Section 1. Start, Stop, and Restart MySQL Server

# **Start MySQL Server**

Start MySQL Server on Linux

On Linux, you can start the server with the following commands using service, init.d, and systemd.

Start MySQL Server using service

sudo service mysql start

Start MySQL Server using using init.d

sudo /etc/init.d/mysql start

Start MySQL Server using systemd

sudo systemctl start mysqld

## Stop MySQL Server on Linux

To stop MySQL Server on Linux, you use the following command:

/etc/init.d/mysqld stop

Some Linux distributions provide server command:

service mysqld stop

Or

service mysql stop

In this tutorial, you have learned how to stop MySQL Server on Windows and Linux.

* Was this tutorial helpful?
* [YesNo](https://www.mysqltutorial.org/mysql-adminsitration/stop-mysql/)

# **Restart MySQL Server**

You use the following command to restart the MySQL server On Linux:

service mysql restart

If the name is MySQL service is mysqld not mysql, you need to change the service name in the command as shown in the following command:

service mysqld restart

Or you can use the init.d to start the MySQL service:

/etc/init.d/mysqld restart

In this tutorial, you have learned how to restart the MySQL server on Windows and Linux.

# **How To Create User Accounts Using MySQL CREATE USER Statement**

## MySQL CREATE USER syntax

The CREATE USER statement creates a new user in the database server.

Here is the basic syntax of the CREATE USER statement:

**CREATE** **USER** [**IF** **NOT** **EXISTS**] account\_name

**IDENTIFIED** **BY** 'password';

In this syntax:

First, specify the account name after the CREATE USER keywords. The account name has two parts: username and hostname, separated by the @ sign:

username@hostname

The username is the name of the user. And hostname is the name of the host from which the user connects to the MySQL Server.

The hostname part of the account name is optional. If you omit it, the user can connect from any host.

An account name without a hostname is equivalent to:

username@%

If the username and hostname contains special characters such as space or -, you need to quote the username and hostname separately as follows:

'username'@'hostname'

Besides the single quote ('), you can use backticks ( `) or double quotation mark ( ").

Second, specify the password for the user after the IDENTIFIED BY keywords.

The IF NOT EXISTS option conditionally create a new user only if it does not exist.

Note that the CREATE USER statement creates a new user without any privileges. To grant privileges to the user, you use the GRANT statement.

## MySQL CREATE USER example

First, connect to the MySQL Server using the mysql client tool:

mysql -u root -p

Enter the password for the root account and press Enter:

Enter password: \*\*\*\*\*\*\*\*

Second, show users from the current MySQL Server:

mysql> **select** **user** **from** mysql.user;

Here is the output:

+------------------+

| user |

+------------------+

| mysql.infoschema |

| mysql.saession |

| mysql.sys |

| root |

+------------------+

Third, create a new user called bob:

mysql> **create** **user** bob@localhost **identified** **by** 'Secure1pass!';

Fourth, show all users again:

mysql> **select** **user** **from** mysql.user;

The output will be:

+------------------+

| user |

+------------------+

| bob |

| mysql.infoschema |

| mysql.session |

| mysql.sys |

| root |

+------------------+

5 rows **in** **set** (0.00 sec)

The user bob has been created successfully.

Fifth, open a second session and log in to the MySQL as bob:

mysql -u bob -p

Input the password for bob and press Enter:

Enter password: \*\*\*\*\*\*\*\*

Sixth, [show the databases](https://www.mysqltutorial.org/mysql-show-databases/) that bob has access:

mysql> **show** **databases**;

Here is the list of databases that bob can access:

+--------------------+

| Database |

+--------------------+

| information\_schema |

+--------------------+

1 row **in** **set** (0.01 sec)

Seventh, go to the session of the user root and [create a new database](https://www.mysqltutorial.org/mysql-create-database/) called bobdb:

mysql> **create** **database** bobdb;

Eight, [select the database](https://www.mysqltutorial.org/mysql-select-database/) bobdb:

mysql> **use** bobdb;

Ninth, [create a new table](https://www.mysqltutorial.org/mysql-create-table/) called lists:

mysql> **create** **table** lists(

-> **id** int auto\_increment primary **key**,

-> todo varchar(100) **not** null,

-> completed bool **default** false);

Notice that when you press Enter, instead of showing the mysql> command, the mysql tool shows the -> that accepts new clause of the statement.

Tenth, grant all privileges on the bobdb to bob:

mysql> **grant** **all** **privileges** **on** bobdb.\* **to** bob@localhost;

Note that you will learn how to grant privileges to a user in the [GRANT](https://www.mysqltutorial.org/mysql-grant.aspx) tutorial.

Eleventh, go to the bob’s session and show databases:

mysql> **show** **databases**;

Now, bob can see the bobdb:

+--------------------+

| Database |

+--------------------+

| bobdb |

| information\_schema |

+--------------------+

2 rows **in** **set** (0.00 sec)

Twelfth, select the database bobdb:

mysql> **use** bobdb;

Thirteenth, [show the tables](https://www.mysqltutorial.org/mysql-show-tables/) from the bobdb database:

mysql> **show** **tables**;

The user bob can see the lists table:

+-----------------+

| Tables\_in\_bobdb |

+-----------------+

| lists |

+-----------------+

1 row **in** **set** (0.00 sec)

Fourteenth, [insert a row](https://www.mysqltutorial.org/mysql-insert-statement.aspx) into the lists table:

mysql> **insert** **into** lists(todo) **values**('Learn MySQL');

Fifteenth, [query data](https://www.mysqltutorial.org/mysql-select-statement-query-data.aspx) from the lists table:

mysql> **select** \* **from** lists;

This is the output:

+----+-------------+-----------+

| id | todo | completed |

+----+-------------+-----------+

| 1 | Learn MySQL | 0 |

+----+-------------+-----------+

1 row **in** **set** (0.00 sec)

So the user bob can do everything in the bobdb database.

Finally, disconnect from the MySQL Server from both sessions:

mysql> exit

In this tutorial, you have learned how to use the MySQL CREATE USER to create a new user in the database server.

# **MySQL GRANT**

## Introduction to the MySQL GRANT statement

The [CREATE USER](https://www.mysqltutorial.org/mysql-create-user.aspx) statement creates one or more user accounts with no privileges. It means that the user accounts can log in to the MySQL Server, but cannot do anything such as [selecting a database](https://www.mysqltutorial.org/mysql-select-database/) and [querying data](https://www.mysqltutorial.org/mysql-select-statement-query-data.aspx) from tables.

To allow user accounts to work with database objects, you need to grant the user accounts privileges. And the GRANT statement grants a user account one or more privileges.

The following illustrates the basic syntax of the GRANT statement:

**GRANT** privilege [,privilege],..

**ON** privilege\_level

**TO** account\_name;

In this syntax:

First, specify one or more privileges after the GRANT keyword. If you grant multiple privileges, you need to separate privileges by commas.

This example grants the SELECT privilege on the table employees  in the sample database to the user acount bob@localhost:

**GRANT** **SELECT**

**ON** employees

**TO** bob@localhost;

The following example grants UPDATE, DELETE, and INSERT privileges on the table employees to bob@localhost:

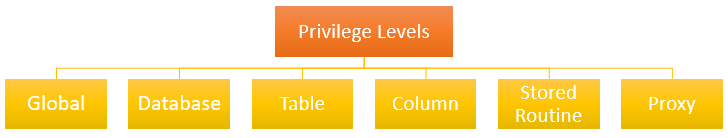
**GRANT** **INSERT**, **UPDATE**, **DELETE**

**ON** employees

**TO** bob@localhost;

Second, specify the privilege\_level that determines the level to which the privileges apply.

MySQL supports the following main privilege levels:



**Global privileges** apply to all databases in a MySQL Server. To assign global privileges, you use the \*.\* syntax, for example:

**GRANT** **SELECT**

**ON** \*.\*

**TO** bob@localhost;

The account user bob@localhost can query data from all tables in all database of the current MySQL Server.

**Database privileges** apply to all objects in a database. To assign database-level privileges, you use the ON database\_name.\* syntax, for example:

**GRANT** **INSERT**

**ON** classicmodels.\*

**TO** bob@localhost;

In this example, bob@localhost can insert data into all tables in the classicmodels database.

**Table privileges** apply to all columns in a table. To assign table-level privileges, you use the ON database\_name.table\_name syntax, for example:

**GRANT** **DELETE**

**ON** classicmodels.employees

**TO** bob@localhsot;

In this example, bob@localhost can delete rows from the table employees in the database classicmodels.

If you skip the database name, MySQL uses the default database or issues an error if there is no default database.

**Column privileges** apply to single columns in a table.  You must specify the column or columns for each privilege, for example:

**GRANT**

**SELECT** (employeeNumner,lastName, firstName,email),

**UPDATE**(lastName)

**ON** employees

**TO** bob@localhost;

In this example, bob@localhost can select data from four columns employeeNumber, lastName, firstName, and email and update only the lastName column in the employees table.

**Stored routine privileges** apply to stored procedures and stored functions, for example:

**GRANT** **EXECUTE**

**ON** **PROCEDURE** CheckCredit

**TO** bob@localhost;

In this example, bob@localhost can execute the stored procedure CheckCredit in the current database.

**Proxy user privileges** allow one user to be a proxy for another. The proxy user gets all privileges of the proxied user. For example:

**GRANT** PROXY

**ON** root

**TO** alice@localhost;

In this example, alice@localhost assumes all privileges of root.

Finally, specify the account name of the user that you want to grant privileges after the TO keyword.

Notice that in order to use the GRANT statement, you must have the GRANT OPTION privilege and the privileges that you are granting. If the [read\_only](http://dev.mysql.com/doc/refman/5.7/en/server-system-variables.html#sysvar_read_only) system variable is enabled, you need to have the [SUPER](http://dev.mysql.com/doc/refman/5.7/en/privileges-provided.html#priv_super) privilege to execute the GRANT statement.

## MySQL GRANT statement examples

Typically, you use the CREATE USER statement to create a new user account first and then use the GRANT statement to grant privileges to the user.

First, create a new user called super@localhost by using the following CREATE TABLE statement:

**CREATE** **USER** super@localhost

**IDENTIFIED** **BY** 'Secure1Pass!';

Second, show the privileges assigned to super@localhost user by using the SHOW GRANTS statement.

**SHOW** **GRANTS** **FOR** super@localhost;

MySQL Grant - No Privilege

The USAGE means that the super@localhost can log in the database but has no privilege.

Third, grant all privileges in all databases in the current database server to super@localhost:

**GRANT** **ALL**

**ON** classicmodels.\*

**TO** super@localhost;

Fourth, use the SHOW GRANTS statement again:

**SHOW** **GRANTS** **FOR** super@localhost;

MySQL Grant example

## Permissible privileges for GRANT statement

The following table illustrates all permissible privileges that you can use for the GRANT and [REVOKE](https://www.mysqltutorial.org/mysql-revoke.aspx) statement:

|  |  |  |
| --- | --- | --- |
| **Privilege** | **Meaning** | **Level** |
| **Global** | **Database** | **Table** | **Column** | **Stored Routine** | **Proxy** |
| ALL [PRIVILEGES] | Grant all privileges at specified access level except GRANT OPTION |  |  |  |  |  |  |
| ALTER | Allow user to use of [ALTER TABLE](https://www.mysqltutorial.org/mysql-alter-table.aspx)statement | X | X | X |  |  |  |
| ALTER ROUTINE | Allow user to alter and drop stored procedures or stored functions. | X | X |  |  | X |  |
| CREATE | Allow user to create databases and tables | X | X | X |  |  |  |
| CREATE ROUTINE | Allow user to create stored procedures and stored functions | X | X |  |  |  |  |
| CREATE TABLESPACE | Allow user to create, alter or drop tablespaces and log file groups | X |  |  |  |  |  |
| CREATE TEMPORARY TABLES | Allow user to create a temporary table by using CREATE TEMPORARY TABLE statement | X | X |  |  |  |  |
| CREATE USER | Allow user to use the CREATE USER, DROP USER, RENAME USER, and REVOKE ALL PRIVILEGES statements. | X |  |  |  |  |  |
| CREATE VIEW | Allow user to create or modify the view. | X | X | X |  |  |  |
| DELETE | Allow user to use DELETE statement | X | X | X |  |  |  |
| DROP | Allow user to drop database, table and view | X | X | X |  |  |  |
| EVENT | Enable use of events for the Event Scheduler. | X | X |  |  |  |  |
| EXECUTE | Allow user to execute stored routines | X | X | X |  |  |  |
| FILE | Allow user to read any file in the database directory. | X |  |  |  |  |  |
| GRANT OPTION | Allow user to have privileges to grant or revoke privileges from other accounts. | X | X | X |  | X | X |
| INDEX | Allow user to create or drop indexes. | X | X | X |  |  |  |
| INSERT | Allow user to use the INSERT statement | X | X | X | X |  |  |
| LOCK TABLES | Allow user to use LOCK TABLES on tables for which you have the SELECT privilege | X | X |  |  |  |  |
| PROCESS | Allow user to see all processes with SHOW PROCESSLIST statement. | X |  |  |  |  |  |
| PROXY | Enable user proxying. |  |  |  |  |  |  |
| REFERENCES | Allow user to create a foreign key | X | X | X | X |  |  |
| RELOAD | Allow user to use FLUSH statement | X |  |  |  |  |  |
| REPLICATION CLIENT | Allow user to query to see where master or slave servers are | X |  |  |  |  |  |
| REPLICATION SLAVE | Allow the user to use replicate slaves to read binary log events from the master. | X |  |  |  |  |  |
| SELECT | Allow user to use SELECT statement | X | X | X | X |  |  |
| SHOW DATABASES | Allow user to show all databases | X |  |  |  |  |  |
| SHOW VIEW | Allow user to use SHOW CREATE VIEW statement | X | X | X |  |  |  |
| SHUTDOWN | Allow user to use mysqladmin shutdown command | X |  |  |  |  |  |
| SUPER | Allow user to use other administrative operations such as CHANGE MASTER TO, KILL, PURGE BINARY LOGS, SET GLOBAL, and mysqladmin command | X |  |  |  |  |  |
| TRIGGER | Allow user to use TRIGGER operations. | X | X | X |  |  |  |
| UPDATE | Allow user to use the UPDATE statement | X | X | X | X |  |  |
| USAGE | Equivalent to “no privileges” |  |  |  |  |  |  |

# **MySQL SHOW GRANTS**

## Introduction to MySQL SHOW GRANTS statement

The MySQL SHOW GRANTS statement returns all privileges and [roles](https://www.mysqltutorial.org/mysql-roles/) granted to an account user or role.

Here is the basic syntax of the SHOW GRANTS statement:

**SHOW** **GRANTS**

[**FOR** {**user** | **role**}

[**USING** **role** [, **role**] ...]]

In this syntax:

* First, specify the name of the user account or role that you want to display the privileges that are previously granted to the user account or role after the FOR keyword. If you skip the FOR clause, the SHOW GRANTS returns the privileges of the current user.
* Second, use the USING clause to examine the privileges associated with roles for the user. The roles that you specify in the USING clause must previously granted to the user.

To execute the SHOW GRANTS statement, you need to have SELECT privilege for the mysql system database, except to show privileges and roles for the current user.

## MySQL SHOW GRANTS statement examples

Let’s take some examples of using the MySQL SHOW GRANTS statement.

### **A) Using MySQL SHOW GRANTS to display the privileges granted for the current user**

The following statement uses the SHOW GRANTS statement to display the privileges granted for the current user:

**SHOW** **GRANTS**;

It is equivalent to the following statement:

**SHOW** **GRANTS** **FOR** **CURRENT\_USER**;

and

**SHOW** **GRANTS** **FOR** **CURRENT\_USER**();

Both CURRENT\_USER and CURRENT\_USER() return the current user.

### **B) Using MySQL SHOW GRANTS to display the privileges granted for a user**

First, [create a new database](https://www.mysqltutorial.org/mysql-create-database/) named vehicles:

**CREATE** **DATABASE** vehicles;

Second, [select the database](https://www.mysqltutorial.org/mysql-select-database/) vehicles :

**USE** vehicles;

Third, [create a new table](https://www.mysqltutorial.org/mysql-create-table/) called cars in the vehicles database:

**CREATE** **TABLE** cars (

**id** INT AUTO\_INCREMENT,

make VARCHAR(100) **NOT** NULL,

**model** VARCHAR(100) **NOT** NULL,

PRIMARY **KEY** (**id**)

);

Fourth, [create a new user](https://www.mysqltutorial.org/mysql-create-user.aspx) called musk@localhost:

**CREATE** **USER** musk@localhost

**IDENTIFIED** **BY** 'Super1Pass!';

Fifth, show the default privileges granted to the user musk@localhost:

**SHOW** **GRANTS**

**FOR** musk@localhost;

show grants

The GRANT USAGE is the synonym of no privilege. By default, when a new user created, it has no privilege.

Sixth, grant all privileges on the vehicles database to the user musk@localhost:

**GRANT** **ALL**

**ON** vehicles.\*

**TO** musk@localhost;

Finally, show the privileges granted for the user musk@localhost:

**SHOW** **GRANTS**

**FOR** musk@localhost;

show grants for a user example

### **C) Using MySQL SHOW GRANTS to display the privileges granted for a role**

First, [create a new role](https://www.mysqltutorial.org/mysql-roles/) called writer@localhost:

**CREATE** **ROLE** writer@localhost;

Second, show privileges granted for the role writer@localhost:

**SHOW** **GRANTS**

**FOR** writer@localhost;

show grants for a role

Third, grant SELECT, INSERT, UPDATE, and DELETE privileges on the vehicles database to the writer@localhost:

**GRANT**

**SELECT**,

**INSERT**,

**UPDATE**,

**DELETE**

**ON** vehicles.\*

**TO** writer@localhost;

Fourth, show privileges granted for the role writer@localhost:

**SHOW** **GRANTS**

**FOR** writer@localhost;

show grants for a role example

### **D) Using MySQL SHOW GRANTS with USING clause example**

First, create a new account user called jame@localhost:

**CREATE** **USER** jame@localhost

**IDENTIFIED** **BY** 'Secret@Pass1';

Second, grant the EXECUTE privilege to the user jame@localhost:

**GRANT** **EXECUTE**

**ON** vehicles.\*

**TO** jame@localhost;

Third, grant the role writer@localhost to the user jame@localhost:

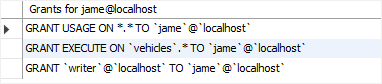
**GRANT** writer@localhost

**TO** jame@localhost;

Fourth, display the privileges granted for the user jame@localhost:

**SHOW** **GRANTS**

**FOR** jame@localhost;

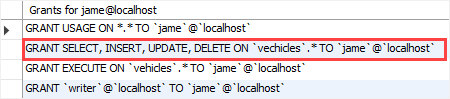


Finally, use the USING clause in the SHOW GRANTS statement to display privileges associated with the writer@localhost role:

**SHOW** **GRANTS**

**FOR** jame@localhost

**USING** writer@localhost;



In this tutorial, you have learned how to use the MySQL SHOW GRANTSstatement to display privileges granted for an account user or role.

# **MySQL REVOKE**

## Introduction to the MySQL REVOKE statement

The REVOKE statement revokes one or more privileges from a user account.

The REVOKE statement has several forms.

### **Revoke one or more privileges**

The following illustrates the basic syntax of the REVOKE statement that revokes one or more privileges from user accounts:

**REVOKE**

privilegee [,privilege]..

**ON** [object\_type] privilege\_level

**FROM** user1 [, user2] ..;

In this syntax:

* First, specify a list of comma-separated privileges that you want to revoke from a user account after the REVOKE keyword.
* Second, specify the object type and privilege level of the privileges after the ON keyword; check it out the [GRANT](https://www.mysqltutorial.org/mysql-grant.aspx) statement for more information on privilege level.
* Third, specify one or more user accounts from which you want to revoke the privileges in the FROM clause.

Note that to execute this form of REVOKE statement, you must have GRANT OPTION privilege or you must have the privileges that you are revoking.

### **Revoke all privileges**

To revoke all privileges from a user, you use the following form of the REVOKE ALL statement:

**REVOKE**

**ALL** [**PRIVILEGES**],

**GRANT** **OPTION**

**FROM** user1 [, user2];

To execute the REVOKE ALL statement, you must have the global CREATE USER privilege or the UPDATE privilege for the mysql system database.

### **Revoke Proxy**

To revoke a proxy user, you use the REVOKE PROXY command:

**REVOKE** PROXY

**ON** proxied\_user

**FROM** proxy\_user1[,proxy\_user1]...;

A proxy user is a valid user in MySQL who can impersonate another user, therefore, the proxy user has all privileges of the user that it impersonates.

It is a good practice to show privileges of the user accounts using the SHOW GRANTS statement before you revoke the privileges from the user:

**SHOW** **GRANTS** **FOR** **user**;

## MySQL REVOKE examples

Let’s take some examples of revoking privileges.

### **A) Using MySQL REVOKE to revoke some privileges from a user account example**

First, create a user account named rfc@localhost:

**CREATE** **USER** rfc@localhost

**IDENTIFIED** **BY** 'Secret1Pass!';

Second, grant rfc@localhost the SELECT, UPDATE, and INSERT privileges on the classicmodels database:

**GRANT** **SELECT**, **UPDATE**, **INSERT**

**ON** classicmodels.\*

**TO** rfc@localhost;

Third, display the granted privileges of the rfc@localhost user account:

**SHOW** **GRANTS** **FOR** rfc@localhost;

https://sp.mysqltutorial.org/wp-content/uploads/2019/09/MySQL-Revoke.png

Fourth, revoke the UPDATE and INSERT privileges from rfc@localhost:

**REVOKE** **INSERT**, **UPDATE**

**ON** classicmodels.\*

**FROM** rfc@localhost;

Fifth, display the privileges of rfc@localhost:

**SHOW** **GRANTS** **FOR** rfc@localhost;

MySQL Revoke privileges example

### **B) Using MySQL REVOKE to revoke all privileges from a user account example**

First, grant the EXECUTE privilege to the rfc@localhost:

**GRANT** **EXECUTE**

**ON** classicmodels.\*

**TO** rfc@localhost;

Second, show the currently granted privileges of rfc@localhost:

https://sp.mysqltutorial.org/wp-content/uploads/2019/09/MySQL-Revoke-All.png

Third, revoke all privileges of the rfc@localhost user account by using the REVOKE ALL statement:

**REVOKE** **ALL**, **GRANT** **OPTION**

**FROM** rfc@localhost;

Finally, show the privileges of the rfc@localhost to verify the revoke:

**SHOW** **GRANTS** **FOR** rfc@localhost;

MySQL Revoke All Example

The rfc@localhost has no privileges. Note that USAGE privilege means no privileges in MySQL.

### **C) Using MySQL REVOKE to revoke PROXY privilege example**

First, grant the PROXY privilege to rfc@localhost user account:

**GRANT** PROXY

**ON** root

**TO** rfc@localhost;

Second, show the granted privileges of rfc@localhost:

**SHOW** **GRANTS** **FOR** rfc@localhost;

MySQL Revoke Proxy

Third, revoke the PROXY privilege from the rfc@localhost:

**REVOKE** PROXY

**ON** root

**FROM** rfc@localhost;

Finally, show the granted privileges of rfc@lcoalhost to verify the revoke:

**SHOW** **GRANTS** **FOR** rfc@localhost;

https://sp.mysqltutorial.org/wp-content/uploads/2019/09/MySQL-Revoke-Proxy-Example.png

## When the MySQL REVOKE command takes effect

The effect of REVOKE statement depends on the privilege level:

**Global level**

The changes take effect when the user account connects to the MySQL Server in the subsequent sessions. The changes are not applied to all currently connected users.

**Database level**

The changes take effect after the next USE statement.

**Table and column levels**

The changes take effect on all subsequent queries.

In this tutorial, you’ve learned how to use the MySQL REVOKE statement to revoke privileges from user accounts.

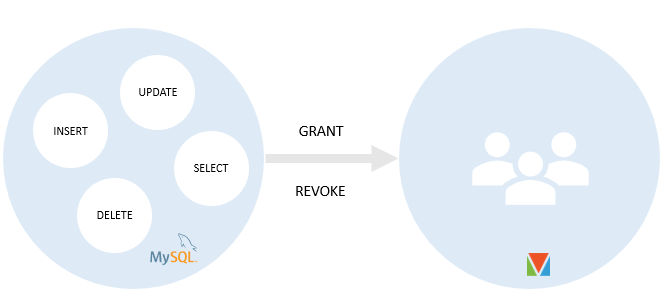
# **The Ultimate Guide To MySQL Roles By Examples**

## Introduction to MySQL roles

Typically, you have multiple users with the same set of privileges. Previously, the only way to [grant](https://www.mysqltutorial.org/mysql-grant.aspx)and [revoke](https://www.mysqltutorial.org/mysql-revoke.aspx)privileges to multiple users is to change the privileges of each user individually, which is time-consuming.

To make it easier, MySQL provided a new object called role. A role is a named collection of privileges.

Like user accounts, you can grant privileges to roles and revoke privileges from them.



If you want to grant the same set of privileges to multiple users, you follow these steps:

* First, create a new role.
* Second, grant privileges to the role.
* Third, grant the role to the users.

In case you want to change the privileges of the users, you need to change the privileges of the granted role only. The changes will take effect to all users to which the role granted.

## MySQL role example

First, [create a new database](https://www.mysqltutorial.org/mysql-create-table/) named CRM, which stands for customer relationship management.

**CREATE** **DATABASE** crm;

Next, use the crm database:

**USE** crm;

Then, create customer table inside the CRM database.

**CREATE** **TABLE** customers(

**id** INT PRIMARY **KEY** AUTO\_INCREMENT,

first\_name VARCHAR(255) **NOT** NULL,

last\_name VARCHAR(255) **NOT** NULL,

phone VARCHAR(15) **NOT** NULL,

email VARCHAR(255)

);

After that, [insert data](https://www.mysqltutorial.org/mysql-insert-statement.aspx) into the customers table.

**INSERT** **INTO** customers(first\_name,last\_name,phone,email)

**VALUES**('John','Doe','(408)-987-7654','john.doe@mysqltutorial.org'),

('Lily','Bush','(408)-987-7985','lily.bush@mysqltutorial.org');

Finally, verify the insert by using the following [SELECT](https://www.mysqltutorial.org/mysql-select-statement-query-data.aspx) statement:

**SELECT** \* **FROM** customers;

mysql role - sample table

## Creating roles

Suppose you develop an application that uses the CRM database. To interact with the CRM database, you need to create accounts for developers who need full access to the database. In addition, you need to create accounts for users who need only read access and others who need both read/write access.

To avoid granting privileges to each user account individually, you create a set of roles and grant the appropriate roles to each user account.

To create new roles, you use CREATE ROLE statement:

**CREATE** **ROLE**

crm\_dev,

crm\_read,

crm\_write;

The role name is similar to the user account that consists of two parts: the name and host:

role\_name@host\_name

If you omit the host part, it defaults to ‘%’ that means any host.

## Granting privileges to roles

To grant privileges to a role, you use GRANT statement. The following statement grants all privileges to crm\_dev role:

**GRANT** **ALL**

**ON** crm.\*

**TO** crm\_dev;

The following statement grants SELECT privilege to crm\_read role:

**GRANT** **SELECT**

**ON** crm.\*

**TO** crm\_read;

The following statement grants INSERT, UPDATE, and DELETE privileges to crm\_write role:

**GRANT** **INSERT**, **UPDATE**, **DELETE**

**ON** crm.\*

**TO** crm\_write;

## Assigning roles to user accounts

Suppose you need one user account as the developer, one user account that can have read-only access and two user accounts that can have read/write access.

To create new users, you use [CREATE USER](https://www.mysqltutorial.org/mysql-create-user.aspx) statements as follows:

*-- developer user*

**CREATE** **USER** crm\_dev1@localhost **IDENTIFIED** **BY** 'Secure$1782';

*-- read access user*

**CREATE** **USER** crm\_read1@localhost **IDENTIFIED** **BY** 'Secure$5432';

*-- read/write users*

**CREATE** **USER** crm\_write1@localhost **IDENTIFIED** **BY** 'Secure$9075';

**CREATE** **USER** crm\_write2@localhost **IDENTIFIED** **BY** 'Secure$3452';

To assign roles to users, you use GRANT statement.

The following statement grants the crm\_rev role to the user account crm\_dev1@localhost:

**GRANT** crm\_dev

**TO** crm\_dev1@localhost;

The following statement grants the crm\_read role to the user account crm\_read1@localhost:

**GRANT** crm\_read

**TO** crm\_read1@localhost;

The following statement grants the crm\_read and crm\_write roles to the user accounts crm\_write1@localhost and crm\_write2@localhost:

**GRANT** crm\_read,

crm\_write

**TO** crm\_write1@localhost,

crm\_write2@localhost;

To verify the role assignments, you use the [SHOW GRANTS](https://www.mysqltutorial.org/mysql-adminsitration/mysql-show-grants/) statement as the following example:

**SHOW** **GRANTS** **FOR** crm\_dev1@localhost;

The statement returned the following result set:

https://sp.mysqltutorial.org/wp-content/uploads/2019/09/mysql-role-show-grants.png

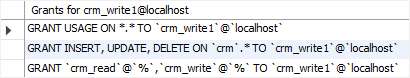
As you can see, it just returned granted roles. To show the privileges that roles represent, you use the USING clause with the name of the granted roles as follows:

**SHOW** **GRANTS**

**FOR** crm\_write1@localhost

**USING** crm\_write;

The statement returns the following output:



## Setting default roles

Now if you connect to the MySQL using the crm\_read1 user account and try to access the CRM database:

>mysql -u crm\_read1 -p

Enter password: \*\*\*\*\*\*\*\*\*\*\*

mysql>**USE** crm;

The statement issued the following error message:

ERROR 1044 (42000): Access denied for user 'crm\_read1'@'localhost' to database 'crm'

This is because when you granted roles to a user account, it did not automatically make the roles to become active when the user account connects to the database server.

If you invoke the CURRENT\_ROLE() function, it will return NONE, meaning no active roles.

**SELECT** current\_role();

Here is the output:

+*----------------+*

| current\_role() |

+*----------------+*

| NONE |

+*----------------+*

1 row in **set** (0.00 sec)

To specify which roles should be active each time a user account connects to the database server, you use the SET DEFAULT ROLE statement.

The following statement sets the default for the crm\_read1@localhost account all its assigned roles.

**SET** **DEFAULT** **ROLE** **ALL** **TO** crm\_read1@localhost;

Now, if you connect to the MySQL database server using the crm\_read1 user account and invoke the CURRENT\_ROLE() function:

>mysql -u crm\_read1 -p

Enter password: \*\*\*\*\*\*\*\*\*\*\*

mysql> **select** current\_role();

You will see the default roles for crm\_read1 user account.

+*----------------+*

| current\_role() |

+*----------------+*

| `crm\_read`@`%` |

+*----------------+*

1 row in **set** (0.00 sec)

You can test the privileges of crm\_read account by switching the current database to CRM, executing a SELECT statement and a DELETE statement as follows:

mysql> **use** crm;

Database changed

mysql> **SELECT** **COUNT**(\*) **FROM** customers;

+*----------+*

| COUNT(\*) |

+*----------+*

| 2 |

+*----------+*

1 row in **set** (0.00 sec)

mysql> **DELETE** **FROM** customers;

ERROR 1142 (42000): **DELETE** command denied **to** **user** 'crm\_read1'@'localhost' **for** **table** 'customers'

It worked as expected. When we issued the DELETE statement, MySQL issued an error because crm\_read1 user account has only read access.

## Setting active roles

A user account can modify the current user’s effective privileges within the current session by specifying which granted role are active.

The following statement set the active role to NONE, meaning no active role.

**SET** **ROLE** **NONE**;

To set active roles to all granted role, you use:

**SET** **ROLE** **ALL**;

To set active roles to default roles that set by the SET DEFAULT ROLE statement, you use:

**SET** **ROLE** **DEFAULT**;

To set active named roles, you use:

**SET** **ROLE**

granted\_role\_1

[,granted\_role\_2, ...]

## Revoking privileges from roles

To revoke privileges from a specific role, you use the [REVOKE](https://www.mysqltutorial.org/mysql-revoke.aspx) statement. The REVOKE statement takes effect not only the role but also any account granted the role.

For example, to temporarily make all read/write users read-only, you change the crm\_write role as follows:

**REVOKE** **INSERT**, **UPDATE**, **DELETE**

**ON** crm.\*

**FROM** crm\_write;

To restore the privileges, you need to re-grant them as follows:

**GRANT** **INSERT**, **UPDATE**, **DELETE**

**ON** crm.\*

**FOR** crm\_write;

## Removing roles

To delete one or more roles, you use the DROP ROLE statement as follows:

**DROP** **ROLE** role\_name[, role\_name, ...];

Like the REVOKE statement, the DROP ROLE statement revokes roles from every user account to which they were granted.

For example, to remove the crm\_read, crm\_write roles, you use the following statement:

**DROP** **ROLE** crm\_read, crm\_write;

## Copying privileges from a user account to another

MySQL treats user accounts like roles, therefore, you can grant a user account to another user account like granting a role to that user account. This allows you to copy privileges from a user to another user.

Suppose you need another developer account for the CRM database:

First, create the new user account:

**CREATE** **USER** crm\_dev2@localhost

**IDENTIFIED** **BY** 'Secure$6275';

Second, copy privileges from the crm\_dev1 user account to crm\_dev2 user account as follows:

**GRANT** crm\_dev1@localhost

**TO** crm\_dev2@localhost;

# **How To Remove User Accounts Using MySQL DROP USER Statement**

## Introduction to MySQL DROP USER statement

To remove a user account from the MySQL Server, you use the DROP USER statement as follows:

**DROP** **USER** account\_name;



In this syntax, you specify the name of the user account that you want to remove after the DROP USER keywords.

If you want to remove multiple user accounts at once, you specify a list of comma-separated user accounts in the DROP USER clause:

**DROP** **USER** account\_name [,account\_name2]...

If you remove a user account that doesn’t exist, MySQL will issue an error.

In MySQL 5.7.8+, you can use the IF EXISTS clause to conditionally drop a user only if it exists:

**DROP** **USER** [**IF** **EXISTS**] account\_name [,account\_name2]...;

Besides removing the user account, the DROP USER statement also removes all privileges of the user from all grant tables.

## MySQL DROP USER examples

Let’s take some examples of dropping users.

### **A) Using MySQL DROP USER statement to drop a user example**

First, connect to the MySQL Server using the root account:

mysql -u root -p

Type the password for the root user and press Enter:

Enter password: \*\*\*\*\*\*\*\*

Second, create four account users  accounts api@localhost, remote, dbadmin@localhost and alice@localhost:

mysql> **create** **user** api@localhost, remote, dbadmin@localhost, alice@localhost **identified** **by** 'Secure1Pass!';

Third, show users from the MySQL Server:

mysql> **select** **user**, host **from** mysql.user;

Here is the current user list:

+*------------------+-----------+*

| user | host |

+*------------------+-----------+*

| remote | % |

| alice | localhost |

| api | localhost |

| bob | localhost |

| dbadmin | localhost |

| mysql.infoschema | localhost |

| mysql.session | localhost |

| mysql.sys | localhost |

| root | localhost |

+*------------------+-----------+*

9 rows in **set** (0.00 sec)

Fourth, drop the user dbadmin@localhost by using the DROP USER statement:

mysql> **drop** **user** dbadmin@localhost

Fifth, show all users again:

mysql> **select** **user**, host **from** mysql.user;

+*------------------+-----------+*

| user | host |

+*------------------+-----------+*

| remote | % |

| alice | localhost |

| api | localhost |

| bob | localhost |

| mysql.infoschema | localhost |

| mysql.session | localhost |

| mysql.sys | localhost |

| root | localhost |

+*------------------+-----------+*

8 rows in **set** (0.00 sec)

The user account dbadmin@localhosthas been removed successfully.

### **B) Using MySQL DROP USER to drop multiple user accounts at once**

First, remove two user accounts api@localhost and remote using the following statement:

mysql> **drop** **user** api@localhost, remote;

Second, show users from the current database:

mysql> **select** **user**, host **from** mysql.user;

Here is the output:

+*------------------+-----------+*

| user | host |

+*------------------+-----------+*

| alice | localhost |

| bob | localhost |

| mysql.infoschema | localhost |

| mysql.session | localhost |

| mysql.sys | localhost |

| root | localhost |

+*------------------+-----------+*

6 rows in **set** (0.00 sec)

### **C) Using MySQL DROP USER to drop a connected user**

First, [create a new database](https://www.mysqltutorial.org/mysql-create-database/) called people:

mysql> **create** **database** people;

Second, select the people database:

mysql> **use** people;

Third, [create a new table](https://www.mysqltutorial.org/mysql-create-table/) persons in the people database:

mysql> **create** **table** persons(

-> **id** int auto\_increment,

-> firstName varchar(100) **not** null,

-> lastName varchar(100) **not** null,

-> primary **key**(**id**));

Fourth, grant all privileges on the people database to the account user alice:

mysql> **grant** **all** **privileges** **on** people.\* **to** alice@localhost;

Fifth, launch another session and connect to the database using the user alice@localhost:

mysql -u alice -p

Type the password for the user account alice and press the Enter key:

Enter password: \*\*\*\*\*\*\*\*\*\*\*\*

Sixth, select the people database:

mysql> **use** people;

Seventh, insert a row into the persons table:

mysql> **insert** **into** persons(firstname, lastname) **values**('John','Doe');

Suppose that you want to drop the user alice@localhost.

However, the user account alice@localhost is still connected to MySQL Server. If you drop a currently connected user, the user can operate as normal until the next login. In this case, you should inform the user first.

If you cannot do so, you can kill user sessions first before dropping the user account.

Eighth, use the [SHOW PROCESSLIST](https://www.mysqltutorial.org/mysql-show-processlist/) statement from the root’s session to find the id of the connection:

+----+-----------------+-----------------+--------+---------+-------+------------------------+------------------+

| Id | User | Host | db | Command | Time | State | Info |

+----+-----------------+-----------------+--------+---------+-------+------------------------+------------------+

| 4 | event\_scheduler | localhost | **NULL** | Daemon | 31803 | Waiting on **empty** queue | **NULL** |

| 20 | root | localhost:63851 | **NULL** | Query | 0 | starting | show processlist |

| 21 | alice | localhost:64060 | people | Sleep | 14 | | **NULL** |

+----+-----------------+-----------------+--------+---------+-------+------------------------+------------------+

As you see, the user account alice@localhost has the connection id id 21.

Ninth, terminate process 21 by using the KILL statement:

**KILL** 21;

The user account alicereceived an error message if he/she issue any query:

ERROR 2013 (HY000): Lost connection to MySQL server during query

Finally, execute the DROP USER statement to remove the user account alice@localhost.

**DROP** **USER** alice@localhost;

In this tutorial, you have learned how to use the MySQL DROP USER statement to remove one or more user accounts.

# **MySQL RENAME USER**

## Introduction to MySQL RENAME USER statement

The RENAME USER statement renames one or more existing accounts. Here is the basic syntax of the RENAME USER statement:

RENAME USER old\_user1

TO new\_user;

In this syntax:

* First, specify the name of the existing user that you want to rename.
* Second, specify the new user name after the TO keyword. The new name must not exist or you will get an error.

If you want to rename multiple user accounts at once, you use the following syntax:

RENAME USER

old\_user1 TO new\_user1,

old\_user2 TO new\_user2,

...

The RENAME USER transfers all privileges of the old users to the new users. However, it does not drop or invalidate database objects that are dependent on old users.

For example, assuming that you have a stored procedure whose the DEFINER attribute specifies the old user. And this stored procedure executes in the definer security context. If you rename the old user, then you will get an error if you execute the stored procedure.

## MySQL RENAME USER examples

Let’s take some examples of using the MySQL RENAME USER statement.

### **A) Using MySQL RENAME USER to rename one user example**

First, create a new user called john@localhost:

CREATE USER john@localhost

IDENTIFIED BY 'Super!pass1';

Second, use the RENAME USER to rename user john@localhost:

RENAME USER john@localhost

TO doe@localhost;

Third, [query data](https://www.mysqltutorial.org/mysql-select-statement-query-data.aspx) from the mysql.user to verify the rename:

SELECT host, user

FROM mysql.user

WHERE user = 'doe' and host = 'localhost';

### **B) Using MySQL RENAME USER to rename multiple user accounts example**

First, [create two user accounts](https://www.mysqltutorial.org/mysql-create-user.aspx) jill@localhost and hill@localhost

CREATE USER jill@localhost

IDENTIFIED BY 'Super!pass1';

CREATE USER hill@localhost

IDENTIFIED BY 'Super!pass1';

Second, use the RENAME USER statement to rename these two users:

RENAME USER

jill@localhost TO jin@localhost,

hill@localhost TO hank@localhost;

Third, query data from the mysql.user to verify the rename:

SELECT host, user

FROM mysql.user

WHERE user IN ('jin','hank');

MySQL RENAME USER - rename multiple users example

### **C) Using MySQL RENAME USER to rename a user account associated with a stored procedure**

First, create a new user account called fx:

CREATE USER fx

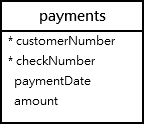
IDENTIFIED BY 'Super!pass2';

Second, grant all privileges to fx:

GRANT ALL ON \*.\*

TO fx;

Third, login as fx and create a procedure that returns all rows from the payments table in the [sample database](https://www.mysqltutorial.org/mysql-sample-database.aspx):



DELIMITER $$

CREATE DEFINER=fx PROCEDURE GetPayments()

SQL SECURITY DEFINER

BEGIN

SELECT \* FROM payments;

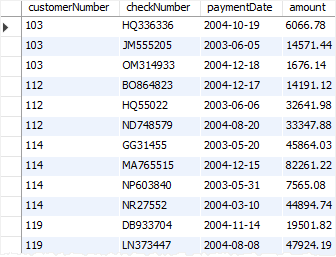
END$$

DELIMITER ;

The definer of the procedure is fx and SQL SECURITY specifies that the procedure will execute with the privileges of the definer.

Fourth, login as root and call the GetPayments() procedure:

CALL GetPayments();



Fifth, rename the user account fx to fc:

RENAME USER fx TO fc;

Sixth, call the GetPayments() procedure again:

CALL GetPayments();

MySQL issued the following message:

Error Code: 1449. The user specified as a definer ('fx'@'%') does not exist

To fix this issue, you need to manually change the definer in the stored procedure GetPayments() and save it.

In this tutorial, you have learned how to use the MySQL RENAME USER statement to rename one or more existing user accounts.

# **3 Best Ways to Change MySQL User Password By Examples**

Before changing the password of a [MySQL user](https://www.mysqltutorial.org/mysql-create-user.aspx) account, you should consider the following important questions:

* Which user account do you want to change the password?
* What application is using the user account whose password is being changed? If you change the password without changing the connection string of the application that is using the user account, the application will not be able to connect to the database server.

After having these questions answered. You can start changing the user account’s password.

MySQL provides various statements that you can use to change the password of a user including the [UPDATE](https://www.mysqltutorial.org/mysql-update-data.aspx), SET PASSWORD, and GRANT USAGE statements.

## Change MySQL user password using UPDATE statement

The first way to change the password is to use the UPDATE statement to update the user table of the mysql database.

After executing the UPDATE statement, you also need to execute the FLUSH PRIVILEGES  statement to reload privileges from the grant table in the mysql database.

Suppose you want to change the password for the dbadmin  user that connects from the localhost to dolphin, you need to execute the following statements:

**USE** mysql;

**UPDATE** **user**

**SET** **password** = **PASSWORD**('dolphin')

**WHERE** **user** = 'dbadmin' **AND**

host = 'localhost';

**FLUSH** **PRIVILEGES**;

Note that from MySQL 5.7.6, the user table uses the authentication\_string column only to store the password. In addition, it removed the password column.

Therefore if you use MySQL 5.7.6+, you must use the authentication\_string column in the UPDATE statement instead:

**USE** mysql;

**UPDATE** **user**

**SET** authentication\_string = **PASSWORD**('dolphin')

**WHERE** **user** = 'dbadmin' **AND**

host = 'localhost';

**FLUSH** **PRIVILEGES**;

Notice that the PASSWORD() function computes the hash value from a plain text.

## Change MySQL user password using the SET PASSWORD statement

The second way to change the password is by using the SET PASSWORD  statement.

You use the user account in user@host format to update the password. If you need to change the password for other accounts, your account needs to have at least UPDATE privilege.

By using the SET PASSOWORD statement, you don’t need to execute the FLUSH PRIVILEGES statement to reload privileges from grant tables.

The following statement changes the password of  dbadmin user account using the SET PASSWORD  statement.

**SET** **PASSWORD** **FOR** 'dbadmin'@'localhost' = **PASSWORD**('bigshark');

Notice that from version 5.7.6, MySQL depreciated this syntax and may remove it in the future releases. Instead, it uses the plaintext password as follows:

**SET** **PASSWORD** **FOR** 'dbadmin'@'localhost' = bigshark;

## Change MySQL user password using ALTER USER statement

The third way to change the password for a user account is to use the ALTER USER statement with the IDENTIFIED BY  clause.

The following ALTER USER statement changes the password of the dbadmin user to littlewhale:

**ALTER** **USER** dbadmin@localhost **IDENTIFIED** **BY** 'littlewhale';

In case you want to reset the password of the MySQL root account, you need to force the MySQL database server to stop and restart without using grant table validation.

# **How To Lock User Accounts in MySQL**

To lock a user account, you can use the CREATE USER .. ACCOUNT LOCK statement:

**CREATE** **USER** account\_name

**IDENTIFIED** **BY** 'password'

**ACCOUNT** **LOCK**;

If you omit the ACCOUNT LOCK clause, the [CREATE USER](https://www.mysqltutorial.org/mysql-create-user.aspx) statement [creates a new user](https://www.mysqltutorial.org/mysql-create-user.aspx) in an unlocked state by default.

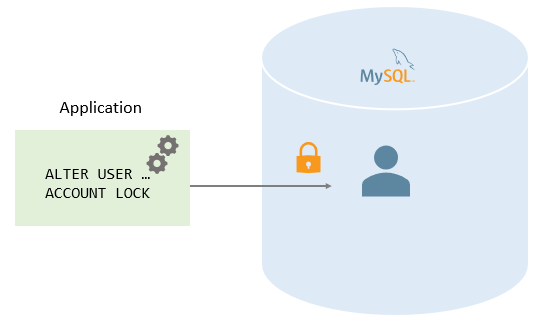
Or you can use the ALTER USER .. ACCOUNT LOCK statement to lock an existing user account:

**ALTER** **USER** account\_name

**IDENTIFIED** **BY** 'password'

**ACCOUNT** **LOCK**;

MySQL records the account locking state in the account\_locked column of the mysql.user system table. The values Y and N for locked and unlocked respectively.



Note that to unlock user accounts, you use the ALTER USER .. ACCOUNT UNLOCK statement.

## Locking user accounts examples

Let’s take some examples of locking user accounts.

### **A) Using ACCOUNT LOCK to lock a new account example**

First, [create a new user account](https://www.mysqltutorial.org/mysql-create-user.aspx) david@localhost in the locked state:

**CREATE** **USER** david@localhost

**IDENTIFIED** **BY** 'Secret!Pass1'

**ACCOUNT** **LOCK**;

Second, show the user account and its status:

**SELECT**

**user**, host, account\_locked

**FROM**

mysql.user

**WHERE**

**user** = 'david' **AND**

host='localhost';

https://sp.mysqltutorial.org/wp-content/uploads/2019/09/MySQL-Lock-Account-example.png

The account\_locked column in the mysql.user table indicates whether an account is locked or not, Y for locked and N for not locked.

Third, use the user account david to connect to the MySQL Server, you will receive an error:

mysql -u david -p

Enter password: \*\*\*\*\*\*\*\*\*\*\*\*

Here is the error message:

ERROR 3118 (HY000): Access denied for user 'david'@'localhost'. Account is locked.

### **B) Using ACCOUNT LOCK to lock an existing user account example**

First, create a user account dolphin@localhost:

**CREATE** **USER** dolphin@localhost

**IDENTIFIED** **BY** 'Secret!pass1';

Second, use the user account dolphin@localhost to login to the MySQL server:

mysql -u dolphin -p

Enter password: \*\*\*\*\*\*\*\*

The user account dolphin@localhost can log in successfully.

Third, use the ALTER TABLE LOCK ACCOUNT statement to lock the user account dolphin@localhost:

**ALTER** **USER** dolphin@localhost

**ACCOUNT** **LOCK**;

Fourth, show the user status:

**SELECT**

**user**, host, account\_locked

**FROM**

mysql.user

**WHERE**

**user** = 'dolphin';

https://sp.mysqltutorial.org/wp-content/uploads/2019/09/MySQL-Lock-User-Account-example.png

The user account dolphin was locked as expected.

The Locked\_connects variables status variable shows the number of attempts to connect to MySQL Server using a locked account.

When a locked account attempts to log in, MySQL increases the Locked\_connects status variable by 1.

To display the number of attempts of the locked accounts, you use this command:

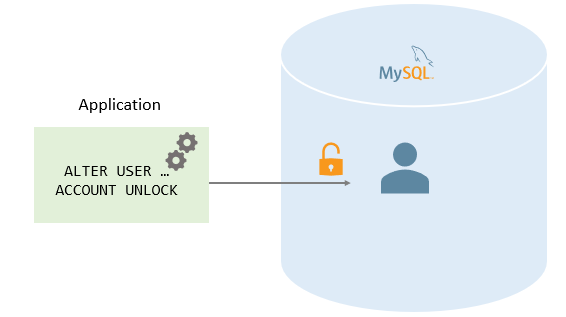
**SHOW** **GLOBAL** **STATUS**

**LIKE** 'Locked\_connects';

# **How To Unlock User Accounts in MySQL Server**

When you create a new user using the [CREATE USER](https://www.mysqltutorial.org/mysql-create-user.aspx) statement with the ACCOUNT LOCK clause, the new user has a locked state.

Similarly, if you use the ALTER USER ACCOUNT LOCK statement to change a user account, the user account is also locked.



To unlock a user account, you use the ALTER USER ACCOUNT LOCK statement:

**ALTER** **USER** [**IF** **EXISTS**] account\_name

**ACCOUNT** **UNLOCK**;

In this syntax:

* First, specify the name of the user account that you want to unlock after the ALTER USER keywords.
* Second, include the ACCOUNT UNLOCK clause after the account name.
* Third, use the IF EXISTS option to conditionally unlock the account if it exists only.

To unlock multiple user accounts at the same time, you use the following syntax:

**ALTER** **USER** [**IF** **EXISTS**]

account\_name1

[, account\_name2, ...]

**ACCOUNT** **UNLOCK**;

In this syntax, you specify a list of comma-separated names of the user accounts that you want to unlock after the ALTER USER keywords.

## Unlocking user accounts example

First, create a user named brad@localhost in a locked state:

**CREATE** **USER** brad@localhost

**IDENTIFIED** **BY** 'Secret!pass1'

**ACCOUNT** **LOCK**;

Second, show the status of the user account:

**SELECT**

**user**,

host,

account\_locked

**FROM**

mysql.user

**WHERE**

**user** = 'brad' **AND**

host = 'localhost';

https://sp.mysqltutorial.org/wp-content/uploads/2019/09/mysql-account-unlocked.png

Third, use the ALTER USER to unlock the user:

**ALTER** **USER** 'brad'@'localhost'

**ACCOUNT** **UNLOCK**;

# **MySQL SHOW DATABASES: List All Databases in MySQL**

## Using the MySQL SHOW DATABASES

To list all databases on a MySQL server host, you use the SHOW DATABASES command as follows:

**SHOW** **DATABASES**;

For example, to list all database in the local MySQL database server, first login to the database server as follows:

>mysql -u root -p

Enter password: \*\*\*\*\*\*\*\*\*\*

mysql>

And then use the SHOW DATABASES command:

mysql> **SHOW** **DATABASES**;

+*--------------------+*

| Database |

+*--------------------+*

| classicmodels |

| information\_schema |

| mysql |

| performance\_schema |

| sys |

| test |

+*--------------------+*

6 rows in **set** (0.00 sec)

The SHOW SCHEMAS command is a synonym for SHOW DATABASES, therefore the following command returns the same result as the one above:

**SHOW** SCHEMAS;

If you want to query the database that matches a specific pattern, you use the [LIKE](https://www.mysqltutorial.org/mysql-like/)clause as follows:

**SHOW** **DATABASES** **LIKE** pattern;

For example, the following statement returns database that ends with the string 'schema';

**SHOW** **DATABASES** **LIKE** '%schema';

+*--------------------+*

| Database (%schema) |

+*--------------------+*

| information\_schema |

| performance\_schema |

+*--------------------+*

2 rows in **set** (0.00 sec)

It is important to note that if the MySQL database server started with --skip-show-database, you cannot use the SHOW DATABASES statement unless you have the SHOW DATABASES privilege.

## Querying database data from information\_schema

If the condition in the LIKE clause is not sufficient, you can query the database information directly from the schemata table in the information\_schema database.

For example, the following query returns the same result as the SHOW DATABASES command.

**SELECT** schema\_name

**FROM** information\_schema.schemata;

The following [SELECT](https://www.mysqltutorial.org/mysql-select-statement-query-data.aspx) statement returns databases whose names end with 'schema' or 's'.

**SELECT** schema\_name

**FROM** information\_schema.schemata

**WHERE** schema\_name **LIKE** '%schema' **OR**

schema\_name **LIKE** '%s';

It returns the following result set:

+*--------------------+*

| SCHEMA\_NAME |

+*--------------------+*

| information\_schema |

| performance\_schema |

| sys |

| classicmodels |

+*--------------------+*

4 rows in **set** (0.00 sec)

[**?**](https://liveburst.com/ref/1/)

**Summary**: in this tutorial, you will learn how to use the MySQL SHOW TABLES command to query tables in a particular database.



To list tables in a MySQL database, you follow these steps:

1. Login to the MySQL database server using a MySQL client such as mysql
2. Switch to a specific database using the USE statement.
3. Use the SHOW TABLES command.

The following illustrates the syntax of the MySQL SHOW TABLES command:

**SHOW** **TABLES**;

## MySQL SHOW TABLES examples

The following example shows you how to list the table in the classicmodels database.

Step 1. Connect to the MySQL database server:

>mysql -u root -p

Enter password: \*\*\*\*\*\*\*\*\*\*

mysql>

Step 2. Switch to classicmodels database:

mysql> **use** classicmodels;

Database changed

mysql>

Step 3. Show tables in the classicmodels database:

> **show** **tables**;

+*-------------------------+*

| Tables\_in\_classicmodels |

+*-------------------------+*

| customers |

| employees |

| offices |

| orderdetails |

| orders |

| payments |

| productlines |

| products |

+*-------------------------+*

8 rows in **set** (0.00 sec)

The SHOW TABLES command allows you to show if a table is a base table or a view. To include the table type in the result, you use the following form of the SHOW TABLES statement.

**SHOW** **FULL** **TABLES**;

Let’s create a view in the classicmodels database called contacts that includes first name, last name and phone from the employees and customers tables for the demonstration.

**CREATE** **VIEW** contacts

**AS**

**SELECT** lastName, firstName, extension **as** phone

**FROM** employees

**UNION**

**SELECT** contactFirstName, contactLastName, phone

**FROM** customers;

Now, you issue the SHOW FULL TABLES command:

> **SHOW** **FULL** **TABLES**

+*-------------------------+------------+*

| Tables\_in\_classicmodels | Table\_type |

+*-------------------------+------------+*

| contacts | **VIEW** |

| customers | BASE **TABLE** |

| employees | BASE **TABLE** |

| offices | BASE **TABLE** |

| orderdetails | BASE **TABLE** |

| orders | BASE **TABLE** |

| payments | BASE **TABLE** |

| productlines | BASE **TABLE** |

| products | BASE **TABLE** |

+*-------------------------+------------+*

9 **rows** **in** **set** (0.00 sec)

As you can see, all the tables are the base tables except for the contacts table which is a view.

For the database that has many tables, showing all tables at a time may not be intuitive.

Fortunately, the SHOW TABLES command provides you with an option that allows you to filter the returned tables using the [LIKE](https://www.mysqltutorial.org/mysql-like/) operator or an expression in the [WHERE](https://www.mysqltutorial.org/mysql-where/) clause as follows:

**SHOW** **TABLES** **LIKE** pattern;

**SHOW** **TABLES** **WHERE** expression;

For example, to shows all tables in the classicmodels database that start with the letter p, you use the following statement:

> **SHOW** **TABLES** **LIKE** 'p%';

+*------------------------------+*

| Tables\_in\_classicmodels (p%) |

+*------------------------------+*

| payments |

| productlines |

| products |

+*------------------------------+*

3 rows in **set** (0.00 sec)

Or to show the tables that end with the string 'es', you use the following statement:

> **SHOW** **TABLES** **LIKE** '%es';

+*-------------------------------+*

| Tables\_in\_classicmodels (%es) |

+*-------------------------------+*

| employees |

| offices |

| productlines |

+*-------------------------------+*

3 rows in **set** (0.00 sec)

The following statement illustrates how to use the WHERE clause in the SHOW TABLES statement to list all the views in the classicmodels database.

> **SHOW** **FULL** **TABLES** **WHERE** table\_type = 'VIEW';

+*-------------------------+------------+*

| Tables\_in\_classicmodels | Table\_type |

+*-------------------------+------------+*

| contacts | VIEW |

+*-------------------------+------------+*

1 row in **set** (0.00 sec)

Sometimes, you want to see the tables in the database that you are not connected to. In this case, you can use the FROM clause of the SHOW TABLES statement to specify the database from which you want to show the tables.

The following example demonstrates how to show tables that start with 'time';

> **SHOW** **TABLES** **FROM** mysql **LIKE** 'time%';

+*---------------------------+*

| Tables\_in\_mysql (time%) |

+*---------------------------+*

| time\_zone |

| time\_zone\_leap\_second |

| time\_zone\_name |

| time\_zone\_transition |

| time\_zone\_transition\_type |

+*---------------------------+*

5 rows in **set** (0.00 sec)

The following statement is equivalent to the statement above but it uses IN instead of FROM.

**SHOW** **TABLES** **IN** mysql **LIKE** 'time%';

It’s important to note that if you don’t have privileges for a base table or view, it won’t show up in the result set of the SHOW TABLES command.

In this tutorial, you have learned how to use the MySQL SHOW TABLES statement to list all tables in a particular database.

# **MySQL SHOW TABLES: List Tables In a MySQL Database**

To list tables in a MySQL database, you follow these steps:

1. Login to the MySQL database server using a MySQL client such as mysql
2. Switch to a specific database using the USE statement.
3. Use the SHOW TABLES command.

The following illustrates the syntax of the MySQL SHOW TABLES command:

**SHOW** **TABLES**;

MySQL SHOW TABLES examples

The following example shows you how to list the table in the classicmodels database.

Step 1. Connect to the MySQL database server:

>mysql -u root -p

Enter password: \*\*\*\*\*\*\*\*\*\*

mysql>

Step 2. Switch to classicmodels database:

mysql> **use** classicmodels;

Database changed

mysql>

Step 3. Show tables in the classicmodels database:

> **show** **tables**;

+*-------------------------+*

| Tables\_in\_classicmodels |

+*-------------------------+*

| customers |

| employees |

| offices |

| orderdetails |

| orders |

| payments |

| productlines |

| products |

+*-------------------------+*

8 rows in **set** (0.00 sec)

The SHOW TABLES command allows you to show if a table is a base table or a view. To include the table type in the result, you use the following form of the SHOW TABLES statement.

**SHOW** **FULL** **TABLES**;

Let’s create a view in the classicmodels database called contacts that includes first name, last name and phone from the employees and customers tables for the demonstration.

**CREATE** **VIEW** contacts

**AS**

**SELECT** lastName, firstName, extension **as** phone

**FROM** employees

**UNION**

**SELECT** contactFirstName, contactLastName, phone

**FROM** customers;

Now, you issue the SHOW FULL TABLES command:

> **SHOW** **FULL** **TABLES**

+*-------------------------+------------+*

| Tables\_in\_classicmodels | Table\_type |

+*-------------------------+------------+*

| contacts | **VIEW** |

| customers | BASE **TABLE** |

| employees | BASE **TABLE** |

| offices | BASE **TABLE** |

| orderdetails | BASE **TABLE** |

| orders | BASE **TABLE** |

| payments | BASE **TABLE** |

| productlines | BASE **TABLE** |

| products | BASE **TABLE** |

+*-------------------------+------------+*

9 **rows** **in** **set** (0.00 sec)

As you can see, all the tables are the base tables except for the contacts table which is a view.

For the database that has many tables, showing all tables at a time may not be intuitive.

Fortunately, the SHOW TABLES command provides you with an option that allows you to filter the returned tables using the [LIKE](https://www.mysqltutorial.org/mysql-like/) operator or an expression in the [WHERE](https://www.mysqltutorial.org/mysql-where/) clause as follows:

**SHOW** **TABLES** **LIKE** pattern;

**SHOW** **TABLES** **WHERE** expression;

For example, to shows all tables in the classicmodels database that start with the letter p, you use the following statement:

> **SHOW** **TABLES** **LIKE** 'p%';

+*------------------------------+*

| Tables\_in\_classicmodels (p%) |

+*------------------------------+*

| payments |

| productlines |

| products |

+*------------------------------+*

3 rows in **set** (0.00 sec)

Or to show the tables that end with the string 'es', you use the following statement:

> **SHOW** **TABLES** **LIKE** '%es';

+*-------------------------------+*

| Tables\_in\_classicmodels (%es) |

+*-------------------------------+*

| employees |

| offices |

| productlines |

+*-------------------------------+*

3 rows in **set** (0.00 sec)

The following statement illustrates how to use the WHERE clause in the SHOW TABLES statement to list all the views in the classicmodels database.

> **SHOW** **FULL** **TABLES** **WHERE** table\_type = 'VIEW';

+*-------------------------+------------+*

| Tables\_in\_classicmodels | Table\_type |

+*-------------------------+------------+*

| contacts | VIEW |

+*-------------------------+------------+*

1 row in **set** (0.00 sec)

Sometimes, you want to see the tables in the database that you are not connected to. In this case, you can use the FROM clause of the SHOW TABLES statement to specify the database from which you want to show the tables.

The following example demonstrates how to show tables that start with 'time';

> **SHOW** **TABLES** **FROM** mysql **LIKE** 'time%';

+*---------------------------+*

| Tables\_in\_mysql (time%) |

+*---------------------------+*

| time\_zone |

| time\_zone\_leap\_second |

| time\_zone\_name |

| time\_zone\_transition |

| time\_zone\_transition\_type |

+*---------------------------+*

5 rows in **set** (0.00 sec)

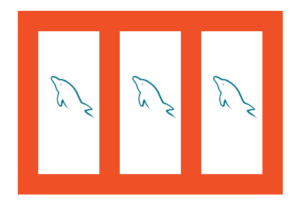
The following statement is equivalent to the statement above but it uses IN instead of FROM.

**SHOW** **TABLES** **IN** mysql **LIKE** 'time%';

It’s important to note that if you don’t have privileges for a base table or view, it won’t show up in the result set of the SHOW TABLES command.

# **MySQL SHOW COLUMNS and DESCRIBE: List All Columns in a Table**

## Using DESCRIBE statement



To show all columns of a table, you use the following steps:

1. Login to the MySQL database server.
2. Switch to a specific database.
3. Use the DESCRIBE statement.

The following example demonstrates how to display columns of the orders table in the classicmodels database.

**Step 1**. Login to the MySQL database.

>mysql -u root -p

Enter password: \*\*\*\*\*\*\*\*\*\*

mysql>

**Step 2**. Issue the USE command to switch to the database to classicmodels:

mysql> **USE** classicmodels;

Database changed

mysql>

**Step 3**. Use the DESCRIBE statement.

mysql> **DESCRIBE** orders;

+*----------------+-------------+------+-----+---------+-------+*

| Field | Type | Null | Key | Default | Extra |

+*----------------+-------------+------+-----+---------+-------+*

| orderNumber | int(11) | NO | PRI | NULL | |

| orderDate | date | NO | | NULL | |

| requiredDate | date | NO | | NULL | |

| shippedDate | date | YES | | NULL | |

| status | varchar(15) | NO | | NULL | |

| comments | text | YES | | NULL | |

| customerNumber | int(11) | NO | MUL | NULL | |

+*----------------+-------------+------+-----+---------+-------+*

7 rows in **set** (0.01 sec)

In practice, you use the DESC statement which is a shorthand of the DESCRIBE statement. For example, the following statement is equivalent to the DESCRIBE above:

DESC orders;

## MySQL SHOW COLUMNS command

The more flexible way to get a list of columns in a table is to use the MySQL SHOW COLUMNS command.

**SHOW** **COLUMNS** **FROM** table\_name;

To show columns of a table, you specific the table name in the FROM clause of the SHOW COLUMNS statement. To show columns of a table in a database that is not the current database, you use the following form:

**SHOW** **COLUMNS** **FROM** database\_name.table\_name;

Or

**SHOW** **COLUMNS** **FROM** table\_name **IN** database\_name;

For example, to get the columns of the orders table, you use the SHOW COLUMNS statement as follows:

**SHOW** **COLUMNS** **FROM** orders;

As you can see the result of this SHOW COLUMNS command is the same as the result of the DESC statement.

To get more information about the column, you add the FULL keyword to the SHOW COLUMNS command as follows:

**SHOW** **FULL** **COLUMNS** **FROM** table\_name;

For example, the following statement lists all columns of the payments table in the classicmodels database.

mysql> **SHOW** **FULL** **COLUMNS** **FROM** payments \G;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Field: customerNumber

Type: int(11)

Collation: NULL

Null: NO

Key: PRI

Default: NULL

Extra:

Privileges: **select**,**insert**,**update**,**references**

**Comment**:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 2. **row** \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Field**: checkNumber

**Type**: varchar(50)

**Collation**: latin1\_swedish\_ci

Null: **NO**

**Key**: PRI

**Default**: NULL

Extra:

**Privileges**: **select**,**insert**,**update**,**references**

**Comment**:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 3. **row** \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Field**: paymentDate

**Type**: date

**Collation**: NULL

Null: **NO**

**Key**:

**Default**: NULL

Extra:

**Privileges**: **select**,**insert**,**update**,**references**

**Comment**:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 4. **row** \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Field**: amount

**Type**: decimal(10,2)

**Collation**: NULL

Null: **NO**

**Key**:

**Default**: NULL

Extra:

**Privileges**: **select**,**insert**,**update**,**references**

**Comment**:

4 **rows** **in** **set** (0.01 sec)

As you can see, the SHOW FULL COLUMNS command adds the collation, privileges, and comment columns to the result set.

The SHOW COLUMNS command allows you to filter the columns of the table by using the  [LIKE](https://www.mysqltutorial.org/mysql-like/) operator or [WHERE](https://www.mysqltutorial.org/mysql-where/)clause:

**SHOW** **COLUMNS** **FROM** table\_name **LIKE** pattern;

**SHOW** **COLUMNS** **FROM** table\_name **WHERE** expression;

For example, to show only columns that start with the letter c, you use the LIKE operator as follows:

mysql> **SHOW** **COLUMNS** **FROM** payments **LIKE** 'c%';

+*----------------+-------------+------+-----+---------+-------+*

| Field | Type | Null | Key | Default | Extra |

+*----------------+-------------+------+-----+---------+-------+*

| customerNumber | int(11) | NO | PRI | NULL | |

| checkNumber | varchar(50) | NO | PRI | NULL | |

+*----------------+-------------+------+-----+---------+-------+*

2 rows in **set** (0.01 sec)

# **MySQL SHOW PROCESSLIST**

Sometimes, you may get the “too many connections” error returned by the MySQL Server. To find out the reasons, you can use the SHOW PROCESSLIST command.

The SHOW PROCESSLIST command returns all currently running threads. You then can terminate the idle threads with the KILL statement.

The following shows the syntax of the SHOW PROCESSLIST command:

**SHOW** [**FULL**] **PROCESSLIST**;

Accounts with the PROCESS [privilege](https://www.mysqltutorial.org/mysql-grant.aspx) can view all threads. Otherwise, they can view only threads associated with their accounts.

The following shows an example of the output of the SHOW PROCESSLIST command:

mysql>SHOW PROCESSLIST;

+----+-----------------+-----------------+---------------+---------+------+------------------------+------------------+

| Id | User | Host | db | Command | Time | State | Info |

+----+-----------------+-----------------+---------------+---------+------+------------------------+------------------+

| 4 | event\_scheduler | localhost | **NULL** | Daemon | 2246 | Waiting on **empty** queue | **NULL** |

| 14 | root | localhost:50924 | **NULL** | Query | 0 | starting | SHOW PROCESSLIST |

| 15 | car | localhost:50933 | classicmodels | Sleep | 2 | | **NULL** |

+----+-----------------+-----------------+---------------+---------+------+------------------------+------------------+

3 rows in set (0.00 sec)

The output of the SHOW PROCESSLIST command consists of the following columns:

### **Id**

The client process’s Id

### **User**

The username associated with the thread.

### **Host**

The host to which the client is connected

### **DB**

The default database if one selected otherwise [NULL](https://www.mysqltutorial.org/mysql-null/)

### **Command**

The command type

### **Time**

The number of seconds that the current thread has been in its current state.

### **State**

The thread state which represents an action, event, or state that indicates what thread is executing.

### **Info**

The statement is being executed, or NULL if it is not executing any statement. If you do not use the FULL keyword in the SHOW PROCESSLIST command, then only the first 100 characters of each statement are returned in the Info column.

# **MySQL Backup: Backing Up Using mysqldump Tool**

## Introduction to mysqldump tool

The mysqldump tool allows you to make a backup of one or more databases by generating a text file that contains SQL statements which can re-create the databases from scratch.

The mysqldump tool is located in the root/bin directory of the MySQL installation directory.

To access the mysqldump tool, you navigate to the root/bin folder and use the mysqldump command with the following options.

Here are the common mysqldump options:

### **add-drop-table**

Includes a DROP TABLE statement for each table in the database.

### **add-locks**

Includes [LOCK TABLES](https://www.mysqltutorial.org/mysql-table-locking/) and UNLOCK TABLES statements before and after each [INSERT](https://www.mysqltutorial.org/mysql-insert-statement.aspx) statement. It improves the data restoration speed from the dump file.

### **all-databases**

Creates a dump of all databases on the MySQL server.

### **create-options**

Includes ENGINE and CHARSET options in the [CREATE TABLE](https://www.mysqltutorial.org/mysql-create-table/) statement for each table.

### **databases**

Creates a dump of one or more databases.

### **disable-keys**

Instructs MySQL to disable index updates during data load for MyISAM tables. MySQL will create indexes after mysqldump completes loading data. This option improves the speed of restoration.

### **extended-insert**

Combines single-row INSERT statements into a single statement that [insert multiple table rows](https://www.mysqltutorial.org/mysql-insert-multiple-rows/); This option also helps speed up data restoration.

### **flush-logs**

Flushes the server logs before dumping the data. This is useful in conjunction with incremental backups.

### **lock-tables**

Ensures that the dump is a consistent snapshot by locking all the tables in a database during the dump.

### **no-data**

create a dump file that contains statements necessary for re-creating the database structure only (only [CREATE DATABASE](https://www.mysqltutorial.org/mysql-create-database/), [CREATE TABLE](https://www.mysqltutorial.org/mysql-create-table/)…), not the data (no INSERT statements).

### **opt**

The mysqldump tool uses the opt by default.

The opt option enables the following options: add-drop-table, add-locks, create-options, disable-keys, extended-insert, lock-tables, quick, and set-charset.

To disable this option, you use skip-opt. If you want to skip each individual option, you use skip-<option\_name>. For example, to skip the disable-keys option, you use skip-disable-keys option.

### **quick**

Instructs mysqldump to not buffer tables in memory before writing to the file. This option speeds up dumps from big tables.

### **result-file**

Specifies the path to the output dump file.

### **set-charset**

Specifies the character set such as latin1 or utf8 of the database.

### **tables**

Creates a dump of one or more tables.

### **where**

Dumps only rows which satisfies a condition in the [WHERE](https://www.mysqltutorial.org/mysql-where/) clause.

## Using the mysqldump tool to make a backup of databases

Let’s take some examples of using the mysqldump tool to backup database examples.

### **1) Using the mysqldump tool to make a backup of a single database**

The following command backs up a single database from a MySQL Server:

mysqldump *--user=<username> --password=<password> --result-file=<path\_to\_backup\_file> --databases <database\_name>*

In this syntax:

* The <username> is the user account that will login to the MySQL Server.
* The <password> is the password for the <username>.
* The <path\_to\_backup\_file> is the path to the backup file.
* The --databases is an option that instructs the mysqldump tool to create a dump of the specified databases.
* The <database\_name> is the name of the database that you want to back up.

For example, the following command creates a backup of the database classicmodels to the file c:\backup\classicmodels.sql:

mysqldump --user=root --password=Supe!rPass1 --result-file=c:\backup\classicmodels.sql --databases classicmodels

### **2) Using the mysqldump tool to make a backup of multiple databases**

To make a backup of multiple databases, you specify a list of the database names after the --database option:

mysqldump --user=<username> --password=<password> --result-file=<path\_to\_backup\_file> --databases <dbname1>[,<dbname2>, ...]

For example, the following command makes a backup of the classicmodels and world databases:

mysqldump --user=root --password=Supe!rPass1 --result-file=c:\backup\classicmodels\_world.sql --databases classicmodels world

### **3) Using the mysqldump tool to make a backup of all databases**

To make a backup of all databases in a MySQL Server, you use the –all-database option:

mysqldump --user=<username> --password=<password> --result-file=<path\_to\_backup\_file> --all-databases

The following statement makes a backup of all databases in the current MySQL server:

mysqldump --user=root --password=Supe!rPass1 --result-file=c:\backup\all\_databases.sql --all-databases

### **4) Using the mysqldump tool to make a backup of specific tables from a database**

To make a backup of specific tables from a database, you use the following command:

mysqldump --user=<username> --password=<password> --result-file=<path\_to\_backup\_file> <database\_name> <table\_name>

You can also specify multiple tables after the database name, one after the other:

mysqldump --user=<username> --password=<password> --result-file=<path\_to\_backup\_file> <database\_name> <table1> <table2> <table3>

For example, to make a backup of the employees table from the classicmodels database, you use the following command:

mysqldump --user=root --password=Supe!rPass1 --result-file=c:\backup\employees.sql classicmodels employees

### **5) Using a mysqldump tool to make a backup of database structure only**

To make a backup of the database structure only, you use the --no-data option:

mysqldump --user=<username> --password=<password> --result-file=<path\_to\_backup\_file> --no-data --databases <database\_name>

The statement will generate a dump file that contains the SQL statement necessary to re-create the database structure. And the dump file does not contain INSERT statements.

For example, the following statement makes a backup of the database structure of the database classicmodels:

mysqldump --user=root --password=Supe!rPass1 --result-file=c:\backup\classicmodels-ddl.sql --no-data --databases classicmodels

### **6) Using the mysqldump tool to make a backup of data only**

To make a backup of the database data only, you use the --no-create-info option:

mysqldump --user=<username> --password=<password> --result-file=<path\_to\_backup\_file> –-no-create-info --databases <database\_name>

The statement will generate a dump file that contains the SQL statements necessary to lock tables and insert data into the tables. It has no CREATE TABLE statements.

The following command makes a backup of data of the database classicmodels:

mysqldump –-user=root --password=Supe!rPass1 --result-file=c:\backup\classicmodels-data.sql –-no-create-info --databases classicmodels

# **MySQL Restore: Restoring from a Dump File**

## Setting up a sample database

First, [create a new database](https://www.mysqltutorial.org/mysql-create-database/) called mydb:

**CREATE** **DATABASE** mydb;

Second, use the mydb database:

**USE** mydb;

Third, [create a new table](https://www.mysqltutorial.org/mysql-create-table/) tests with one column:

**CREATE** **TABLE** tests(

**id** INT PRIMARY **KEY**

);

Fourth, [insert some rows](https://www.mysqltutorial.org/mysql-insert-multiple-rows/) into the tests table:

**INSERT** **INTO** tests(**id**)

**VALUES**(1),(2),(3);

Finally, use the mysqldump to dump the mydb database:

mysqldump *--user=root --password=Supe!rPass1 --result-file=c:\backup\mydb.sql --databases mydb*

Note that you must have the c:\backup directory available on your computer or server.

## Restoring an SQL dump file using the SOURCE command

To restore the mydb.sql SQL dump file, you follow these steps:

First, connect to MySQL server:

C:\>mysql -u root -p

Enter password: \*\*\*\*\*\*\*\*

Second, drop the mydb database:

mysql>**drop** **database** mydb;

Third, use the SOURCE command to load the dump file:

mysql>source c:\**backup**\mydb.sql

The command created a database mydb, select the database, and execute other SQL statements. In addition, it showed all the possible warnings and errors.

It is recommended that you use the SOURCE command to restore a dump file because the SOURCE command returns very detailed information including warnings and errors.

# **Maintaining MySQL Database Tables**

MySQL provides several useful statements that allow you to maintain database tables effectively. Those statements enable you to analyze, optimize, check, and repair database tables.

## Analyze table statement

MySQL query optimizer is an important component of the MySQL server that creates an optimal query execution plan for a query. For a particular query, the query optimizer uses the stored key distribution and other factors to decide the order in which tables should be joined when you performing the [join](https://www.mysqltutorial.org/mysql-inner-join.aspx), and which [index](https://www.mysqltutorial.org/mysql-index/mysql-create-index/) should be used for a specific table.

However, the key distributions can be sometimes inaccurate e.g., after you have done a lot of data changes in the table including [insert](https://www.mysqltutorial.org/mysql-insert-statement.aspx), [delete](https://www.mysqltutorial.org/mysql-delete-statement.aspx), or [update](https://www.mysqltutorial.org/mysql-update-data.aspx). If the key distribution is not accurate, the query optimizer may pick a bad query execution plan that may cause a severe performance issue.

To solve this problem, you can run the ANALYZE TABLE statement for the table e.g., the following statement analyzes the payments table in the [sample database](https://www.mysqltutorial.org/mysql-sample-database.aspx).

**ANALYZE** **TABLE** payments;

mysql analyze table statement

If there is no change to the table since the ANALYZE TABLE statement ran, MySQL will not analyze the table again. If you run the above statement again:

**ANALYZE** **TABLE** payments;

mysql analyze table again

It is saying that the table is already up to date.

## Optimize table statement

While working with the database, you do a lot of changes such as [insert](https://www.mysqltutorial.org/mysql-insert-statement.aspx), [update](https://www.mysqltutorial.org/mysql-update-data.aspx)and [delete](https://www.mysqltutorial.org/mysql-delete-statement.aspx)data in the table that may cause the physical storage of the table fragmented. As a result, the performance of database server is degraded.

MySQL provides you with a statement that allows you to optimize the table to avoid this defragmenting problem. The following illustrates how to optimize a table:

**OPTIMIZE** **TABLE** table\_name;

It is recommended that you execute this statement for the tables that are updated frequently. For example, if you want to optimize the orders table to defragment it, you can perform the following statement:

**OPTIMIZE** **TABLE** orders;

mysql optimize table statement

## Check table statement

Something wrong can happen to the database server e.g., the server was shutdown unexpectedly, error while writing data to the hard disk, etc. These situations could make the database operate incorrectly and in the worst case, it can be crashed.

MySQL allows you to check the integrity of database tables by using the CHECK TABLE statement. The following illustrates the syntax of the CHECK TABLE statement:

**CHECK** **TABLE** table\_name;

The CHECK TABLE statement checks both table and its indexes. For example, you can use the CHECK TABLE statement to check the orders table as follows:

**CHECK** **TABLE** orders;

mysql check table statement

The CHECK TABLE statement only detects problems in a database table but it does not repair them. To repair the table, you use the REPAIR TABLE statement.

## Repair table statement

The REPAIR TABLE statement allows you to repair some errors occurred in database tables. MySQL does not guarantee that the REPAIR TABLE statement can repair all errors that the tables may have.

The following is the syntax of the REPAIR TABLE statement:

**REPAIR** **TABLE** table\_name;

Suppose there are some errors in the orders table and you need to fix them, you can use the REPAIR TABLE statement as the following query:

**REPAIR** **TABLE** employees;

MySQL returns what it has done to the table and shows you whether the table was repaired or not.

mysql repair statement

# **MySQL Cheat Sheet**

## MySQL command-line client Commands

[Connect to MySQL server](https://www.mysqltutorial.org/getting-started-with-mysql/connect-to-mysql-server/) using mysql  command-line client with a username and password (MySQL will prompt for a password):

mysql -u [username] -p;

Connect to MySQL Server with a specified database using a username and password:

mysql -u [username] -p [database];

Exit mysql command-line client:

exit;

[Export data using mysqldump tool](https://www.mysqltutorial.org/how-to-backup-database-using-mysqldump.aspx)

mysqldump -u [username] -p [database] > data\_backup.sql;

To clear MySQL screen console window on Linux, you use the following command:

mysql> system clear;

Currently, there is no command available on Windows OS for clearing MySQL screen console window.

## Working with databases

[Create a database](https://www.mysqltutorial.org/mysql-create-drop-database.aspx) with a specified name if it does not exist in the database server

**CREATE** **DATABASE** [**IF** **NOT** **EXISTS**] database\_name;

Use a database or change the current database to another database that you are working with:

**USE** database\_name;

Drop a database with a specified name permanently. All physical files associated with the database will be deleted.

**DROP** **DATABASE** [**IF** **EXISTS**] database\_name;

Show all available databases in the current MySQL database server

**SHOW** **DATABASE**;

## Working with tables

Show all tables in a current database.

**SHOW** **TABLES**;

[Create a new table](https://www.mysqltutorial.org/mysql-create-table/)

**CREATE** **TABLE** [**IF** **NOT** **EXISTS**] table\_name(

column\_list

);

[Add a new column](https://www.mysqltutorial.org/mysql-add-column/) into a table:

**ALTER** **TABLE** **table**

**ADD** [**COLUMN**] column\_name;

[Drop a column](https://www.mysqltutorial.org/mysql-drop-column/) from a table:

**ALTER** **TABLE** table\_name

**DROP** [**COLUMN**] column\_name;

Add index with a specific name to a table on a column:

**ALTER** **TABLE** **table**

**ADD** **INDEX** [**name**](**column**, ...);

Add [primary key](https://www.mysqltutorial.org/mysql-primary-key/) into a table:

ALTER TABLE table\_name

ADD PRIMARY KEY (column\_name,...);

Remove the primary key of a table:

ALTER TABLE table\_name

DROP PRIMARY KEY;

[Drop a table](https://www.mysqltutorial.org/mysql-drop-table):

**DROP** **TABLE** [**IF** **EXISTS**] table\_name;

[Show the columns](https://www.mysqltutorial.org/mysql-show-columns/) of a table:

**DESCRIBE** table\_name;

Show the information of a column in a table:

**DESCRIBE** table\_name column\_name;

## Working with indexes

[Creating an index](https://www.mysqltutorial.org/mysql-index/mysql-create-index/) with the specified name on a table:

**CREATE** **INDEX** index\_name

**ON** table\_name (**column**,...);

[Drop an index](https://www.mysqltutorial.org/mysql-index/mysql-drop-index/):

**DROP** **INDEX** index\_name;

[Create a unique index](https://www.mysqltutorial.org/mysql-unique/):

**CREATE** **UNIQUE** **INDEX** index\_name

**ON** table\_name (**column**,...);

## Working with views

Create a new view:

**CREATE** **VIEW** [**IF** **NOT** **EXISTS**] view\_name

**AS**

select\_statement;

Create a new view with the WITH CHECK OPTION:

**CREATE** **VIEW** [**IF** **NOT** **EXISTS**] view\_name

**AS** select\_statement

**WITH** **CHECK** **OPTION**;

Create or replace a view:

**CREATE** **OR** **REPLACE** view\_name

**AS**

select\_statement;

Drop a view:

**DROP** **VIEW** [**IF** **EXISTS**] view\_name;

Drop multiple views:

**DROP** **VIEW** [**IF** **EXISTS**] view1, view2, ...;

Rename a view:

**RENAME** **TABLE** view\_name

**TO** new\_view\_name;

Show views from a database:

**SHOW** **FULL** **TABLES**

[{**FROM** | **IN** } database\_name]

**WHERE** table\_type = 'VIEW';

## Working with triggers

Create a new trigger:

**CREATE** **TRIGGER** trigger\_name

{**BEFORE** | **AFTER**} {**INSERT** | **UPDATE**| **DELETE** }

**ON** table\_name **FOR** **EACH** **ROW**

trigger\_body;

Drop a trigger:

**DROP** **TRIGGER** [**IF** **EXISTS**] trigger\_name;

Show triggers in a database:

**SHOW** **TRIGGERS**

[{**FROM** | **IN**} database\_name]

[**LIKE** 'pattern' | **WHERE** search\_condition];

## Working with stored procedures

Create a stored procedure:

DELIMITER $$

**CREATE** **PROCEDURE** procedure\_name(parameter\_list)

**BEGIN**

**body**;

**END** $$

DELIMITER ;

Drop a stored procedure:

**DROP** **PROCEDURE** [**IF** **EXISTS**] procedure\_name;

Show stored procedures:

**SHOW** **PROCEDURE** **STATUS**

[**LIKE** 'pattern' | **WHERE** search\_condition];

## Working with stored functions

Create a new stored function:

DELIMITER $$

**CREATE** **FUNCTION** function\_name(parameter\_list)

**RETURNS** datatype

[**NOT**] **DETERMINISTIC**

**BEGIN**

*-- statements*

**END** $$

DELIMITER ;

Drop a stored function:

**DROP** **FUNCTION** [**IF** **EXISTS**] function\_name;

Show stored functions:

**SHOW** **FUNCTION** **STATUS**

[**LIKE** 'pattern' | **WHERE** search\_condition];

## Querying data from tables

[Query all data](https://www.mysqltutorial.org/mysql-select-statement-query-data.aspx) from a table:

**SELECT** \* **FROM** table\_name;

Query data from one or more column of a table:

**SELECT**

column1, column2, ...

**FROM**

table\_name;

Remove duplicate rows from the result of a query:

**SELECT**

**DISTINCT** (**column**)

**FROM**

table\_name;

Query data with a filter using a [WHERE](https://www.mysqltutorial.org/mysql-where/) clause:

**SELECT** select\_list

**FROM** table\_name

**WHERE** condition;

Change the output of the column name using [column alias](https://www.mysqltutorial.org/mysql-alias/):

**SELECT**

column1 **AS** alias\_name,

expression **AS** **alias**,

...

**FROM**

table\_name;

Query data from multiple tables using [inner join](https://www.mysqltutorial.org/mysql-inner-join.aspx):

**SELECT** select\_list

**FROM** table1

**INNER** **JOIN** table2 **ON** condition;

Query data from multiple tables using [left join](https://www.mysqltutorial.org/mysql-left-join.aspx):

**SELECT** select\_list

**FROM** table1

**LEFT** **JOIN** table2 **ON** condition;

Query data from multiple tables using [right join](https://www.mysqltutorial.org/mysql-right-join/):

**SELECT** select\_list

**FROM** table1

**RIGHT** **JOIN** table2 **ON** condition;

Make a Cartesian product of rows:

**SELECT** select\_list

**FROM** table1

**CROSS** **JOIN** table2;

[Counting rows](https://www.mysqltutorial.org/mysql-count/) in a table.

**SELECT** **COUNT**(\*)

**FROM** table\_name;

Sorting a result set:

**SELECT**

select\_list

**FROM**

table\_name

**ORDER** **BY**

column1 **ASC** [**DESC**],

column2 **ASC** [**DESC**];

Group rows using the [GROUP BY](https://www.mysqltutorial.org/mysql-group-by.aspx) clause.

**SELECT** select\_list

**FROM** table\_name

**GROUP** **BY** column\_1, column\_2, ...;

Filter group using the [HAVING](https://www.mysqltutorial.org/mysql-having.aspx) clause:

**SELECT** select\_list

**FROM** table\_name

**GROUP** **BY** column1

**HAVING** condition;

## Modifying data in tables

[Insert a new row](https://www.mysqltutorial.org/mysql-insert-statement.aspx) into a table:

**INSERT** **INTO** table\_name(column\_list)

**VALUES**(value\_list);

[Insert multiple rows](https://www.mysqltutorial.org/mysql-insert-multiple-rows/) into a table:

**INSERT** **INTO** table\_name(column\_list)

**VALUES**(value\_list1),

(value\_list2),

(value\_list3),

...;

[Update](https://www.mysqltutorial.org/mysql-update-data.aspx) all rows in a table:

**UPDATE** table\_name

**SET** column1 = value1,

...;

Update data for a set of rows specified by a condition in WHERE clause.

**UPDATE** table\_name

**SET** column\_1 = value\_1,

...

**WHERE** condition

[Update with join](https://www.mysqltutorial.org/mysql-update-join/)

**UPDATE**

table1,

table2

**INNER** **JOIN** table1 **ON** table1.column1 = table2.column2

**SET** column1 = value1,

**WHERE** condition;

[Delete all rows in a table](https://www.mysqltutorial.org/mysql-delete-statement.aspx)

**DELETE** **FROM** table\_name;

Delete rows specified by a condition:

**DELETE** **FROM** table\_name

**WHERE** condition;

[Delete with join](https://www.mysqltutorial.org/mysql-delete-join/)

**DELETE** table1, table2

**FROM** table1

**INNER** **JOIN** table2

**ON** table1.column1 = table2.column2

**WHERE** condition;

## Searching

Search for data using the [LIKE](https://www.mysqltutorial.org/mysql-like/) operator:

**SELECT** select\_list

**FROM** table\_name

**WHERE** **column** **LIKE** '%pattern%';

Text search using a [regular expression](https://www.mysqltutorial.org/mysql-regular-expression-regexp.aspx) with RLIKE operator.

**SELECT** select\_list

**FROM** table\_name

**WHERE** **column** **RLIKE** 'regular\_expression';