESTRING(174 149, 176 151)');
SET @g2 = ST_GEOFROMTEXT('POLYGON((175 150, 20 40, 50 60, 125 100, 175 150))');

SELECT ST_CROSSES(@g1,@g2);
+-----+
| ST_CROSSES(@g1,@g2) |
+-----+
|          1          |
+-----+

SET @g1 = ST_GEOFROMTEXT('LINESTRING(176 149, 176 151)');
SELECT ST_CROSSES(@g1,@g2);
+-----+
| ST_CROSSES(@g1,@g2) |
+-----+
|          0          |
+-----+

```

ST_DISJOINT

Syntax

```
ST_DISJOINT(g1,g2)
```

Description

Returns `1` or `0` to indicate whether geometry `g1` is spatially disjoint from (does not intersect with) geometry `g2`.

`ST_DISJOINT()` uses object shapes, while `DISJOINT()`, based on the original MySQL implementation, uses object bounding rectangles.

`ST_DISJOINT()` tests the opposite relationship to `ST_INTERSECTS()`.

Examples

```

SET @g1 = ST_GEOFROMTEXT('POINT(0 0)');
SET @g2 = ST_GEOFROMTEXT('LINESTRING(2 0, 0 2)');

SELECT ST_DISJOINT(@g1,@g2);
+-----+
| ST_DISJOINT(@g1,@g2) |
+-----+
|          1          |
+-----+

SET @g2 = ST_GEOFROMTEXT('LINESTRING(0 0, 0 2)');
SELECT ST_DISJOINT(@g1,@g2);
+-----+
| ST_DISJOINT(@g1,@g2) |
+-----+
|          0          |
+-----+

```

ST_EQUALS

Syntax

```
ST_EQUALS(g1,g2)
```

Description

Returns `1` or `0` to indicate whether geometry `g1` is spatially equal to geometry `g2`.

`ST_EQUALS()` uses object shapes, while [EQUALS\(\)](#), based on the original MySQL implementation, uses object bounding rectangles.

Examples

```
SET @g1 = ST_GEOFROMTEXT('LINESTRING(174 149, 176 151)');
SET @g2 = ST_GEOFROMTEXT('LINESTRING(176 151, 174 149)');
SELECT ST_EQUALS(@g1,@g2);
+-----+
| ST_EQUALS(@g1,@g2) |
+-----+
|          1          |
+-----+
```

```
SET @g1 = ST_GEOFROMTEXT('POINT(0 2)');
SET @g1 = ST_GEOFROMTEXT('POINT(2 0)');
SELECT ST_EQUALS(@g1,@g2);
+-----+
| ST_EQUALS(@g1,@g2) |
+-----+
|          0          |
+-----+
```

ST_INTERSECTS

Syntax

```
ST_INTERSECTS(g1,g2)
```

Description

Returns `1` or `0` to indicate whether geometry `g1` spatially intersects geometry `g2`.

`ST_INTERSECTS()` uses object shapes, while [INTERSECTS\(\)](#), based on the original MySQL implementation, uses object bounding rectangles.

`ST_INTERSECTS()` tests the opposite relationship to [ST_DISJOINT\(\)](#).

Examples

```
SET @g1 = ST_GEOFROMTEXT('POINT(0 0)');
SET @g2 = ST_GEOFROMTEXT('LINESTRING(0 0, 0 2)');
SELECT ST_INTERSECTS(@g1,@g2);
+-----+
| ST_INTERSECTS(@g1,@g2) |
+-----+
|          1          |
+-----+
```

```
SET @g2 = ST_GEOMFROMTEXT('LINESTRING(2 0, 0 2)';

SELECT ST_INTERSECTS(@g1,@g2);
+-----+
| ST_INTERSECTS(@g1,@g2) |
+-----+
|          0 |
+-----+
```

ST_TOUCHES

Syntax

```
ST_TOUCHES(g1,g2)
```

Description

Returns `1` or `0` to indicate whether geometry g_1 spatially touches geometry g_2 . Two geometries spatially touch if the interiors of the geometries do not intersect, but the boundary of one of the geometries intersects either the boundary or the interior of the other.

`ST_TOUCHES()` uses object shapes, while [TOUCHES\(\)](#), based on the original MySQL implementation, uses object bounding rectangles.

Examples

```
SET @g1 = ST_GEOMFROMTEXT('POINT(2 0)');
SET @g2 = ST_GEOMFROMTEXT('LINESTRING(2 0, 0 2)';

SELECT ST_TOUCHES(@g1,@g2);
+-----+
| ST_TOUCHES(@g1,@g2) |
+-----+
|          1 |
+-----+

SET @g1 = ST_GEOMFROMTEXT('POINT(2 1)');

SELECT ST_TOUCHES(@g1,@g2);
+-----+
| ST_TOUCHES(@g1,@g2) |
+-----+
|          0 |
+-----+
```

ST_WITHIN

Syntax

```
ST_WITHIN(g1,g2)
```

Description

Returns `1` or `0` to indicate whether geometry g_1 is spatially within geometry g_2 .

This tests the opposite relationship as [ST_CONTAINS\(\)](#).

`ST_WITHIN()` uses object shapes, while [WITHIN\(\)](#), based on the original MySQL implementation, uses object bounding rectangles.

Examples

```
SET @g1 = ST_GEOMFROMTEXT('POINT(174 149)');
SET @g2 = ST_GEOMFROMTEXT('POLYGON((175 150, 20 40, 50 60, 125 100, 175 150))');
SELECT ST_WITHIN(@g1,@g2);
+-----+
| ST_WITHIN(@g1,@g2) |
+-----+
|          1 |
+-----+
SET @g1 = ST_GEOMFROMTEXT('POINT(176 151)');
SELECT ST_WITHIN(@g1,@g2);
+-----+
| ST_WITHIN(@g1,@g2) |
+-----+
|          0 |
+-----+
```

WITHIN

Syntax

```
Within(g1,g2)
```

Description

Returns `1` or `0` to indicate whether `g1` is spatially within `g2`. This tests the opposite relationship as [Contains\(\)](#).

`WITHIN()` is based on the original MySQL implementation, and uses object bounding rectangles, while [ST_WITHIN\(\)](#) uses object shapes.

Examples

```
SET @g1 = GEOMFROMTEXT('POINT(174 149)');
SET @g2 = GEOMFROMTEXT('POINT(176 151)');
SET @g3 = GEOMFROMTEXT('POLYGON((175 150, 20 40, 50 60, 125 100, 175 150))');

SELECT within(@g1,@g3);
+-----+
| within(@g1,@g3) |
+-----+
|          1 |
+-----+
SELECT within(@g2,@g3);
+-----+
| within(@g2,@g3) |
+-----+
|          0 |
+-----+
```

ADDDATE

Syntax

```
ADDDATE(date,INTERVAL expr unit), ADDDATE(expr,days)
```

Description

When invoked with the `INTERVAL` form of the second argument, `ADDDATE()` is a synonym for `DATE_ADD()`. The related function `SUBDATE()` is a synonym for `DATE_SUB()`. For information on the `INTERVAL` unit argument, see the discussion for `DATE_ADD()`.

When invoked with the days form of the second argument, MariaDB treats it as an integer number of days to be added to `expr`.

Examples

```
SELECT DATE_ADD('2008-01-02', INTERVAL 31 DAY);
+-----+
| DATE_ADD('2008-01-02', INTERVAL 31 DAY) |
+-----+
| 2008-02-02 |
+-----+
```

```
SELECT ADDDATE('2008-01-02', INTERVAL 31 DAY);
+-----+
| ADDDATE('2008-01-02', INTERVAL 31 DAY) |
+-----+
| 2008-02-02 |
+-----+
```

```
SELECT ADDDATE('2008-01-02', 31);
+-----+
| ADDDATE('2008-01-02', 31) |
+-----+
| 2008-02-02 |
+-----+
```

```
CREATE TABLE t1 (d DATETIME);
INSERT INTO t1 VALUES
    ("2007-01-30 21:31:07"),
    ("1983-10-15 06:42:51"),
    ("2011-04-21 12:34:56"),
    ("2011-10-30 06:31:41"),
    ("2011-01-30 14:03:25"),
    ("2004-10-07 11:19:34");
```

```
SELECT d, ADDDATE(d, 10) from t1;
+-----+
| d           | ADDDATE(d, 10)   |
+-----+
| 2007-01-30 21:31:07 | 2007-02-09 21:31:07 |
| 1983-10-15 06:42:51 | 1983-10-25 06:42:51 |
| 2011-04-21 12:34:56 | 2011-05-01 12:34:56 |
| 2011-10-30 06:31:41 | 2011-11-09 06:31:41 |
| 2011-01-30 14:03:25 | 2011-02-09 14:03:25 |
| 2004-10-07 11:19:34 | 2004-10-17 11:19:34 |
+-----+
```

```
SELECT d, ADDDATE(d, INTERVAL 10 HOUR) from t1;
+-----+
| d           | ADDDATE(d, INTERVAL 10 HOUR) |
+-----+
| 2007-01-30 21:31:07 | 2007-01-31 07:31:07 |
| 1983-10-15 06:42:51 | 1983-10-15 16:42:51 |
| 2011-04-21 12:34:56 | 2011-04-21 22:34:56 |
| 2011-10-30 06:31:41 | 2011-10-30 16:31:41 |
| 2011-01-30 14:03:25 | 2011-01-31 00:03:25 |
| 2004-10-07 11:19:34 | 2004-10-07 21:19:34 |
+-----+
```

ADDTIME

Syntax

```
ADDTIME(expr1,expr2)
```

Description

ADDTIME() adds *expr2* to *expr1* and returns the result. *expr1* is a time or datetime expression, and *expr2* is a time expression.

Examples

```
SELECT ADDTIME('2007-12-31 23:59:59.999999', '1 1:1:1.000002');
+-----+
| ADDTIME('2007-12-31 23:59:59.999999', '1 1:1:1.000002') |
+-----+
| 2008-01-02 01:01:01.000001 |
+-----+

SELECT ADDTIME('01:00:00.999999', '02:00:00.999998');
+-----+
| ADDTIME('01:00:00.999999', '02:00:00.999998') |
+-----+
| 03:00:01.999997 |
+-----+
```

CONVERT_TZ

Syntax

```
CONVERT_TZ(dt,from_tz,to_tz)
```

Description

CONVERT_TZ() converts a datetime value *dt* from the [time zone](#) given by *from_tz* to the time zone given by *to_tz* and returns the resulting value.

In order to use named time zones, such as GMT, MET or Africa/Johannesburg, the `time_zone` tables must be loaded (see [mysql_tzinfo_to_sql](#)).

No conversion will take place if the value falls outside of the supported `TIMESTAMP` range ('1970-01-01 00:00:01' to '2038-01-19 05:14:07' UTC) when converted from *from_tz* to UTC.

This function returns NULL if the arguments are invalid (or named time zones have not been loaded).

See [time zones](#) for more information.

Examples

```
SELECT CONVERT_TZ('2016-01-01 12:00:00','+00:00','+10:00');
+-----+
| CONVERT_TZ('2016-01-01 12:00:00','+00:00','+10:00') |
+-----+
| 2016-01-01 22:00:00 |
+-----+
```

Using named time zones (with the time zone tables loaded):

```
SELECT CONVERT_TZ('2016-01-01 12:00:00','GMT','Africa/Johannesburg');
+-----+
| CONVERT_TZ('2016-01-01 12:00:00','GMT','Africa/Johannesburg') |
+-----+
| 2016-01-01 14:00:00 |
+-----+
```

The value is out of the TIMESTAMP range, so no conversion takes place:

```
SELECT CONVERT_TZ('1969-12-31 22:00:00','+00:00','+10:00');
+-----+
| CONVERT_TZ('1969-12-31 22:00:00','+00:00','+10:00') |
+-----+
| 1969-12-31 22:00:00 |
+-----+
```

CURDATE

Syntax

```
CURDATE()
CURRENT_DATE
CURRENT_DATE()
```

Description

`CURDATE` returns the current date as a value in 'YYYY-MM-DD' or YYYYMMDD format, depending on whether the function is used in a string or numeric context.

`CURRENT_DATE` and `CURRENT_DATE()` are synonyms.

Examples

```
SELECT CURDATE();
+-----+
| CURDATE() |
+-----+
| 2019-03-05 |
+-----+
```

In a numeric context (note this is not performing date calculations):

```
SELECT CURDATE() +0;
+-----+
| CURDATE() +0 |
+-----+
| 20190305 |
+-----+
```

Data calculation:

```
SELECT CURDATE() - INTERVAL 5 DAY;
+-----+
| CURDATE() - INTERVAL 5 DAY |
+-----+
| 2019-02-28 |
+-----+
```

DATEDIFF

Syntax

```
DATEDIFF(expr1,expr2)
```

Description

DATEDIFF() returns (*expr1 – expr2*) expressed as a value in days from one date to the other. *expr1* and *expr2* are date or date-and-time expressions. Only the date parts of the values are used in the calculation.

Examples

```
SELECT DATEDIFF('2007-12-31 23:59:59','2007-12-30');
+-----+
| DATEDIFF('2007-12-31 23:59:59','2007-12-30') |
+-----+
| 1 |
+-----+

SELECT DATEDIFF('2010-11-30 23:59:59','2010-12-31');
+-----+
| DATEDIFF('2010-11-30 23:59:59','2010-12-31') |
+-----+
| -31 |
+-----+
```

```
CREATE TABLE t1 (d DATETIME);
INSERT INTO t1 VALUES
    ("2007-01-30 21:31:07"),
    ("1983-10-15 06:42:51"),
    ("2011-04-21 12:34:56"),
    ("2011-10-30 06:31:41"),
    ("2011-01-30 14:03:25"),
    ("2004-10-07 11:19:34");
```

```
SELECT NOW();
+-----+
| NOW()           |
+-----+
| 2011-05-23 10:56:05 |
+-----+

SELECT d, DATEDIFF(NOW(),d) FROM t1;
+-----+-----+
| d           | DATEDIFF(NOW(),d) |
+-----+-----+
| 2007-01-30 21:31:07 |      1574 |
| 1983-10-15 06:42:51 |     10082 |
| 2011-04-21 12:34:56 |       32 |
| 2011-10-30 06:31:41 |      -160 |
| 2011-01-30 14:03:25 |       113 |
| 2004-10-07 11:19:34 |      2419 |
+-----+-----+
```

DATE_ADD

Syntax

```
DATE_ADD(date,INTERVAL expr unit)
```

Description

Performs date arithmetic. The *date* argument specifies the starting date or datetime value. *expr* is an expression specifying the interval value to be added or subtracted from the starting date. *expr* is a string; it may start with a " - " for negative intervals. *unit* is a keyword indicating the units in which the expression should be interpreted. See [Date and Time Units](#) for a complete list of permitted units.

Contents

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

Examples

```
SELECT '2008-12-31 23:59:59' + INTERVAL 1 SECOND;  
+-----+  
| '2008-12-31 23:59:59' + INTERVAL 1 SECOND |  
+-----+  
| 2009-01-01 00:00:00 |  
+-----+
```

```
SELECT INTERVAL 1 DAY + '2008-12-31';  
+-----+  
| INTERVAL 1 DAY + '2008-12-31' |  
+-----+  
| 2009-01-01 |  
+-----+
```

```
SELECT '2005-01-01' - INTERVAL 1 SECOND;  
+-----+  
| '2005-01-01' - INTERVAL 1 SECOND |  
+-----+  
| 2004-12-31 23:59:59 |  
+-----+
```

```
SELECT DATE_ADD('2000-12-31 23:59:59', INTERVAL 1 SECOND);  
+-----+  
| DATE_ADD('2000-12-31 23:59:59', INTERVAL 1 SECOND) |  
+-----+  
| 2001-01-01 00:00:00 |  
+-----+
```

```
SELECT DATE_ADD('2010-12-31 23:59:59', INTERVAL 1 DAY);  
+-----+  
| DATE_ADD('2010-12-31 23:59:59', INTERVAL 1 DAY) |  
+-----+  
| 2011-01-01 23:59:59 |  
+-----+
```

```
SELECT DATE_ADD('2100-12-31 23:59:59', INTERVAL '1:1' MINUTE_SECOND);  
+-----+  
| DATE_ADD('2100-12-31 23:59:59', INTERVAL '1:1' MINUTE_SECOND) |  
+-----+  
| 2101-01-01 00:01:00 |  
+-----+
```

```
SELECT DATE_ADD('1900-01-01 00:00:00', INTERVAL '-1 10' DAY_HOUR);  
+-----+  
| DATE_ADD('1900-01-01 00:00:00', INTERVAL '-1 10' DAY_HOUR) |  
+-----+  
| 1899-12-30 14:00:00 |  
+-----+
```

```

SELECT DATE_ADD('1992-12-31 23:59:59.000002', INTERVAL '1.999999' SECOND_MICROSECOND);
+-----+
| DATE_ADD('1992-12-31 23:59:59.000002', INTERVAL '1.999999' SECOND_MICROSECOND) |
+-----+
| 1993-01-01 00:00:01.000001 |
+-----+

```

See Also

- [DATE_SUB](#)
- [ADD_MONTHS](#)

DATE_FORMAT

Syntax

```
DATE_FORMAT(date, format[, locale])
```

Description

Formats the date value according to the format string.

The language used for the names is controlled by the value of the [lc_time_names](#) system variable. See [server locale](#) for more on the supported locales.

The options that can be used by DATE_FORMAT(), as well as its inverse [STR_TO_DATE\(\)](#) and the [FROM_UNIXTIME\(\)](#) function, are:

Contents

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

Option	Description
%a	Short weekday name in current locale (Variable lc_time_names).
%b	Short form month name in current locale. For locale en_US this is one of: Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov or Dec.
%c	Month with 1 or 2 digits.
%D	Day with English suffix 'th', 'nd', 'st' or 'rd'. (1st, 2nd, 3rd...).
%d	Day with 2 digits.
%e	Day with 1 or 2 digits.
%f	Microseconds 6 digits.
%H	Hour with 2 digits between 00-23.
%h	Hour with 2 digits between 01-12.
%I	Hour with 2 digits between 01-12.
%i	Minute with 2 digits.
%j	Day of the year (001-366)
%k	Hour with 1 digits between 0-23.
%l	Hour with 1 digits between 1-12.
%M	Full month name in current locale (Variable lc_time_names).
%m	Month with 2 digits.
%p	AM/PM according to current locale (Variable lc_time_names).
%r	Time in 12 hour format, followed by AM/PM. Short for '%l:%i:%S %p'.
%S	Seconds with 2 digits.
%s	Seconds with 2 digits.
%T	Time in 24 hour format. Short for '%H:%i:%S'.

%U	Week number (00-53), when first day of the week is Sunday.
%u	Week number (00-53), when first day of the week is Monday.
%V	Week number (01-53), when first day of the week is Sunday. Used with %X.
%v	Week number (01-53), when first day of the week is Monday. Used with %x.
%W	Full weekday name in current locale (Variable <code>lc_time_names</code>).
%w	Day of the week. 0 = Sunday, 6 = Saturday.
%X	Year with 4 digits when first day of the week is Sunday. Used with %V.
%x	Year with 4 digits when first day of the week is Monday. Used with %v.
%Y	Year with 4 digits.
%y	Year with 2 digits.
%#	For <code>str_to_date()</code> , skip all numbers.
%.	For <code>str_to_date()</code> , skip all punctuation characters.
%@	For <code>str_to_date()</code> , skip all alpha characters.
%%	A literal % character.

To get a date in one of the standard formats, `GET_FORMAT()` can be used.

Examples

```

SELECT DATE_FORMAT('2009-10-04 22:23:00', '%W %M %Y');
+-----+
| DATE_FORMAT('2009-10-04 22:23:00', '%W %M %Y') |
+-----+
| Sunday October 2009 |
+-----+

SELECT DATE_FORMAT('2007-10-04 22:23:00', '%H:%i:%s');
+-----+
| DATE_FORMAT('2007-10-04 22:23:00', '%H:%i:%s') |
+-----+
| 22:23:00 |
+-----+

SELECT DATE_FORMAT('1900-10-04 22:23:00', '%D %y %a %d %m %b %j');
+-----+
| DATE_FORMAT('1900-10-04 22:23:00', '%D %y %a %d %m %b %j') |
+-----+
| 4th 00 Thu 04 10 Oct 277 |
+-----+

SELECT DATE_FORMAT('1997-10-04 22:23:00', '%H %k %I %r %T %S %w');
+-----+
| DATE_FORMAT('1997-10-04 22:23:00', '%H %k %I %r %T %S %w') |
+-----+
| 22 22 10 10:23:00 PM 22:23:00 00 6 |
+-----+

SELECT DATE_FORMAT('1999-01-01', '%X %V');
+-----+
| DATE_FORMAT('1999-01-01', '%X %V') |
+-----+
| 1998 52 |
+-----+

SELECT DATE_FORMAT('2006-06-00', '%d');
+-----+
| DATE_FORMAT('2006-06-00', '%d') |
+-----+
| 00 |
+-----+

```

MariaDB starting with 10.3.2

Optionally, the locale can be explicitly specified as the third DATE_FORMAT() argument. Doing so makes the function independent from the session settings, and the three argument version of DATE_FORMAT() can be used in virtual indexed and persistent generated-columns:

```

SELECT DATE_FORMAT('2006-01-01', '%W', 'el_GR');
+-----+
| DATE_FORMAT('2006-01-01', '%W', 'el_GR') |
+-----+
| Κυριακή |
+-----+

```

See Also

- [STR_TO_DATE\(\)](#)
- [FROM_UNIXTIME\(\)](#)

DATE_SUB

Syntax

```
DATE_SUB(date,INTERVAL expr unit)
```

Description

Performs date arithmetic. The *date* argument specifies the starting date or datetime value. *expr* is an expression specifying the interval value to be added or subtracted from the starting date. *expr* is a string; it may start with a " - " for negative intervals. *unit* is a keyword indicating the units in which the expression should be interpreted. See [Date and Time Units](#) for a complete list of permitted units.

See also [DATE_ADD\(\)](#).

Examples

```
SELECT DATE_SUB('1998-01-02', INTERVAL 31 DAY);
+-----+
| DATE_SUB('1998-01-02', INTERVAL 31 DAY) |
+-----+
| 1997-12-02                                |
+-----+
```

```
SELECT DATE_SUB('2005-01-01 00:00:00', INTERVAL '1 1:1:1' DAY_SECOND);
+-----+-----+
| DATE_SUB('2005-01-01 00:00:00', INTERVAL '1 1:1:1' DAY_SECOND) |
+-----+-----+
| 2004-12-30 22:58:59                         |
+-----+-----+
```

DAYNAME

Syntax

```
DAYNAME(date)
```

Description

Returns the name of the weekday for date. The language used for the name is controlled by the value of the [lc_time_names](#) system variable. See [server locale](#) for more on the supported locales.

Examples

```
SELECT DAYNAME('2007-02-03');
+-----+
| DAYNAME('2007-02-03') |
+-----+
| Saturday                |
+-----+
```

```
CREATE TABLE t1 (d DATETIME);
INSERT INTO t1 VALUES
    ("2007-01-30 21:31:07"),
    ("1983-10-15 06:42:51"),
    ("2011-04-21 12:34:56"),
    ("2011-10-30 06:31:41"),
    ("2011-01-30 14:03:25"),
    ("2004-10-07 11:19:34");
```

```

SELECT d, DAYNAME(d) FROM t1;
+-----+-----+
| d | DAYNAME(d) |
+-----+-----+
| 2007-01-30 21:31:07 | Tuesday |
| 1983-10-15 06:42:51 | Saturday |
| 2011-04-21 12:34:56 | Thursday |
| 2011-10-30 06:31:41 | Sunday |
| 2011-01-30 14:03:25 | Sunday |
| 2004-10-07 11:19:34 | Thursday |
+-----+-----+

```

Changing the locale:

```

SET lc_time_names = 'fr_CA';

SELECT DAYNAME('2013-04-01');
+-----+
| DAYNAME('2013-04-01') |
+-----+
| lundi |
+-----+

```

DAYOFMONTH

Syntax

```
DAYOFMONTH(date)
```

Description

Returns the day of the month for date, in the range 1 to 31, or 0 for dates such as '0000-00-00' or '2008-00-00' which have a zero day part.

DAY() is a synonym.

Examples

```

SELECT DAYOFMONTH('2007-02-03');
+-----+
| DAYOFMONTH('2007-02-03') |
+-----+
| 3 |
+-----+

```

```

CREATE TABLE t1 (d DATETIME);
INSERT INTO t1 VALUES
    ("2007-01-30 21:31:07"),
    ("1983-10-15 06:42:51"),
    ("2011-04-21 12:34:56"),
    ("2011-10-30 06:31:41"),
    ("2011-01-30 14:03:25"),
    ("2004-10-07 11:19:34");

```

```
SELECT d FROM t1 WHERE DAYOFMONTH(d) = 30;
+-----+
| d           |
+-----+
| 2007-01-30 21:31:07 |
| 2011-10-30 06:31:41 |
| 2011-01-30 14:03:25 |
+-----+
```

DAYOFWEEK

Syntax

```
DAYOFWEEK(date)
```

Description

Returns the day of the week index for the date (1 = Sunday, 2 = Monday, ..., 7 = Saturday). These index values correspond to the ODBC standard.

This contrasts with `WEEKDAY()` which follows a different index numbering (0 = Monday, 1 = Tuesday, ... 6 = Sunday).

Examples

```
SELECT DAYOFWEEK('2007-02-03');
+-----+
| DAYOFWEEK('2007-02-03') |
+-----+
| 7 |
+-----+
```

```
CREATE TABLE t1 (d DATETIME);
INSERT INTO t1 VALUES
    ("2007-01-30 21:31:07"),
    ("1983-10-15 06:42:51"),
    ("2011-04-21 12:34:56"),
    ("2011-10-30 06:31:41"),
    ("2011-01-30 14:03:25"),
    ("2004-10-07 11:19:34");
```

```
SELECT d, DAYNAME(d), DAYOFWEEK(d), WEEKDAY(d) FROM t1;
+-----+-----+-----+-----+
| d           | DAYNAME(d) | DAYOFWEEK(d) | WEEKDAY(d) |
+-----+-----+-----+-----+
| 2007-01-30 21:31:07 | Tuesday     | 3           | 1           |
| 1983-10-15 06:42:51 | Saturday    | 7           | 5           |
| 2011-04-21 12:34:56 | Thursday    | 5           | 3           |
| 2011-10-30 06:31:41 | Sunday      | 1           | 6           |
| 2011-01-30 14:03:25 | Sunday      | 1           | 6           |
| 2004-10-07 11:19:34 | Thursday    | 5           | 3           |
+-----+-----+-----+-----+
```

EXTRACT

Syntax

```
EXTRACT(unit FROM date)
```

Description

The EXTRACT() function extracts the required unit from the date. See [Date and Time Units](#) for a complete list of permitted units.

In [MariaDB 10.0.7](#) and [MariaDB 5.5.35](#), EXTRACT (HOUR FROM ...) was changed to return a value from 0 to 23, adhering to the SQL standard. Until [MariaDB 10.0.6](#) and [MariaDB 5.5.34](#), and in all versions of MySQL at least as of MySQL 5.7, it could return a value > 23. HOUR() is not a standard function, so continues to adhere to the old behaviour inherited from MySQL.

Examples

```
SELECT EXTRACT(YEAR FROM '2009-07-02');
+-----+
| EXTRACT(YEAR FROM '2009-07-02') |
+-----+
|          2009 |
+-----+


SELECT EXTRACT(YEAR_MONTH FROM '2009-07-02 01:02:03');
+-----+
| EXTRACT(YEAR_MONTH FROM '2009-07-02 01:02:03') |
+-----+
|          200907 |
+-----+


SELECT EXTRACT(DAY_MINUTE FROM '2009-07-02 01:02:03');
+-----+
| EXTRACT(DAY_MINUTE FROM '2009-07-02 01:02:03') |
+-----+
|          20102 |
+-----+


SELECT EXTRACT(MICROSECOND FROM '2003-01-02 10:30:00.000123');
+-----+
| EXTRACT(MICROSECOND FROM '2003-01-02 10:30:00.000123') |
+-----+
|          123 |
+-----+
```

From [MariaDB 10.0.7](#) and [MariaDB 5.5.35](#), EXTRACT (HOUR FROM...) returns a value from 0 to 23, as per the SQL standard. HOUR is not a standard function, so continues to adhere to the old behaviour inherited from MySQL.

```
SELECT EXTRACT(HOUR FROM '26:30:00'), HOUR('26:30:00');
+-----+-----+
| EXTRACT(HOUR FROM '26:30:00') | HOUR('26:30:00') |
+-----+-----+
|          2 |          26 |
+-----+-----+
```

See Also

- [Date and Time Units](#)
- [Date and Time Literals](#)
- [HOUR\(\)](#)

FROM_UNIXTIME

Syntax

```
FROM_UNIXTIME(unix_timestamp), FROM_UNIXTIME(unix_timestamp,format)
```

Contents

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

Description

Returns a representation of the unix_timestamp argument as a value in 'YYYY-MM-DD HH:MM:SS' or YYYYMMDDHHMMSS.uduuuu format, depending on whether the function is used in a string or numeric context. The value is expressed in the current [time zone](#). unix_timestamp is an internal timestamp value such as is produced by the [UNIX_TIMESTAMP\(\)](#) function.

If format is given, the result is formatted according to the format string, which is used the same way as listed in the entry for the [DATE_FORMAT\(\)](#) function.

Contents

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

Timestamps in MariaDB have a maximum value of 2147483647, equivalent to 2038-01-19 05:14:07. This is due to the underlying 32-bit limitation. Using the function on a timestamp beyond this will result in NULL being returned. Use [DATETIME](#) as a storage type if you require dates beyond this.

The options that can be used by FROM_UNIXTIME(), as well as [DATE_FORMAT\(\)](#) and [STR_TO_DATE\(\)](#), are:

Option	Description
%a	Short weekday name in current locale (Variable lc_time_names).
%b	Short form month name in current locale. For locale en_US this is one of: Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov or Dec.
%c	Month with 1 or 2 digits.
%D	Day with English suffix 'th', 'nd', 'st' or 'rd'. (1st, 2nd, 3rd...).
%d	Day with 2 digits.
%e	Day with 1 or 2 digits.
%f	Microseconds 6 digits.
%H	Hour with 2 digits between 00-23.
%h	Hour with 2 digits between 01-12.
%I	Hour with 2 digits between 01-12.
%i	Minute with 2 digits.
%j	Day of the year (001-366)
%k	Hour with 1 digits between 0-23.
%l	Hour with 1 digits between 1-12.
%M	Full month name in current locale (Variable lc_time_names).
%m	Month with 2 digits.
%p	AM/PM according to current locale (Variable lc_time_names).
%r	Time in 12 hour format, followed by AM/PM. Short for '%l:%i:%S %p'.
%S	Seconds with 2 digits.
%s	Seconds with 2 digits.
%T	Time in 24 hour format. Short for '%H:%i:%S'.
%U	Week number (00-53), when first day of the week is Sunday.
%u	Week number (00-53), when first day of the week is Monday.
%v	Week number (01-53), when first day of the week is Sunday. Used with %X.
%v	Week number (01-53), when first day of the week is Monday. Used with %x.
%W	Full weekday name in current locale (Variable lc_time_names).
%w	Day of the week. 0 = Sunday, 6 = Saturday.
%X	Year with 4 digits when first day of the week is Sunday. Used with %V.
%x	Year with 4 digits when first day of the week is Sunday. Used with %v.

%Y	Year with 4 digits.
%y	Year with 2 digits.
%#	For <code>str_to_date()</code> , skip all numbers.
%.	For <code>str_to_date()</code> , skip all punctuation characters.
%@	For <code>str_to_date()</code> , skip all alpha characters.
%%	A literal % character.

Performance Considerations

If your `session time zone` is set to `SYSTEM` (the default), `FROM_UNIXTIME()` will call the OS function to convert the data using the system time zone. At least on Linux, the corresponding function (`localtime_r`) uses a global mutex inside glibc that can cause contention under high concurrent load.

Set your time zone to a named time zone to avoid this issue. See [mysql time zone tables](#) for details on how to do this.

Examples

```
SELECT FROM_UNIXTIME(1196440219);
+-----+
| FROM_UNIXTIME(1196440219) |
+-----+
| 2007-11-30 11:30:19 |
+-----+

SELECT FROM_UNIXTIME(1196440219) + 0;
+-----+
| FROM_UNIXTIME(1196440219) + 0 |
+-----+
| 20071130113019.000000 |
+-----+

SELECT FROM_UNIXTIME(UNIX_TIMESTAMP(), '%Y %D %M %h:%i:%s %x');
+-----+
| FROM_UNIXTIME(UNIX_TIMESTAMP(), '%Y %D %M %h:%i:%s %x') |
+-----+
| 2010 27th March 01:03:47 2010 |
+-----+
```

See Also

- [UNIX_TIMESTAMP\(\)](#)
- [DATE_FORMAT\(\)](#)
- [STR_TO_DATE\(\)](#)

GET_FORMAT

Syntax

```
GET_FORMAT({DATE|DATETIME|TIME}, {'EUR' | 'USA' | 'JIS' | 'ISO' | 'INTERNAL'})
```

Description

Returns a format string. This function is useful in combination with the `DATE_FORMAT()` and the `STR_TO_DATE()` functions.

Possible result formats are:

Function Call	Result Format
GET_FORMAT(DATE,'EUR')	'%d.%m.%Y'

GET_FORMAT(DATE,'USA')	'%m.%d.%Y'
GET_FORMAT(DATE,'JIS')	'%Y-%m-%d'
GET_FORMAT(DATE,'ISO')	'%Y-%m-%d'
GET_FORMAT(DATE,'INTERNAL')	'%Y%m%d'
GET_FORMAT(DATETIME,'EUR')	'%Y-%m-%d %H.%i.%s'
GET_FORMAT(DATETIME,'USA')	'%Y-%m-%d %H.%i.%s'
GET_FORMAT(DATETIME,'JIS')	'%Y-%m-%d %H:%i:%s'
GET_FORMAT(DATETIME,'ISO')	'%Y-%m-%d %H:%i:%s'
GET_FORMAT(DATETIME,'INTERNAL')	'%Y%m%d%H%i%s'
GET_FORMAT(TIME,'EUR')	'%H.%i.%s'
GET_FORMAT(TIME,'USA')	'%h.%i.%s %p'
GET_FORMAT(TIME,'JIS')	'%H.%i.%s'
GET_FORMAT(TIME,'ISO')	'%H.%i.%s'
GET_FORMAT(TIME,'INTERNAL')	'%H%i%s'

Examples

Obtaining the string matching to the standard European date format:

```
SELECT GET_FORMAT(DATE, 'EUR');
+-----+
| GET_FORMAT(DATE, 'EUR') |
+-----+
| %d.%m.%Y |
+-----+
```

Using the same string to format a date:

```
SELECT DATE_FORMAT('2003-10-03',GET_FORMAT(DATE,'EUR'));
+-----+
| DATE_FORMAT('2003-10-03',GET_FORMAT(DATE,'EUR')) |
+-----+
| 03.10.2003 |
+-----+

SELECT STR_TO_DATE('10.31.2003',GET_FORMAT(DATE,'USA'));
+-----+
| STR_TO_DATE('10.31.2003',GET_FORMAT(DATE,'USA')) |
+-----+
| 2003-10-31 |
+-----+
```

HOUR

Syntax

```
HOUR(time)
```

Description

Returns the hour for time. The range of the return value is 0 to 23 for time-of-day values. However, the range of `TIME` values actually is much larger, so `HOUR` can return values greater than 23.

The return value is always positive, even if a negative `TIME` value is provided.

Examples

```
SELECT HOUR('10:05:03');
+-----+
| HOUR('10:05:03') |
+-----+
|          10 |
+-----+

SELECT HOUR('272:59:59');
+-----+
| HOUR('272:59:59') |
+-----+
|         272 |
+-----+
```

Difference between `EXTRACT (HOUR FROM ...)` (\geq MariaDB 10.0.7 and MariaDB 5.5.35) and `HOUR`:

```
SELECT EXTRACT(HOUR FROM '26:30:00'), HOUR('26:30:00');
+-----+-----+
| EXTRACT(HOUR FROM '26:30:00') | HOUR('26:30:00') |
+-----+-----+
|             2 |           26 |
+-----+-----+
```

See Also

- [Date and Time Units](#)
- [Date and Time Literals](#)
- [EXTRACT\(\)](#)

LAST_DAY

Syntax

```
LAST_DAY(date)
```

Description

Takes a date or datetime value and returns the corresponding value for the last day of the month. Returns NULL if the argument is invalid.

Examples

```

SELECT LAST_DAY('2003-02-05');
+-----+
| LAST_DAY('2003-02-05') |
+-----+
| 2003-02-28 |
+-----+

SELECT LAST_DAY('2004-02-05');
+-----+
| LAST_DAY('2004-02-05') |
+-----+
| 2004-02-29 |
+-----+

SELECT LAST_DAY('2004-01-01 01:01:01');
+-----+
| LAST_DAY('2004-01-01 01:01:01') |
+-----+
| 2004-01-31 |
+-----+

SELECT LAST_DAY('2003-03-32');
+-----+
| LAST_DAY('2003-03-32') |
+-----+
| NULL |
+-----+
1 row in set, 1 warning (0.00 sec)

Warning (Code 1292): Incorrect datetime value: '2003-03-32'

```

MAKEDATE

Syntax

```
MAKEDATE(year,dayofyear)
```

Description

Returns a date, given `year` and `day-of-year` values. `dayofyear` must be greater than 0 or the result is `NULL`.

Examples

```

SELECT MAKEDATE(2011,31), MAKEDATE(2011,32);
+-----+-----+
| MAKEDATE(2011,31) | MAKEDATE(2011,32) |
+-----+-----+
| 2011-01-31 | 2011-02-01 |
+-----+-----+

SELECT MAKEDATE(2011,365), MAKEDATE(2014,365);
+-----+-----+
| MAKEDATE(2011,365) | MAKEDATE(2014,365) |
+-----+-----+
| 2011-12-31 | 2014-12-31 |
+-----+-----+

SELECT MAKEDATE(2011,0);
+-----+
| MAKEDATE(2011,0) |
+-----+
| NULL |
+-----+

```

MAKETIME

Syntax

```
MAKETIME(hour,minute,second)
```

Description

Returns a time value calculated from the `hour`, `minute`, and `second` arguments.

If `minute` or `second` are out of the range 0 to 60, `NULL` is returned. The `hour` can be in the range -838 to 838, outside of which the value is truncated with a warning.

Examples

```
SELECT MAKETIME(13,57,33);
+-----+
| MAKETIME(13,57,33) |
+-----+
| 13:57:33           |
+-----+

SELECT MAKETIME(-13,57,33);
+-----+
| MAKETIME(-13,57,33) |
+-----+
| -13:57:33          |
+-----+

SELECT MAKETIME(13,67,33);
+-----+
| MAKETIME(13,67,33) |
+-----+
| NULL               |
+-----+

SELECT MAKETIME(-1000,57,33);
+-----+
| MAKETIME(-1000,57,33) |
+-----+
| -838:59:59          |
+-----+
1 row in set, 1 warning (0.00 sec)

SHOW WARNINGS;
+-----+
| Level | Code | Message           |
+-----+
| Warning | 1292 | Truncated incorrect time value: '-1000:57:33' |
+-----+
```

MICROSECOND

Syntax

```
MICROSECOND(expr)
```

Description

Returns the microseconds from the time or datetime expression `expr` as a number in the range from 0 to 999999.

If *expr* is a time with no microseconds, zero is returned, while if *expr* is a date with no time, zero with a warning is returned.

Examples

```
SELECT MICROSECOND('12:00:00.123456');
+-----+
| MICROSECOND('12:00:00.123456') |
+-----+
|          123456 |
+-----+


SELECT MICROSECOND('2009-12-31 23:59:59.000010');
+-----+
| MICROSECOND('2009-12-31 23:59:59.000010') |
+-----+
|                  10 |
+-----+


SELECT MICROSECOND('2013-08-07 12:13:14');
+-----+
| MICROSECOND('2013-08-07 12:13:14') |
+-----+
|                      0 |
+-----+


SELECT MICROSECOND('2013-08-07');
+-----+
| MICROSECOND('2013-08-07') |
+-----+
|                      0 |
+-----+
1 row in set, 1 warning (0.00 sec)

SHOW WARNINGS;
+-----+
| Level | Code | Message           |
+-----+
| Warning | 1292 | Truncated incorrect time value: '2013-08-07' |
+-----+
```

See Also

- [Microseconds in MariaDB](#)

NOW

Syntax

```
NOW([precision])
CURRENT_TIMESTAMP
CURRENT_TIMESTAMP([precision])
LOCALTIME, LOCALTIME([precision])
LOCALTIMESTAMP
LOCALTIMESTAMP([precision])
```

Description

Returns the current date and time as a value in 'YYYY-MM-DD HH:MM:SS' or YYYYMMDDHHMMSS.uuuuuu format, depending on whether the function is used in a string or numeric context. The value is expressed in the current [time zone](#).

The optional *precision* determines the microsecond precision. See [Microseconds in MariaDB](#).

Contents

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

Contents

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

`NOW()` (or its synonyms) can be used as the default value for `TIMESTAMP` columns as well as, since MariaDB 10.0.1, `DATETIME` columns. Before MariaDB 10.0.1, it was only possible for a single `TIMESTAMP` column per table to contain the `CURRENT_TIMESTAMP` as its default.

When displayed in the `INFORMATION_SCHEMA.COLUMNS` table, a default `CURRENT_TIMESTAMP` is displayed as `CURRENT_TIMESTAMP` up until MariaDB 10.2.2, and as `current_timestamp()` from MariaDB 10.2.3, due to to MariaDB 10.2 accepting expressions in the `DEFAULT` clause.

Examples

```
SELECT NOW();
+-----+
| NOW()           |
+-----+
| 2010-03-27 13:13:25 |
+-----+

SELECT NOW() + 0;
+-----+
| NOW() + 0          |
+-----+
| 20100327131329.000000 |
+-----+
```

With precision:

```
SELECT CURRENT_TIMESTAMP(2);
+-----+
| CURRENT_TIMESTAMP(2)   |
+-----+
| 2018-07-10 09:47:26.24 |
+-----+
```

Used as a default `TIMESTAMP`:

```
CREATE TABLE t (createdTS TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP);
```

From MariaDB 10.2.2:

```
SELECT * FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_SCHEMA='test'
  AND COLUMN_NAME LIKE '%ts%'

***** 1. row *****
  TABLE_CATALOG: def
  TABLE_SCHEMA: test
  TABLE_NAME: t
  COLUMN_NAME: ts
  ORDINAL_POSITION: 1
  COLUMN_DEFAULT: current_timestamp()
...
```

<= MariaDB 10.2.1

```
SELECT * FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_SCHEMA='test'
  AND COLUMN_NAME LIKE '%ts%'

***** 1. row *****
  TABLE_CATALOG: def
  TABLE_SCHEMA: test
  TABLE_NAME: t
  COLUMN_NAME: createdTS
  ORDINAL_POSITION: 1
  COLUMN_DEFAULT: CURRENT_TIMESTAMP
...
```

See Also

- [Microseconds in MariaDB](#)

- timestamp server system variable

PERIOD_ADD

Syntax

```
PERIOD_ADD(P,N)
```

Description

Adds N months to period P . P is in the format YYMM or YYYYMM, and is not a date value. If P contains a two-digit year, values from 00 to 69 are converted to from 2000 to 2069, while values from 70 are converted to 1970 upwards.

Returns a value in the format YYYYMM.

Examples

```
SELECT PERIOD_ADD('200801',2);
+-----+
| PERIOD_ADD('200801',2) |
+-----+
|          200803 |
+-----+

SELECT PERIOD_ADD('6910',2);
+-----+
| PERIOD_ADD('6910',2) |
+-----+
|          206912 |
+-----+

SELECT PERIOD_ADD('7010',2);
+-----+
| PERIOD_ADD('7010',2) |
+-----+
|          197012 |
+-----+
```

PERIOD_DIFF

Syntax

```
PERIOD_DIFF(P1,P2)
```

Description

Returns the number of months between periods P1 and P2. P1 and P2 can be in the format YYMM or YYYYMM , and are not date values.

If P1 or P2 contains a two-digit year, values from 00 to 69 are converted to from 2000 to 2069, while values from 70 are converted to 1970 upwards.

Examples

```
SELECT PERIOD_DIFF(200802,200703);
+-----+
| PERIOD_DIFF(200802,200703) |
+-----+
|          11 |
+-----+

SELECT PERIOD_DIFF(6902,6803);
+-----+
| PERIOD_DIFF(6902,6803) |
+-----+
|          11 |
+-----+

SELECT PERIOD_DIFF(7002,6803);
+-----+
| PERIOD_DIFF(7002,6803) |
+-----+
|        -1177 |
+-----+
```

SEC_TO_TIME

Syntax

```
SEC_TO_TIME(seconds)
```

Description

Returns the seconds argument, converted to hours, minutes, and seconds, as a TIME value. The range of the result is constrained to that of the [TIME data type](#). A warning occurs if the argument corresponds to a value outside that range.

The time will be returned in the format `hh:mm:ss`, or `hhmmss` if used in a numeric calculation.

Examples

```

SELECT SEC_TO_TIME(12414);
+-----+
| SEC_TO_TIME(12414) |
+-----+
| 03:26:54 |
+-----+

SELECT SEC_TO_TIME(12414)+0;
+-----+
| SEC_TO_TIME(12414)+0 |
+-----+
|      32654 |
+-----+

SELECT SEC_TO_TIME(9999999);
+-----+
| SEC_TO_TIME(9999999) |
+-----+
| 838:59:59 |
+-----+
1 row in set, 1 warning (0.00 sec)

SHOW WARNINGS;
+-----+
| Level | Code | Message           |
+-----+
| Warning | 1292 | Truncated incorrect time value: '9999999' |
+-----+

```

STR_TO_DATE

Syntax

`STR_TO_DATE(str,format)`

Description

This is the inverse of the `DATE_FORMAT()` function. It takes a string `str` and a format string `format`. `STR_TO_DATE()` returns a `DATETIME` value if the format string contains both date and time parts, or a `DATE` or `TIME` value if the string contains only date or time parts.

The date, time, or datetime values contained in `str` should be given in the format indicated by `format`. If `str` contains an illegal date, time, or datetime value, `STR_TO_DATE()` returns `NULL`. An illegal value also produces a warning.

The options that can be used by `STR_TO_DATE()`, as well as its inverse `DATE_FORMAT()` and the `FROM_UNIXTIME()` function, are:

Contents

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

Option	Description
%a	Short weekday name in current locale (Variable <code>lc_time_names</code>).
%b	Short form month name in current locale. For locale en_US this is one of: Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov or Dec.
%c	Month with 1 or 2 digits.
%D	Day with English suffix 'th', 'nd', 'st' or 'rd'. (1st, 2nd, 3rd...).
%d	Day with 2 digits.
%e	Day with 1 or 2 digits.
%f	Microseconds 6 digits.
%H	Hour with 2 digits between 00-23.
%h	Hour with 2 digits between 01-12.
%I	Hour with 2 digits between 01-12.

%i	Minute with 2 digits.
%j	Day of the year (001-366)
%k	Hour with 1 digits between 0-23.
%l	Hour with 1 digits between 1-12.
%M	Full month name in current locale (Variable lc_time_names).
%m	Month with 2 digits.
%p	AM/PM according to current locale (Variable lc_time_names).
%r	Time in 12 hour format, followed by AM/PM. Short for '%l:%i:%S %p'.
%S	Seconds with 2 digits.
%s	Seconds with 2 digits.
%T	Time in 24 hour format. Short for '%H:%i:%S'.
%U	Week number (00-53), when first day of the week is Sunday.
%u	Week number (00-53), when first day of the week is Monday.
%V	Week number (01-53), when first day of the week is Sunday. Used with %X.
%v	Week number (01-53), when first day of the week is Monday. Used with %x.
%W	Full weekday name in current locale (Variable lc_time_names).
%w	Day of the week. 0 = Sunday, 6 = Saturday.
%X	Year with 4 digits when first day of the week is Sunday. Used with %V.
%x	Year with 4 digits when first day of the week is Monday. Used with %v.
%Y	Year with 4 digits.
%y	Year with 2 digits.
%#	For str_to_date() , skip all numbers.
%.	For str_to_date() , skip all punctuation characters.
%@	For str_to_date() , skip all alpha characters.
%%	A literal % character.

Examples

```

SELECT STR_TO_DATE('Wednesday, June 2, 2014', '%W, %M %e, %Y');
+-----+
| STR_TO_DATE('Wednesday, June 2, 2014', '%W, %M %e, %Y') |
+-----+
| 2014-06-02 |
+-----+


SELECT STR_TO_DATE('Wednesday23423, June 2, 2014', '%W, %M %e, %Y');
+-----+
| STR_TO_DATE('Wednesday23423, June 2, 2014', '%W, %M %e, %Y') |
+-----+
| NULL |
+-----+
1 row in set, 1 warning (0.00 sec)

SHOW WARNINGS;
+-----+
| Level | Code | Message |
+-----+
| Warning | 1411 | Incorrect datetime value: 'Wednesday23423, June 2, 2014' for function str_to_date |
+-----+


SELECT STR_TO_DATE('Wednesday23423, June 2, 2014', '%W##, %M %e, %Y');
+-----+
| STR_TO_DATE('Wednesday23423, June 2, 2014', '%W##, %M %e, %Y') |
+-----+
| 2014-06-02 |
+-----+

```

See Also

- [DATE_FORMAT\(\)](#)
- [FROM_UNIXTIME\(\)](#)

SUBDATE

Syntax

```
SUBDATE(date,INTERVAL expr unit), SUBDATE(expr,days)
```

Description

When invoked with the `INTERVAL` form of the second argument, `SUBDATE()` is a synonym for `DATE_SUB()`. See [Date and Time Units](#) for a complete list of permitted units.

The second form allows the use of an integer value for days. In such cases, it is interpreted as the number of days to be subtracted from the date or datetime expression expr.

Examples

```

SELECT DATE_SUB('2008-01-02', INTERVAL 31 DAY);
+-----+
| DATE_SUB('2008-01-02', INTERVAL 31 DAY) |
+-----+
| 2007-12-02                                |
+-----+


SELECT SUBDATE('2008-01-02', INTERVAL 31 DAY);
+-----+
| SUBDATE('2008-01-02', INTERVAL 31 DAY) |
+-----+
| 2007-12-02                                |
+-----+

```

```

SELECT SUBDATE('2008-01-02 12:00:00', 31);
+-----+
| SUBDATE('2008-01-02 12:00:00', 31) |
+-----+
| 2007-12-02 12:00:00                  |
+-----+

```

```

CREATE TABLE t1 (d DATETIME);
INSERT INTO t1 VALUES
    ("2007-01-30 21:31:07"),
    ("1983-10-15 06:42:51"),
    ("2011-04-21 12:34:56"),
    ("2011-10-30 06:31:41"),
    ("2011-01-30 14:03:25"),
    ("2004-10-07 11:19:34");

```

```

SELECT d, SUBDATE(d, 10) from t1;
+-----+
| d           | SUBDATE(d, 10)   |
+-----+
| 2007-01-30 21:31:07 | 2007-01-20 21:31:07 |
| 1983-10-15 06:42:51 | 1983-10-05 06:42:51 |
| 2011-04-21 12:34:56 | 2011-04-11 12:34:56 |
| 2011-10-30 06:31:41 | 2011-10-20 06:31:41 |
| 2011-01-30 14:03:25 | 2011-01-20 14:03:25 |
| 2004-10-07 11:19:34 | 2004-09-27 11:19:34 |
+-----+


SELECT d, SUBDATE(d, INTERVAL 10 MINUTE) from t1;
+-----+
| d           | SUBDATE(d, INTERVAL 10 MINUTE) |
+-----+
| 2007-01-30 21:31:07 | 2007-01-30 21:21:07 |
| 1983-10-15 06:42:51 | 1983-10-15 06:32:51 |
| 2011-04-21 12:34:56 | 2011-04-21 12:24:56 |
| 2011-10-30 06:31:41 | 2011-10-30 06:21:41 |
| 2011-01-30 14:03:25 | 2011-01-30 13:53:25 |
| 2004-10-07 11:19:34 | 2004-10-07 11:09:34 |
+-----+

```

SUBTIME

Syntax

```
SUBTIME(expr1,expr2)
```

Description

SUBTIME() returns expr1 - expr2 expressed as a value in the same format as expr1 . expr1 is a time or datetime expression, and expr2 is a time expression.

Examples

```
SELECT SUBTIME('2007-12-31 23:59:59.999999','1 1:1:1.000002');
+-----+
| SUBTIME('2007-12-31 23:59:59.999999','1 1:1:1.000002') |
+-----+
| 2007-12-30 22:58:58.999997 |
+-----+

SELECT SUBTIME('01:00:00.999999', '02:00:00.999998');
+-----+
| SUBTIME('01:00:00.999999', '02:00:00.999998') |
+-----+
| -00:59:59.999999 |
+-----+
```

SYSDATE

Syntax

```
SYSDATE([precision])
```

Description

Returns the current date and time as a value in 'YYYY-MM-DD HH:MM:SS' or YYYYMMDDHHMMSS.uduuuu format, depending on whether the function is used in a string or numeric context.

The optional *precision* determines the microsecond precision. See [Microseconds in MariaDB](#).

SYSDATE() returns the time at which it executes. This differs from the behavior for [NOW\(\)](#), which returns a constant time that indicates the time at which the statement began to execute. (Within a stored routine or trigger, NOW() returns the time at which the routine or triggering statement began to execute.)

In addition, changing the [timestamp system variable](#) with a `SET timestamp` statement affects the value returned by NOW() but not by SYSDATE(). This means that timestamp settings in the [binary log](#) have no effect on invocations of SYSDATE().

Because SYSDATE() can return different values even within the same statement, and is not affected by SET TIMESTAMP, it is non-deterministic and therefore unsafe for replication if statement-based binary logging is used. If that is a problem, you can use row-based logging, or start the server with the mysqld option `--sysdate-is-now` to cause SYSDATE() to be an alias for NOW(). The non-deterministic nature of SYSDATE() also means that indexes cannot be used for evaluating expressions that refer to it, and that statements using the SYSDATE() function are [unsafe for statement-based replication](#).

Contents

- [Syntax](#)
- [Description](#)
- [Examples](#)
- [See Also](#)

Examples

Difference between NOW() and SYSDATE():

```

SELECT NOW(), SLEEP(2), NOW();
+-----+-----+-----+
| NOW() | SLEEP(2) | NOW() |
+-----+-----+-----+
| 2010-03-27 13:23:40 | 0 | 2010-03-27 13:23:40 |
+-----+-----+-----+

SELECT SYSDATE(), SLEEP(2), SYSDATE();
+-----+-----+-----+
| SYSDATE() | SLEEP(2) | SYSDATE() |
+-----+-----+-----+
| 2010-03-27 13:23:52 | 0 | 2010-03-27 13:23:54 |
+-----+-----+-----+

```

With precision:

```

SELECT SYSDATE(4);
+-----+
| SYSDATE(4) |
+-----+
| 2018-07-10 10:17:13.1689 |
+-----+

```

See Also

- [Microseconds in MariaDB](#)
- [timestamp server system variable](#)

TIMEDIFF

Syntax

```
TIMEDIFF(expr1,expr2)
```

Description

`TIMEDIFF()` returns `expr1 - expr2` expressed as a time value. `expr1` and `expr2` are time or date-and-time expressions, but both must be of the same type.

Examples

```

SELECT TIMEDIFF('2000:01:01 00:00:00', '2000:01:01 00:00:00.000001');
+-----+
| TIMEDIFF('2000:01:01 00:00:00', '2000:01:01 00:00:00.000001') |
+-----+
| -00:00:00.000001 |
+-----+

SELECT TIMEDIFF('2008-12-31 23:59:59.000001', '2008-12-30 01:01:01.000002');
+-----+
| TIMEDIFF('2008-12-31 23:59:59.000001', '2008-12-30 01:01:01.000002') |
+-----+
| 46:58:57.999999 |
+-----+

```

TIMESTAMP FUNCTION

Syntax

```
TIMESTAMP(expr), TIMESTAMP(expr1,expr2)
```

Description

With a single argument, this function returns the date or datetime expression `expr` as a datetime value. With two arguments, it adds the time expression `expr2` to the date or datetime expression `expr1` and returns the result as a datetime value.

Examples

```
SELECT TIMESTAMP('2003-12-31');
+-----+
| TIMESTAMP('2003-12-31') |
+-----+
| 2003-12-31 00:00:00 |
+-----+

SELECT TIMESTAMP('2003-12-31 12:00:00','6:30:00');
+-----+
| TIMESTAMP('2003-12-31 12:00:00','6:30:00') |
+-----+
| 2003-12-31 18:30:00 |
+-----+
```

TIMESTAMPADD

Syntax

```
TIMESTAMPADD(unit,interval,datetime_expr)
```

Description

Adds the integer expression `interval` to the date or datetime expression `datetime_expr`. The unit for `interval` is given by the `unit` argument, which should be one of the following values: MICROSECOND, SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER, or YEAR.

The `unit` value may be specified using one of keywords as shown, or with a prefix of `SQL_TSI_`. For example, `DAY` and `SQL_TSI_DAY` both are legal.

Before [MariaDB 5.5](#), `FRAC_SECOND` was permitted as a synonym for `MICROSECOND`.

Examples

```
SELECT TIMESTAMPADD(MINUTE,1,'2003-01-02');
+-----+
| TIMESTAMPADD(MINUTE,1,'2003-01-02') |
+-----+
| 2003-01-02 00:01:00 |
+-----+

SELECT TIMESTAMPADD(WEEK,1,'2003-01-02');
+-----+
| TIMESTAMPADD(WEEK,1,'2003-01-02') |
+-----+
| 2003-01-09 |
+-----+
```

TIMESTAMPDIFF

Syntax

```
TIMESTAMPDIFF(unit,datetime_expr1,datetime_expr2)
```

Description

Returns `datetime_expr2 - datetime_expr1`, where `datetime_expr1` and `datetime_expr2` are date or datetime expressions. One expression may be a date and the other a datetime; a date value is treated as a datetime having the time part '00:00:00' where necessary. The unit for the result (an integer) is given by the unit argument. The legal values for unit are the same as those listed in the description of the `TIMESTAMPADD()` function, i.e MICROSECOND, SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER, or YEAR.

`TIMESTAMPDIFF` can also be used to calculate age.

Examples

```
SELECT TIMESTAMPDIFF(MONTH,'2003-02-01','2003-05-01');
+-----+
| TIMESTAMPDIFF(MONTH,'2003-02-01','2003-05-01') |
+-----+
|                      3 |
+-----+


SELECT TIMESTAMPDIFF(YEAR,'2002-05-01','2001-01-01');
+-----+
| TIMESTAMPDIFF(YEAR,'2002-05-01','2001-01-01') |
+-----+
|                      -1 |
+-----+


SELECT TIMESTAMPDIFF(MINUTE,'2003-02-01','2003-05-01 12:05:55');
+-----+
| TIMESTAMPDIFF(MINUTE,'2003-02-01','2003-05-01 12:05:55') |
+-----+
|                      128885 |
+-----+
```

Calculating age:

```
SELECT CURDATE();
+-----+
| CURDATE() |
+-----+
| 2019-05-27 |
+-----+


SELECT TIMESTAMPDIFF(YEAR, '1971-06-06', CURDATE()) AS age;
+---+
| age |
+---+
| 47 |
+---+


SELECT TIMESTAMPDIFF(YEAR, '1971-05-06', CURDATE()) AS age;
+---+
| age |
+---+
| 48 |
+---+
```

Age as of 2014-08-02:

```

SELECT name, date_of_birth, TIMESTAMPDIFF(YEAR,date_of_birth,'2014-08-02') AS age
FROM student_details;
+-----+-----+
| name | date_of_birth | age |
+-----+-----+
| Chun | 1993-12-31 | 20 |
| Esben | 1946-01-01 | 68 |
| Kaolin | 1996-07-16 | 18 |
| Tatiana | 1988-04-13 | 26 |
+-----+-----+

```

TO_DAYS

Syntax

```
TO_DAYS(date)
```

Description

Given a date `date`, returns the number of days since the start of the current calendar (0000-00-00).

The function is not designed for use with dates before the advent of the Gregorian calendar in October 1582. Results will not be reliable since it doesn't account for the lost days when the calendar changed from the Julian calendar.

This is the converse of the [FROM_DAYS\(\)](#) function.

Examples

```

SELECT TO_DAYS('2007-10-07');
+-----+
| TO_DAYS('2007-10-07') |
+-----+
| 733321 |
+-----+

SELECT TO_DAYS('0000-01-01');
+-----+
| TO_DAYS('0000-01-01') |
+-----+
| 1 |
+-----+

SELECT TO_DAYS(950501);
+-----+
| TO_DAYS(950501) |
+-----+
| 728779 |
+-----+

```

TO_SECONDS

Syntax

```
TO_SECONDS(expr)
```

Description

Returns the number of seconds from year 0 till `expr`, or NULL if `expr` is not a valid date or [datetime](#).

Examples

```
SELECT TO_SECONDS('2013-06-13');
+-----+
| TO_SECONDS('2013-06-13') |
+-----+
|          63538300800 |
+-----+

SELECT TO_SECONDS('2013-06-13 21:45:13');
+-----+
| TO_SECONDS('2013-06-13 21:45:13') |
+-----+
|          63538379113 |
+-----+

SELECT TO_SECONDS(NOW());
+-----+
| TO_SECONDS(NOW()) |
+-----+
|      63543530875 |
+-----+

SELECT TO_SECONDS(20130513);
+-----+
| TO_SECONDS(20130513) |
+-----+
|          63535622400 |
+-----+
1 row in set (0.00 sec)

SELECT TO_SECONDS(130513);
+-----+
| TO_SECONDS(130513) |
+-----+
|          63535622400 |
+-----+
```

UNIX_TIMESTAMP

Syntax

```
UNIX_TIMESTAMP()
UNIX_TIMESTAMP(date)
```

Description

If called with no argument, returns a Unix timestamp (seconds since '1970-01-01 00:00:00' UTC) as an unsigned integer. If `UNIX_TIMESTAMP()` is called with a date argument, it returns the value of the argument as seconds since '1970-01-01 00:00:00' UTC. date may be a `DATE` string, a `DATETIME` string, a `TIMESTAMP`, or a number in the format YYMMDD or YYYYMMDD. The server interprets date as a value in the current `time zone` and converts it to an internal value in `UTC`. Clients can set their time zone as described in `time zones`.

The inverse function of `UNIX_TIMESTAMP()` is `FROM_UNIXTIME()`

`UNIX_TIMESTAMP()` supports `microseconds`.

Contents

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

Error Handling

Returns NULL for wrong arguments to `UNIX_TIMESTAMP()`. In MySQL and MariaDB before 5.3 wrong arguments to `UNIX_TIMESTAMP()` returned 0.

Compatibility

As you can see in the examples above, `UNIX_TIMESTAMP(constant-date-string)` returns a timestamp with 6 decimals while [MariaDB 5.2](#) and before returns it without decimals. This can cause a problem if you are using `UNIX_TIMESTAMP()` as a partitioning function. You can fix this by using `FLOOR(UNIX_TIMESTAMP(..))` or changing the date string to a date number, like 20080101000000.

Examples

```
SELECT UNIX_TIMESTAMP();
+-----+
| UNIX_TIMESTAMP() |
+-----+
|      1269711082 |
+-----+

SELECT UNIX_TIMESTAMP('2007-11-30 10:30:19');
+-----+
| UNIX_TIMESTAMP('2007-11-30 10:30:19') |
+-----+
|          1196436619.000000 |
+-----+

SELECT UNIX_TIMESTAMP("2007-11-30 10:30:19.123456");
+-----+
| unix_timestamp("2007-11-30 10:30:19.123456") |
+-----+
|          1196411419.123456 |
+-----+

SELECT FROM_UNIXTIME(UNIX_TIMESTAMP('2007-11-30 10:30:19'));
+-----+
| FROM_UNIXTIME(UNIX_TIMESTAMP('2007-11-30 10:30:19')) |
+-----+
| 2007-11-30 10:30:19.000000 |
+-----+

SELECT FROM_UNIXTIME(FLOOR(UNIX_TIMESTAMP('2007-11-30 10:30:19')));
+-----+
| FROM_UNIXTIME(FLOOR(UNIX_TIMESTAMP('2007-11-30 10:30:19'))) |
+-----+
| 2007-11-30 10:30:19 |
+-----+
```

See Also

- [FROM_UNIXTIME\(\)](#)

UTC_TIMESTAMP

Syntax

```
UTC_TIMESTAMP
UTC_TIMESTAMP([precision])
```

Description

Returns the current [UTC](#) date and time as a value in 'YYYY-MM-DD HH:MM:SS' or YYYYMMDDHHMMSS.uduuuu format, depending on whether the function is used in a string or numeric context.

The optional *precision* determines the microsecond precision. See [Microseconds in MariaDB](#).

Examples

```
SELECT UTC_TIMESTAMP(), UTC_TIMESTAMP() + 0;
+-----+
| UTC_TIMESTAMP()      | UTC_TIMESTAMP() + 0    |
+-----+
| 2010-03-27 17:33:16 | 20100327173316.000000 |
+-----+
```

With precision:

```
SELECT UTC_TIMESTAMP(4);
+-----+
| UTC_TIMESTAMP(4)   |
+-----+
| 2018-07-10 07:51:09.1019 |
+-----+
```

See Also

- [Time Zones](#)
- [Microseconds in MariaDB](#)

WEEK

Syntax

```
WEEK(date[,mode])
```

Description

This function returns the week number for `date`. The two-argument form of `WEEK()` allows you to specify whether the week starts on Sunday or Monday and whether the return value should be in the range from 0 to 53 or from 1 to 53. If the `mode` argument is omitted, the value of the `default_week_format` system variable is used.

Modes

Mode	1st day of week	Range	Week 1 is the 1st week with
0	Sunday	0-53	a Sunday in this year
1	Monday	0-53	more than 3 days this year
2	Sunday	1-53	a Sunday in this year
3	Monday	1-53	more than 3 days this year
4	Sunday	0-53	more than 3 days this year
5	Monday	0-53	a Monday in this year
6	Sunday	1-53	more than 3 days this year
7	Monday	1-53	a Monday in this year

With the mode value of 3, which means “more than 3 days this year”, weeks are numbered according to ISO 8601:1988.

Examples

```

SELECT WEEK('2008-02-20');
+-----+
| WEEK('2008-02-20') |
+-----+
|          7 |
+-----+

SELECT WEEK('2008-02-20',0);
+-----+
| WEEK('2008-02-20',0) |
+-----+
|          7 |
+-----+

SELECT WEEK('2008-02-20',1);
+-----+
| WEEK('2008-02-20',1) |
+-----+
|          8 |
+-----+

SELECT WEEK('2008-12-31',0);
+-----+
| WEEK('2008-12-31',0) |
+-----+
|          52 |
+-----+

SELECT WEEK('2008-12-31',1);
+-----+
| WEEK('2008-12-31',1) |
+-----+
|          53 |
+-----+

SELECT WEEK('2019-12-30',3);
+-----+
| WEEK('2019-12-30',3) |
+-----+
|          1 |
+-----+

```

```

CREATE TABLE t1 (d DATETIME);
INSERT INTO t1 VALUES
    ("2007-01-30 21:31:07"),
    ("1983-10-15 06:42:51"),
    ("2011-04-21 12:34:56"),
    ("2011-10-30 06:31:41"),
    ("2011-01-30 14:03:25"),
    ("2004-10-07 11:19:34");

```

```

SELECT d, WEEK(d,0), WEEK(d,1) from t1;
+-----+-----+-----+
| d           | WEEK(d,0) | WEEK(d,1) |
+-----+-----+-----+
| 2007-01-30 21:31:07 |      4 |      5 |
| 1983-10-15 06:42:51 |     41 |     41 |
| 2011-04-21 12:34:56 |     16 |     16 |
| 2011-10-30 06:31:41 |     44 |     43 |
| 2011-01-30 14:03:25 |      5 |      4 |
| 2004-10-07 11:19:34 |     40 |     41 |
+-----+-----+-----+

```

WEEKDAY

Syntax

```
WEEKDAY(date)
```

Description

Returns the weekday index for date (0 = Monday, 1 = Tuesday, ... 6 = Sunday).

This contrasts with [DAYOFWEEK\(\)](#) which follows the ODBC standard (1 = Sunday, 2 = Monday, ..., 7 = Saturday).

Examples

```
SELECT WEEKDAY('2008-02-03 22:23:00');
+-----+
| WEEKDAY('2008-02-03 22:23:00') |
+-----+
|                               6 |
+-----+  
  
SELECT WEEKDAY('2007-11-06');
+-----+
| WEEKDAY('2007-11-06') |
+-----+
|                           1 |
+-----+
```

```
CREATE TABLE t1 (d DATETIME);
INSERT INTO t1 VALUES
    ("2007-01-30 21:31:07"),
    ("1983-10-15 06:42:51"),
    ("2011-04-21 12:34:56"),
    ("2011-10-30 06:31:41"),
    ("2011-01-30 14:03:25"),
    ("2004-10-07 11:19:34");
```

```
SELECT d FROM t1 WHERE WEEKDAY(d) = 6;
+-----+
| d           |
+-----+
| 2011-10-30 06:31:41 |
| 2011-01-30 14:03:25 |
+-----+
```

WEEKOFYEAR

Syntax

```
WEEKOFYEAR(date)
```

Description

Returns the calendar week of the date as a number in the range from 1 to 53. `WEEKOFYEAR()` is a compatibility function that is equivalent to `WEEK(date,3)`.

Examples

```
SELECT WEEKOFYEAR('2008-02-20');
+-----+
| WEEKOFYEAR('2008-02-20') |
+-----+
|          8 |
+-----+
```

```
CREATE TABLE t1 (d DATETIME);
INSERT INTO t1 VALUES
    ("2007-01-30 21:31:07"),
    ("1983-10-15 06:42:51"),
    ("2011-04-21 12:34:56"),
    ("2011-10-30 06:31:41"),
    ("2011-01-30 14:03:25"),
    ("2004-10-07 11:19:34");
```

```
select * from t1;
+-----+
| d           |
+-----+
| 2007-01-30 21:31:07 |
| 1983-10-15 06:42:51 |
| 2011-04-21 12:34:56 |
| 2011-10-30 06:31:41 |
| 2011-01-30 14:03:25 |
| 2004-10-07 11:19:34 |
+-----+
```

```
SELECT d, WEEKOFYEAR(d), WEEK(d,3) from t1;
+-----+-----+-----+
| d           | WEEKOFYEAR(d) | WEEK(d,3) |
+-----+-----+-----+
| 2007-01-30 21:31:07 |      5 |      5 |
| 1983-10-15 06:42:51 |     41 |     41 |
| 2011-04-21 12:34:56 |     16 |     16 |
| 2011-10-30 06:31:41 |     43 |     43 |
| 2011-01-30 14:03:25 |      4 |      4 |
| 2004-10-07 11:19:34 |     41 |     41 |
+-----+-----+-----+
```

YEAR

Syntax

```
YEAR(date)
```

Description

Returns the year for the given date, in the range 1000 to 9999, or 0 for the "zero" date.

Examples

```
CREATE TABLE t1 (d DATETIME);
INSERT INTO t1 VALUES
    ("2007-01-30 21:31:07"),
    ("1983-10-15 06:42:51"),
    ("2011-04-21 12:34:56"),
    ("2011-10-30 06:31:41"),
    ("2011-01-30 14:03:25"),
    ("2004-10-07 11:19:34");
```

```
SELECT * FROM t1;
+-----+
| d           |
+-----+
| 2007-01-30 21:31:07 |
| 1983-10-15 06:42:51 |
| 2011-04-21 12:34:56 |
| 2011-10-30 06:31:41 |
| 2011-01-30 14:03:25 |
| 2004-10-07 11:19:34 |
+-----+

SELECT * FROM t1 WHERE YEAR(d) = 2011;
+-----+
| d           |
+-----+
| 2011-04-21 12:34:56 |
| 2011-10-30 06:31:41 |
| 2011-01-30 14:03:25 |
+-----+
```

```
SELECT YEAR('1987-01-01');
+-----+
| YEAR('1987-01-01') |
+-----+
|          1987      |
+-----+
```

See Also

- [YEAR data type](#)

YEARWEEK

Syntax

```
YEARWEEK(date), YEARWEEK(date,mode)
```

Description

Returns year and week for a date. The mode argument works exactly like the mode argument to [WEEK\(\)](#). The year in the result may be different from the year in the date argument for the first and the last week of the year.

Examples

```
SELECT YEARWEEK('1987-01-01');
+-----+
| YEARWEEK('1987-01-01') |
+-----+
|          198652      |
+-----+
```

```

CREATE TABLE t1 (d DATETIME);
INSERT INTO t1 VALUES
    ("2007-01-30 21:31:07"),
    ("1983-10-15 06:42:51"),
    ("2011-04-21 12:34:56"),
    ("2011-10-30 06:31:41"),
    ("2011-01-30 14:03:25"),
    ("2004-10-07 11:19:34");

```

```

SELECT * FROM t1;
+-----+
| d           |
+-----+
| 2007-01-30 21:31:07 |
| 1983-10-15 06:42:51 |
| 2011-04-21 12:34:56 |
| 2011-10-30 06:31:41 |
| 2011-01-30 14:03:25 |
| 2004-10-07 11:19:34 |
+-----+
6 rows in set (0.02 sec)

```

```

SELECT YEARWEEK(d) FROM t1 WHERE YEAR(d) = 2011;
+-----+
| YEARWEEK(d) |
+-----+
| 201116      |
| 201144      |
| 201105      |
+-----+
3 rows in set (0.03 sec)

```

Well-Known Binary (WKB) Format

WKB stands for Well-Known Binary, a format for representing geographical and geometrical data.

WKB uses 1-byte unsigned integers, 4-byte unsigned integers, and 8-byte double-precision numbers.

- The first byte indicates the byte order. 00 for big endian, or 01 for little endian.
- The next 4 bytes indicate the geometry type. Values from 1 to 7 indicate whether the type is Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon, or GeometryCollection respectively.
- The 8-byte floats represent the co-ordinates.

Take the following example, a sequence of 21 bytes each represented by two hex digits:

```
00000000140000000000000004010000000000000
```

- It's big endian
 - 0000000014000000000000000401000000000000
- It's a POINT
 - 0000000014000000000000000401000000000000
- The X co-ordinate is 2.0
 - 0000000014000000000000000401000000000000
- The Y co-ordinate is 4.0
 - 0000000014000000000000000401000000000000

MPolyFromWKB

Syntax

```

MPolyFromWKB(wkb[,srid])
MultiPolygonFromWKB(wkb[,srid])

```

Description

Constructs a **MULTIPOLYGON** value using its **WKB** representation and **SRID**.

`MPolyFromWKB()` and `MultiPolygonFromWKB()` are synonyms.

Examples

```
SET @g = ST_AsBinary(MPointFromText('MULTIPOINT((28 26,28 0,84 0,84 42,28 26),(52 18,66 23,73 9,48 6,52 18)),((59 18,67 18,67 13,15,59 18))'));

SELECT ST_AsText(MPolyFromWKB(@g));
+-----+
| ST_AsText(MPolyFromWKB(@g)) |
+-----+
| MULTIPOINT((28 26,28 0,84 0,84 42,28 26),(52 18,66 23,73 9,48 6,52 18)),((59 18,67 18,67 13,15,59 18))) |
+-----+
```

ST_GeomCollFromWKB

Syntax

```
ST_GeomCollFromWKB(wkb[,srid])
ST_GeometryCollectionFromWKB(wkb[,srid])
GeomCollFromWKB(wkb[,srid])
GeometryCollectionFromWKB(wkb[,srid])
```

Description

Constructs a GEOMETRYCOLLECTION value using its WKB representation and SRID.

`ST_GeomCollFromWKB()`, `ST_GeometryCollectionFromWKB()`, `GeomCollFromWKB()` and `GeometryCollectionFromWKB()` are synonyms.

Examples

```
SET @g = ST_AsBinary(ST_GeomFromText('GEOMETRYCOLLECTION(POLYGON((5 5,10 5,10 10,5 5)),POINT(10 10))');

SELECT ST_AsText(ST_GeomCollFromWKB(@g));
+-----+
| ST_AsText(ST_GeomCollFromWKB(@g)) |
+-----+
| GEOMETRYCOLLECTION(POLYGON((5 5,10 5,10 10,5 5)),POINT(10 10)) |
+-----+
```

ST BOUNDARY

MariaDB starting with 10.1.2

The `ST_BOUNDARY` function was introduced in MariaDB 10.1.2.

Syntax

ST_BOUNDARY(g)
BOUNDARY(g)

Description

Returns a geometry that is the closure of the combinatorial boundary of the geometry value `g`.

`BOUNDARY()` is a synonym.

Examples

```
SELECT ST_AsText(ST_Boundary(ST_GeomFromText('LINESTRING(3 3,0 0, -3 3)')));  
+-----+  
| ST_AsText(ST_Boundary(ST_GeomFromText('LINESTRING(3 3,0 0, -3 3)'))) |  
+-----+  
| MULTIPOINT(3 3,-3 3) |  
+-----+  
  
SELECT ST_AsText(ST_Boundary(ST_GeomFromText('POLYGON((3 3,0 0, -3 3, 3 3))')));  
+-----+  
| ST_AsText(ST_Boundary(ST_GeomFromText('POLYGON((3 3,0 0, -3 3, 3 3))'))) |  
+-----+  
| LINESTRING(3 3,0 0,-3 3,3 3) |  
+-----+
```

ST_ENVELOPE

Syntax

```
ST_ENVELOPE(g)  
ENVELOPE(g)
```

Description

Returns the Minimum Bounding Rectangle (MBR) for the geometry value `g`. The result is returned as a Polygon value.

The polygon is defined by the corner points of the bounding box:

```
POLYGON((MINX MINY, MAXX MINY, MAXX MAXY, MINX MAXY, MINX MINY))
```

`ST_ENVELOPE()` and `ENVELOPE()` are synonyms.

Examples

```
SELECT AsText(ST_ENVELOPE(GeomFromText('LineString(1 1,4 4)')));  
+-----+  
| AsText(ST_ENVELOPE(GeomFromText('LineString(1 1,4 4)'))) |  
+-----+  
| POLYGON((1 1,4 1,4 4,1 4,1 1)) |  
+-----+
```

ST_ISCLOSED

Syntax

```
ST_IsClosed(g)  
IsClosed(g)
```

Description

Returns 1 if a given [LINESTRING](#)'s start and end points are the same, or 0 if they are not the same. Before MariaDB 10.1.5, returns NULL if not given a LINESTRING. After MariaDB 10.1.5, returns -1.

`ST_IsClosed()` and `IsClosed()` are synonyms.

Examples

```
SET @ls = 'LineString(0 0, 0 4, 4 4, 0 0)';
SELECT ST_ISCLOSED(GEOMFROMTEXT(@ls));
+-----+
| ST_ISCLOSED(GEOMFROMTEXT(@ls)) |
+-----+
|          1          |
+-----+

SET @ls = 'LineString(0 0, 0 4, 4 4, 0 1)';
SELECT ST_ISCLOSED(GEOMFROMTEXT(@ls));
+-----+
| ST_ISCLOSED(GEOMFROMTEXT(@ls)) |
+-----+
|          0          |
+-----+
```

BINARY Operator

This page describes the `BINARY` operator. For details about the data type, see [Binary Data Type](#).

Syntax

```
BINARY
```

Description

The `BINARY` operator casts the string following it to a binary string. This is an easy way to force a column comparison to be done byte by byte rather than character by character. This causes the comparison to be case sensitive even if the column isn't defined as `BINARY` or `BLOB`.

`BINARY` also causes trailing spaces to be significant.

Examples

```

SELECT 'a' = 'A';
+-----+
| 'a' = 'A' |
+-----+
|      1 |
+-----+

SELECT BINARY 'a' = 'A';
+-----+
| BINARY 'a' = 'A' |
+-----+
|          0 |
+-----+

SELECT 'a' = 'a ';
+-----+
| 'a' = 'a ' |
+-----+
|      1 |
+-----+

SELECT BINARY 'a' = 'a ';
+-----+
| BINARY 'a' = 'a ' |
+-----+
|          0 |
+-----+

```

CAST

Syntax

`CAST(expr AS type)`

Description

The `CAST()` function takes a value of one [type](#) and produces a value of another type, similar to the [CONVERT\(\)](#) function.

The type can be one of the following values:

- [BINARY](#)
- [CHAR](#)
- [DATE](#)
- [DATETIME](#)
- [DECIMAL\[M,D\]](#)
- [DOUBLE](#)
- [FLOAT](#) (from [MariaDB 10.4.5](#))
- [INTEGER](#)
 - Short for [SIGNED INTEGER](#)
- [SIGNED \[INTEGER\]](#)
- [UNSIGNED \[INTEGER\]](#)
- [TIME](#)
- [VARCHAR](#) (in [Oracle mode](#), from [MariaDB 10.3](#))

Contents

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

The main difference between `CAST` and `CONVERT()` is that `CONVERT(expr,type)` is ODBC syntax while `CAST(expr as type)` and `CONVERT(... USING ...)` are SQL92 syntax.

In [MariaDB 10.4](#) and later, you can use the `CAST()` function with the `INTERVAL` keyword.

MariaDB starting with [5.5.31](#)

Until [MariaDB 5.5.31](#), `x'HHHH'`, the standard SQL syntax for binary string literals, erroneously worked in the same way as `0xHHHH`. In 5.5.31 it was intentionally changed to behave as a string in all contexts (and never as a number).

This introduces an incompatibility with previous versions of MariaDB, and all versions of MySQL (see the example below).

Examples

Simple casts:

```
SELECT CAST("abc" AS BINARY);
SELECT CAST("1" AS UNSIGNED INTEGER);
SELECT CAST(123 AS CHAR CHARACTER SET utf8)
```

Note that when one casts to `CHAR` without specifying the character set, the `collation_connection` character set collation will be used. When used with `CHAR CHARACTER SET`, the default collation for that character set will be used.

```
SELECT COLLATION(CAST(123 AS CHAR));
+-----+
| COLLATION(CAST(123 AS CHAR)) |
+-----+
| latin1_swedish_ci           |
+-----+

SELECT COLLATION(CAST(123 AS CHAR CHARACTER SET utf8));
+-----+
| COLLATION(CAST(123 AS CHAR CHARACTER SET utf8)) |
+-----+
| utf8_general_ci              |
+-----+
```

If you also want to change the collation, you have to use the `COLLATE` operator:

```
SELECT COLLATION(CAST(123 AS CHAR CHARACTER SET utf8)
    COLLATE utf8_unicode_ci);
+-----+
| COLLATION(CAST(123 AS CHAR CHARACTER SET utf8) COLLATE utf8_unicode_ci) |
+-----+
| utf8_unicode_ci               |
+-----+
```

Using `CAST()` to order an `ENUM` field as a `CHAR` rather than the internal numerical value:

```
CREATE TABLE enum_list (enum_field enum('c','a','b'));

INSERT INTO enum_list (enum_field)
VALUES('c'),('a'),('c'),('b');
```

```
SELECT * FROM enum_list
ORDER BY enum_field;
```

```
+-----+
| enum_field |
+-----+
| c          |
| c          |
| a          |
| b          |
+-----+
```

```
SELECT * FROM enum_list
ORDER BY CAST(enum_field AS CHAR);
```

```
+-----+
| enum_field |
+-----+
| a          |
| b          |
| c          |
| c          |
+-----+
```

From MariaDB 5.5.31, the following will trigger warnings, since `x'aa'` and `X'aa'` no longer behave as a number. Previously, and in all versions of MySQL, no warnings are triggered since they did erroneously behave as a number:

```
SELECT CAST(0xAA AS UNSIGNED), CAST(x'aa' AS UNSIGNED), CAST(X'aa' AS UNSIGNED);
+-----+-----+
| CAST(0xAA AS UNSIGNED) | CAST(x'aa' AS UNSIGNED) | CAST(X'aa' AS UNSIGNED) |
+-----+-----+
|      170 |          0 |          0 |
+-----+-----+
1 row in set, 2 warnings (0.00 sec)

Warning (Code 1292): Truncated incorrect INTEGER value: '\xAA'
Warning (Code 1292): Truncated incorrect INTEGER value: '\xAA'
```

Casting to intervals:

```
SELECT CAST('2019-01-04 INTERVAL AS DAY_SECOND(2)' AS "Cast");
+-----+
| Cast   |
+-----+
| 00:20:17.00 |
+-----+
```

See Also

- [Supported data types](#)
- [Microseconds in MariaDB](#)
- [String literals](#)
- [COLLATION\(\)](#)
- [CONVERT\(\)](#)

CHAR Function

Syntax

```
CHAR(N, ... [USING charset_name])
```

Description

`CHAR()` interprets each argument as an `INT` and returns a string consisting of the characters given by the code values of those integers. `NUL` values are skipped. By default, `CHAR()` returns a binary string. To produce a string in a given `character set`, use the optional `USING` clause:

```
SELECT CHARSET(CHAR(0x65)), CHARSET(CHAR(0x65 USING utf8));
+-----+-----+
| CHARSET(CHAR(0x65)) | CHARSET(CHAR(0x65 USING utf8)) |
+-----+-----+
| binary            | utf8              |
+-----+-----+
```

If `USING` is given and the result string is illegal for the given character set, a warning is issued. Also, if strict `SQL mode` is enabled, the result from `CHAR()` becomes `NULL`.

Examples

```

SELECT CHAR(77,97,114,'105',97,'68',66);
+-----+
| CHAR(77,97,114,'105',97,'68',66) |
+-----+
| MariaDB |
+-----+

SELECT CHAR(77,77.3,'77.3');
+-----+
| CHAR(77,77.3,'77.3') |
+-----+
| MMM |
+-----+
1 row in set, 1 warning (0.00 sec)

Warning (Code 1292): Truncated incorrect INTEGER value: '77.3'

```

See Also

- [Character Sets and Collations](#)
- [ASCII\(\)](#) - Return ASCII value of first character
- [ORD\(\)](#) - Return value for character in single or multi-byte character sets
- [CHR](#) - Similar, Oracle-compatible, function

CHAR_LENGTH

Syntax

```

CHAR_LENGTH(str)
CHARACTER_LENGTH(str)

```

Description

Returns the length of the given string argument, measured in characters. A multi-byte character counts as a single character. This means that for a string containing five two-byte characters, `LENGTH()` (or `OCTET_LENGTH()` in Oracle mode) returns 10, whereas `CHAR_LENGTH()` returns 5. If the argument is `NULL`, it returns `NULL`.

If the argument is not a string value, it is converted into a string.

It is synonymous with the `CHARACTER_LENGTH()` function.

Contents

- [Syntax](#)
- [Description](#)
- [Examples](#)
- [See Also](#)

Examples

```

SELECT CHAR_LENGTH('MariaDB');
+-----+
| CHAR_LENGTH('MariaDB') |
+-----+
| 7 |
+-----+

```

When Oracle mode from MariaDB 10.3 is not set:

```

SELECT CHAR_LENGTH('π'), LENGTH('π'), LENGTHB('π'), OCTET_LENGTH('π');
+-----+-----+-----+
| CHAR_LENGTH('π') | LENGTH('π') | LENGTHB('π') | OCTET_LENGTH('π') |
+-----+-----+-----+
| 1 | 2 | 2 | 2 |
+-----+-----+-----+

```

In Oracle mode from MariaDB 10.3:

```
SELECT CHAR_LENGTH('π'), LENGTH('π'), LENGTHB('π'), OCTET_LENGTH('π');
+-----+-----+-----+
| CHAR_LENGTH('π') | LENGTH('π') | LENGTHB('π') | OCTET_LENGTH('π') |
+-----+-----+-----+
|           1 |         1 |         2 |         2 |
+-----+-----+-----+
```

See Also

- [LENGTH\(\)](#)
- [LENGTHB\(\)](#)
- [OCTET_LENGTH\(\)](#)
- Oracle mode from MariaDB 10.3

CHR

MariaDB starting with [10.3.1](#)

The `CHR()` function was introduced in [MariaDB 10.3.1](#) to provide Oracle compatibility

Syntax

`CHR(N)`

Description

`CHR()` interprets each argument N as an integer and returns a `VARCHAR(1)` string consisting of the character given by the code values of the integer. The character set and collation of the string are set according to the values of the `character_set_database` and `collation_database` system variables.

`CHR()` is similar to the `CHAR()` function, but only accepts a single argument.

`CHR()` is available in all `sql_modes`.

Examples

```

SELECT CHR(67);
+-----+
| CHR(67) |
+-----+
| C       |
+-----+

SELECT CHR('67');
+-----+
| CHR('67') |
+-----+
| C          |
+-----+

SELECT CHR('C');
+-----+
| CHR('C') |
+-----+
|          |
+-----+
1 row in set, 1 warning (0.000 sec)

SHOW WARNINGS;
+-----+
| Level | Code | Message           |
+-----+
| Warning | 1292 | Truncated incorrect INTEGER value: 'C' |
+-----+

```

See Also

- [Character Sets and Collations](#)
- [ASCII\(\)](#) - Return ASCII value of first character
- [ORD\(\)](#) - Return value for character in single or multi-byte character sets
- [CHAR\(\)](#) - Similar function which accepts multiple integers

CONCAT

Syntax

```
CONCAT(str1,str2,...)
```

Description

Returns the string that results from concatenating the arguments. May have one or more arguments. If all arguments are non-binary strings, the result is a non-binary string. If the arguments include any binary strings, the result is a binary string. A numeric argument is converted to its equivalent binary string form; if you want to avoid that, you can use an explicit type cast, as in this example:

```
SELECT CONCAT(CAST(int_col AS CHAR), char_col);
```

`CONCAT()` returns `NULL` if any argument is `NULL`.

A `NULL` parameter hides all information contained in other parameters from the result. Sometimes this is not desirable; to avoid this, you can:

- Use the `CONCAT_WS()` function with an empty separator, because that function is `NULL`-safe.
- Use `IFNULL()` to turn `NULLs` into empty strings.

Oracle Mode

Contents

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

In Oracle mode from MariaDB 10.3, CONCAT ignores NULL.

Examples

```
SELECT CONCAT('Ma', 'ria', 'DB');
+-----+
| CONCAT('Ma', 'ria', 'DB') |
+-----+
| MariaDB                |
+-----+

SELECT CONCAT('Ma', 'ria', NULL, 'DB');
+-----+
| CONCAT('Ma', 'ria', NULL, 'DB') |
+-----+
| NULL                      |
+-----+

SELECT CONCAT(42.0);
+-----+
| CONCAT(42.0) |
+-----+
| 42.0          |
+-----+
```

Using IFNULL() to handle NULLs:

```
SELECT CONCAT('The value of @v is: ', IFNULL(@v, ''));
```

In Oracle mode, from MariaDB 10.3:

```
SELECT CONCAT('Ma', 'ria', NULL, 'DB');
+-----+
| CONCAT('Ma', 'ria', NULL, 'DB') |
+-----+
| MariaDB                |
+-----+
```

See Also

- [GROUP_CONCAT\(\)](#)
- [Oracle mode from MariaDB 10.3](#)

CONCAT_WS

Syntax

```
CONCAT_WS(separator,str1,str2,...)
```

Description

CONCAT_WS() stands for Concatenate With Separator and is a special form of [CONCAT\(\)](#). The first argument is the separator for the rest of the arguments. The separator is added between the strings to be concatenated. The separator can be a string, as can the rest of the arguments.

If the separator is NULL, the result is NULL; all other NULL values are skipped. This makes CONCAT_WS() suitable when you want to concatenate

some values and avoid losing all information if one of them is `NULL`.

Examples

```
SELECT CONCAT_WS(',', 'First name', 'Second name', 'Last Name');
+-----+
| CONCAT_WS(',', 'First name', 'Second name', 'Last Name') |
+-----+
| First name,Second name,Last Name                         |
+-----+  
  
SELECT CONCAT_WS('-', 'Floor', NULL, 'Room');
+-----+
| CONCAT_WS('-', 'Floor', NULL, 'Room') |
+-----+
| Floor-Room                                         |
+-----+
```

In some cases, remember to include a space in the separator string:

```
SET @a = 'gnu', @b = 'penguin', @c = 'sea lion';
Query OK, 0 rows affected (0.00 sec)

SELECT CONCAT_WS(' ', @a, @b, @c);
+-----+
| CONCAT_WS(' ', @a, @b, @c) |
+-----+
| gnu, penguin, sea lion      |
+-----+
```

Using `CONCAT_WS()` to handle `NULL`s:

```
SET @a = 'a', @b = NULL, @c = 'c';

SELECT CONCAT_WS('', @a, @b, @c);
+-----+
| CONCAT_WS('', @a, @b, @c) |
+-----+
| ac                           |
+-----+
```

See Also

- [GROUP_CONCAT\(\)](#)

CONVERT

Syntax

```
CONVERT(expr,type), CONVERT(expr USING transcoding_name)
```

Description

The `CONVERT()` and `CAST()` functions take a value of one type and produce a value of another type.

The type can be one of the following values:

- [BINARY](#)
- [CHAR](#)
- [DATE](#)
- [DATETIME](#)

Contents

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

- [DECIMAL\[\(M,D\)\]](#)
- [DOUBLE](#)
- [FLOAT](#) (from MariaDB 10.4.5)
- [INTEGER](#)
 - Short for [SIGNED INTEGER](#)
- [SIGNED \[INTEGER\]](#)
- [UNSIGNED \[INTEGER\]](#)
- [TIME](#)
- [VARCHAR](#) (in Oracle mode, from MariaDB 10.3)

Note that in MariaDB, `INT` and `INTEGER` are the same thing.

`BINARY` produces a string with the `BINARY` data type. If the optional length is given, `BINARY(N)` causes the cast to use no more than `N` bytes of the argument. Values shorter than the given number in bytes are padded with `0x00` bytes to make them equal the length value.

`CHAR(N)` causes the cast to use no more than the number of characters given in the argument.

The main difference between the `CAST()` and `CONVERT()` is that `CONVERT(expr,type)` is ODBC syntax while `CAST(expr as type)` and `CONVERT(... USING ...)` are SQL92 syntax.

`CONVERT()` with `USING` is used to convert data between different [character sets](#). In MariaDB, transcoding names are the same as the corresponding character set names. For example, this statement converts the string 'abc' in the default character set to the corresponding string in the `utf8` character set:

```
SELECT CONVERT('abc' USING utf8);
```

Examples

```
SELECT enum_col FROM tbl_name
ORDER BY CAST(enum_col AS CHAR);
```

Converting a `BINARY` to string to permit the `LOWER` function to work:

```
SET @x = 'AardVark';
SET @x = BINARY 'AardVark';

SELECT LOWER(@x), LOWER(CONVERT (@x USING latin1));
+-----+-----+
| LOWER(@x) | LOWER(CONVERT (@x USING latin1)) |
+-----+-----+
| AardVark | aardvark
+-----+-----+
```

See Also

- [Character Sets and Collations](#)

ELT

Syntax

```
ELT(N, str1[, str2, str3,...])
```

Description

Takes a numeric argument and a series of string arguments. Returns the string that corresponds to the given numeric position. For instance, it returns `str1` if `N` is 1, `str2` if `N` is 2, and so on. If the numeric argument is a `FLOAT`, MariaDB rounds it to the nearest `INTEGER`. If the numeric argument is less than 1, greater than the total number of arguments, or not a number, `ELT()` returns `NULL`. It must have at least two arguments.

It is complementary to the `FIELD()` function.

Examples

```
SELECT ELT(1, 'ej', 'Heja', 'hej', 'foo');
+-----+
| ELT(1, 'ej', 'Heja', 'hej', 'foo') |
+-----+
| ej                                |
+-----+  
  
SELECT ELT(4, 'ej', 'Heja', 'hej', 'foo');
+-----+
| ELT(4, 'ej', 'Heja', 'hej', 'foo') |
+-----+
| foo                               |
+-----+
```

See also

- [FIND_IN_SET\(\)](#) function. Returns the position of a string in a set of strings.
- [FIELD\(\)](#) function. Returns the index position of a string in a list.

EXPORT_SET

Syntax

```
EXPORT_SET(bits, on, off[, separator[, number_of_bits]])
```

Description

Takes a minimum of three arguments. Returns a string where each bit in the given `bits` argument is returned, with the string values given for `on` and `off`.

Bits are examined from right to left, (from low-order to high-order bits). Strings are added to the result from left to right, separated by a separator string (defaults as `' , '`). You can optionally limit the number of bits the `EXPORT_SET()` function examines using the `number_of_bits` option.

If any of the arguments are set as `NULL`, the function returns `NULL`.

Examples

```
SELECT EXPORT_SET(5,'Y','N','','',4);
+-----+
| EXPORT_SET(5,'Y','N','','',4) |
+-----+
| Y,N,Y,N                         |
+-----+  
  
SELECT EXPORT_SET(6,'1','0','','',10);
+-----+
| EXPORT_SET(6,'1','0','','',10) |
+-----+
| 0,1,1,0,0,0,0,0,0,0             |
+-----+
```

FIELD

Syntax

```
FIELD(pattern, str1[,str2,...])
```

Description

Returns the index position of the string or number matching the given pattern. Returns `0` in the event that none of the arguments match the pattern. Raises an Error 1582 if not given at least two arguments.

When all arguments given to the `FIELD()` function are strings, they are treated as case-insensitive. When all the arguments are numbers, they are treated as numbers. Otherwise, they are treated as doubles.

If the given pattern occurs more than once, the `FIELD()` function only returns the index of the first instance. If the given pattern is `NULL`, the function returns `0`, as a `NULL` pattern always fails to match.

This function is complementary to the `ELT()` function.

Examples

```
SELECT FIELD('ej', 'Hej', 'ej', 'Heja', 'hej', 'foo')
      AS 'Field Results';
+-----+
| Field Results |
+-----+
|          2 |
+-----+  
  
SELECT FIELD('fo', 'Hej', 'ej', 'Heja', 'hej', 'foo')
      AS 'Field Results';
+-----+
| Field Results |
+-----+
|          0 |
+-----+  
  
SELECT FIELD(1, 2, 3, 4, 5, 1) AS 'Field Results';
+-----+
| Field Results |
+-----+
|          5 |
+-----+  
  
SELECT FIELD(NULL, 2, 3) AS 'Field Results';
+-----+
| Field Results |
+-----+
|          0 |
+-----+  
  
SELECT FIELD('fail') AS 'Field Results';
Error 1582 (42000): Incorrect parameter count in call
to native function 'field'
```

See also

- [ELT\(\)](#) function. Returns the N'th element from a set of strings.

EXTRACTVALUE

Syntax

```
EXTRACTVALUE(xml_frag, xpath_expr)
```

Description

The `EXTRACTVALUE()` function takes two string arguments: a fragment of XML markup and an XPath expression, (also known as a locator). It returns the text (That is, CDDATA), of the first text node which is a child of the element or elements matching the XPath expression.

In cases where a valid XPath expression does not match any text nodes in a valid XML fragment, (including the implicit `/text()` expression), the `EXTRACTVALUE()` function returns an empty string.

Invalid Arguments

When either the XML fragment or the XPath expression is `NULL`, the `EXTRACTVALUE()` function returns `NULL`. When the XML fragment is invalid, it raises a warning Code 1525:

```
Warning (Code 1525): Incorrect XML value: 'parse error at line 1 pos 11: unexpected END-C'
```

Contents

1. [Syntax](#)
2. [Description](#)
 1. [Invalid Arguments](#)
 2. [Explicit text\(\) Expressions](#)
 3. [Count Matches](#)
 4. [Matches](#)
3. [Examples](#)

When the XPath value is invalid, it generates an Error 1105:

```
ERROR 1105 (HY000): XPATH syntax error: ''
```

Explicit text() Expressions

This function is the equivalent of performing a match using the XPath expression after appending `/text()`. In other words:

```
SELECT
EXTRACTVALUE('<cases><case>example</case></cases>', '/cases/case') AS 'Base Example',
EXTRACTVALUE('<cases><case>example</case></cases>', '/cases/case/text()') AS 'text() Example';

+-----+-----+
| Base Example | text() Example |
+-----+-----+
| example      | example        |
+-----+-----+
```

Count Matches

When `EXTRACTVALUE()` returns multiple matches, it returns the content of the first child text node of each matching element, in the matched order, as a single, space-delimited string.

By design, the `EXTRACTVALUE()` function makes no distinction between a match on an empty element and no match at all. If you need to determine whether no matching element was found in the XML fragment or if an element was found that contained no child text nodes, use the XPath `count()` function.

For instance, when looking for a value that exists, but contains no child text nodes, you would get a count of the number of matching instances:

```
SELECT
EXTRACTVALUE('<cases><case/></cases>', '/cases/case') AS 'Empty Example',
EXTRACTVALUE('<cases><case/></cases>', 'count(/cases/case)') AS 'count() Example';

+-----+-----+
| Empty Example | count() Example |
+-----+-----+
|             |          1 |
+-----+-----+
```

Alternatively, when looking for a value that doesn't exist, `count()` returns 0.

```
SELECT
EXTRACTVALUE('<cases><case/></cases>', '/cases/person') AS 'No Match Example',
EXTRACTVALUE('<cases><case/></cases>', 'count(/cases/person)') AS 'count() Example';

+-----+-----+
| No Match Example | count() Example |
+-----+-----+
|                 |          0 |
+-----+-----+
```

Matches

Important: The `EXTRACTVALUE()` function only returns CDDATA. It does not return tags that the element might contain or the text that these child elements contain.

```
SELECT EXTRACTVALUE('<cases><case>Person<email>x@example.com</email></case></cases>', '/cases') AS Case;  
+-----+  
| Case |  
+-----+  
| Person |  
+-----+
```

Note, in the above example, while the XPath expression matches to the parent `<case>` instance, it does not return the contained `<email>` tag or its content.

Examples

```
SELECT  
    ExtractValue('<a>ccc<b>ddd</b></a>', '/a')          AS val1,  
    ExtractValue('<a>ccc<b>ddd</b></a>', '/a/b')        AS val2,  
    ExtractValue('<a>ccc<b>ddd</b></a>', '//b')         AS val3,  
    ExtractValue('<a>ccc<b>ddd</b></a>', '/b')          AS val4,  
    ExtractValue('<a>ccc<b>ddd</b><b>eee</b></a>', '//b') AS val5;  
+-----+-----+-----+-----+  
| val1 | val2 | val3 | val4 | val5 |  
+-----+-----+-----+-----+  
| ccc  | ddd  | ddd  |      | ddd eee |  
+-----+-----+-----+-----+
```

FIND_IN_SET

Syntax

```
FIND_IN_SET(pattern, strlist)
```

Description

Returns the index position where the given pattern occurs in a string list. The first argument is the pattern you want to search for. The second argument is a string containing comma-separated variables. If the second argument is of the `SET` data-type, the function is optimized to use bit arithmetic.

If the pattern does not occur in the string list or if the string list is an empty string, the function returns `0`. If either argument is `NULL`, the function returns `NULL`. The function does not return the correct result if the pattern contains a comma (" , ") character.

Examples

```
SELECT FIND_IN_SET('b','a,b,c,d') AS "Found Results";  
+-----+  
| Found Results |  
+-----+  
|      2 |  
+-----+
```

See Also

- [ELT\(\)](#) function. Returns the N'th element from a set of strings.

FORMAT

Syntax

```
FORMAT(num, decimal_position[, locale])
```

Description

Formats the given number for display as a string, adding separators to appropriate position and rounding the results to the given decimal position. For instance, it would format `15233.345` to `15,233.35`.

If the given decimal position is `0`, it rounds to return no decimal point or fractional part. You can optionally specify a `locale` value to format numbers to the pattern appropriate for the given region.

Examples

```
SELECT FORMAT(1234567890.09876543210, 4) AS 'Format';
+-----+
| Format      |
+-----+
| 1,234,567,890.0988 |
+-----+

SELECT FORMAT(1234567.89, 4) AS 'Format';
+-----+
| Format      |
+-----+
| 1,234,567.8900 |
+-----+

SELECT FORMAT(1234567.89, 0) AS 'Format';
+-----+
| Format      |
+-----+
| 1,234,568 |
+-----+

SELECT FORMAT(123456789,2,'rm_CH') AS 'Format';
+-----+
| Format      |
+-----+
| 123'456'789,00 |
+-----+
```

FROM_BASE64

Syntax

```
FROM_BASE64(str)
```

Description

Decodes the given base-64 encode string, returning the result as a binary string. Returns `NULL` if the given string is `NULL` or if it's invalid.

It is the reverse of the `TO_BASE64` function.

There are numerous methods to base-64 encode a string. MariaDB uses the following:

- It encodes alphabet value 64 as `'+'`.
- It encodes alphabet value 63 as `'/'`.
- It codes output in groups of four printable characters. Each three byte of data encoded uses four characters. If the final group is incomplete, it pads the difference with the `'='` character.
- It divides long output, adding a new line every 76 characters.

- In decoding, it recognizes and ignores newlines, carriage returns, tabs and space whitespace characters.

```
SELECT TO_BASE64('Maria') AS 'Input';
+-----+
| Input |
+-----+
| TWFyaWE= |
+-----+

SELECT FROM_BASE64('TWFyaWE=') AS 'Output';
+-----+
| Output |
+-----+
| Maria |
+-----+
```

HEX

Syntax

HEX(*N_or_S*)

Description

If *N_or_S* is a number, returns a string representation of the hexadecimal value of *N*, where *N* is a longlong (`BIGINT`) number. This is equivalent to `CONV(N,10,16)`.

If *N_or_S* is a string, returns a hexadecimal string representation of *N_or_S* where each byte of each character in *N_or_S* is converted to two hexadecimal digits. If *N_or_S* is NULL, returns NULL. The inverse of this operation is performed by the `UNHEX()` function.

MariaDB starting with 10.5.0

HEX() with an INET6 argument returns a hexadecimal representation of the underlying 16-byte binary string.

Examples

```
SELECT HEX(255);
+-----+
| HEX(255) |
+-----+
| FF         |
+-----+

SELECT 0x4D617269614442;
+-----+
| 0x4D617269614442 |
+-----+
| MariaDB      |
+-----+

SELECT HEX('MariaDB');
+-----+
| HEX('MariaDB') |
+-----+
| 4D617269614442 |
+-----+
```

From MariaDB 10.5.0:

```
SELECT HEX(CAST('2001:db8::ff00:42:8329' AS INET6));
+-----+
| HEX(CAST('2001:db8::ff00:42:8329' AS INET6)) |
+-----+
| 20010DB800000000000FF0000428329 |
+-----+
```

See Also

- [Hexadecimal literals](#)
- [UNHEX\(\)](#)
- [CONV\(\)](#)
- [BIN\(\)](#)
- [OCT\(\)](#)

INSERT Function

Syntax

```
INSERT(str,pos,len,newstr)
```

Description

Returns the string `str`, with the substring beginning at position `pos` and `len` characters long replaced by the string `newstr`. Returns the original string if `pos` is not within the length of the string. Replaces the rest of the string from position `pos` if `len` is not within the length of the rest of the string. Returns NULL if any argument is NULL.

Examples

```
SELECT INSERT('Quadratic', 3, 4, 'What');
+-----+
| INSERT('Quadratic', 3, 4, 'What') |
+-----+
| QuWhattic |
+-----+

SELECT INSERT('Quadratic', -1, 4, 'What');
+-----+
| INSERT('Quadratic', -1, 4, 'What') |
+-----+
| Quadratic |
+-----+

SELECT INSERT('Quadratic', 3, 100, 'What');
+-----+
| INSERT('Quadratic', 3, 100, 'What') |
+-----+
| QuWhat |
+-----+
```

LENGTH

Syntax

```
LENGTH(str)
```

Description

Returns the length of the string `str`.

In the default mode, when [Oracle mode from MariaDB 10.3](#) is not set, the length is measured in bytes. In this case, a multi-byte character counts as multiple bytes. This means that for a string containing five two-byte characters, `LENGTH()` returns 10, whereas [CHAR_LENGTH\(\)](#) returns 5.

When running [Oracle mode from MariaDB 10.3](#), the length is measured in characters, and `LENGTH` is a synonym for [CHAR_LENGTH\(\)](#).

If `str` is not a string value, it is converted into a string. If `str` is `NULL`, the function returns `NULL`.

Examples

```
SELECT LENGTH('MariaDB');
+-----+
| LENGTH('MariaDB') |
+-----+
|          7 |
+-----+
```

When [Oracle mode from MariaDB 10.3](#) is not set:

```
SELECT CHAR_LENGTH('π'), LENGTH('π'), LENGTHB('π'), OCTET_LENGTH('π');
+-----+-----+-----+
| CHAR_LENGTH('π') | LENGTH('π') | LENGTHB('π') | OCTET_LENGTH('π') |
+-----+-----+-----+
|          1 |         2 |         2 |         2 |
+-----+-----+-----+
```

In [Oracle mode from MariaDB 10.3](#):

```
SELECT CHAR_LENGTH('π'), LENGTH('π'), LENGTHB('π'), OCTET_LENGTH('π');
+-----+-----+-----+
| CHAR_LENGTH('π') | LENGTH('π') | LENGTHB('π') | OCTET_LENGTH('π') |
+-----+-----+-----+
|          1 |         1 |         2 |         2 |
+-----+-----+-----+
```

See Also

- [CHAR_LENGTH\(\)](#)
- [LENGTHB\(\)](#)
- [OCTET_LENGTH\(\)](#)
- [Oracle mode from MariaDB 10.3](#)

LOAD_FILE

Syntax

```
LOAD_FILE(file_name)
```

Description

Reads the file and returns the file contents as a string. To use this function, the file must be located on the server host, you must specify the full path name to the file, and you must have the FILE privilege. The file must be readable by all and it must be less than the size, in bytes, of the `max_allowed_packet` system variable. If the `secure_file_priv` system variable is set to a non-empty directory name, the file to be loaded must be located in that directory.

If the file does not exist or cannot be read because one of the preceding conditions is not satisfied, the function returns `NULL`.

Since [MariaDB 5.1](#), the `character_set_filesystem` system variable has controlled interpretation of file names that are given as literal strings.

Contents

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

Contents

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

Statements using the LOAD_FILE() function are not [safe for statement based replication](#). This is because the slave will execute the LOAD_FILE() command itself. If the file doesn't exist on the slave, the function will return NULL.

Examples

```
UPDATE t SET blob_col=LOAD_FILE('/tmp/picture') WHERE id=1;
```

See Also

- [SELECT INTO DUMPFILE](#)

LIKE

Syntax

```
expr LIKE pat [ESCAPE 'escape_char']
expr NOT LIKE pat [ESCAPE 'escape_char']
```

Description

Tests whether *expr* matches the pattern *pat*. Returns either 1 (TRUE) or 0 (FALSE). Both *expr* and *pat* may be any valid expression and are evaluated to strings. Patterns may use the following wildcard characters:

- % matches any number of characters, including zero.
- _ matches any single character.

Use NOT LIKE to test if a string does not match a pattern. This is equivalent to using the NOT operator on the entire LIKE expression.

If either the expression or the pattern is NULL , the result is NULL .

LIKE performs case-insensitive substring matches if the collation for the expression and pattern is case-insensitive. For case-sensitive matches, declare either argument to use a binary collation using COLLATE , or coerce either of them to a BINARY string using CAST . Use SHOW COLLATION to get a list of available collations. Collations ending in _bin are case-sensitive.

Numeric arguments are coerced to binary strings.

The _ wildcard matches a single character, not byte. It will only match a multi-byte character if it is valid in the expression's character set. For example, _ will match _utf8"€" , but it will not match _latin1"€" because the Euro sign is not a valid latin1 character. If necessary, use CONVERT to use the expression in a different character set.

If you need to match the characters _ or % , you must escape them. By default, you can prefix the wildcard characters the backslash character \ to escape them. The backslash is used both to encode special characters like newlines when a string is parsed as well as to escape wildcards in a pattern after parsing. Thus, to match an actual backslash, you sometimes need to double-escape it as "\ \\ \" .

To avoid difficulties with the backslash character, you can change the wildcard escape character using ESCAPE in a LIKE expression. The argument to ESCAPE must be a single-character string.

Examples

Select the days that begin with "T":

```
CREATE TABLE t1 (d VARCHAR(16));
INSERT INTO t1 VALUES ("Monday"), ("Tuesday"), ("Wednesday"), ("Thursday"), ("Friday"), ("Saturday"), ("Sunday");
SELECT * FROM t1 WHERE d LIKE "T%";
```

Contents

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

```
SELECT * FROM t1 WHERE d LIKE "T%";  
+-----+  
| d      |  
+-----+  
| Tuesday |  
| Thursday |  
+-----+
```

Select the days that contain the substring "es":

```
SELECT * FROM t1 WHERE d LIKE "%es%";
```

```
SELECT * FROM t1 WHERE d LIKE "%es%";  
+-----+  
| d      |  
+-----+  
| Tuesday |  
| Wednesday |  
+-----+
```

Select the six-character day names:

```
SELECT * FROM t1 WHERE d like "__day";
```

```
SELECT * FROM t1 WHERE d like "__day";  
+-----+  
| d      |  
+-----+  
| Monday |  
| Friday |  
| Sunday |  
+-----+
```

With the default collations, `LIKE` is case-insensitive:

```
SELECT * FROM t1 where d like "t%";
```

```
SELECT * FROM t1 where d like "t%";  
+-----+  
| d      |  
+-----+  
| Tuesday |  
| Thursday |  
+-----+
```

Use `COLLATE` to specify a binary collation, forcing case-sensitive matches:

```
SELECT * FROM t1 WHERE d like "t%" COLLATE latin1_bin;
```

```
SELECT * FROM t1 WHERE d like "t%" COLLATE latin1_bin;  
Empty set (0.00 sec)
```

You can include functions and operators in the expression to match. Select dates based on their day name:

```

CREATE TABLE t2 (d DATETIME);
INSERT INTO t2 VALUES
    ("2007-01-30 21:31:07"),
    ("1983-10-15 06:42:51"),
    ("2011-04-21 12:34:56"),
    ("2011-10-30 06:31:41"),
    ("2011-01-30 14:03:25"),
    ("2004-10-07 11:19:34");
SELECT * FROM t2 WHERE DAYNAME(d) LIKE "T%";

```

```

SELECT * FROM t2 WHERE DAYNAME(d) LIKE "T%";
+-----+
| d      |
+-----+
| 2007-01-30 21:31 |
| 2011-04-21 12:34 |
| 2004-10-07 11:19 |
+-----+
3 rows in set, 7 warnings (0.00 sec)

```

Optimizing LIKE

- MariaDB can use indexes for LIKE on string columns in the case where the LIKE doesn't start with % or _.
- Starting from [MariaDB 10.0](#), one can set the `optimizer_use_condition_selectivity` variable to 5. If this is done, then the optimizer will read `optimizer_selectivity_sampling_limit` rows to calculate the selectivity of the LIKE expression before starting to calculate the query plan. This can help speed up some LIKE queries by providing the optimizer with more information about your data.

See Also

- For searches on text columns, with results sorted by relevance, see [full-text](#) indexes.
- For more complex searches and operations on strings, you can use [regular expressions](#), which were enhanced in MariaDB 10 (see [PCRE Regular Expressions](#)).

LOCATE

Syntax

```
LOCATE(substr,str), LOCATE(substr,str,pos)
```

Description

The first syntax returns the position of the first occurrence of substring `substr` in string `str`. The second syntax returns the position of the first occurrence of substring `substr` in string `str`, starting at position `pos`. Returns 0 if `substr` is not in `str`.

`LOCATE()` performs a case-insensitive search.

If any argument is `NULL`, returns `NULL`.

[INSTR\(\)](#) is the same as the two-argument form of `LOCATE()`, except that the order of the arguments is reversed.

Examples

```

SELECT LOCATE('bar', 'foobarbar');
+-----+
| LOCATE('bar', 'foobarbar') |
+-----+
|          4 |
+-----+

SELECT LOCATE('My', 'Maria');
+-----+
| LOCATE('My', 'Maria') |
+-----+
|          0 |
+-----+

SELECT LOCATE('bar', 'foobarbar', 5);
+-----+
| LOCATE('bar', 'foobarbar', 5) |
+-----+
|          7 |
+-----+

```

See Also

- [INSTR\(\)](#) ; Returns the position of a string within a string
- [SUBSTRING_INDEX\(\)](#) ; Returns the substring from string before count occurrences of a delimiter

LOWER

Syntax

```
LOWER(str)
```

Description

Returns the string `str` with all characters changed to lowercase according to the current character set mapping. The default is latin1 (cp1252 West European).

Contents

- [Syntax](#)
- [Description](#)
- [Examples](#)

Examples

```

SELECT LOWER('QUADRATICALLY');
+-----+
| LOWER('QUADRATICALLY') |
+-----+
| quadratically         |
+-----+

```

`LOWER()` (and `UPPER()`) are ineffective when applied to binary strings (`BINARY`, `VARBINARY`, `BLOB`). To perform lettercase conversion, `CONVERT` the string to a non-binary string:

```

SET @str = BINARY 'North Carolina';

SELECT LOWER(@str), LOWER(CONVERT(@str USING latin1));
+-----+
| LOWER(@str)    | LOWER(CONVERT(@str USING latin1)) |
+-----+
| North Carolina | north carolina                 |
+-----+

```

LPAD

Syntax

```
LPAD(str, len [,padstr])
```

Description

Returns the string `str`, left-padded with the string `padstr` to a length of `len` characters. If `str` is longer than `len`, the return value is shortened to `len` characters. If `padstr` is omitted, the LPAD function pads spaces.

Prior to MariaDB 10.3.1, the `padstr` parameter was mandatory.

Returns NULL if given a NULL argument. If the result is empty (zero length), returns either an empty string or, from MariaDB 10.3.6 with `SQL_MODE=Oracle`, NULL.

The Oracle mode version of the function can be accessed outside of Oracle mode by using `LPAD_ORACLE` as the function name.

Examples

```
SELECT LPAD('hello',10,'.');
+-----+
| LPAD('hello',10,'.') |
+-----+
| .....hello         |
+-----+

SELECT LPAD('hello',2,'.');
+-----+
| LPAD('hello',2,'.') |
+-----+
| he                  |
+-----+
```

From MariaDB 10.3.1, with the pad string defaulting to space.

```
SELECT LPAD('hello',10);
+-----+
| LPAD('hello',10) |
+-----+
|      hello      |
+-----+
```

Oracle mode version from MariaDB 10.3.6:

```
SELECT LPAD(' ',0),LPAD_ORACLE(' ',0);
+-----+-----+
| LPAD(' ',0) | LPAD_ORACLE(' ',0) |
+-----+-----+
|          | NULL           |
+-----+-----+
```

See Also

- [RPAD](#) - Right-padding instead of left-padding.

LTRIM

Syntax

```
LTRIM(str)
```

Contents

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

Description

Returns the string `str` with leading space characters removed.

Returns `NULL` if given a `NULL` argument. If the result is empty, returns either an empty string, or, from MariaDB 10.3.6 with `SQL_MODE=Oracle`, `NULL`.

The Oracle mode version of the function can be accessed outside of Oracle mode by using `LTRIM_ORACLE` as the function name.

Contents

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

Examples

```
SELECT QUOTE(LTRIM(' MariaDB '));  
+-----+  
| QUOTE(LTRIM(' MariaDB ')) |  
+-----+  
| 'MariaDB' |  
+-----+
```

Oracle mode version from MariaDB 10.3.6:

```
SELECT LTRIM(''),LTRIM_ORACLE('');  
+-----+-----+  
| LTRIM('') | LTRIM_ORACLE('') |  
+-----+-----+  
| NULL |  
+-----+
```

See Also

- [RTRIM](#) - trailing spaces removed
- [TRIM](#) - removes all given prefixes or suffixes

MAKE_SET

Syntax

```
MAKE_SET(bits,str1,str2,...)
```

Description

Returns a set value (a string containing substrings separated by "," characters) consisting of the strings that have the corresponding bit in `bits` set. `str1` corresponds to bit 0, `str2` to bit 1, and so on. `NULL` values in `str1`, `str2`, ... are not appended to the result.

Examples

```

SELECT MAKE_SET(1, 'a', 'b', 'c');
+-----+
| MAKE_SET(1, 'a', 'b', 'c') |
+-----+
| a |
+-----+

SELECT MAKE_SET(1 | 4, 'hello', 'nice', 'world');
+-----+
| MAKE_SET(1 | 4, 'hello', 'nice', 'world') |
+-----+
| hello,world |
+-----+

SELECT MAKE_SET(1 | 4, 'hello', 'nice', NULL, 'world');
+-----+
| MAKE_SET(1 | 4, 'hello', 'nice', NULL, 'world') |
+-----+
| hello |
+-----+

SELECT QUOTE(MAKE_SET(0, 'a', 'b', 'c'));
+-----+
| QUOTE(MAKE_SET(0, 'a', 'b', 'c')) |
+-----+
| '' |
+-----+

```

MATCH AGAINST

Syntax

```
MATCH (col1,col2,...) AGAINST (expr [search_modifier])
```

Description

A special construct used to perform a fulltext search on a fulltext index.

See [Fulltext Index Overview](#) for a full description, and [Full-text Indexes](#) for more articles on the topic.

Examples

```

CREATE TABLE ft_myisam(copy TEXT,FULLTEXT(copy)) ENGINE=MyISAM;

INSERT INTO ft_myisam(copy) VALUES ('Once upon a time'), ('There was a wicked witch'),
('Who ate everybody up');

SELECT * FROM ft_myisam WHERE MATCH(copy) AGAINST('wicked');
+-----+
| copy |
+-----+
| There was a wicked witch |
+-----+

```

```

SELECT id, body, MATCH (title,body) AGAINST
('Security implications of running MySQL as root'
IN NATURAL LANGUAGE MODE) AS score
FROM articles WHERE MATCH (title,body) AGAINST
('Security implications of running MySQL as root'
IN NATURAL LANGUAGE MODE);
+-----+-----+
| id | body | score |
+-----+-----+
| 4 | 1. Never run mysqld as root. 2. ... | 1.5219271183014 |
| 6 | When configured properly, MySQL ... | 1.3114095926285 |
+-----+-----+

```

REGEXP_INSTR

MariaDB starting with 10.0.5

REGEXP_INSTR was introduced in MariaDB 10.0.5.

Syntax

```
REGEXP_INSTR(subject, pattern)
```

Returns the position of the first occurrence of the regular expression `pattern` in the string `subject`, or 0 if pattern was not found.

The positions start with 1 and are measured in characters (i.e. not in bytes), which is important for multi-byte character sets. You can cast a multi-byte character set to `BINARY` to get offsets in bytes.

The function follows the case sensitivity rules of the effective `collation`. Matching is performed case insensitively for case insensitive collations, and case sensitively for case sensitive collations and for binary data.

The collation case sensitivity can be overwritten using the (?i) and (?-i) PCRE flags.

MariaDB 10.0.5 switched to the `PCRE regular expression` library for enhanced regular expression performance, and REGEXP_INSTR was introduced as part of this enhancement.

Examples

```

SELECT REGEXP_INSTR('abc','b');
-> 2

SELECT REGEXP_INSTR('abc','x');
-> 0

SELECT REGEXP_INSTR('BJÖRN','N');
-> 5

```

Casting a multi-byte character set as `BINARY` to get offsets in bytes:

```

SELECT REGEXP_INSTR(BINARY 'BJÖRN','N') AS cast_utf8_to_binary;
-> 6

```

Case sensitivity:

```
SELECT REGEXP_INSTR('ABC','b');
-> 2

SELECT REGEXP_INSTR('ABC' COLLATE utf8_bin,'b');
-> 0

SELECT REGEXP_INSTR(BINARY'ABC','b');
-> 0

SELECT REGEXP_INSTR('ABC','(?-i)b');
-> 0

SELECT REGEXP_INSTR('ABC' COLLATE utf8_bin,'(?i)b');
-> 2
```

REGEXP_REPLACE

MariaDB starting with [10.0.5](#)

REGEXP_REPLACE was introduced in [MariaDB 10.0.5](#).

Syntax

```
REGEXP_REPLACE(subject, pattern, replace)
```

Description

`REGEXP_REPLACE` returns the string `subject` with all occurrences of the regular expression `pattern` replaced by the string `replace`. If no occurrences are found, then `subject` is returned as is.

The replace string can have backreferences to the subexpressions in the form `\N`, where N is a number from 1 to 9.

The function follows the case sensitivity rules of the effective [collation](#). Matching is performed case insensitively for case insensitive collations, and case sensitively for case sensitive collations and for binary data.

The collation case sensitivity can be overwritten using the `(?i)` and `(?-i)` PCRE flags.

[MariaDB 10.0.5](#) switched to the [PCRE regular expression](#) library for enhanced regular expression performance, and `REGEXP_REPLACE` was introduced as part of this enhancement.

[MariaDB 10.0.11](#) introduced the `default_regex_flags` variable to address the remaining compatibilities between PCRE and the old regex library.

Examples

```
SELECT REGEXP_REPLACE('ab12cd','[0-9]', '') AS remove_digits;
-> abcd

SELECT REGEXP_REPLACE('<html><head><title>title</title><body>body</body></html>', '<.+?>', ' ')
AS strip_html;
-> title body
```

Backreferences to the subexpressions in the form `\N`, where N is a number from 1 to 9:

```
SELECT REGEXP_REPLACE('James Bond','^(.*)(.*$)', '\2, \1') AS reorder_name;
-> Bond, James
```

Case insensitive and case sensitive matches:

```
SELECT REGEXP_REPLACE('ABC','b','-') AS case_insensitive;
-> A-C

SELECT REGEXP_REPLACE('ABC' COLLATE utf8_bin,'b','-') AS case_sensitive;
-> ABC

SELECT REGEXP_REPLACE(BINARY 'ABC','b','-') AS binary_data;
-> ABC
```

Overwriting the collation case sensitivity using the (?i) and (?-i) PCRE flags.

```
SELECT REGEXP_REPLACE('ABC','(?-i)b','-') AS force_case_sensitive;
-> ABC

SELECT REGEXP_REPLACE(BINARY 'ABC','(?i)b','-') AS force_case_insensitive;
-> A-C
```

REGEXP_SUBSTR

MariaDB starting with 10.0.5

REGEXP_SUBSTR was introduced in [MariaDB 10.0.5](#).

Syntax

```
REGEXP_SUBSTR(subject,pattern)
```

Description

Returns the part of the string `subject` that matches the regular expression `pattern`, or an empty string if `pattern` was not found.

The function follows the case sensitivity rules of the effective [collation](#). Matching is performed case insensitively for case insensitive collations, and case sensitively for case sensitive collations and for binary data.

The collation case sensitivity can be overwritten using the (?i) and (?-i) PCRE flags.

[MariaDB 10.0.5](#) switched to the [PCRE regular expression](#) library for enhanced regular expression performance, and `REGEXP_SUBSTR` was introduced as part of this enhancement.

[MariaDB 10.0.11](#) introduced the `default_regex_flags` variable to address the remaining compatibilities between PCRE and the old regex library.

Examples

```
SELECT REGEXP_SUBSTR('ab12cd','[0-9]+');
-> 12

SELECT REGEXP_SUBSTR(
  'See https://mariadb.org/en/foundation/ for details',
  'https?://[^/]*');
-> https://mariadb.org
```

```

SELECT REGEXP_SUBSTR('ABC','b');
-> B

SELECT REGEXP_SUBSTR('ABC' COLLATE utf8_bin,'b');
->

SELECT REGEXP_SUBSTR(BINARY'ABC','b');
->

SELECT REGEXP_SUBSTR('ABC','(?i)b');
-> B

SELECT REGEXP_SUBSTR('ABC' COLLATE utf8_bin,'(?+i)b');
-> B

```

RPAD

Syntax

```
RPAD(str, len [, padstr])
```

Description

Returns the string `str`, right-padded with the string `padstr` to a length of `len` characters. If `str` is longer than `len`, the return value is shortened to `len` characters. If `padstr` is omitted, the RPAD function pads spaces.

Prior to MariaDB 10.3.1, the `padstr` parameter was mandatory.

Returns NULL if given a NULL argument. If the result is empty (a length of zero), returns either an empty string, or, from MariaDB 10.3.6 with `SQL_MODE=Oracle`, NULL.

The Oracle mode version of the function can be accessed outside of Oracle mode by using `RPAD_ORACLE` as the function name.

Contents

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

Examples

```

SELECT RPAD('hello',10,'.');
+-----+
| RPAD('hello',10,'.') |
+-----+
| hello.....          |
+-----+

SELECT RPAD('hello',2,'.');
+-----+
| RPAD('hello',2,'.') |
+-----+
| he                  |
+-----+

```

From MariaDB 10.3.1, with the pad string defaulting to space.

```

SELECT RPAD('hello',30);
+-----+
| RPAD('hello',30)           |
+-----+
| hello                      |
+-----+

```

Oracle mode version from MariaDB 10.3.6:

```
SELECT RPAD(' ',0),RPAD_ORACLE(' ',0);
+-----+-----+
| RPAD(' ',0) | RPAD_ORACLE(' ',0) |
+-----+-----+
|          | NULL           |
+-----+-----+
```

See Also

- [LPAD](#) - Left-padding instead of right-padding.

RTRIM

Syntax

```
RTRIM(str)
```

Description

Returns the string `str` with trailing space characters removed.

Returns `NULL` if given a `NULL` argument. If the result is empty, returns either an empty string, or, from [MariaDB 10.3.6](#) with `SQL_MODE=Oracle`, `NULL`.

The Oracle mode version of the function can be accessed outside of Oracle mode by using `RTRIM_ORACLE` as the function name.

Contents

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

Examples

```
SELECT QUOTE(RTRIM('MariaDB    '));
+-----+
| QUOTE(RTRIM('MariaDB    ')) |
+-----+
| 'MariaDB'                   |
+-----+
```

Oracle mode version from [MariaDB 10.3.6](#):

```
SELECT RTRIM(''),RTRIM_ORACLE('');
+-----+-----+
| RTRIM('') | RTRIM_ORACLE('') |
+-----+-----+
|          | NULL           |
+-----+-----+
```

See Also

- [LTRIM](#) - leading spaces removed
- [TRIM](#) - removes all given prefixes or suffixes

SOUNDEX

Syntax

```
SOUNDEX(str)
```

Description

Returns a soundex string from `str`. Two strings that sound almost the same should have identical soundex strings. A standard soundex string is four characters long, but the `SOUNDEX()` function returns an arbitrarily long string. You can use `SUBSTRING()` on the result to get a standard soundex string. All non-alphabetic characters in `str` are ignored. All international alphabetic characters outside the A-Z range are treated as vowels.

Important: When using `SOUNDEX()`, you should be aware of the following details:

- This function, as currently implemented, is intended to work well with strings that are in the English language only. Strings in other languages may not produce reasonable results.
- This function implements the original Soundex algorithm, not the more popular enhanced version (also described by D. Knuth). The difference is that original version discards vowels first and duplicates second, whereas the enhanced version discards duplicates first and vowels second.

Examples

```
SOUNDEX('Hello');
+-----+
| SOUNDEX('Hello') |
+-----+
| H400           |
+-----+
```

```
SELECT SOUNDEX('MariaDB');
+-----+
| SOUNDEX('MariaDB') |
+-----+
| M631            |
+-----+
```

```
SELECT SOUNDEX('Knowledgebase');
+-----+
| SOUNDEX('Knowledgebase') |
+-----+
| K543212          |
+-----+
```

```
SELECT givenname, surname FROM users WHERE SOUNDEX(givenname) = SOUNDEX("robert");
+-----+
| givenname | surname |
+-----+
| Roberto   | Castro  |
+-----+
```

See Also

- [SOUNDS LIKE\(\)](#)

SUBSTRING

Syntax

```
SUBSTRING(str,pos),  
SUBSTRING(str FROM pos),  
SUBSTRING(str,pos,len),  
SUBSTRING(str FROM pos FOR len)
```

```
SUBSTR(str,pos),  
SUBSTR(str FROM pos),  
SUBSTR(str,pos,len),  
SUBSTR(str FROM pos FOR len)
```

Description

The forms without a *Len* argument return a substring from string *str* starting at position *pos*.

The forms with a *Len* argument return a substring *Len* characters long from string *str*, starting at position *pos*.

The forms that use *FROM* are standard SQL syntax.

It is also possible to use a negative value for *pos*. In this case, the beginning of the substring is *pos* characters from the end of the string, rather than the beginning. A negative value may be used for *pos* in any of the forms of this function.

By default, the position of the first character in the string from which the substring is to be extracted is reckoned as 1. For [Oracle-compatibility](#), from [MariaDB 10.3.3](#), when `sql_mode` is set to 'oracle', position zero is treated as position 1 (although the first character is still reckoned as 1).

If any argument is `NULL`, returns `NULL`.

Contents

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

Examples

```

SELECT SUBSTRING('Knowledgebase',5);
+-----+
| SUBSTRING('Knowledgebase',5) |
+-----+
| ledgebase |
+-----+

SELECT SUBSTRING('MariaDB' FROM 6);
+-----+
| SUBSTRING('MariaDB' FROM 6) |
+-----+
| DB |
+-----+

SELECT SUBSTRING('Knowledgebase',3,7);
+-----+
| SUBSTRING('Knowledgebase',3,7) |
+-----+
| owledge |
+-----+

SELECT SUBSTRING('Knowledgebase', -4);
+-----+
| SUBSTRING('Knowledgebase', -4) |
+-----+
| base |
+-----+

SELECT SUBSTRING('Knowledgebase', -8, 4);
+-----+
| SUBSTRING('Knowledgebase', -8, 4) |
+-----+
| edge |
+-----+

SELECT SUBSTRING('Knowledgebase' FROM -8 FOR 4);
+-----+
| SUBSTRING('Knowledgebase' FROM -8 FOR 4) |
+-----+
| edge |
+-----+

```

Oracle mode from MariaDB 10.3.3:

```

SELECT SUBSTR('abc',0,3);
+-----+
| SUBSTR('abc',0,3) |
+-----+
| abc |
+-----+

SELECT SUBSTR('abc',1,2);
+-----+
| SUBSTR('abc',1,2) |
+-----+
| ab |
+-----+

SET sql_mode='oracle';

SELECT SUBSTR('abc',0,3);
+-----+
| SUBSTR('abc',0,3) |
+-----+
| abc |
+-----+

SELECT SUBSTR('abc',1,2);
+-----+
| SUBSTR('abc',1,2) |
+-----+
| ab |
+-----+

```

See Also

- [INSTR\(\)](#) - Returns the position of a string within a string
- [LOCATE\(\)](#) - Returns the position of a string within a string
- [SUBSTRING_INDEX\(\)](#) - Returns a string based on substring

SUBSTRING_INDEX

Syntax

```
SUBSTRING_INDEX(str,delim,count)
```

Description

Returns the substring from string *str* before count occurrences of the delimiter *delim*. If *count* is positive, everything to the left of the final delimiter (counting from the left) is returned. If *count* is negative, everything to the right of the final delimiter (counting from the right) is returned.

SUBSTRING_INDEX() performs a case-sensitive match when searching for *delim*.

If any argument is `NULL`, returns `NULL`.

For example

```
SUBSTRING_INDEX('www.mariadb.org', '.', 2)
```

means "Return all of the characters up to the 2nd occurrence of '.'"

Contents

- [Syntax](#)
- [Description](#)
- [Examples](#)
- [See Also](#)

Examples

```
SELECT SUBSTRING_INDEX('www.mariadb.org', '.', 2);
+-----+
| SUBSTRING_INDEX('www.mariadb.org', '.', 2) |
+-----+
| www.mariadb                         |
+-----+

SELECT SUBSTRING_INDEX('www.mariadb.org', '.', -2);
+-----+
| SUBSTRING_INDEX('www.mariadb.org', '.', -2) |
+-----+
| mariadb.org                          |
+-----+
```

See Also

- [INSTR\(\)](#) - Returns the position of a string within a string
- [LOCATE\(\)](#) - Returns the position of a string within a string
- [SUBSTRING\(\)](#) - Returns a string based on position

TO_BASE64

MariaDB starting with [10.0.5](#)

The TO_BASE64() function was introduced in [MariaDB 10.0.5](#).

Syntax

```
TO_BASE64(str)
```

Description

Converts the string argument `str` to its base-64 encoded form, returning the result as a character string in the connection character set and collation.

The argument `str` will be converted to string first if it is not a string. A NULL argument will return a NULL result.

The reverse function, [FROM_BASE64\(\)](#), decodes an encoded base-64 string.

There are a numerous different methods to base-64 encode a string. The following are used by MariaDB and MySQL:

- Alphabet value 64 is encoded as '+'.
- Alphabet value 63 is encoded as '/'.
- Encoding output is made up of groups of four printable characters, with each three bytes of data encoded using four characters. If the final group is not complete, it is padded with '=' characters to make up a length of four.
- To divide long output, a newline is added after every 76 characters.
- Decoding will recognize and ignore newlines, carriage returns, tabs, and spaces.

Examples

```
SELECT TO_BASE64('Maria');
+-----+
| TO_BASE64('Maria') |
+-----+
| TWFyaWE=          |
+-----+
```

TRIM

Syntax

```
TRIM([{BOTH | LEADING | TRAILING} [remstr] FROM] str), TRIM([remstr FROM] str)
```

From MariaDB 10.3.6

```
TRIM_ORACLE([{BOTH | LEADING | TRAILING} [remstr] FROM] str), TRIM([remstr FROM] str)
```

Description

Returns the string `str` with all `remstr` prefixes or suffixes removed. If none of the specifiers `BOTH`, `LEADING`, or `TRAILING` is given, `BOTH` is assumed. `remstr` is optional and, if not specified, spaces are removed.

Returns `NULL` if given a `NULL` argument. If the result is empty, returns either an empty string, or, from MariaDB 10.3.6 with `SQL_MODE=Oracle`, `NULL`. `SQL_MODE=Oracle` is not set by default.

The Oracle mode version of the function can be accessed in any mode by using `TRIM_ORACLE` as the function name.

Examples

```
SELECT TRIM(' bar ')\\G
*****
1. row *****
TRIM(' bar '): bar

SELECT TRIM(LEADING 'x' FROM 'xxxbarxxx')\\G
*****
1. row *****
TRIM(LEADING 'x' FROM 'xxxbarxxx'): barxxx

SELECT TRIM(BOTH 'x' FROM 'xxxbarxxx')\\G
*****
1. row *****
TRIM(BOTH 'x' FROM 'xxxbarxxx'): bar

SELECT TRIM(TRAILING 'xyz' FROM 'barxyz')\\G
*****
1. row *****
TRIM(TRAILING 'xyz' FROM 'barxyz'): barx
```

From MariaDB 10.3.6, with `SQL_MODE=Oracle` not set:

```
SELECT TRIM(''),TRIM_ORACLE('');
+-----+
| TRIM('') | TRIM_ORACLE('') |
+-----+
|      | NULL      |
+-----+
```

From MariaDB 10.3.6, with `SQL_MODE=Oracle` set:

```
SELECT TRIM(''),TRIM_ORACLE('');
+-----+
| TRIM('') | TRIM_ORACLE('') |
+-----+
| NULL     | NULL     |
+-----+
```

See Also

- [LTRIM](#) - leading spaces removed
- [RTRIM](#) - trailing spaces removed

UNHEX

Contents

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

Syntax

```
UNHEX(str)
```

Description

Performs the inverse operation of [HEX\(str\)](#). That is, it interprets each pair of hexadecimal digits in the argument as a number and converts it to the character represented by the number. The resulting characters are returned as a binary string.

If str is NULL , UNHEX() returns NULL .

Examples

```
SELECT HEX('MariaDB');
+-----+
| HEX('MariaDB') |
+-----+
| 4D617269614442 |
+-----+

SELECT UNHEX('4D617269614442');
+-----+
| UNHEX('4D617269614442') |
+-----+
| MariaDB |
+-----+

SELECT 0x4D617269614442;
+-----+
| 0x4D617269614442 |
+-----+
| MariaDB |
+-----+

SELECT UNHEX(HEX('string'));
+-----+
| UNHEX(HEX('string')) |
+-----+
| string |
+-----+

SELECT HEX(UNHEX('1267'));
+-----+
| HEX(UNHEX('1267')) |
+-----+
| 1267 |
+-----+
```

See Also

- [Hexadecimal literals](#)
- [HEX\(\)](#)
- [CONV\(\)](#)

UPDATEXML

Syntax

```
UpdateXML(xml_target, xpath_expr, new_xml)
```

Description

This function replaces a single portion of a given fragment of XML markup `xml_target` with a new XML fragment `new_xml`, and then returns the changed XML. The portion of `xml_target` that is replaced matches an XPath expression `xpath_expr` supplied by the user. If no expression matching `xpath_expr` is found, or if multiple matches are found, the function returns the original `xml_target` XML fragment. All three arguments should be strings.

Examples

```
SELECT
  UpdateXML('<a><b>ccc</b><d></d></a>', '/a', '<e>ffff</e>') AS val1,
  UpdateXML('<a><b>ccc</b><d></d></a>', '/b', '<e>ffff</e>') AS val2,
  UpdateXML('<a><b>ccc</b><d></d></a>', '//b', '<e>ffff</e>') AS val3,
  UpdateXML('<a><b>ccc</b><d></d></a>', '/a/d', '<e>ffff</e>') AS val4,
  UpdateXML('<a><d></d><b>ccc</b><d></d></a>', '/a/d', '<e>ffff</e>') AS val5
  NG
*****
1. row *****
val1: <e>ffff</e>
val2: <a><b>ccc</b><d></d></a>
val3: <a><e>ffff</e><d></d></a>
val4: <a><b>ccc</b><e>ffff</e></a>
val5: <a><d></d><b>ccc</b><d></d></a>
1 row in set (0.00 sec)
```

WEIGHT_STRING

MariaDB starting with **10.0.5**

The `WEIGHT_STRING` function was introduced in [MariaDB 10.0.5](#).

Syntax

```
WEIGHT_STRING(str [AS {CHAR|BINARY}(N)] [LEVEL levels] [flags])
  levels: N [ASC|DESC|REVERSE] [, N [ASC|DESC|REVERSE]] ...
```

Description

Returns a binary string representing the string's sorting and comparison value. A string with a lower result means that for sorting purposes the string appears before a string with a higher result.

`WEIGHT_STRING()` is particularly useful when adding new collations, for testing purposes.

If `str` is a non-binary string (`CHAR`, `VARCHAR` or `TEXT`), `WEIGHT_STRING` returns the string's collation weight. If `str` is a binary string (`BINARY`, `VARBINARY` or `BLOB`), the return value is simply the input value, since the weight for each byte in a binary string is the byte value.

`WEIGHT_STRING()` returns `NULL` if given a `NULL` input.

The optional `AS` clause permits casting the input string to a binary or non-binary string, as well as to a particular length.

`AS BINARY(N)` measures the length in bytes rather than characters, and right pads with `0x00` bytes to the desired length.

`AS CHAR(N)` measures the length in characters, and right pads with spaces to the desired length.

`N` has a minimum value of 1, and if it is less than the length of the input string, the string is truncated without warning.

The optional `LEVEL` clause specifies that the return value should contain weights for specific collation levels. The `levels` specifier can either be a single integer, a comma-separated list of integers, or a range of integers separated by a dash (whitespace is ignored). Integers can range from 1 to a maximum of 6, dependent on the collation, and need to be listed in ascending order.

If the `LEVEL` clause is not provided, a default of 1 to the maximum for the collation is assumed.

If the `LEVEL` is specified without using a range, an optional modifier is permitted.

`ASC`, the default, returns the weights without any modification.

`DESC` returns bitwise-inverted weights.

`REVERSE` returns the weights in reverse order.

Examples

The examples below use the `HEX()` function to represent non-printable results in hexadecimal format.

```
SELECT HEX(WEIGHT_STRING('x'));
+-----+
| HEX(WEIGHT_STRING('x')) |
+-----+
| 0058 |
+-----+

SELECT HEX(WEIGHT_STRING('x' AS BINARY(4)));
+-----+
| HEX(WEIGHT_STRING('x' AS BINARY(4))) |
+-----+
| 78000000 |
+-----+

SELECT HEX(WEIGHT_STRING('x' AS CHAR(4)));
+-----+
| HEX(WEIGHT_STRING('x' AS CHAR(4))) |
+-----+
| 0058002000200020 |
+-----+

SELECT HEX(WEIGHT_STRING(@xaa22ee LEVEL 1));
+-----+
| HEX(WEIGHT_STRING(@xaa22ee LEVEL 1)) |
+-----+
| AA22EE |
+-----+

SELECT HEX(WEIGHT_STRING(@xaa22ee LEVEL 1 DESC));
+-----+
| HEX(WEIGHT_STRING(@xaa22ee LEVEL 1 DESC)) |
+-----+
| 55DD11 |
+-----+

SELECT HEX(WEIGHT_STRING(@xaa22ee LEVEL 1 REVERSE));
+-----+
| HEX(WEIGHT_STRING(@xaa22ee LEVEL 1 REVERSE)) |
+-----+
| EE22AA |
+-----+
```

CREATE DATABASE

Syntax

```
CREATE [OR REPLACE] {DATABASE | SCHEMA} [IF NOT EXISTS] db_name
[create_specification] ...

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

Description

`CREATE DATABASE` creates a database with the given name. To use this statement, you need the [CREATE privilege](#) for the database. `CREATE SCHEMA`

is a synonym for `CREATE DATABASE`.

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

OR REPLACE

MariaDB starting with 10.1.3

The `OR REPLACE` clause was added in [MariaDB 10.1.3](#)

Contents

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

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

```
DROP DATABASE IF EXISTS db_name;
CREATE DATABASE db_name ...;
```

IF NOT EXISTS

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

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, a 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

```
CREATE DATABASE db1;
Query OK, 1 row affected (0.18 sec)

CREATE DATABASE db1;
ERROR 1007 (HY000): Can't create database 'db1'; database exists

CREATE OR REPLACE DATABASE db1;
Query OK, 2 rows affected (0.00 sec)

CREATE DATABASE IF NOT EXISTS db1;
Query OK, 1 row affected, 1 warning (0.01 sec)

SHOW WARNINGS;
+-----+-----+
| Level | Code | Message           |
+-----+-----+
| Note  | 1007 | Can't create database 'db1'; database exists |
+-----+-----+
```

Setting the [character sets and collation](#). See [Setting Character Sets and Collations](#) for more details.

```
CREATE DATABASE czech_slovak_names
CHARACTER SET = 'keybcs2'
COLLATE = 'keybcs2_bin';
```

Comments, from [MariaDB 10.5.0](#):

```
CREATE DATABASE presentations COMMENT 'Presentations for conferences';
```

See Also

- [Identifier Names](#)
- [DROP DATABASE](#)

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

CONSTRAINT

MariaDB supports the implementation of constraints at the table-level using either [CREATE TABLE](#) or [ALTER TABLE](#) statements. A table constraint restricts the data you can add to the table. If you attempt to insert invalid data on a column, MariaDB throws an error.

Syntax

```
[CONSTRAINT [symbol]] constraint_expression

constraint_expression:
| PRIMARY KEY [index_type] (index_col_name, ...) [index_option] ...
| FOREIGN KEY [index_name] (index_col_name, ...)
    REFERENCES tbl_name (index_col_name, ...)
    [ON DELETE reference_option]
    [ON UPDATE reference_option]
| UNIQUE [INDEX|KEY] [index_name]
    [index_type] (index_col_name, ...) [index_option] ...
| CHECK (check_constraints)

index_type:
    USING {BTREE | HASH | RTREE}

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

index_option:
    | KEY_BLOCK_SIZE [=] value
    | index_type
    | WITH PARSER parser_name
    | COMMENT 'string'
    | CLUSTERING={YES|NO}

reference_option:
    RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT
```

Description

Constraints provide restrictions on the data you can add to a table. This allows you to enforce data integrity from MariaDB, rather than through application logic. When a statement violates a constraint, MariaDB throws an error.

There are four types of table constraints:

Constraint	Description
PRIMARY KEY	Sets the column for referencing rows. Values must be unique and not null.
FOREIGN KEY	Sets the column to reference the primary key on another table.
UNIQUE	Requires values in column or columns only occur once in the table.
CHECK	Checks whether the data meets the given condition.

Contents

1. [Syntax](#)
2. [Description](#)
 1. [FOREIGN KEY Constraints](#)
 2. [CHECK Constraints](#)
 3. [Replication](#)
 4. [Auto_increment](#)
3. [Examples](#)
4. [See Also](#)

The [Information Schema TABLE_CONSTRAINTS Table](#) contains information about tables that have constraints.

FOREIGN KEY Constraints

InnoDB supports [foreign key](#) constraints. The syntax for a foreign key constraint definition in InnoDB looks like this:

```
[CONSTRAINT [symbol]] FOREIGN KEY  
[index_name] (index_col_name, ...)  
REFERENCES tbl_name (index_col_name,...)  
[ON DELETE reference_option]  
[ON UPDATE reference_option]  
  
reference_option:  
RESTRICT | CASCADE | SET NULL | NO ACTION
```

The [Information Schema REFERENTIAL_CONSTRAINTS](#) table has more information about foreign keys.

CHECK Constraints

MariaDB starting with [10.2.1](#)

From [MariaDB 10.2.1](#), constraints are enforced. Before [MariaDB 10.2.1](#) constraint expressions were accepted in the syntax but ignored.

In [MariaDB 10.2.1](#) you can define constraints in 2 different ways:

- `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 constraint expression returns false, then the row will not be inserted or updated. One can use most deterministic functions in a constraint, including [UDFs](#).

```
CREATE TABLE t1 (a INT CHECK (a>2), b INT CHECK (b>2), CONSTRAINT a_greater CHECK (a>b));
```

If you use the second format and you don't give a name to the constraint, then the constraint will get an automatically 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 `check_constraint_checks` variable 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.

Replication

In [row-based replication](#), only the master checks constraints, and failed statements will not be replicated. In [statement-based](#) replication, the slaves will also check constraints. Constraints should therefore be identical, as well as deterministic, in a replication environment.

Auto_increment

MariaDB starting with [10.2.6](#)

- From [MariaDB 10.2.6](#), `auto_increment` columns are no longer permitted in check constraints. Previously they were permitted, but would not work correctly. See [MDEV-11117](#).

Examples

```

CREATE TABLE product (category INT NOT NULL, id INT NOT NULL,
    price DECIMAL,
    PRIMARY KEY(category, id)) ENGINE=INNODB;
CREATE TABLE customer (id INT NOT NULL,
    PRIMARY KEY (id)) ENGINE=INNODB;
CREATE TABLE product_order (no INT NOT NULL AUTO_INCREMENT,
    product_category INT NOT NULL,
    product_id INT NOT NULL,
    customer_id INT NOT NULL,
    PRIMARY KEY(no),
    INDEX (product_category, product_id),
    FOREIGN KEY (product_category, product_id)
        REFERENCES product(category, id)
        ON UPDATE CASCADE ON DELETE RESTRICT,
    INDEX (customer_id),
    FOREIGN KEY (customer_id)
        REFERENCES customer(id)) ENGINE=INNODB;

```

MariaDB starting with 10.2.1

The following examples will work from MariaDB 10.2.1 onwards.

Numeric constraints and comparisons:

```

CREATE TABLE t1 (a INT CHECK (a>2), b INT CHECK (b>2), CONSTRAINT a_greater CHECK (a>b));

INSERT INTO t1(a) VALUES (1);
ERROR 4222 (23000): CONSTRAINT `a` failed for `test`.`t1`

INSERT INTO t1(a,b) VALUES (3,4);
ERROR 4222 (23000): CONSTRAINT `a_greater` failed for `test`.`t1`

INSERT INTO t1(a,b) VALUES (4,3);
Query OK, 1 row affected (0.04 sec)

```

Dropping a constraint:

```
ALTER TABLE t1 DROP CONSTRAINT a_greater;
```

Adding a constraint:

```
ALTER TABLE t1 ADD CONSTRAINT a_greater CHECK (a>b);
```

Date comparisons and character length:

```

CREATE TABLE t2 (name VARCHAR(30) CHECK (CHAR_LENGTH(name)>2), start_date DATE,
    end_date DATE CHECK (start_date IS NULL OR end_date IS NULL OR start_date<end_date));

INSERT INTO t2(name, start_date, end_date) VALUES('Ione', '2003-12-15', '2014-11-09');
Query OK, 1 row affected (0.04 sec)

INSERT INTO t2(name, start_date, end_date) VALUES('Io', '2003-12-15', '2014-11-09');
ERROR 4222 (23000): CONSTRAINT `name` failed for `test`.`t2`

INSERT INTO t2(name, start_date, end_date) VALUES('Ione', NULL, '2014-11-09');
Query OK, 1 row affected (0.04 sec)

INSERT INTO t2(name, start_date, end_date) VALUES('Ione', '2015-12-15', '2014-11-09');
ERROR 4222 (23000): CONSTRAINT `end_date` failed for `test`.`t2`

```

A misplaced parenthesis:

```
CREATE TABLE t3 (name VARCHAR(30) CHECK (CHAR_LENGTH(name)>2)), start_date DATE,  
end_date DATE CHECK (start_date IS NULL OR end_date IS NULL OR start_date<end_date));  
Query OK, 0 rows affected (0.32 sec)
```

```
INSERT INTO t3(name, start_date, end_date) VALUES('Io', '2003-12-15', '2014-11-09');  
Query OK, 1 row affected, 1 warning (0.04 sec)
```

```
SHOW WARNINGS;  
+-----+-----+  
| Level | Code | Message |  
+-----+-----+  
| Warning | 1292 | Truncated incorrect DOUBLE value: 'Io' |  
+-----+-----+
```

Compare the definition of table *t2* to table *t3*. `CHAR_LENGTH(name)>2` is very different to `CHAR_LENGTH(name)>2` as the latter mistakenly performs a numeric comparison on the *name* field, leading to unexpected results.

See Also

- [Foreign Keys](#)

CREATE EVENT

Syntax

```
CREATE [OR REPLACE]  
[DEFINER = { user | CURRENT_USER | role | CURRENT_ROLE }]  
EVENT  
[IF NOT EXISTS]  
event_name  
ON SCHEDULE schedule  
[ON COMPLETION [NOT] PRESERVE]  
[ENABLE | DISABLE | DISABLE ON SLAVE]  
[COMMENT 'comment']  
DO sql_statement;  
  
schedule:  
    AT timestamp [+ INTERVAL interval] ...  
    | EVERY interval  
    [STARTS timestamp [+ INTERVAL interval] ...]  
    [ENDS timestamp [+ INTERVAL interval] ...]  
  
interval:  
    quantity {YEAR | QUARTER | MONTH | DAY | HOUR | MINUTE |  
              WEEK | SECOND | YEAR_MONTH | DAY_HOUR | DAY_MINUTE |  
              DAY_SECOND | HOUR_MINUTE | HOUR_SECOND | MINUTE_SECOND}
```

Description

This statement creates and schedules a new [event](#). It requires the [EVENT](#) privilege for the schema in which the event is to be created.

The minimum requirements for a valid CREATE EVENT statement are as follows:

- The keywords `CREATE EVENT` plus an event name, which uniquely identifies the event in the current schema. (Prior to MySQL 5.1.12, the event name needed to be unique only among events created by the same user on a given database.)
- An `ON SCHEDULE` clause, which determines when and how often the event executes.
- A `DO` clause, which contains the SQL statement to be executed by an event.

Here is an example of a minimal `CREATE EVENT` statement:

Contents

1. [Syntax](#)
2. [Description](#)
 1. [OR REPLACE](#)
 2. [IF NOT EXISTS](#)
 3. [ON SCHEDULE](#)
 4. [AT](#)
 5. [ON COMPLETION \[NOT\] PRESERVE](#)
 6. [ENABLE/DISABLE/DISABLE ON SLAVE](#)
 7. [COMMENT](#)
3. [Examples](#)
4. [See Also](#)

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

The previous statement creates an event named `myevent`. This event executes once — one hour following its creation — by running an SQL statement that increments the value of the `myschema.mytable` table's `mycol` column by 1.

The `event_name` must be a valid MariaDB identifier with a maximum length of 64 characters. It may be delimited using back ticks, and may be qualified with the name of a database schema. An event is associated with both a MariaDB user (the definer) and a schema, and its name must be unique among names of events within that schema. In general, the rules governing event names are the same as those for names of stored routines. See [Identifier Names](#).

If no schema is indicated as part of `event_name`, the default (current) schema is assumed.

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

OR REPLACE

The `OR REPLACE` clause was included in [MariaDB 10.1.4](#). If used and the event already exists, instead of an error being returned, the existing event will be dropped and replaced by the newly defined event.

IF NOT EXISTS

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

ON SCHEDULE

The `ON SCHEDULE` clause can be used to specify when the event must be triggered.

AT

If you want to execute the event only once (one time event), you can use the `AT` keyword, followed by a timestamp. If you use `CURRENT_TIMESTAMP`, the event acts as soon as it is created. As a convenience, you can add one or more intervals to that timestamp. You can also specify a timestamp in the past, so that the event is stored but not triggered, until you modify it via [ALTER EVENT](#).

The following example shows how to create an event that will be triggered tomorrow at a certain time:

```
CREATE EVENT example
  ON SCHEDULE AT CURRENT_TIMESTAMP + INTERVAL 1 DAY + INTERVAL 3 HOUR
  DO something;
```

You can also specify that an event must be triggered at a regular interval (recurring event). In such cases, use the `EVERY` clause followed by the interval.

If an event is recurring, you can specify when the first execution must happen via the `STARTS` clause and a maximum time for the last execution via the `ENDS` clause. `STARTS` and `ENDS` clauses are followed by a timestamp and, optionally, one or more intervals. The `ENDS` clause can specify a timestamp in the past, so that the event is stored but not executed until you modify it via [ALTER EVENT](#).

In the following example, next month a recurring event will be triggered hourly for a week:

```
CREATE EVENT example
  ON SCHEDULE EVERY 1 HOUR
  STARTS CURRENT_TIMESTAMP + INTERVAL 1 MONTH
  ENDS CURRENT_TIMESTAMP + INTERVAL 1 MONTH + INTERVAL 1 WEEK
  DO some_task;
```

Intervals consist of a quantity and a time unit. The time units are the same used for other statements and time functions, except that you can't use microseconds for events. For simple time units, like `HOUR` or `MINUTE`, the quantity is an integer number, for example '10 MINUTE'. For composite time units, like `HOUR_MINUTE` or `HOUR_SECOND`, the quantity must be a string with all involved simple values and their separators, for example '2:30' or '2:30:30'.

ON COMPLETION [NOT] PRESERVE

The `ON COMPLETION` clause can be used to specify if the event must be deleted after its last execution (that is, after its `AT` or `ENDS` timestamp is past). By default, events are dropped when they are expired. To explicitly state that this is the desired behaviour, you can use `ON COMPLETION NOT PRESERVE`. Instead, if you want the event to be preserved, you can use `ON COMPLETION PRESERVE`.

If you specify `ON COMPLETION NOT PRESERVE`, and you specify a timestamp in the past for `AT` or `ENDS` clause, the event will be immediately dropped. In such cases, you will get a Note 1558: "Event execution time is in the past and ON COMPLETION NOT PRESERVE is set. The event was dropped immediately after creation".

ENABLE/DISABLE/DISABLE ON SLAVE

Events are `ENABLE`d by default. If you want to stop MariaDB from executing an event, you may specify `DISABLE`. When it is ready to be activated, you may enable it using `ALTER EVENT`. Another option is `DISABLE ON SLAVE`, which indicates that an event was created on a master and has been replicated to the slave, which is prevented from executing the event. If `DISABLE ON SLAVE` is specifically set, the event will not be executed.

COMMENT

The `COMMENT` clause may be used to set a comment for the event. Maximum length for comments is 64 characters. The comment is a string, so it must be quoted. To see events comments, you can query the `INFORMATION_SCHEMA.EVENTS` table (the column is named `EVENT_COMMENT`).

Examples

Minimal `CREATE EVENT` statement:

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

An event that will be triggered tomorrow at a certain time:

```
CREATE EVENT example
  ON SCHEDULE AT CURRENT_TIMESTAMP + INTERVAL 1 DAY + INTERVAL 3 HOUR
  DO something;
```

Next month a recurring event will be triggered hourly for a week:

```
CREATE EVENT example
  ON SCHEDULE EVERY 1 HOUR
  STARTS CURRENT_TIMESTAMP + INTERVAL 1 MONTH
  ENDS CURRENT_TIMESTAMP + INTERVAL 1 MONTH + INTERVAL 1 WEEK
  DO some_task;
```

OR REPLACE and IF NOT EXISTS:

```

CREATE EVENT myevent
  ON SCHEDULE AT CURRENT_TIMESTAMP + INTERVAL 1 HOUR
  DO
    UPDATE myschema.mytable SET mycol = mycol + 1;
ERROR 1537 (HY000): Event 'myevent' already exists

CREATE OR REPLACE EVENT myevent
  ON SCHEDULE AT CURRENT_TIMESTAMP + INTERVAL 1 HOUR
  DO
    UPDATE myschema.mytable SET mycol = mycol + 1;;
Query OK, 0 rows affected (0.00 sec)

CREATE EVENT IF NOT EXISTS myevent
  ON SCHEDULE AT CURRENT_TIMESTAMP + INTERVAL 1 HOUR
  DO
    UPDATE myschema.mytable SET mycol = mycol + 1;
Query OK, 0 rows affected, 1 warning (0.00 sec)

SHOW WARNINGS;
+-----+-----+
| Level | Code | Message           |
+-----+-----+
| Note  | 1537 | Event 'myevent' already exists |
+-----+-----+

```

See Also

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

CREATE INDEX

Syntax

```

CREATE [OR REPLACE] [UNIQUE|FULLTEXT|SPATIAL] INDEX
[IF NOT EXISTS] index_name
  [index_type]
  ON tbl_name (index_col_name,...)
  [WAIT n | NOWAIT]
  [index_option]
  [algorithm_option | lock_option] ...

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 ]

algorithm_option:
  ALGORITHM [=] {DEFAULT|INPLACE|COPY|NOCOPY|INSTANT}

lock_option:
  LOCK [=] {DEFAULT|NONE|SHARED|EXCLUSIVE}

```

Description

CREATE INDEX is mapped to an ALTER TABLE statement to create indexes. See [ALTER TABLE](#). CREATE INDEX cannot be used to create a PRIMARY KEY; use ALTER TABLE instead.

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.

Another shortcut, [DROP INDEX](#), allows the removal of an index.

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

Note that KEY_BLOCK_SIZE is currently ignored in CREATE INDEX, although it is included in the output of [SHOW CREATE TABLE](#).

Contents

1. [Syntax](#)
2. [Description](#)
3. [Privileges](#)
4. [Online DDL](#)
5. [CREATE OR REPLACE INDEX](#)
6. [CREATE INDEX IF NOT EXISTS](#)
7. [Index Definitions](#)
8. [WAIT/NOWAIT](#)
9. [ALGORITHM](#)
10. [LOCK](#)
11. [Progress Reporting](#)
12. [WITHOUT OVERLAPS](#)
13. [Examples](#)
14. [See Also](#)

Privileges

Executing the `CREATE INDEX` statement requires the [INDEX](#) privilege for the table or the database.

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.

CREATE OR REPLACE INDEX

MariaDB starting with [10.1.4](#)

The `OR REPLACE` clause was added in [MariaDB 10.1.4](#).

If the `OR REPLACE` clause is used and if the index already exists, then instead of returning an error, the server will drop the existing index and replace it with the newly defined index.

CREATE INDEX IF NOT EXISTS

If the `IF NOT EXISTS` clause is used, then the index will only be created if an index with the same name does not already exist. If the index already exists, then a warning will be triggered by default.

Index Definitions

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

WAIT/NOWAIT

MariaDB starting with [10.3.0](#)

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

ALGORITHM

See [ALTER TABLE: ALGORITHM](#) for more information.

LOCK

See [ALTER TABLE: LOCK](#) for more information.

Progress Reporting

MariaDB provides progress reporting for `CREATE INDEX` 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::

```
CREATE INDEX ON tab (num);  
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.

WITHOUT OVERLAPS

MariaDB starting with 10.5.3

The `WITHOUT OVERLAPS` clause allows one to constrain a primary or unique index such that application-time periods cannot overlap.

Examples

Creating a unique index:

```
CREATE UNIQUE INDEX HomePhone ON Employees(Home_Phone);
```

OR REPLACE and IF NOT EXISTS:

```
CREATE INDEX xi ON xx5 (x);  
Query OK, 0 rows affected (0.03 sec)  
  
CREATE INDEX xi ON xx5 (x);  
ERROR 1061 (42000): Duplicate key name 'xi'  
  
CREATE OR REPLACE INDEX xi ON xx5 (x);  
Query OK, 0 rows affected (0.03 sec)  
  
CREATE INDEX IF NOT EXISTS xi ON xx5 (x);  
Query OK, 0 rows affected, 1 warning (0.00 sec)  
  
SHOW WARNINGS;
```

Level	Code	Message
Note	1061	Duplicate key name 'xi'

From MariaDB 10.5.3, creating a unique index for an application-time period table with a `WITHOUT OVERLAPS` constraint:

```
CREATE UNIQUE INDEX u ON rooms (room_number, p WITHOUT OVERLAPS);
```

See Also

- [Identifier Names](#)
- [Getting Started with Indexes](#)
- [What is an Index?](#)
- [ALTER TABLE](#)
- [DROP INDEX](#)
- [SHOW INDEX](#)
- [SPATIAL INDEX](#)
- [Full-text Indexes](#)
- [WITHOUT OVERLAPS](#)

- Ignored Indexes

CREATE PACKAGE

MariaDB starting with 10.3.5

Oracle-style packages were introduced in MariaDB 10.3.5.

Syntax

```

CREATE
  [ OR REPLACE]
  [DEFINER = { user | CURRENT_USER | role | CURRENT_ROLE }]
  PACKAGE [ IF NOT EXISTS ]
  [ db_name . ] package_name
  [ package_characteristic ... ]
{ AS | IS }
  [ package_specification_element ... ]
END [ package_name ]

package_characteristic:
  COMMENT 'string'
  | SQL SECURITY { DEFINER | INVOKER }

package_specification_element:
  FUNCTION_SYM package_specification_function ;
  | PROCEDURE_SYM package_specification_procedure ;

package_specification_function:
  func_name [ ( func_param [, func_param]... ) ]
  RETURNS func_return_type
  [ package_routine_characteristic... ]

package_specification_procedure:
  proc_name [ ( proc_param [, proc_param]... ) ]
  [ package_routine_characteristic... ]

func_return_type:
  type

func_param:
  param_name [ IN | OUT | INOUT | IN OUT ] type

proc_param:
  param_name [ IN | OUT | INOUT | IN OUT ] type

type:
  Any valid MariaDB explicit or anchored data type

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

```

Description

The `CREATE PACKAGE` statement can be used when `Oracle SQL_MODE` is set.

The `CREATE PACKAGE` creates the specification for a stored package (a collection of logically related stored objects). A stored package specification declares public routines (procedures and functions) of the package, but does not implement these routines.

A package whose specification was created by the `CREATE PACKAGE` statement, should later be implemented using the `CREATE PACKAGE BODY` statement.

Contents

1. [Syntax](#)
2. [Description](#)
3. [Function parameter quantifiers IN | OUT | INOUT | IN OUT](#)
4. [Examples](#)
5. [See Also](#)

MariaDB starting with [10.8.0](#)

The function parameter quantifiers for `IN`, `OUT`, `INOUT`, and `IN OUT` were added in a 10.8.0 preview release. Prior to 10.8.0 quantifiers were supported only in procedures.

`OUT`, `INOUT` and its equivalent `IN OUT`, are only valid if called from `SET` and not `SELECT`. These quantifiers are especially useful for creating functions and procedures with more than one return value. This allows functions and procedures to be more complex and nested.

Examples

```
SET sql_mode=ORACLE;
DELIMITER $$

CREATE OR REPLACE 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;
$$
DELIMITER ;
```

See Also

- [CREATE PACKAGE BODY](#)
- [SHOW CREATE PACKAGE](#)
- [DROP PACKAGE](#)
- [Oracle SQL_MODE](#)

CREATE PACKAGE BODY

MariaDB starting with [10.3.5](#)

Oracle-style packages were introduced in [MariaDB 10.3.5](#).

Syntax

```
CREATE [ OR REPLACE ]
[DEFINER = { user | CURRENT_USER | role | CURRENT_ROLE }]
PACKAGE BODY
[ IF NOT EXISTS ]
[ db_name . ] package_name
[ package_characteristic... ]
{ AS | IS }
package_implementation_declare_section
package_implementation_executable_section
END [ package_name]
```

```
package_implementation_declare_section:
    package_implementation_item_declaration
    [ package_implementation_item_declaration... ]
    [ package_implementation_routine_definition... ]
| package_implementation_routine_definition
    [ package_implementation_routine_definition... ]
```

```

package_implementation_item_declarator:
    variable_declarator ;

variable_declarator:
    variable_name[,...] type [:= expr ] ;

package_implementation_routine_definition:
    FUNCTION package_specification_function
        [ package_implementation_function_body ] ;
    | PROCEDURE package_specification_procedure
        [ package_implementation_procedure_body ] ;

package_implementation_function_body:
    { AS | IS } package_routine_body [func_name]

package_implementation_procedure_body:
    { AS | IS } package_routine_body [proc_name]

package_routine_body:
    [ package_routine_declarations ]
    BEGIN
        statements [ EXCEPTION exception_handlers ]
    END

package_routine_declarations:
    package_routine_declaration ';' [package_routine_declaration ';'...] ;

package_routine_declaration:
    variable_declarator
    | condition_name CONDITION FOR condition_value
    | user_exception_name EXCEPTION
    | CURSOR_SYM cursor_name
        [ ( cursor_formal_parameters ) ]
    IS select_statement
    ;

package_implementation_executable_section:
    END
    | BEGIN
        statement ; [statement ; ]...
        [EXCEPTION exception_handlers]
    END

exception_handlers:
    exception_handler [exception_handler...]

exception_handler:
    WHEN_SYM condition_value [, condition_value]...
        THEN_SYM statement ; [statement ;]...

condition_value:
    condition_name
    | user_exception_name
    | SQLWARNING
    | SQLEXCEPTION
    | NOT FOUND
    | OTHERS_SYM
    | SQLSTATE [VALUE] sqlstate_value
    | mariadb_error_code

```

Description

The `CREATE PACKAGE BODY` statement can be used when [Oracle SQL_MODE](#) is set.

The `CREATE PACKAGE BODY` statement creates the package body for a stored package. The package specification must be previously created using the `CREATE PACKAGE` statement.

A package body provides implementations of the package public routines and can optionally have:

- package-wide private variables
- package private routines
- forward declarations for private routines
- an executable initialization section

Contents

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

Examples

```
SET sql_mode=ORACLE;
DELIMITER $$

CREATE OR REPLACE 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;
$$

CREATE PACKAGE BODY employee_tools AS
    -- package body variables
    stdRaiseAmount DECIMAL(10,2):=500;

    -- private routines
    PROCEDURE log (eid INT, ecmnt TEXT) AS
    BEGIN
        INSERT INTO employee_log (id, cmnt) VALUES (eid, ecmnt);
    END;

    -- public routines
    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
        -- This code is executed when the current session
        -- accesses any of the package routines for the first time
        log(0, 'Session ' || connection_id() || ' ' || current_user || ' started');
    END;
$$

DELIMITER ;
```

See Also

- [CREATE PACKAGE](#)
- [SHOW CREATE PACKAGE BODY](#)
- [DROP PACKAGE BODY](#)
- [Oracle SQL_MODE](#)

DROP DATABASE

Syntax

```
DROP {DATABASE | SCHEMA} [IF EXISTS] db_name
```

Description

`DROP DATABASE` drops all tables in the database and deletes the database. Be very careful with this statement! To use `DROP DATABASE`, you need the [DROP privilege](#) on the database. `DROP SCHEMA` is a synonym for `DROP DATABASE`.

Important: When a database is dropped, user privileges on the database are not automatically dropped. See [GRANT](#).

Contents

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

IF EXISTS

Use `IF EXISTS` to prevent an error from occurring for databases that do not exist. A `NOTE` is generated for each non-existent database when using `IF EXISTS`. See [SHOW WARNINGS](#).

Atomic DDL

MariaDB starting with 10.6.1

MariaDB 10.6.1 supports [Atomic DDL](#).

`DROP DATABASE` is implemented as

```
loop over all tables
  DROP TABLE table
```

Each individual [DROP TABLE](#) is atomic while `DROP DATABASE` as a whole is crash-safe.

Examples

```
DROP DATABASE bufg;
Query OK, 0 rows affected (0.39 sec)

DROP DATABASE bufg;
ERROR 1008 (HY000): Can't drop database 'bufg'; database doesn't exist

\W
Show warnings enabled.

DROP DATABASE IF EXISTS bufg;
Query OK, 0 rows affected, 1 warning (0.00 sec)
Note (Code 1008): Can't drop database 'bufg'; database doesn't exist
```

See Also

- [CREATE DATABASE](#)
- [ALTER DATABASE](#)

- [SHOW DATABASES](#)
- [Information Schema SCHEMATA Table](#)
- [SHOW CREATE DATABASE](#)

CREATE SEQUENCE

MariaDB starting with [10.3](#)

`CREATE SEQUENCE` was introduced in [MariaDB 10.3](#).

Syntax

```
CREATE [OR REPLACE] [TEMPORARY] SEQUENCE [IF NOT EXISTS] sequence_name
[ INCREMENT [ BY | = ] increment ]
[ MINVALUE [=] minvalue | NO MINVALUE | NOMINVALUE ]
[ MAXVALUE [=] maxvalue | NO MAXVALUE | NOMAXVALUE ]
[ START [ WITH | = ] start ]
[ CACHE [=] cache | NOCACHE ] [ CYCLE | NOCYCLE]
[table_options]
```

The options for `CREATE SEQUENCE` can be given in any order, optionally followed by `table_options`.

`table_options` can be any of the normal table options in [CREATE TABLE](#) but the most usable ones are `ENGINE=...` and `COMMENT=`.

`NOMAXVALUE` and `NOMINVALUE` are there to allow one to create SEQUENCES using the Oracle syntax.

Description

`CREATE SEQUENCE` will create a sequence that generates new values when called with `NEXT VALUE FOR sequence_name`. It's an alternative to [AUTO INCREMENT](#) when one wants to have more control of how the numbers are generated. As the SEQUENCE caches values (up to `CACHE`) it can in some cases be much faster than [AUTO INCREMENT](#). Another benefit is that one can access the last value generated by all used sequences, which solves one of the limitations with [LAST_INSERT_ID\(\)](#).

`CREATE SEQUENCE` requires the [CREATE privilege](#).

[DROP SEQUENCE](#) can be used to drop a sequence, and [ALTER SEQUENCE](#) to change it.

Arguments to Create

The following options may be used:

Option	Default value	Description
INCREMENT	1	Increment to use for values. May be negative. Setting an increment of 0 causes the sequence to use the value of the auto_increment_increment system variable at the time of creation, which is always a positive number. (see MDEV-16035).
MINVALUE	1 if INCREMENT > 0 and -9223372036854775807 if INCREMENT < 0	Minimum value for the sequence
MAXVALUE	9223372036854775806 if INCREMENT > 0 and -1 if INCREMENT < 0	Max value for sequence
START	MINVALUE if INCREMENT > 0 and MAX_VALUE if INCREMENT < 0	First value that the sequence will generate
CACHE	1000	Number of values that should be cached. 0 if no CACHE. The underlying table will be updated first time a new sequence number is generated and each time the cache runs out.

If `CYCLE` is used then the sequence should start again from `MINVALUE` after it has run out of values. Default value is `NOCYCLE`.

Contents

1. [Syntax](#)
2. [Description](#)
 1. [Arguments to Create](#)
 2. [Constraints on Create Arguments](#)
 3. [Atomic DDL](#)
3. [Examples](#)
4. [See Also](#)

Constraints on Create Arguments

To be able to create a legal sequence, the following must hold:

- MAXVALUE >= start
- MAXVALUE > MINVALUE
- START >= MINVALUE
- MAXVALUE <= 9223372036854775806 (LONGLONG_MAX-1)
- MINVALUE >= -9223372036854775807 (LONGLONG_MIN+1)

Note that sequences can't generate the maximum/minimum 64 bit number because of the constraint of MINVALUE and MAXVALUE .

Atomic DDL

MariaDB starting with 10.6.1

MariaDB 10.6.1 supports [Atomic DDL](#) and CREATE SEQUENCE is atomic.

Examples

```
CREATE SEQUENCE s START WITH 100 INCREMENT BY 10;  
CREATE SEQUENCE s2 START WITH -100 INCREMENT BY -10;
```

The following statement fails, as the increment conflicts with the defaults

```
CREATE SEQUENCE s3 START WITH -100 INCREMENT BY 10;  
ERROR 4082 (HY000): Sequence 'test.s3' values are conflicting
```

The sequence can be created by specifying workable minimum and maximum values:

```
CREATE SEQUENCE s3 START WITH -100 INCREMENT BY 10 MINVALUE=-100 MAXVALUE=1000;
```

See Also

- [Sequence Overview](#)
- [ALTER SEQUENCE](#)
- [DROP SEQUENCE](#)
- [NEXT VALUE FOR](#)
- [PREVIOUS VALUE FOR](#)
- [SETVAL\(\)](#). Set next value for the sequence.
- [AUTO INCREMENT](#)
- [SHOW CREATE SEQUENCE](#)

DROP EVENT

Syntax

```
DROP EVENT [IF EXISTS] event_name
```

Description

This statement drops the `event` named `event_name` . The event immediately ceases being active, and is deleted completely from the server.

If the event does not exist, the error ERROR 1517 (HY000): Unknown event 'event_name' results. You can override this and cause the statement to generate a NOTE for non-existent events instead by using IF EXISTS . See [SHOW WARNINGS](#) .

This statement requires the `EVENT` privilege. In MySQL 5.1.11 and earlier, an event could be dropped only by its definer, or by a user having the `SUPER` privilege.

Examples

```
DROP EVENT myevent3;
```

Using the IF EXISTS clause:

```
DROP EVENT IF EXISTS myevent3;
Query OK, 0 rows affected, 1 warning (0.01 sec)

SHOW WARNINGS;
+-----+-----+
| Level | Code | Message           |
+-----+-----+
| Note  | 1305 | Event myevent3 does not exist |
+-----+-----+
```

See also

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

DROP FUNCTION

Syntax

```
DROP FUNCTION [IF EXISTS] f_name
```

Description

The `DROP FUNCTION` statement is used to drop a [stored function](#) or a user-defined function (UDF). That is, the specified routine is removed from the server, along with all privileges specific to the function. You must have the `ALTER ROUTINE` [privilege](#) for the routine in order to drop it. If the `automatic_sp_privileges` server system variable is set, both the `ALTER ROUTINE` and `EXECUTE` privileges are granted automatically to the routine creator - see [Stored Routine Privileges](#).

Contents

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

IF EXISTS

The `IF EXISTS` clause is a MySQL/MariaDB extension. It prevents an error from occurring if the function does not exist. A `NOTE` is produced that can be viewed with `SHOW WARNINGS`.

For dropping a [user-defined functions](#) (UDF), see [DROP FUNCTION UDF](#).

Examples

```

DROP FUNCTION hello;
Query OK, 0 rows affected (0.042 sec)

DROP FUNCTION hello;
ERROR 1305 (42000): FUNCTION test.hello does not exist

DROP FUNCTION IF EXISTS hello;
Query OK, 0 rows affected, 1 warning (0.000 sec)

SHOW WARNINGS;
+-----+-----+
| Level | Code | Message
+-----+-----+
| Note  | 1305 | FUNCTION test.hello does not exist |
+-----+-----+

```

See Also

- [DROP PROCEDURE](#)
- [Stored Function Overview](#)
- [CREATE FUNCTION](#)
- [CREATE FUNCTION UDF](#)
- [ALTER FUNCTION](#)
- [SHOW CREATE FUNCTION](#)
- [SHOW FUNCTION STATUS](#)
- [Stored Routine Privileges](#)
- [INFORMATION_SCHEMA ROUTINES Table](#)

CREATE SERVER

Syntax

```

CREATE [OR REPLACE] SERVER [IF NOT EXISTS] server_name
    FOREIGN DATA WRAPPER wrapper_name
    OPTIONS (option [, option] ...)

option:
{ HOST character-literal
| DATABASE character-literal
| USER character-literal
| PASSWORD character-literal
| SOCKET character-literal
| OWNER character-literal
| PORT numeric-literal }

```

Description

This statement creates the definition of a server for use with the [Spider](#), [Connect](#), [FEDERATED](#) or [FederatedX](#) storage engine. The CREATE SERVER statement creates a new row within the [servers](#) table within the mysql database. This statement requires the [SUPER](#) privilege or, from MariaDB 10.5.2, the [FEDERATED ADMIN](#) privilege.

The server_name should be a unique reference to the server. Server definitions are global within the scope of the server, it is not possible to qualify the server definition to a specific database.

server_name has a maximum length of 64 characters (names longer than 64 characters are silently truncated), and is case insensitive. You may specify the name as a quoted string.

The wrapper_name may be quoted with single quotes. Supported values are:

- mysql
- mariadb (in [MariaDB 10.3](#) and later)

For each option you must specify either a character literal or numeric literal. Character literals are UTF-8, support a maximum length of 64 characters and default to a blank (empty) string. String literals are silently truncated to 64 characters. Numeric literals must be a number between 0

Contents

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

and 9999, default value is 0.

Note: The `OWNER` option is currently not applied, and has no effect on the ownership or operation of the server connection that is created.

The `CREATE SERVER` statement creates an entry in the `mysql.servers` table that can later be used with the `CREATE TABLE` statement when creating a `Spider`, `Connect`, `FederatedX` or `FEDERATED` table. The options that you specify will be used to populate the columns in the `mysql.servers` table. The table columns are `Server_name`, `Host`, `Db`, `Username`, `Password`, `Port` and `Socket`.

`DROP SERVER` removes a previously created server definition.

`CREATE 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.

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

OR REPLACE

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

```
DROP SERVER IF EXISTS name;
CREATE SERVER server_name ...;
```

IF NOT EXISTS

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

Examples

```
CREATE SERVER s
FOREIGN DATA WRAPPER mysql
OPTIONS (USER 'Remote', HOST '192.168.1.106', DATABASE 'test');
```

`OR REPLACE` and `IF NOT EXISTS`:

```
CREATE SERVER s
FOREIGN DATA WRAPPER mysql
OPTIONS (USER 'Remote', HOST '192.168.1.106', DATABASE 'test');
ERROR 1476 (HY000): The foreign server, s, you are trying to create already exists

CREATE OR REPLACE SERVER s
FOREIGN DATA WRAPPER mysql
OPTIONS (USER 'Remote', HOST '192.168.1.106', DATABASE 'test');
Query OK, 0 rows affected (0.00 sec)

CREATE SERVER IF NOT EXISTS s
FOREIGN DATA WRAPPER mysql
OPTIONS (USER 'Remote', HOST '192.168.1.106', DATABASE 'test');
Query OK, 0 rows affected, 1 warning (0.00 sec)

SHOW WARNINGS;
+-----+-----+
| Level | Code | Message |
+-----+-----+
| Note  | 1476 | The foreign server, s, you are trying to create already exists |
+-----+-----+
```

See Also

- [Identifier Names](#)
- [ALTER SERVER](#)
- [DROP SERVER](#)
- [Spider Storage Engine](#)
- [Connect Storage Engine](#)

DROP INDEX

Syntax

```
DROP INDEX [IF EXISTS] index_name ON tbl_name  
[WAIT n |NOWAIT]
```

Description

`DROP INDEX` drops the `index` named `index_name` from the table `tbl_name`. This statement is mapped to an `ALTER TABLE` statement to drop the index.

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.

See [ALTER TABLE](#).

Another shortcut, `CREATE INDEX`, allows the creation of an index.

To remove the primary key, ``PRIMARY`` must be specified as `index_name`. Note that the quotes are necessary, because `PRIMARY` is a keyword.

Contents

1. [Syntax](#)
2. [Description](#)
3. [Privileges](#)
4. [Online DDL](#)
5. [DROP INDEX IF EXISTS ...](#)
6. [WAIT/NOWAIT](#)
7. [Progress Reporting](#)
8. [See Also](#)

Privileges

Executing the `DROP INDEX` statement requires the `INDEX` privilege for the table or the database.

Online DDL

Online DDL is used by default with InnoDB, when the drop index operation supports it.

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

DROP INDEX IF EXISTS ...

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

WAIT/NOWAIT

MariaDB starting with 10.3.0

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

Progress Reporting

MariaDB provides progress reporting for `DROP INDEX` 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::

See Also

- [Getting Started with Indexes](#)
- [CREATE INDEX](#)
- [ALTER TABLE](#)

CREATE TRIGGER

Syntax

```

CREATE [OR REPLACE]
[DEFINER = { user | CURRENT_USER | role | CURRENT_ROLE }]
TRIGGER [IF NOT EXISTS] trigger_name trigger_time trigger_event
ON tbl_name FOR EACH ROW
[ { FOLLOWES | PRECEDES } other_trigger_name ]
trigger_stmt;

```

Description

This statement creates a new [trigger](#). A trigger is a named database object that is associated with a table, and that activates when a particular event occurs for the table. The trigger becomes associated with the table named `tbl_name`, which must refer to a permanent table. You cannot associate a trigger with a `TEMPORARY` table or a view.

`CREATE TRIGGER` requires the [TRIGGER](#) privilege for the table associated with the trigger.

MariaDB starting with [10.2.3](#)

You can have multiple triggers for the same `trigger_time` and `trigger_event`.

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

Contents

1. [Syntax](#)
2. [Description](#)
 1. [OR REPLACE](#)
 2. [DEFINER](#)
 3. [IF NOT EXISTS](#)
 4. [trigger_time](#)
 5. [trigger_event](#)
 6. [FOLLOWES/PRECEDES other_trigger_name](#)
 7. [Atomic DDL](#)
3. [Examples](#)
4. [See Also](#)

OR REPLACE

MariaDB starting with [10.1.4](#)

If used and the trigger already exists, instead of an error being returned, the existing trigger will be dropped and replaced by the newly defined trigger.

DEFINER

The `DEFINER` clause determines the security context to be used when checking access privileges at trigger activation time. Usage requires the [SUPER](#) privilege, or, from [MariaDB 10.5.2](#), the [SET USER](#) privilege.

IF NOT EXISTS

MariaDB starting with [10.1.4](#)

If the `IF NOT EXISTS` clause is used, the trigger will only be created if a trigger of the same name does not exist. If the trigger already exists, by default a warning will be returned.

trigger_time

`trigger_time` is the trigger action time. It can be `BEFORE` or `AFTER` to indicate that the trigger activates before or after each row to be modified.

trigger_event

`trigger_event` indicates the kind of statement that activates the trigger. The `trigger_event` can be one of the following:

- `INSERT` : The trigger is activated whenever a new row is inserted into the table; for example, through [INSERT](#), [LOAD DATA](#), and [REPLACE](#) statements.
- `UPDATE` : The trigger is activated whenever a row is modified; for example, through [UPDATE](#) statements.
- `DELETE` : The trigger is activated whenever a row is deleted from the table; for example, through [DELETE](#) and [REPLACE](#) statements. However, [DROP TABLE](#) and [TRUNCATE](#) statements on the table do not activate this trigger, because they do not use `DELETE`. Dropping a partition does not activate `DELETE` triggers, either.

FOLLOWES/PRECEDES other_trigger_name

MariaDB starting with [10.2.3](#)

The `FOLLOWS` `other_trigger_name` and `PRECEDES` `other_trigger_name` options were added in MariaDB 10.2.3 as part of supporting multiple triggers per action time. This is the same syntax used by MySQL 5.7, although MySQL 5.7 does not have multi-trigger support.

`FOLLOWS` adds the new trigger after another trigger while `PRECEDES` adds the new trigger before another trigger. If neither option is used, the new trigger is added last for the given action and time.

`FOLLOWS` and `PRECEDES` are not stored in the trigger definition. However the trigger order is guaranteed to not change over time. `mariadb-dump/mysqldump` and other backup methods will not change trigger order. You can verify the trigger order from the `ACTION_ORDER` column in `INFORMATION_SCHEMA.TRIGGERS` table.

```
SELECT trigger_name, action_order FROM information_schema.triggers
  WHERE event_object_table='t1';
```

Atomic DDL

MariaDB starting with 10.6.1

MariaDB 10.6.1 supports [Atomic DDL](#) and `CREATE TRIGGER` is atomic.

Examples

```
CREATE DEFINER=`root`@`localhost` TRIGGER increment_animal
  AFTER INSERT ON animals FOR EACH ROW
    UPDATE animal_count SET animal_count.animals = animal_count.animals+1;
```

OR REPLACE and IF NOT EXISTS

```
CREATE DEFINER=`root`@`localhost` TRIGGER increment_animal
  AFTER INSERT ON animals FOR EACH ROW
    UPDATE animal_count SET animal_count.animals = animal_count.animals+1;
ERROR 1359 (HY000): Trigger already exists

CREATE OR REPLACE DEFINER=`root`@`localhost` TRIGGER increment_animal
  AFTER INSERT ON animals FOR EACH ROW
    UPDATE animal_count SET animal_count.animals = animal_count.animals+1;
Query OK, 0 rows affected (0.12 sec)

CREATE DEFINER=`root`@`localhost` TRIGGER IF NOT EXISTS increment_animal
  AFTER INSERT ON animals FOR EACH ROW
    UPDATE animal_count SET animal_count.animals = animal_count.animals+1;
Query OK, 0 rows affected, 1 warning (0.00 sec)

SHOW WARNINGS;
+-----+-----+
| Level | Code | Message          |
+-----+-----+
| Note  | 1359 | Trigger already exists |
+-----+-----+
1 row in set (0.00 sec)
```

See Also

- [Identifier Names](#)
- [Trigger Overview](#)
- [DROP TRIGGER](#)
- [Information Schema TRIGGERS Table](#)
- [SHOW TRIGGERS](#)
- [SHOW CREATE TRIGGER](#)
- [Trigger Limitations](#)

DROP PROCEDURE

Syntax

```
DROP PROCEDURE [IF EXISTS] sp_name
```

Description

This statement is used to drop a [stored procedure](#). That is, the specified routine is removed from the server along with all privileges specific to the [procedure](#). You must have the `ALTER ROUTINE` privilege for the routine. If the `automatic_sp_privileges` server system variable is set, that privilege and `EXECUTE` are granted automatically to the routine creator - see [Stored Routine Privileges](#).

The `IF EXISTS` clause is a MySQL/MariaDB extension. It prevents an error from occurring if the procedure or function does not exist. A `NOTE` is produced that can be viewed with `SHOW WARNINGS`.

While this statement takes effect immediately, threads which are executing a procedure can continue execution.

Examples

```
DROP PROCEDURE simpleproc;
```

IF EXISTS:

```
DROP PROCEDURE simpleproc;
ERROR 1305 (42000): PROCEDURE test.simpleproc does not exist

DROP PROCEDURE IF EXISTS simpleproc;
Query OK, 0 rows affected, 1 warning (0.00 sec)

SHOW WARNINGS;
+-----+-----+
| Level | Code | Message           |
+-----+-----+
| Note  | 1305 | PROCEDURE test.simpleproc does not exist |
+-----+-----+
```

See Also

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

DROP SEQUENCE

MariaDB starting with 10.3

`DROP SEQUENCE` was introduced in [MariaDB 10.3](#).

Syntax

```
DROP [TEMPORARY] SEQUENCE [IF EXISTS] /*COMMENT TO SAVE*/
sequence_name [, sequence_name] ...
```

Description

Contents

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

`DROP SEQUENCE` removes one or more sequences created with [CREATE SEQUENCE](#). You must have the `CREATE` privilege for each sequence. 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 connection is using the sequence, a metadata lock is active, and this statement will wait until the lock is released. This is also true for non-transactional tables.

For each referenced sequence, `DROP SEQUENCE` drops a temporary sequence with that name, if it exists. If it does not exist, and the `TEMPORARY` keyword is not used, it drops a non-temporary sequence with the same name, if it exists. The `TEMPORARY` keyword ensures that a non-temporary sequence will not accidentally be dropped.

Use `IF EXISTS` to prevent an error from occurring for sequences that do not exist. A NOTE is generated for each non-existent sequence when using `IF EXISTS`. See [SHOW WARNINGS](#).

`DROP SEQUENCE` requires the [DROP](#) privilege.

Notes

`DROP SEQUENCE` only removes sequences, not tables. However, [DROP TABLE](#) can remove both sequences and tables.

See Also

- [Sequence Overview](#)
- [CREATE SEQUENCE](#)
- [ALTER SEQUENCE](#)
- [DROP TABLE](#)

DROP SERVER

Syntax

```
DROP SERVER [ IF EXISTS ] server_name
```

Description

Drops the server definition for the server named `server_name`. The corresponding row within the `mysql.servers` table will be deleted. This statement requires the `SUPER` privilege or, from MariaDB 10.5.2, the `FEDERATED ADMIN` privilege.

Dropping a server for a table does not affect any `FederatedX`, `FEDERATED`, `Connect` or `Spider` tables that used this connection information when they were created.

`DROP 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.

IF EXISTS

If the `IF EXISTS` clause is used, MariaDB will not return an error if the server does not exist. Unlike all other statements, `DROP SERVER IF EXISTS` does not issue a note if the server does not exist. See [MDEV-9400](#).

Examples

```
DROP SERVER s;
```

IF EXISTS:

Contents

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

Contents

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

```
DROP SERVER s;
ERROR 1477 (HY000): The foreign server name you are trying to reference
does not exist. Data source error: s

DROP SERVER IF EXISTS s;
Query OK, 0 rows affected (0.00 sec)
```

See Also

- [CREATE SERVER](#)
- [ALTER SERVER](#)
- [Spider Storage Engine](#)
- [FederatedX Storage Engine](#)
- [Connect Storage Engine](#)

DROP TRIGGER

Syntax

```
DROP TRIGGER [IF EXISTS] [schema_name.]trigger_name
```

Description

This statement drops a [trigger](#). The schema (database) name is optional. If the schema is omitted, the trigger is dropped from the default schema. Its use requires the `TRIGGER` privilege for the table associated with the trigger.

Use `IF EXISTS` to prevent an error from occurring for a trigger that does not exist. A `NOTE` is generated for a non-existent trigger when using `IF EXISTS`. See [SHOW WARNINGS](#).

Note: Triggers for a table are also dropped if you drop the table.

Contents

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

Atomic DDL

MariaDB starting with 10.6.1

MariaDB 10.6.1 supports [Atomic DDL](#) and `DROP TRIGGER` is atomic.

Examples

```
DROP TRIGGER test.example_trigger;
```

Using the IF EXISTS clause:

```
DROP TRIGGER IF EXISTS test.example_trigger;
Query OK, 0 rows affected, 1 warning (0.01 sec)

SHOW WARNINGS;
+-----+-----+
| Level | Code | Message           |
+-----+-----+
| Note  | 1360 | Trigger does not exist |
+-----+-----+
```

See Also

- [Trigger Overview](#)
- [CREATE TRIGGER](#)

- [Information Schema TRIGGERS Table](#)
- [SHOW TRIGGERS](#)
- [SHOW CREATE TRIGGER](#)
- [Trigger Limitations](#)

DROP VIEW

Syntax

```
DROP VIEW [IF EXISTS]
  view_name [, view_name] ...
[RESTRICT | CASCADE]
```

Description

`DROP VIEW` removes one or more [views](#). You must have the `DROP` privilege for each view. If any of the views named in the argument list do not exist, MariaDB returns an error indicating by name which non-existing views it was unable to drop, but it also drops all of the views in the list that do exist.

The `IF EXISTS` clause prevents an error from occurring for views that don't exist. When this clause is given, a `NOTE` is generated for each non-existent view. See [SHOW WARNINGS](#).

`RESTRICT` and `CASCADE`, if given, are parsed and ignored.

It is possible to specify view names as `db_name.view_name`. This is useful to delete views 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.

If a view references another view, it will be possible to drop the referenced view. However, the other view will reference a view which does not exist any more. Thus, querying it will produce an error similar to the following:

```
ERROR 1356 (HY000): View 'db_name.view_name' references invalid table(s) or
column(s) or function(s) or definer/invoker of view lack rights to use them
```

This problem is reported in the output of [CHECK TABLE](#).

Note that it is not necessary to use `DROP VIEW` to replace an existing view, because [CREATE VIEW](#) has an `OR REPLACE` clause.

Atomic DDL

MariaDB starting with [10.6.1](#)

MariaDB 10.6.1 supports [Atomic DDL](#) and `DROP VIEW` for a singular view is atomic. Dropping multiple views is crash-safe.

Examples

```
DROP VIEW v,v2;
```

Given views `v` and `v2`, but no view `v3`

```
DROP VIEW v,v2,v3;
ERROR 1051 (42S02): Unknown table 'v3'
```

Contents

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

```
DROP VIEW IF EXISTS v,v2,v3;
Query OK, 0 rows affected, 1 warning (0.01 sec)
```

```
SHOW WARNINGS;
+-----+-----+
| Level | Code | Message           |
+-----+-----+
| Note  | 1051 | Unknown table 'test.v3' |
+-----+-----+
```

See Also

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

MERGE

Description

The MERGE storage engine, also known as the MRG_MyISAM engine, is a collection of identical MyISAM tables that can be used as one. "Identical" means that all tables have identical column and index information. You cannot merge MyISAM tables in which the columns are listed in a different order, do not have exactly the same columns, or have the indexes in different order. However, any or all of the MyISAM tables can be compressed with [myisampack](#). Columns names and indexes names can be different, as long as data types and NULL/NOT NULL clauses are the same. Differences in table options such as AVG_ROW_LENGTH, MAX_ROWS, or PACK_KEYS do not matter.

Each index in a MERGE table must match an index in underlying MyISAM tables, but the opposite is not true. Also, a MERGE table cannot have a PRIMARY KEY or UNIQUE indexes, because it cannot enforce uniqueness over all underlying tables.

The following options are meaningful for MERGE tables:

- `UNION` . This option specifies the list of the underlying MyISAM tables. The list is enclosed between parentheses and separated with commas.
- `INSERT_METHOD` . This option specifies whether, and how, INSERTs are allowed for the table. Allowed values are: `NO` (INSERTs are not allowed), `FIRST` (new rows will be written into the first table specified in the `UNION` list), `LAST` (new rows will be written into the last table specified in the `UNION` list). The default value is `NO` .

If you define a MERGE table with a definition which is different from the underlying MyISAM tables, or one of the underlying tables is not MyISAM, the CREATE TABLE statement will not return any error. But any statement which involves the table will produce an error like the following:

```
ERROR 1168 (HY000): Unable to open underlying table which is differently defined
or of non-MyISAM type or doesn't exist
```

A [CHECK TABLE](#) will show more information about the problem.

The error is also produced if the table is properly defined, but an underlying table's definition changes at some point in time.

If you try to insert a new row into a MERGE table with `INSERT_METHOD=NO`, you will get an error like the following:

```
ERROR 1036 (HY000): Table 'tbl_name' is read only
```

It is possible to build a MERGE table on MyISAM tables which have one or more [virtual columns](#). MERGE itself does not support virtual columns, thus such columns will be seen as regular columns. The data types and sizes will still need to be identical, and they cannot be NOT NULL.

Examples

Contents

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

```

CREATE TABLE t1 (
    a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    message CHAR(20)) ENGINE=MyISAM;

CREATE TABLE t2 (
    a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    message CHAR(20)) ENGINE=MyISAM;

INSERT INTO t1 (message) VALUES ('Testing'),('table'),('t1');

INSERT INTO t2 (message) VALUES ('Testing'),('table'),('t2');

CREATE TABLE total (
    a INT NOT NULL AUTO_INCREMENT,
    message CHAR(20), INDEX(a))
ENGINE=MERGE UNION=(t1,t2) INSERT_METHOD=LAST;

SELECT * FROM total;
+---+-----+
| a | message |
+---+-----+
| 1 | Testing |
| 2 | table   |
| 3 | t1      |
| 1 | Testing |
| 2 | table   |
| 3 | t2      |
+---+-----+

```

In the following example, we'll create three MyISAM tables, and then a MERGE table on them. However, one of them uses a different data type for the column b, so a SELECT will produce an error:

```

CREATE TABLE t1 (
    a INT,
    b INT
) ENGINE = MyISAM;

CREATE TABLE t2 (
    a INT,
    b INT
) ENGINE = MyISAM;

CREATE TABLE t3 (
    a INT,
    b TINYINT
) ENGINE = MyISAM;

CREATE TABLE t_mrg (
    a INT,
    b INT
) ENGINE = MERGE,UNION=(t1,t2,t3);

SELECT * FROM t_mrg;
ERROR 1168 (HY000): Unable to open underlying table which is differently defined
or of non-MyISAM type or doesn't exist

```

To find out what's wrong, we'll use a CHECK TABLE:

```

CHECK TABLE t_mrg;
+-----+-----+-----+
| Table | Op   | Msg_type | Msg_text
+-----+-----+-----+
| test.t_mrg | check | Error   | Table 'test.t3' is differently defined or of non-MyISAM type or doesn't exist
| test.t_mrg | check | Error   | Unable to open underlying table which is differently defined or of non-MyISAM type or doesn't exist
| test.t_mrg | check | error   | Corrupt
+-----+-----+-----+

```

Now, we know that the problem is in `t3`'s definition.

NEXT VALUE for sequence_name

MariaDB starting with 10.3

SEQUENCEs were introduced in [MariaDB 10.3](#).

Syntax

```
NEXT VALUE FOR sequence
```

or

```
NEXTVAL(sequence_name)
```

or in Oracle mode ([SQL_MODE=ORACLE](#))

```
sequence_name.nextval
```

`NEXT VALUE FOR` is ANSI SQL syntax while `NEXTVAL()` is PostgreSQL syntax.

Contents

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

Description

Generate next value for a `SEQUENCE`.

- You can greatly speed up `NEXT VALUE` by creating the sequence with the `CACHE` option. If not, every `NEXT VALUE` usage will cause changes in the stored `SEQUENCE` table.
- When using `NEXT VALUE` the value will be reserved at once and will not be reused, except if the `SEQUENCE` was created with `CYCLE`. This means that when you are using `SEQUENCE`s you have to expect gaps in the generated sequence numbers.
- If one updates the `SEQUENCE` with `SETVAL()` or `ALTER SEQUENCE ... RESTART`, `NEXT VALUE FOR` will notice this and start from the next requested value.
- `FLUSH TABLES` will close the sequence and the next sequence number generated will be according to what's stored in the `SEQUENCE` object. In effect, this will discard the cached values.
- A server restart (or closing the current connection) also causes a drop of all cached values. The cached sequence numbers are reserved only for the current connection.
- `NEXT VALUE` requires the `INSERT` privilege.

MariaDB starting with 10.3.3

- You can also use `NEXT VALUE FOR sequence` for column `DEFAULT`.

See Also

- [Sequence Overview](#)
- [CREATE SEQUENCE](#)
- [ALTER SEQUENCE](#)
- [PREVIOUS VALUE FOR](#)
- [SETVAL\(\)](#). Set next value for the sequence.
- [AUTO_INCREMENT](#)

PREVIOUS VALUE FOR sequence_name

MariaDB starting with 10.3

SEQUENCEs were introduced in [MariaDB 10.3](#).

Syntax

```
PREVIOUS VALUE FOR sequence_name
```

or

```
LASTVAL(sequence_name)
```

or in Oracle mode ([SQL_MODE=ORACLE](#))

```
sequence_name.currval
```

PREVIOUS VALUE FOR is IBM DB2 syntax while LASTVAL() is PostgreSQL syntax.

Description

Get last value in the current connection generated from a sequence.

Contents

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

- If the sequence has not yet been used by the connection, PREVIOUS VALUE FOR returns NULL (the same thing applies with a new connection which doesn't see a last value for an existing sequence).
- If a SEQUENCE has been dropped and re-created then it's treated as a new SEQUENCE and PREVIOUS VALUE FOR will return NULL .
- FLUSH TABLES has no effect on PREVIOUS VALUE FOR .
- Previous values for all used sequences are stored per connection until connection ends.
- PREVIOUS VALUE FOR requires the [SELECT privilege](#).

Example

```
MariaDB [test]> CREATE SEQUENCE s START WITH 100 INCREMENT BY 10;
Query OK, 0 rows affected (0.026 sec)

MariaDB [test]> SELECT PREVIOUS VALUE FOR s;
+-----+
| PREVIOUS VALUE FOR s |
+-----+
|          NULL         |
+-----+
1 row in set (0.000 sec)

# The function works for sequences only, if the table is used an error is generated
MariaDB [test]> SELECT PREVIOUS VALUE FOR t;
ERROR 4089 (42S02): 'test.t' is not a SEQUENCE

# Call the NEXT VALUE FOR s:
MariaDB [test]> SELECT NEXT VALUE FOR s;
+-----+
| NEXT VALUE FOR s |
+-----+
|          100        |
+-----+
1 row in set (0.000 sec)

MariaDB [test]> SELECT PREVIOUS VALUE FOR s;
+-----+
| PREVIOUS VALUE FOR s |
+-----+
|          100        |
+-----+
1 row in set (0.000 sec)
```

Now try to start the new connection and check that the last value is still NULL, before updating the value in the new connection after the output of the new connection gets current value (110 in the example below). Note that first connection cannot see this change and the result of last value still remains the same (100 in the example above).

```
$ .mysql -uroot test -e"SELECT PREVIOUS VALUE FOR s; SELECT NEXT VALUE FOR s; SELECT PREVIOUS VALUE FOR s;"
```

PREVIOUS VALUE FOR s
NULL
NEXT VALUE FOR s
110
PREVIOUS VALUE FOR s
110

See Also

- [Sequence Overview](#)
- [CREATE SEQUENCE](#)
- [ALTER SEQUENCE](#)
- [NEXT VALUE FOR](#)
- [SETVAL\(\)](#). Set next value for the sequence.
- [AUTO_INCREMENT](#)

Sequence Overview

MariaDB starting with 10.3

Sequences were introduced in [MariaDB 10.3](#).

Introduction

A sequence is an object that generates a sequence of numeric values, as specified by the [CREATE SEQUENCE](#) statement.

`CREATE SEQUENCE` will create a sequence that generates new values when called with `NEXT VALUE FOR sequence_name`. It's an alternative to [AUTO_INCREMENT](#) when one wants to have more control of how the numbers are generated. As the `SEQUENCE` caches values (up to the `CACHE` value in the `CREATE SEQUENCE` statement, by default 1000) it can in some cases be much faster than `AUTO_INCREMENT`. Another benefit is that one can access the last value generated by all used sequences, which solves one of the limitations with [LAST_INSERT_ID\(\)](#).

Creating a Sequence

The `CREATE SEQUENCE` statement is used to create a sequence. Here is an example of a sequence starting at 100, incrementing by 10 each time:

```
CREATE SEQUENCE s START WITH 100 INCREMENT BY 10;
```

The `CREATE SEQUENCE` statement, along with defaults, can be viewed with the `SHOW CREATE SEQUENCE` statement, for example:

```
SHOW CREATE SEQUENCE s
*****
1. row *****
Table: s
Create Table: CREATE SEQUENCE `s` start with 100 minvalue 1 maxvalue 9223372036854775806
increment by 10 cache 1000 nocycle ENGINE=InnoDB
```

Contents

1. [Introduction](#)
2. [Creating a Sequence](#)
3. [Using Sequence Objects](#)
 1. [Using Sequences in DEFAULT](#)
4. [Changing a Sequence](#)
5. [Dropping a Sequence](#)
6. [Replication](#)
7. [Standards Compliance](#)
8. [Notes](#)
9. [Table Operations that Work with Sequences](#)
10. [Implementation](#)
11. [Underlying Table Structure](#)
12. [Credits](#)
13. [See Also](#)

Using Sequence Objects

To get the [next value from a sequence](#), use

```
NEXT VALUE FOR sequence_name
```

or

```
NEXTVAL(sequence_name)
```

or in Oracle mode ([SQL_MODE=ORACLE](#))

```
sequence_name.nextval
```

For [retrieving the last value](#) used by the current connection from a sequence use:

```
PREVIOUS VALUE FOR sequence_name
```

or

```
LASTVAL(sequence_name)
```

or in Oracle mode ([SQL_MODE=ORACLE](#))

```
sequence_name.currvla
```

For example:

```
SELECT NEXTVAL(s);
+-----+
| NEXTVAL(s) |
+-----+
|      100   |
+-----+

SELECT NEXTVAL(s);
+-----+
| NEXTVAL(s) |
+-----+
|      110   |
+-----+

SELECT LASTVAL(s);
+-----+
| LASTVAL(s) |
+-----+
|      110   |
+-----+
```

Using Sequences in DEFAULT

MariaDB starting with [10.3.3](#)

Starting from 10.3.3 you can use Sequences in DEFAULT:

```

create sequence s1;
create table t1 (a int primary key default (next value for s1), b int);
insert into t1 (b) values (1),(2);
select * from t1;
+----+-----+
| a | b    |
+----+-----+
| 1 | 1   |
| 2 | 2   |
+----+-----+

```

Changing a Sequence

The [ALTER SEQUENCE](#) statement is used for changing sequences. For example, to restart the sequence at another value:

```

ALTER SEQUENCE s RESTART 50;

SELECT NEXTVAL(s);
+-----+
| NEXTVAL(s) |
+-----+
|      50   |
+-----+

```

The [SETVAL function](#) can also be used to set the next value to be returned for a SEQUENCE, for example:

```

SELECT SETVAL(s, 100);
+-----+
| SETVAL(s, 100) |
+-----+
|      100   |
+-----+

```

SETVAL can only be used to increase the sequence value. Attempting to set a lower value will fail, returning NULL:

```

SELECT SETVAL(s, 50);
+-----+
| SETVAL(s, 50) |
+-----+
|      NULL   |
+-----+

```

Dropping a Sequence

The [DROP SEQUENCE](#) statement is used to drop a sequence, for example:

```

DROP SEQUENCE s;

```

Replication

If one wants to use Sequences in a master-master setup or with Galera one should use `INCREMENT=0`. This will tell the Sequence to use `auto_increment_increment` and `auto_increment_offset` to generate unique values for each server.

Standards Compliance

[MariaDB 10.3](#) supports both ANSI SQL and Oracle syntax for sequences.

However as `SEQUENCE` is implemented as a special kind of table, it uses the same namespace as tables. The benefits are that sequences show up in [SHOW TABLES](#), and one can also create a sequence with [CREATE TABLE](#) and drop it with [DROP TABLE](#). One can [SELECT](#) from it as from any other table. This ensures that all old tools that work with tables should work with sequences.

Since sequence objects act as regular tables in many contexts, they will be affected by [LOCK TABLES](#). This is not the case in other DBMS, such as

Oracle, where LOCK TABLE does not affect sequences.

Notes

One of the goals with the Sequence implementation is that all old tools, such as `mysqldump`, should work unchanged, while still keeping the normal usage of sequence standard compatibility.

To make this possible, `sequence` is currently implemented as a table with a few exclusive properties.

The special properties for sequence tables are:

- A sequence table has always one row.
- When one creates a sequence, either with `CREATE TABLE` or `CREATE SEQUENCE`, one row will be inserted.
- If one tries to insert into a sequence table, the single row will be updated. This allows `mysqldump` to work but also gives the additional benefit that one can change all properties of a sequence with a single insert. New applications should of course also use `ALTER SEQUENCE`.
- `UPDATE` or `DELETE` can't be performed on Sequence objects.
- Doing a select on the sequence shows the current state of the sequence, except the values that are reserved in the cache. The `next_value` column shows the next value not reserved by the cache.
- `FLUSH TABLES` will close the sequence and the next sequence number generated will be according to what's stored in the Sequence object. In effect, this will discard the cached values.
- A number of normal table operations work on Sequence tables. See next section.

Table Operations that Work with Sequences

- `SHOW CREATE TABLE sequence_name`. This shows the table structure that is behind the `SEQUENCE` including the field names that can be used with `SELECT` or even `CREATE TABLE`.
- `CREATE TABLE sequence-structure ... SEQUENCE=1`
- `ALTER TABLE sequence RENAME TO sequence2`
- `RENAME TABLE sequence_name TO new_sequence_name`
- `DROP TABLE sequence_name`. This is allowed mainly to get old tools like `mysqldump` to work with sequence tables.
- `SHOW TABLES`

Implementation

Internally, sequence tables are created as a normal table without rollback (the `InnoDB`, `Aria` and `MySAM` engines support this), wrapped by a sequence engine object. This allowed us to create sequences with almost no performance impact for normal tables. (The cost is one 'if' per insert if the `binary log` is enabled).

Underlying Table Structure

The following example shows the table structure of sequences and how it can be used as a table. (Output of results are slightly edited to make them easier to read)

```

create sequence t1;
show create sequence t1\G
***** 1. row *****
CREATE SEQUENCE `t1` start with 1 minvalue 1 maxvalue 9223372036854775806
increment by 1 cache 1000 nocycle ENGINE=InnoDB

show create table t1\G
***** 1. row *****
Create Table: CREATE TABLE `t1` (
`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 max value is reached',
`cycle_count` bigint(21) NOT NULL COMMENT 'How many cycles have been done'
) ENGINE=InnoDB SEQUENCE=1

select * from t1\G
next_not_cached_value: 1
minimum_value: 1
maximum_value: 9223372036854775806
start_value: 1
increment: 1
cache_size: 1000
cycle_option: 0
cycle_count: 0

```

The `cycle_count` column is incremented every time the sequence wraps around.

Credits

- Thanks to Jianwe Zhao from Aliyun for his work on SEQUENCE in AliSQL, which gave ideas and inspiration for this work.
- Thanks to Peter Gulutzan, who helped test and gave useful comments about the implementation.

See Also

- [CREATE SEQUENCE](#)
- [ALTER SEQUENCE](#)
- [DROP SEQUENCE](#)
- [NEXT VALUE FOR](#)
- [PREVIOUS VALUE FOR](#)
- [SETVAL\(\)](#). Set next value for the sequence.
- [AUTO INCREMENT](#)

SETVAL()

MariaDB starting with [10.3.1](#)

SEQUENCEs were introduced in [MariaDB 10.3](#).

Syntax

```
SETVAL(sequence_name, next_value, [is_used, [round]])
```

Description

Set the next value to be returned for a `SEQUENCE`.

This function is compatible with PostgreSQL syntax, extended with the `round` argument.

If the `is_used` argument is not given or is `1` or `true`, then the next used value will one after the given value. If `is_used` is `0` or `false` then the next generated value will be the given value.

If `round` is used then it will set the `round` value (or the internal cycle count, starting at zero) for the sequence. If `round` is not used, it's assumed to be `0`.

`next_value` must be an integer literal.

For `SEQUENCE` tables defined with `CYCLE` (see [CREATE SEQUENCE](#)) one should use both `next_value` and `round` to define the next value. In this case the current sequence value is defined to be `round`, `next_value`.

The result returned by `SETVAL()` is `next_value` or `NULL` if the given `next_value` and `round` is smaller than the current value.

`SETVAL()` will not set the `SEQUENCE` value to a something that is less than its current value. This is needed to ensure that `SETVAL()` is replication safe. If you want to set the `SEQUENCE` to a smaller number use [ALTER SEQUENCE](#).

If `CYCLE` is used, first `round` and then `next_value` are compared to see if the value is bigger than the current value.

Internally, in the MariaDB server, `SETVAL()` is used to inform slaves that a `SEQUENCE` has changed value. The slave may get `SETVAL()` statements out of order, but this is ok as only the biggest one will have an effect.

`SETVAL` requires the [INSERT privilege](#).

Contents

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

Examples

```
SELECT setval(foo, 42);          -- Next nextval will return 43
SELECT setval(foo, 42, true);    -- Same as above
SELECT setval(foo, 42, false);   -- Next nextval will return 42
```

`SETVAL` setting higher and lower values on a sequence with an increment of 10:

```
SELECT NEXTVAL(s);
+-----+
| NEXTVAL(s) |
+-----+
|      50     |
+-----+

SELECT SETVAL(s, 100);
+-----+
| SETVAL(s, 100) |
+-----+
|        100    |
+-----+

SELECT NEXTVAL(s);
+-----+
| NEXTVAL(s) |
+-----+
|      110    |
+-----+

SELECT SETVAL(s, 50);
+-----+
| SETVAL(s, 50) |
+-----+
|       NULL    |
+-----+

SELECT NEXTVAL(s);
+-----+
| NEXTVAL(s) |
+-----+
|      120    |
+-----+
```

Example demonstrating round :

```
CREATE OR REPLACE SEQUENCE s1
  START WITH 1
  MINVALUE 1
  MAXVALUE 99
  INCREMENT BY 1
  CACHE 20
  CYCLE;

SELECT SETVAL(s1, 99, 1, 0);
+-----+
| SETVAL(s1, 99, 1, 0) |
+-----+
|          99 |
+-----+

SELECT NEXTVAL(s1);
+-----+
| NEXTVAL(s1) |
+-----+
|          1 |
+-----+
```

The following statement returns NULL, as the given next_value and round is smaller than the current value.

```
SELECT SETVAL(s1, 99, 1, 0);
+-----+
| SETVAL(s1, 99, 1, 0) |
+-----+
|          NULL |
+-----+

SELECT NEXTVAL(s1);
+-----+
| NEXTVAL(s1) |
+-----+
|          2 |
+-----+
```

Increasing the round from zero to 1 will allow next_value to be returned.

```
SELECT SETVAL(s1, 99, 1, 1);
+-----+
| SETVAL(s1, 99, 1, 1) |
+-----+
|          99 |
+-----+

SELECT NEXTVAL(s1);
+-----+
| NEXTVAL(s1) |
+-----+
|          1 |
+-----+
```

See Also

- [Sequence Overview](#)
- [ALTER SEQUENCE](#)
- [CREATE SEQUENCE](#)
- [NEXT VALUE FOR](#)
- [PREVIOUS VALUE FOR](#)

JSON_ARRAY_APPEND

MariaDB starting with 10.2.3

JSON functions were added in MariaDB 10.2.3.

Syntax

```
JSON_ARRAY_APPEND(json_doc, path, value[, path, value] ...)
```

Description

Appends values to the end of the specified arrays within a JSON document, returning the result, or NULL if any of the arguments are NULL.

Evaluation is performed from left to right, with the resulting document from the previous pair becoming the new value against which the next pair is evaluated.

If the `json_doc` is not a valid JSON document, or if any of the paths are not valid, or contain a * or ** wildcard, an error is returned.

Examples

```
SET @json = '[1, 2, [3, 4]]';

SELECT JSON_ARRAY_APPEND(@json, '$[0]', 5)
+-----+
| JSON_ARRAY_APPEND(@json, '$[0]', 5) |
+-----+
| [[1, 5], 2, [3, 4]]                |
+-----+

SELECT JSON_ARRAY_APPEND(@json, '$[1]', 6);
+-----+
| JSON_ARRAY_APPEND(@json, '$[1]', 6) |
+-----+
| [1, [2, 6], [3, 4]]                |
+-----+

SELECT JSON_ARRAY_APPEND(@json, '$[1]', 6, '$[2]', 7);
+-----+
| JSON_ARRAY_APPEND(@json, '$[1]', 6, '$[2]', 7) |
+-----+
| [1, [2, 6], [3, 4, 7]]                |
+-----+

SELECT JSON_ARRAY_APPEND(@json, '$', 5);
+-----+
| JSON_ARRAY_APPEND(@json, '$', 5) |
+-----+
| [1, 2, [3, 4], 5]                |
+-----+

SET @json = '{"A": 1, "B": [2], "C": [3, 4]}';

SELECT JSON_ARRAY_APPEND(@json, '$.B', 5);
+-----+
| JSON_ARRAY_APPEND(@json, '$.B', 5) |
+-----+
| [{"A": 1, "B": [2, 5], "C": [3, 4]}] |
+-----+
```

JSON_ARRAY_INSERT

MariaDB starting with 10.2.3

JSON functions were added in MariaDB 10.2.3.

Syntax

```
JSON_ARRAY_INSERT(json_doc, path, value[, path, value] ...)
```

Description

Inserts a value into a JSON document, returning the modified document, or NULL if any of the arguments are NULL.

Evaluation is performed from left to right, with the resulting document from the previous pair becoming the new value against which the next pair is evaluated.

If the `json_doc` is not a valid JSON document, or if any of the paths are not valid, or contain a `*` or `**` wildcard, an error is returned.

Examples

```
SET @json = '[1, 2, [3, 4]]';

SELECT JSON_ARRAY_INSERT(@json, '$[0]', 5);
+-----+
| JSON_ARRAY_INSERT(@json, '$[0]', 5) |
+-----+
| [5, 1, 2, [3, 4]]                   |
+-----+

SELECT JSON_ARRAY_INSERT(@json, '$[1]', 6);
+-----+
| JSON_ARRAY_INSERT(@json, '$[1]', 6) |
+-----+
| [1, 6, 2, [3, 4]]                   |
+-----+

SELECT JSON_ARRAY_INSERT(@json, '$[1]', 6, '$[2]', 7);
+-----+
| JSON_ARRAY_INSERT(@json, '$[1]', 6, '$[2]', 7) |
+-----+
| [1, 6, 7, 2, [3, 4]]                 |
+-----+
```

JSON_CONTAINS

MariaDB starting with 10.2.3

JSON functions were added in MariaDB 10.2.3.

Syntax

```
JSON_CONTAINS(json_doc, val[, path])
```

Description

Returns whether or not the specified value is found in the given JSON document or, optionally, at the specified path within the document. Returns 1 if it does, 0 if not and NULL if any of the arguments are null. An error occurs if the document or path is not valid, or contains the `*` or `**` wildcards.

Examples

```

SET @json = '{"A": 0, "B": {"C": 1}, "D": 2}';

SELECT JSON_CONTAINS(@json, '2', '$.A');
+-----+
| JSON_CONTAINS(@json, '2', '$.A') |
+-----+
|          0 |
+-----+

SELECT JSON_CONTAINS(@json, '2', '$.D');
+-----+
| JSON_CONTAINS(@json, '2', '$.D') |
+-----+
|          1 |
+-----+

SELECT JSON_CONTAINS(@json, '{"C": 1}', '$.A');
+-----+
| JSON_CONTAINS(@json, '{"C": 1}', '$.A') |
+-----+
|          0 |
+-----+

SELECT JSON_CONTAINS(@json, '{"C": 1}', '$.B');
+-----+
| JSON_CONTAINS(@json, '{"C": 1}', '$.B') |
+-----+
|          1 |
+-----+

```

JSON_CONTAINS_PATH

MariaDB starting with 10.2.3

JSON functions were added in MariaDB 10.2.3.

Syntax

```
JSON_CONTAINS_PATH(json_doc, return_arg, path[, path] ...)
```

Description

Indicates whether the given JSON document contains data at the specified path or paths. Returns 1 if it does, 0 if not and NULL if any of the arguments are null.

The *return_arg* can be `one` or `all`:

- `one` - Returns 1 if at least one path exists within the JSON document.
- `all` - Returns 1 only if all paths exist within the JSON document.

Examples

```

SET @json = '{"A": 1, "B": [2], "C": [3, 4]}';

SELECT JSON_CONTAINS_PATH(@json, 'one', '$.A', '$.D');
+-----+
| JSON_CONTAINS_PATH(@json, 'one', '$.A', '$.D') |
+-----+
| 1 |
+-----+
1 row in set (0.00 sec)

SELECT JSON_CONTAINS_PATH(@json, 'all', '$.A', '$.D');
+-----+
| JSON_CONTAINS_PATH(@json, 'all', '$.A', '$.D') |
+-----+
| 0 |
+-----+

```

JSON_DEPTH

MariaDB starting with 10.2.3

JSON functions were added in MariaDB 10.2.3.

Syntax

```
JSON_DEPTH(json_doc)
```

Description

Returns the maximum depth of the given JSON document, or NULL if the argument is null. An error will occur if the argument is an invalid JSON document.

- Scalar values or empty arrays or objects have a depth of 1.
- Arrays or objects that are not empty but contain only elements or member values of depth 1 will have a depth of 2.
- In other cases, the depth will be greater than 2.

Examples

```

SELECT JSON_DEPTH('[]'), JSON_DEPTH('true'), JSON_DEPTH('{}');
+-----+
| JSON_DEPTH('[]') | JSON_DEPTH('true') | JSON_DEPTH('{}') |
+-----+
| 1 | 1 | 1 |
+-----+

SELECT JSON_DEPTH('[1, 2, 3)'), JSON_DEPTH('[], {}, []');
+-----+
| JSON_DEPTH('[1, 2, 3]') | JSON_DEPTH('[], {}, []') |
+-----+
| 2 | 2 |
+-----+

SELECT JSON_DEPTH('[1, 2, [3, 4, 5, 6], 7]');
+-----+
| JSON_DEPTH('[1, 2, [3, 4, 5, 6], 7]') |
+-----+
| 3 |
+-----+

```

JSON_EXISTS

MariaDB starting with 10.2.3

JSON functions were added in MariaDB 10.2.3.

Syntax

Description

Determines whether a specified JSON value exists in the given data. Returns 1 if found, 0 if not, or NULL if any of the inputs were NULL.

Examples

```
SELECT JSON_EXISTS('{"key1":"xxxx", "key2": [1, 2, 3]}',("$.key2");
+-----+
| JSON_EXISTS('{"key1":"xxxx", "key2": [1, 2, 3]}',("$.key2") |
+-----+
|          1 |
+-----+  
  
SELECT JSON_EXISTS('{"key1":"xxxx", "key2": [1, 2, 3]}',("$.key3");
+-----+
| JSON_EXISTS('{"key1":"xxxx", "key2": [1, 2, 3]}',("$.key3") |
+-----+
|          0 |
+-----+  
  
SELECT JSON_EXISTS('{"key1":"xxxx", "key2": [1, 2, 3]}',("$.key2[1]");
+-----+
| JSON_EXISTS('{"key1":"xxxx", "key2": [1, 2, 3]}',("$.key2[1]) |
+-----+
|          1 |
+-----+  
  
SELECT JSON_EXISTS('{"key1":"xxxx", "key2": [1, 2, 3]}',("$.key2[10]");
+-----+
| JSON_EXISTS('{"key1":"xxxx", "key2": [1, 2, 3]}',("$.key2[10]) |
+-----+
|          0 |
+-----+
```

JSON_EXTRACT

MariaDB starting with 10.2.3

JSON functions were added in MariaDB 10.2.3.

Syntax

```
JSON_EXTRACT(json_doc, path[, path] ...)
```

Description

Extracts data from a JSON document. The extracted data is selected from the parts matching the path arguments. Returns all matched values; either as a single matched value, or, if the arguments could return multiple values, a result autowrapped as an array in the matching order.

Returns NULL if no paths match or if any of the arguments are NULL.

An error will occur if any path argument is not a valid path, or if the json_doc argument is not a valid JSON document.

The path expression be a [JSONPath expression](#) as supported by MariaDB

Contents

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

Examples

```
SET @json = '[1, 2, [3, 4]]';

SELECT JSON_EXTRACT(@json, '$[1]');
+-----+
| JSON_EXTRACT(@json, '$[1]') |
+-----+
| 2 |
+-----+

SELECT JSON_EXTRACT(@json, '$[2]');
+-----+
| JSON_EXTRACT(@json, '$[2]') |
+-----+
| [3, 4] |
+-----+

SELECT JSON_EXTRACT(@json, '$[2][1]');
+-----+
| JSON_EXTRACT(@json, '$[2][1]') |
+-----+
| 4 |
+-----+
```

See Also

- [JSON video tutorial](#) covering JSON_EXTRACT.

JSON_INSERT

MariaDB starting with [10.2.3](#)

JSON functions were added in [MariaDB 10.2.3](#).

Syntax

```
JSON_INSERT(json_doc, path, val[, path, val] ...)
```

Description

Inserts data into a JSON document, returning the resulting document or NULL if any argument is null.

An error will occur if the JSON document is not valid, or if any of the paths are invalid or contain a * or ** wildcard.

JSON_INSERT can only insert data while [JSON_REPLACE](#) can only update. [JSON_SET](#) can update or insert data.

Contents

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

Examples

```
SET @json = '{ "A": 0, "B": [1, 2]}';

SELECT JSON_INSERT(@json, '$.C', '[3, 4]');
+-----+
| JSON_INSERT(@json, '$.C', '[3, 4]') |
+-----+
| { "A": 0, "B": [1, 2], "C": "[3, 4]" } |
+-----+
```

See Also

- [JSON video tutorial](#) covering JSON_INSERT.

JSON_KEYS

MariaDB starting with [10.2.3](#)

JSON functions were added in [MariaDB 10.2.3](#).

Syntax

```
JSON_KEYS(json_doc[, path])
```

Description

Returns the keys as a JSON array from the top-level value of a JSON object or, if the optional path argument is provided, the top-level keys from the path.

Excludes keys from nested sub-objects in the top level value. The resulting array will be empty if the selected object is empty.

Returns NULL if any of the arguments are null, a given path does not locate an object, or if the json_doc argument is not an object.

An error will occur if JSON document is invalid, the path is invalid or if the path contains a * or ** wildcard.

Examples

```
SELECT JSON_KEYS('{"A": 1, "B": {"C": 2}}');
+-----+
| JSON_KEYS('{"A": 1, "B": {"C": 2}}') |
+-----+
| ["A", "B"] |
+-----+
+-----+
SELECT JSON_KEYS('{"A": 1, "B": 2, "C": {"D": 3}}', '$.C');
+-----+
| JSON_KEYS('{"A": 1, "B": 2, "C": {"D": 3}}', '$.C') |
+-----+
| ["D"] |
+-----+
```

JSON_LENGTH

MariaDB starting with [10.2.3](#)

JSON functions were added in [MariaDB 10.2.3](#).

Syntax

```
JSON_LENGTH(json_doc[, path])
```

Description

Returns the length of a JSON document, or, if the optional path argument is given, the length of the value within the document specified by the path.

Returns NULL if any of the arguments argument are null or the path argument does not identify a value in the document.

An error will occur if the JSON document is invalid, the path is invalid or if the path contains a * or ** wildcard.

Length will be determined as follow:

- A scalar's length is always 1.
- If an array, the number of elements in the array.
- If an object, the number of members in the object.

The length of nested arrays or objects are not counted.

Examples

JSON_QUERY

MariaDB starting with [10.2.3](#)

JSON functions were added in [MariaDB 10.2.3](#).

Syntax

```
JSON_QUERY(json_doc, path)
```

Description

Given a JSON document, returns an object or array specified by the path. Returns NULL if not given a valid JSON document, or if there is no match.

Examples

```
select json_query('{"key1":{"a":1, "b":[1,2]}', '$.key1');
+-----+
| json_query('{"key1":{"a":1, "b":[1,2]}', '$.key1') |
+-----+
| [{"a":1, "b":[1,2]}] |
+-----+

select json_query('{"key1":123, "key1": [1,2,3]}', '$.key1');
+-----+
| json_query('{"key1":123, "key1": [1,2,3]}', '$.key1') |
+-----+
| [1,2,3] |
+-----+
```

JSON_QUOTE

MariaDB starting with [10.2.3](#)

JSON functions were added in [MariaDB 10.2.3](#).

Syntax

```
JSON_QUOTE(json_value)
```

Description

Quotes a string as a JSON value, usually for producing valid JSON string literals for inclusion in JSON documents. Wraps the string with double quote characters and escapes interior quotes and other special characters, returning a utf8mb4 string.

Returns NULL if the argument is NULL.

Examples

```
SELECT JSON_QUOTE('A'), JSON_QUOTE("B"), JSON_QUOTE('"C"');
+-----+-----+-----+
| JSON_QUOTE('A') | JSON_QUOTE("B") | JSON_QUOTE('"C"') |
+-----+-----+-----+
| "A"           | "B"           | "\"C\""
+-----+-----+-----+
```

JSON_REMOVE

MariaDB starting with 10.2.3

JSON functions were added in MariaDB 10.2.3.

Syntax

```
JSON_REMOVE(json_doc, path[, path] ...)
```

Description

Removes data from a JSON document returning the result, or NULL if any of the arguments are null. If the element does not exist in the document, no changes are made.

An error will occur if JSON document is invalid, the path is invalid or if the path contains a * or ** wildcard.

Path arguments are evaluated from left to right, with the result from the earlier evaluation being used as the value for the next.

Contents

- [Syntax](#)
- [Description](#)
- [Examples](#)
- [See Also](#)

Examples

```
SELECT JSON_REMOVE('{"A": 1, "B": 2, "C": {"D": 3}}', '$.C');
+-----+
| JSON_REMOVE('{"A": 1, "B": 2, "C": {"D": 3}}', '$.C') |
+-----+
| {"A": 1, "B": 2} |
+-----+

SELECT JSON_REMOVE('["A", "B", ["C", "D"], "E"]', '$[1]');
+-----+
| JSON_REMOVE('["A", "B", ["C", "D"], "E"]', '$[1]') |
+-----+
| ["A", ["C", "D"], "E"] |
+-----+
```

See Also

- [JSON video tutorial](#) covering JSON_REMOVE.

JSON_REPLACE

MariaDB starting with 10.2.3

JSON functions were added in MariaDB 10.2.3.

Syntax

```
JSON_REPLACE(json_doc, path, val[, path, val] ...)
```

Description

Replaces existing values in a JSON document, returning the result, or NULL if any of the arguments are NULL.

An error will occur if the JSON document is invalid, the path is invalid or if the path contains a * or ** wildcard.

Paths and values are evaluated from left to right, with the result from the earlier evaluation being used as the value for the next.

JSON_REPLACE can only update data, while [JSON_INSERT](#) can only insert. [JSON_SET](#) can update or insert data.

Examples

```
SELECT JSON_REPLACE('{ "A": 1, "B": [2, 3]}', '$.B[1]', 4);
+-----+
| JSON_REPLACE('{ "A": 1, "B": [2, 3]}', '$.B[1]', 4) |
+-----+
| { "A": 1, "B": [2, 4]} |
+-----+
```

See Also

- [JSON video tutorial](#) covering JSON_REPLACE.

JSON_SEARCH

MariaDB starting with [10.2.3](#)

JSON functions were added in [MariaDB 10.2.3](#).

Syntax

```
JSON_SEARCH(json_doc, return_arg, search_str[, escape_char[, path] ...])
```

Description

Returns the path to the given string within a JSON document, or NULL if any of *json_doc*, *search_str* or a path argument is NULL; if the search string is not found, or if no path exists within the document.

A warning will occur if the JSON document is not valid, any of the path arguments are not valid, if *return_arg* is neither *one* nor *all*, or if the escape character is not a constant. NULL will be returned.

return_arg can be one of two values:

- 'one' : Terminates after finding the first match, so will return one path string. If there is more than one match, it is undefined which is considered first.
- 'all' : Returns all matching path strings, without duplicates. Multiple strings are autowrapped as an array. The order is undefined.

Examples

Contents

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

```

SET @json = '[{"A": [{"B": "1"}], {"C":"AB"}, {"D":"BC"}]';

SELECT JSON_SEARCH(@json, 'one', 'AB');
+-----+
| JSON_SEARCH(@json, 'one', 'AB') |
+-----+
| "$[2].C" |
+-----+

```

JSON_TYPE

MariaDB starting with [10.2.3](#)

JSON functions were added in [MariaDB 10.2.3](#).

Syntax

```
JSON_TYPE(json_val)
```

Description

Returns the type of a JSON value (as a string), or NULL if the argument is null.

An error will occur if the argument is an invalid JSON value.

The following is a complete list of the possible return types:

Return type	Value	Example
ARRAY	JSON array	[1, 2, {"key": "value"}]
OBJECT	JSON object	{"key": "value"}
BOOLEAN	JSON true/false literals	true, false
DOUBLE	A number with at least one floating point decimal.	1.2
INTEGER	A number without a floating point decimal.	1
NULL	JSON null literal (this is returned as a string, not to be confused with the SQL NULL value!)	null
STRING	JSON String	"a sample string"

Examples

```

SELECT JSON_TYPE('{"A": 1, "B": 2, "C": 3}');
+-----+
| JSON_TYPE('{"A": 1, "B": 2, "C": 3}') |
+-----+
| OBJECT |
+-----+

```

JSON_UNQUOTE

MariaDB starting with [10.2.3](#)

JSON functions were added in [MariaDB 10.2.3](#).

Syntax

```
JSON_UNQUOTE(val)
```

Description

Unquotes a JSON value, returning a string, or NULL if the argument is null.

An error will occur if the given value begins and ends with double quotes and is an invalid JSON string literal.

If the given value is not a JSON string, value is passed through unmodified.

Certain character sequences have special meanings within a string. Usually, a backslash is ignored, but the escape sequences in the table below are recognised by MariaDB, unless the [SQL Mode](#) is set to NO_BACKSLASH_ESCAPES SQL.

Escape sequence	Character
\"	Double quote ("")
\b	Backslash
\f	Formfeed
\n	Newline (linefeed)
\r	Carriage return
\t	Tab
\\	Backslash ()
\uXXXX	UTF-8 bytes for Unicode value XXXX

Examples

```
SELECT JSON_UNQUOTE('"Monty"');
+-----+
| JSON_UNQUOTE('"Monty") |
+-----+
| Monty |
+-----+
```

With the default [SQL Mode](#):

```
SELECT JSON_UNQUOTE('Si\bng\ting');
+-----+
| JSON_UNQUOTE('Si\bng\ting') |
+-----+
| Sng ing |
+-----+
```

Setting NO_BACKSLASH_ESCAPES:

```
SET @@sql_mode = 'NO_BACKSLASH_ESCAPES';
SELECT JSON_UNQUOTE('Si\bng\ting');
+-----+
| JSON_UNQUOTE('Si\bng\ting') |
+-----+
| Si\bng\ting |
+-----+
```

JSON_VALID

MariaDB starting with [10.2.3](#)

JSON functions were added in [MariaDB 10.2.3](#).

Syntax

```
JSON_VALID(value)
```

Description

Indicates whether the given value is a valid JSON document or not. Returns `1` if valid, `0` if not, and `NULL` if the argument is `NULL`.

From MariaDB 10.4.3, the `JSON_VALID` function is automatically used as a `CHECK constraint` for the `JSON data type alias` in order to ensure that a valid json document is inserted.

Contents

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

Examples

```
SELECT JSON_VALID('{"id": 1, "name": "Monty"}');
+-----+
| JSON_VALID('{"id": 1, "name": "Monty"}') |
+-----+
|                               1 |
+-----+  
  
SELECT JSON_VALID('{"id": 1, "name": "Monty", "oddfield"}');
+-----+
| JSON_VALID('{"id": 1, "name": "Monty", "oddfield"}') |
+-----+
|                               0 |
+-----+
```

See Also

- [JSON video tutorial](#) covering `JSON_VALID`.

JSON_VALUE

MariaDB starting with [10.2.3](#)

JSON functions were added in [MariaDB 10.2.3](#).

Syntax

```
JSON_VALUE(json_doc, path)
```

Description

Given a JSON document, returns the scalar specified by the path. Returns `NULL` if not given a valid JSON document, or if there is no match.

Examples

```
select json_value('{"key1":123}', '$.key1');
+-----+
| json_value('{"key1":123}', '$.key1') |
+-----+
| 123 |
+-----+

select json_value('{"key1": [1,2,3], "key1":123}', '$.key1');
+-----+
| json_value('{"key1": [1,2,3], "key1":123}', '$.key1') |
+-----+
| 123 |
+-----+
```

DENSE_RANK

MariaDB starting with 10.2

The DENSE_RANK() function was first introduced with [window functions](#) in MariaDB 10.2.0.

Syntax

```
DENSE_RANK() OVER (
    [ PARTITION BY partition_expression ]
    [ ORDER BY order_list ]
)
```

Contents

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

Description

DENSE_RANK() is a [window function](#) that displays the number of a given row, starting at one and following the [ORDER BY](#) sequence of the window function, with identical values receiving the same result. Unlike the [RANK\(\)](#) function, there are no skipped values if the preceding results are identical. It is also similar to the [ROW_NUMBER\(\)](#) function except that in that function, identical values will receive a different row number for each result.

Examples

The distinction between DENSE_RANK(), [RANK\(\)](#) and [ROW_NUMBER\(\)](#):

```

CREATE TABLE student(course VARCHAR(10), mark int, name varchar(10));

INSERT INTO student VALUES
('Maths', 60, 'Thulile'),
('Maths', 60, 'Pritha'),
('Maths', 70, 'Voitto'),
('Maths', 55, 'Chun'),
('Biology', 60, 'Bilal'),
('Biology', 70, 'Roger');

SELECT
    RANK() OVER (PARTITION BY course ORDER BY mark DESC) AS rank,
    DENSE_RANK() OVER (PARTITION BY course ORDER BY mark DESC) AS dense_rank,
    ROW_NUMBER() OVER (PARTITION BY course ORDER BY mark DESC) AS row_num,
    course, mark, name
FROM student ORDER BY course, mark DESC;

```

rank	dense_rank	row_num	course	mark	name
1	1	1	Biology	70	Roger
2	2	2	Biology	60	Bilal
1	1	1	Maths	70	Voitto
2	2	2	Maths	60	Thulile
2	2	3	Maths	60	Pritha
4	3	4	Maths	55	Chun

See Also

- [RANK\(\)](#)
- [ROW_NUMBER\(\)](#)
- [ORDER BY](#)

FIRST_VALUE

MariaDB starting with [10.2.2](#)

The FIRST_VALUE() function was first introduced with other [window functions](#) in MariaDB 10.2.

Syntax

```

FIRST_VALUE(expr) OVER (
    [ PARTITION BY partition_expression ]
    [ ORDER BY order_list ]
)

```

Contents

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

Description

`FIRST_VALUE` returns the first result from an ordered set, or `NULL` if no such result exists.

Examples

```

CREATE TABLE t1 (
    pk int primary key,
    a int,
    b int,
    c char(10),
    d decimal(10, 3),
    e real
);

INSERT INTO t1 VALUES
( 1, 0, 1, 'one', 0.1, 0.001),
( 2, 0, 2, 'two', 0.2, 0.002),
( 3, 0, 3, 'three', 0.3, 0.003),
( 4, 1, 2, 'three', 0.4, 0.004),
( 5, 1, 1, 'two', 0.5, 0.005),
( 6, 1, 1, 'one', 0.6, 0.006),
( 7, 2, NULL, 'n_one', 0.5, 0.007),
( 8, 2, 1, 'n_two', NULL, 0.008),
( 9, 2, 2, NULL, 0.7, 0.009),
(10, 2, 0, 'n_four', 0.8, 0.010),
(11, 2, 10, NULL, 0.9, NULL);

SELECT pk, FIRST_VALUE(pk) OVER (ORDER BY pk) AS first_asc,
       LAST_VALUE(pk) OVER (ORDER BY pk) AS last_asc,
       FIRST_VALUE(pk) OVER (ORDER BY pk DESC) AS first_desc,
       LAST_VALUE(pk) OVER (ORDER BY pk DESC) AS last_desc
FROM t1
ORDER BY pk DESC;

```

pk	first_asc	last_asc	first_desc	last_desc
11	1	11	11	11
10	1	10	11	10
9	1	9	11	9
8	1	8	11	8
7	1	7	11	7
6	1	6	11	6
5	1	5	11	5
4	1	4	11	4
3	1	3	11	3
2	1	2	11	2
1	1	1	11	1

```

CREATE OR REPLACE TABLE t1 (i int);
INSERT INTO t1 VALUES (1),(2),(3),(4),(5),(6),(7),(8),(9),(10);

SELECT i,
       FIRST_VALUE(i) OVER (ORDER BY i ROWS BETWEEN CURRENT ROW and 1 FOLLOWING) AS f_1f,
       LAST_VALUE(i) OVER (ORDER BY i ROWS BETWEEN CURRENT ROW and 1 FOLLOWING) AS l_1f,
       FIRST_VALUE(i) OVER (ORDER BY i ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS f_1p1f,
       LAST_VALUE(i) OVER (ORDER BY i ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS f_1p1f,
       FIRST_VALUE(i) OVER (ORDER BY i ROWS BETWEEN 2 PRECEDING AND 1 PRECEDING) AS f_2p1p,
       LAST_VALUE(i) OVER (ORDER BY i ROWS BETWEEN 2 PRECEDING AND 1 PRECEDING) AS f_2p1p,
       FIRST_VALUE(i) OVER (ORDER BY i ROWS BETWEEN 1 FOLLOWING AND 2 FOLLOWING) AS f_1f2f,
       LAST_VALUE(i) OVER (ORDER BY i ROWS BETWEEN 1 FOLLOWING AND 2 FOLLOWING) AS f_1f2f
  FROM t1;

```

i	f_1f	l_1f	f_1p1f	f_1p1f	f_2p1p	f_2p1p	f_1f2f	f_1f2f
1	1	2	1	2	NULL	NULL	2	3
2	2	3	1	3	1	1	3	4
3	3	4	2	4	1	2	4	5
4	4	5	3	5	2	3	5	6
5	5	6	4	6	3	4	6	7
6	6	7	5	7	4	5	7	8
7	7	8	6	8	5	6	8	9
8	8	9	7	9	6	7	9	10
9	9	10	8	10	7	8	10	10
10	10	10	9	10	8	9	NULL	NULL

See Also

- [LAST_VALUE](#)

LAG

MariaDB starting with 10.2.2

The LAG() function was first introduced with other [window functions](#) in MariaDB 10.2.

Syntax

```

LAG (expr[, offset]) OVER (
  [ PARTITION BY partition_expression ]
  < ORDER BY order_list >
)

```

Contents

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

Description

The *LAG* function accesses data from a previous row according to the *ORDER BY* clause without the need for a self-join. The specific row is determined by the *offset* (default 1), which specifies the number of rows behind the current row to use. An offset of 0 is the current row.

Examples

```

CREATE TABLE t1 (pk int primary key, a int, b int, c char(10), d decimal(10, 3), e real);

INSERT INTO t1 VALUES
( 1, 0, 1, 'one', 0.1, 0.001),
( 2, 0, 2, 'two', 0.2, 0.002),
( 3, 0, 3, 'three', 0.3, 0.003),
( 4, 1, 2, 'three', 0.4, 0.004),
( 5, 1, 1, 'two', 0.5, 0.005),
( 6, 1, 1, 'one', 0.6, 0.006),
( 7, 2, NULL, 'n_one', 0.5, 0.007),
( 8, 2, 1, 'n_two', NULL, 0.008),
( 9, 2, 2, NULL, 0.7, 0.009),
(10, 2, 0, 'n_four', 0.8, 0.010),
(11, 2, 10, NULL, 0.9, NULL);

SELECT pk, LAG(pk) OVER (ORDER BY pk) AS l1,
       LAG(pk,1) OVER (ORDER BY pk) AS l11,
       LAG(pk,2) OVER (ORDER BY pk) AS l12,
       LAG(pk,0) OVER (ORDER BY pk) AS l10,
       LAG(pk,-1) OVER (ORDER BY pk) AS lm1,
       LAG(pk,-2) OVER (ORDER BY pk) AS lm2
FROM t1;

```

pk	l1	l11	l12	l10	lm1	lm2
1	NULL	NULL	NULL	1	2	3
2	1	1	NULL	2	3	4
3	2	2	1	3	4	5
4	3	3	2	4	5	6
5	4	4	3	5	6	7
6	5	5	4	6	7	8
7	6	6	5	7	8	9
8	7	7	6	8	9	10
9	8	8	7	9	10	11
10	9	9	8	10	11	NULL
11	10	10	9	11	NULL	NULL

See Also

- [LEAD](#) - Window function to access a following row

MEDIAN

MariaDB starting with [10.3.3](#)

The `MEDIAN()` [window function](#) was first introduced with in [MariaDB 10.3.3](#).

Syntax

```

MEDIAN(median expression) OVER (
    [ PARTITION BY partition_expression ]
)

```

Contents

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

Description

`MEDIAN()` is a [window function](#) that returns the median value of a range of values.

It is a specific case of `PERCENTILE_CONT`, with an argument of 0.5 and the `ORDER BY` column the one in `MEDIAN`'s argument.

```

MEDIAN(<median-arg>) OVER ( [ PARTITION BY partition_expression ] )

```

Is equivalent to:

```
PERCENTILE_CONT(0.5) WITHIN
GROUP (ORDER BY <median-arg>) OVER ( [ PARTITION BY partition_expression ] )
```

Examples

```
CREATE TABLE book_rating (name CHAR(30), star_rating TINYINT);

INSERT INTO book_rating VALUES ('Lord of the Ladybirds', 5);
INSERT INTO book_rating VALUES ('Lord of the Ladybirds', 3);
INSERT INTO book_rating VALUES ('Lady of the Flies', 1);
INSERT INTO book_rating VALUES ('Lady of the Flies', 2);
INSERT INTO book_rating VALUES ('Lady of the Flies', 5);

SELECT name, median(star_rating) OVER (PARTITION BY name) FROM book_rating;
+-----+-----+
| name | median(star_rating) OVER (PARTITION BY name) |
+-----+-----+
| Lord of the Ladybirds | 4.0000000000 |
| Lord of the Ladybirds | 4.0000000000 |
| Lady of the Flies | 2.0000000000 |
| Lady of the Flies | 2.0000000000 |
| Lady of the Flies | 2.0000000000 |
+-----+-----+
```

See Also

- [PERCENTILE_CONT](#)

NTILE

MariaDB starting with 10.2

The NTILE() function was first introduced with [window functions](#) in MariaDB 10.2.0.

Syntax

```
NTILE (expr) OVER (
  [ PARTITION BY partition_expression ]
  [ ORDER BY order_list ]
)
```

Contents

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

Description

NTILE() is a [window function](#) that returns an integer indicating which group a given row falls into. The number of groups is specified in the argument (*expr*), starting at one. Ordered rows in the partition are divided into the specified number of groups with as equal a size as possible.

Examples

```

create table t1 (
    pk int primary key,
    a int,
    b int
);

insert into t1 values
    (11 , 0, 10),
    (12 , 0, 10),
    (13 , 1, 10),
    (14 , 1, 10),
    (18 , 2, 10),
    (15 , 2, 20),
    (16 , 2, 20),
    (17 , 2, 20),
    (19 , 4, 20),
    (20 , 4, 20);

select pk, a, b,
       ntile(1) over (order by pk)
  from t1;
+-----+-----+-----+
| pk | a   | b   | ntile(1) over (order by pk) |
+-----+-----+-----+
| 11 | 0   | 10  | 1   |
| 12 | 0   | 10  | 1   |
| 13 | 1   | 10  | 1   |
| 14 | 1   | 10  | 1   |
| 15 | 2   | 20  | 1   |
| 16 | 2   | 20  | 1   |
| 17 | 2   | 20  | 1   |
| 18 | 2   | 10  | 1   |
| 19 | 4   | 20  | 1   |
| 20 | 4   | 20  | 1   |
+-----+-----+-----+

select pk, a, b,
       ntile(4) over (order by pk)
  from t1;
+-----+-----+-----+
| pk | a   | b   | ntile(4) over (order by pk) |
+-----+-----+-----+
| 11 | 0   | 10  | 1   |
| 12 | 0   | 10  | 1   |
| 13 | 1   | 10  | 1   |
| 14 | 1   | 10  | 2   |
| 15 | 2   | 20  | 2   |
| 16 | 2   | 20  | 2   |
| 17 | 2   | 20  | 3   |
| 18 | 2   | 10  | 3   |
| 19 | 4   | 20  | 4   |
| 20 | 4   | 20  | 4   |
+-----+-----+-----+

```

PERCENT_RANK

MariaDB starting with 10.2

The PERCENT_RANK() function was first introduced with [window functions](#) in MariaDB 10.2.0.

Syntax

```

PERCENT_RANK() OVER (
    [ PARTITION BY partition_expression ]
    [ ORDER BY order_list ]
)

```

Contents

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

Description

PERCENT_RANK() is a [window function](#) that returns the relative percent rank of a given row. The following formula is used to calculate the percent rank:

```
(rank - 1) / (number of rows in the window or partition - 1)
```

Examples

```
create table t1 (
    pk int primary key,
    a int,
    b int
);

insert into t1 values
( 1 , 0, 10),
( 2 , 0, 10),
( 3 , 1, 10),
( 4 , 1, 10),
( 8 , 2, 10),
( 5 , 2, 20),
( 6 , 2, 20),
( 7 , 2, 20),
( 9 , 4, 20),
(10 , 4, 20);

select pk, a, b,
       rank() over (order by a) as rank,
       percent_rank() over (order by a) as pct_rank,
       cume_dist() over (order by a) as cume_dist
  from t1;
+-----+-----+-----+-----+-----+
| pk | a   | b   | rank | pct_rank      | cume_dist     |
+-----+-----+-----+-----+-----+
| 1  | 0   | 10  | 1   | 0.0000000000 | 0.2000000000 |
| 2  | 0   | 10  | 1   | 0.0000000000 | 0.2000000000 |
| 3  | 1   | 10  | 3   | 0.2222222222 | 0.4000000000 |
| 4  | 1   | 10  | 3   | 0.2222222222 | 0.4000000000 |
| 5  | 2   | 20  | 5   | 0.4444444444 | 0.8000000000 |
| 6  | 2   | 20  | 5   | 0.4444444444 | 0.8000000000 |
| 7  | 2   | 20  | 5   | 0.4444444444 | 0.8000000000 |
| 8  | 2   | 10  | 5   | 0.4444444444 | 0.8000000000 |
| 9  | 4   | 20  | 9   | 0.8888888889 | 1.0000000000 |
|10  | 4   | 20  | 9   | 0.8888888889 | 1.0000000000 |
+-----+-----+-----+-----+-----+

select pk, a, b,
       percent_rank() over (order by pk) as pct_rank,
       cume_dist() over (order by pk) as cume_dist
  from t1 order by pk;
+-----+-----+-----+-----+
| pk | a   | b   | pct_rank      | cume_dist     |
+-----+-----+-----+-----+
| 1  | 0   | 10  | 0.0000000000 | 0.1000000000 |
| 2  | 0   | 10  | 0.1111111111 | 0.2000000000 |
| 3  | 1   | 10  | 0.2222222222 | 0.3000000000 |
| 4  | 1   | 10  | 0.3333333333 | 0.4000000000 |
| 5  | 2   | 20  | 0.4444444444 | 0.5000000000 |
| 6  | 2   | 20  | 0.5555555556 | 0.6000000000 |
| 7  | 2   | 20  | 0.6666666667 | 0.7000000000 |
| 8  | 2   | 10  | 0.7777777778 | 0.8000000000 |
| 9  | 4   | 20  | 0.8888888889 | 0.9000000000 |
|10  | 4   | 20  | 1.0000000000 | 1.0000000000 |
+-----+-----+-----+-----+

select pk, a, b,
```

```

percent_rank() over (partition by a order by a) as pct_rank,
cume_dist() over (partition by a order by a) as cume_dist
from t1;
+-----+-----+
| pk | a   | b   | pct_rank      | cume_dist     |
+-----+-----+
| 1  | 0   | 10  | 0.0000000000 | 1.0000000000 |
| 2  | 0   | 10  | 0.0000000000 | 1.0000000000 |
| 3  | 1   | 10  | 0.0000000000 | 1.0000000000 |
| 4  | 1   | 10  | 0.0000000000 | 1.0000000000 |
| 5  | 2   | 20  | 0.0000000000 | 1.0000000000 |
| 6  | 2   | 20  | 0.0000000000 | 1.0000000000 |
| 7  | 2   | 20  | 0.0000000000 | 1.0000000000 |
| 8  | 2   | 10  | 0.0000000000 | 1.0000000000 |
| 9  | 4   | 20  | 0.0000000000 | 1.0000000000 |
| 10 | 4   | 20  | 0.0000000000 | 1.0000000000 |
+-----+-----+

```

See Also

- [CUME_DIST\(\)](#)

PERCENTILE_DISC

MariaDB starting with 10.3.3

The PERCENTILE_DISC() window function was first introduced with in MariaDB 10.3.3.

Syntax

Description

PERCENTILE_DISC() (standing for discrete percentile) is a [window function](#) which returns the first value in the set whose ordered position is the same or more than the specified fraction.

Essentially, the following process is followed to find the value to return:

- Get the number of rows in the partition.
- Walk through the partition, in order, until finding the the first row with [CUME_DIST\(\) >= function_argument](#).

Contents

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

Examples

```

CREATE TABLE book_rating (name CHAR(30), star_rating TINYINT);

INSERT INTO book_rating VALUES ('Lord of the Ladybirds', 5);
INSERT INTO book_rating VALUES ('Lord of the Ladybirds', 3);
INSERT INTO book_rating VALUES ('Lady of the Flies', 1);
INSERT INTO book_rating VALUES ('Lady of the Flies', 2);
INSERT INTO book_rating VALUES ('Lady of the Flies', 5);

SELECT name, PERCENTILE_DISC(0.5) WITHIN GROUP (ORDER BY star_rating)
OVER (PARTITION BY name) AS pc FROM book_rating;
+-----+
| name | pc |
+-----+
| Lord of the Ladybirds | 3 |
| Lord of the Ladybirds | 3 |
| Lady of the Flies | 2 |
| Lady of the Flies | 2 |
| Lady of the Flies | 2 |
+-----+
5 rows in set (0.000 sec)

SELECT name, PERCENTILE_DISC(0) WITHIN GROUP (ORDER BY star_rating)
OVER (PARTITION BY name) AS pc FROM book_rating;
+-----+
| name | pc |
+-----+
| Lord of the Ladybirds | 3 |
| Lord of the Ladybirds | 3 |
| Lady of the Flies | 1 |
| Lady of the Flies | 1 |
| Lady of the Flies | 1 |
+-----+
5 rows in set (0.000 sec)

SELECT name, PERCENTILE_DISC(1) WITHIN GROUP (ORDER BY star_rating)
OVER (PARTITION BY name) AS pc FROM book_rating;
+-----+
| name | pc |
+-----+
| Lord of the Ladybirds | 5 |
| Lord of the Ladybirds | 5 |
| Lady of the Flies | 5 |
| Lady of the Flies | 5 |
| Lady of the Flies | 5 |
+-----+
5 rows in set (0.000 sec)

SELECT name, PERCENTILE_DISC(0.6) WITHIN GROUP (ORDER BY star_rating)
OVER (PARTITION BY name) AS pc FROM book_rating;
+-----+
| name | pc |
+-----+
| Lord of the Ladybirds | 5 |
| Lord of the Ladybirds | 5 |
| Lady of the Flies | 2 |
| Lady of the Flies | 2 |
| Lady of the Flies | 2 |
+-----+

```

See Also

- [CUME_DIST\(\)](#)

PERCENTILE_CONT

MariaDB starting with 10.3.3

The PERCENTILE_CONT() window function was first introduced with in MariaDB 10.3.3.

Syntax

Description

`PERCENTILE_CONT()` (standing for continuous percentile) is a [window function](#) which returns a value which corresponds to the given fraction in the sort order. If required, it will interpolate between adjacent input items.

Essentially, the following process is followed to find the value to return:

- Get the number of rows in the partition, denoted by N
- $RN = p*(N-1)$, where p denotes the argument to the `PERCENTILE_CONT` function
- calculate the FRN(floor row number) and CRN(column row number for the group(FRN= floor(RN) and CRN = ceil(RN))
- look up rows FRN and CRN
- If (CRN = FRN = RN) then the result is (value of expression from row at RN)
- Otherwise the result is
 - $(CRN - RN) * (\text{value of expression for row at FRN}) +$
 - $(RN - FRN) * (\text{value of expression for row at CRN})$

The [MEDIAN function](#) is a specific case of `PERCENTILE_CONT` , equivalent to `PERCENTILE_CONT(0.5)` .

Contents

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

```

CREATE TABLE book_rating (name CHAR(30), star_rating TINYINT);

INSERT INTO book_rating VALUES ('Lord of the Ladybirds', 5);
INSERT INTO book_rating VALUES ('Lord of the Ladybirds', 3);
INSERT INTO book_rating VALUES ('Lady of the Flies', 1);
INSERT INTO book_rating VALUES ('Lady of the Flies', 2);
INSERT INTO book_rating VALUES ('Lady of the Flies', 5);

SELECT name, PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY star_rating)
OVER (PARTITION BY name) AS pc
FROM book_rating;
+-----+-----+
| name | pc   |
+-----+-----+
| Lord of the Ladybirds | 4.0000000000 |
| Lord of the Ladybirds | 4.0000000000 |
| Lady of the Flies    | 2.0000000000 |
| Lady of the Flies    | 2.0000000000 |
| Lady of the Flies    | 2.0000000000 |
+-----+-----+

SELECT name, PERCENTILE_CONT(1) WITHIN GROUP (ORDER BY star_rating)
OVER (PARTITION BY name) AS pc
FROM book_rating;
+-----+-----+
| name | pc   |
+-----+-----+
| Lord of the Ladybirds | 5.0000000000 |
| Lord of the Ladybirds | 5.0000000000 |
| Lady of the Flies    | 5.0000000000 |
| Lady of the Flies    | 5.0000000000 |
| Lady of the Flies    | 5.0000000000 |
+-----+-----+

SELECT name, PERCENTILE_CONT(0) WITHIN GROUP (ORDER BY star_rating)
OVER (PARTITION BY name) AS pc
FROM book_rating;
+-----+-----+
| name | pc   |
+-----+-----+
| Lord of the Ladybirds | 3.0000000000 |
| Lord of the Ladybirds | 3.0000000000 |
| Lady of the Flies    | 1.0000000000 |
| Lady of the Flies    | 1.0000000000 |
| Lady of the Flies    | 1.0000000000 |
+-----+-----+

SELECT name, PERCENTILE_CONT(0.6) WITHIN GROUP (ORDER BY star_rating)
OVER (PARTITION BY name) AS pc
FROM book_rating;
+-----+-----+
| name | pc   |
+-----+-----+
| Lord of the Ladybirds | 4.2000000000 |
| Lord of the Ladybirds | 4.2000000000 |
| Lady of the Flies    | 2.6000000000 |
| Lady of the Flies    | 2.6000000000 |
| Lady of the Flies    | 2.6000000000 |
+-----+-----+

```

See Also

- [MEDIAN\(\)](#) - a special case of `PERCENTILE_CONT` equivalent to `PERCENTILE_CONT(0.5)`

RANK

The RANK() function was first introduced with [window functions](#) in MariaDB 10.2.0.

Syntax

```
RANK() OVER (
    [ PARTITION BY partition_expression ]
    [ ORDER BY order_list ]
)
```

Contents

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

Description

RANK() is a [window function](#) that displays the number of a given row, starting at one and following the [ORDER BY](#) sequence of the window function, with identical values receiving the same result. It is similar to the [ROW_NUMBER\(\)](#) function except that in that function, identical values will receive a different row number for each result.

Examples

The distinction between [DENSE_RANK\(\)](#), RANK() and [ROW_NUMBER\(\)](#):

```
CREATE TABLE student(course VARCHAR(10), mark int, name varchar(10));

INSERT INTO student VALUES
('Maths', 60, 'Thulile'),
('Maths', 60, 'Pritha'),
('Maths', 70, 'Voitto'),
('Maths', 55, 'Chun'),
('Biology', 60, 'Bilal'),
('Biology', 70, 'Roger');

SELECT
    RANK() OVER (PARTITION BY course ORDER BY mark DESC) AS rank,
    DENSE_RANK() OVER (PARTITION BY course ORDER BY mark DESC) AS dense_rank,
    ROW_NUMBER() OVER (PARTITION BY course ORDER BY mark DESC) AS row_num,
    course, mark, name
FROM student ORDER BY course, mark DESC;
```

rank	dense_rank	row_num	course	mark	name
1	1	1	Biology	70	Roger
2	2	2	Biology	60	Bilal
1	1	1	Maths	70	Voitto
2	2	2	Maths	60	Thulile
2	2	3	Maths	60	Pritha
4	3	4	Maths	55	Chun

See Also

- [DENSE_RANK\(\)](#)
- [ROW_NUMBER\(\)](#)
- [ORDER BY](#)

ROW_NUMBER

MariaDB starting with 10.2

ROW_NUMBER() was first introduced with [window functions](#) in MariaDB 10.2.0.

Syntax

```
ROW_NUMBER() OVER (
    [ PARTITION BY partition_expression ]
    [ ORDER BY order_list ]
)
```

Contents

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

Description

`ROW_NUMBER()` is a [window function](#) that displays the number of a given row, starting at one and following the `ORDER BY` sequence of the window function, with identical values receiving different row numbers. It is similar to the [RANK\(\)](#) and [DENSE_RANK\(\)](#) functions except that in that function, identical values will receive the same rank for each result.

Examples

The distinction between [DENSE_RANK\(\)](#), [RANK\(\)](#) and [ROW_NUMBER\(\)](#):

```
CREATE TABLE student(course VARCHAR(10), mark int, name varchar(10));

INSERT INTO student VALUES
('Maths', 60, 'Thulile'),
('Maths', 60, 'Pritha'),
('Maths', 70, 'Voitto'),
('Maths', 55, 'Chun'),
('Biology', 60, 'Bilal'),
('Biology', 70, 'Roger');

SELECT
    RANK() OVER (PARTITION BY course ORDER BY mark DESC) AS rank,
    DENSE_RANK() OVER (PARTITION BY course ORDER BY mark DESC) AS dense_rank,
    ROW_NUMBER() OVER (PARTITION BY course ORDER BY mark DESC) AS row_num,
    course, mark, name
FROM student ORDER BY course, mark DESC;
```

rank	dense_rank	row_num	course	mark	name
1	1	1	Biology	70	Roger
2	2	2	Biology	60	Bilal
1	1	1	Maths	70	Voitto
2	2	2	Maths	60	Thulile
2	2	3	Maths	60	Pritha
4	3	4	Maths	55	Chun

See Also

- [RANK\(\)](#)
- [DENSE_RANK\(\)](#)
- [ORDER BY](#)

SPIDER_BG_DIRECT_SQL

Syntax

```
SPIDER_BG_DIRECT_SQL('sql', 'tmp_table_list', 'parameters')
```

Description

Executes the given SQL statement in the background on the remote server, as defined in the parameters listing. If the query returns a result-set, it stores the results in the given temporary table. When the given SQL statement executes successfully, this function returns the number of called UDF's. It returns 0 when the given SQL statement fails.

This function is a [UDF](#) installed with the [Spider](#) storage engine.

Examples

```
SELECT SPIDER_BG_DIRECT_SQL('SELECT * FROM example_table', '',
    'srv "node1", port "8607"' ) AS "Direct Query";
+-----+
| Direct Query |
+-----+
|      1 |
+-----+
```

Parameters

error_rw_mode

- **Description:** Returns empty results on network error.
 - 0 : Return error on getting network error.
 - 1 : Return 0 records on getting network error.
- **Default Table Value:** 0
- **DSN Parameter Name:** erwm

See also

- [SPIDER_DIRECT_SQL](#)

SPIDER_COPY_TABLES

Syntax

```
SPIDER_COPY_TABLES(spider_table_name,
    source_link_id, destination_link_id_list [,parameters])
```

Description

A [UDF](#) installed with the [Spider Storage Engine](#), this function copies table data from `source_link_id` to `destination_link_id_list`. The service does not need to be stopped in order to copy.

If the Spider table is partitioned, the name must be of the format `table_name#P#partition_name`. The partition name can be viewed in the `mysql.spider_tables` table, for example:

```
SELECT table_name FROM mysql.spider_tables;
+-----+
| table_name   |
+-----+
| spt_a#P#pt1 |
| spt_a#P#pt2 |
| spt_a#P#pt3 |
+-----+
```

Returns 1 if the data was copied successfully, or 0 if copying the data failed.

SPIDER_DIRECT_SQL

Syntax

```
SPIDER_DIRECT_SQL('sql', 'tmp_table_list', 'parameters')
```

Description

A UDF installed with the [Spider Storage Engine](#), this function is used to execute the SQL string `sql` on the remote server, as defined in `parameters`. If any resultsets are returned, they are stored in the `tmp_table_list`.

The function returns `1` if the SQL executes successfully, or `0` if it fails.

Examples

```
SELECT SPIDER_DIRECT_SQL('SELECT * FROM s', '', 'srv "node1", port "8607"');
+-----+
| SPIDER_DIRECT_SQL('SELECT * FROM s', '', 'srv "node1", port "8607"') |
+-----+
|                               1 |
+-----+
```

See also

- [SPIDER_BG_DIRECT_SQL](#)

COLUMN_ADD

Syntax

```
COLUMN_ADD(dyncol_blob, column_nr, value [as type], [column_nr, value [as type]]...);
COLUMN_ADD(dyncol_blob, column_name, value [as type], [column_name, value [as type]]...);
```

Description

Adds or updates [dynamic columns](#).

- `dyncol_blob` must be either a valid dynamic columns blob (for example, `COLUMN_CREATE` returns such blob), or an empty string.
- `column_name` specifies the name of the column to be added. If `dyncol_blob` already has a column with this name, it will be overwritten.
- `value` specifies the new value for the column. Passing a `NULL` value will cause the column to be deleted.
- `as type` is optional. See [#datatypes](#) section for a discussion about types.

The return value is a dynamic column blob after the modifications.

Examples

```
UPDATE t1 SET dyncol_blob=COLUMN_ADD(dyncol_blob, "column_name", "value") WHERE id=1;
```

Note: `COLUMN_ADD()` is a regular function (just like `CONCAT()`), hence, in order to update the value in the table you have to use the `UPDATE ... SET dynamic_col=COLUMN_ADD(dynamic_col,)` pattern.

COLUMN_CREATE

Syntax

```
COLUMN_CREATE(column_nr, value [as type], [column_nr, value [as type]]...);
COLUMN_CREATE(column_name, value [as type], [column_name, value [as type]]...);
```

Description

Returns a [dynamic columns](#) blob that stores the specified columns with values.

The return value is suitable for

- storing in a table
- further modification with other dynamic columns functions

The `as type` part allows one to specify the value type. In most cases, this is redundant because MariaDB will be able to deduce the type of the value. Explicit type specification may be needed when the type of the value is not apparent. For example, a literal '`'2012-12-01'`' has a CHAR type by default, one will need to specify '`'2012-12-01' AS DATE`' to have it stored as a date. See [Dynamic Columns:Datatypes](#) for further details.

Examples

```
INSERT INTO tbl SET dyncol_blob=COLUMN_CREATE("column_name", "value");
```

COLUMN_GET

Syntax

```
COLUMN_GET(dyncol_blob, column_nr as type);
COLUMN_GET(dyncol_blob, column_name as type);
```

Description

Gets the value of a [dynamic column](#) by its name. If no column with the given name exists, `NULL` will be returned.

`column_name as type` requires that one specify the datatype of the dynamic column they are reading.

This may seem counter-intuitive: why would one need to specify which datatype they're retrieving? Can't the dynamic columns system figure the datatype from the data being stored?

The answer is: SQL is a statically-typed language. The SQL interpreter needs to know the datatypes of all expressions before the query is run (for example, when one is using prepared statements and runs "`select COLUMN_GET(...)`", the prepared statement API requires the server to inform the client about the datatype of the column being read before the query is executed and the server can see what datatype the column actually has).

Lengths

If you're running queries like:

```
SELECT COLUMN_GET(blob, 'colname' as CHAR) ...
```

without specifying a maximum length (i.e. using `as CHAR`, not `as CHAR(n)`), MariaDB will report the maximum length of the resultset column to be 53,6870,911 for [MariaDB 5.3-10.0.0](#) and 16,777,216 for [MariaDB 10.0.1+](#). This may cause excessive memory usage in some client libraries, because they try to pre-allocate a buffer of maximum resultset width. To avoid this problem, use `CHAR(n)` whenever you're using `COLUMN_GET` in the select list.

See [Dynamic Columns:Datatypes](#) for more information about datatypes.

COLUMN_JSON

Syntax

```
COLUMN_JSON(dyncol_blob)
```

Description

Returns a JSON representation of data in `dyncol_blob`. Can also be used to display nested columns. See [dynamic columns](#) for more information.

Example

```
select item_name, COLUMN_JSON(dynamic_cols) from assets;
+-----+-----+
| item_name | COLUMN_JSON(dynamic_cols) |
+-----+-----+
| MariaDB T-shirt | {"size":"XL","color":"blue"} |
| Thinkpad Laptop | {"color":"black","warranty":"3 years"} |
+-----+-----+
```

Limitation: `COLUMN_JSON` will decode nested dynamic columns at a nesting level of not more than 10 levels deep. Dynamic columns that are nested deeper than 10 levels will be shown as `BINARY` string, without encoding.