***MySQL Database***

# **MySQL Create Database**

A database is used to store the collection of records in an organized form. It allows us to hold the data into tables, rows, columns, and indexes to find the relevant information frequently. We can access and manage the records through the database very easily.

**MySQL**

implements a database as a directory that stores all files in the form of a table. It allows us to create a database mainly in **two ways**:

1. MySQL Command Line Client
2. MySQL Workbench

### **MySQL Command Line Client**

We can create a new database in MySQL by using the **CREATE DATABASE** statement with the below syntax:

1. CREATEDATABASE**[IF NOT EXISTS] database\_name**
2. **[**CHARACTERSET**charset\_name]**
3. **[**COLLATE**collation\_name];**

**Parameter Explanation**

The parameter descriptions of the above syntax are as follows:

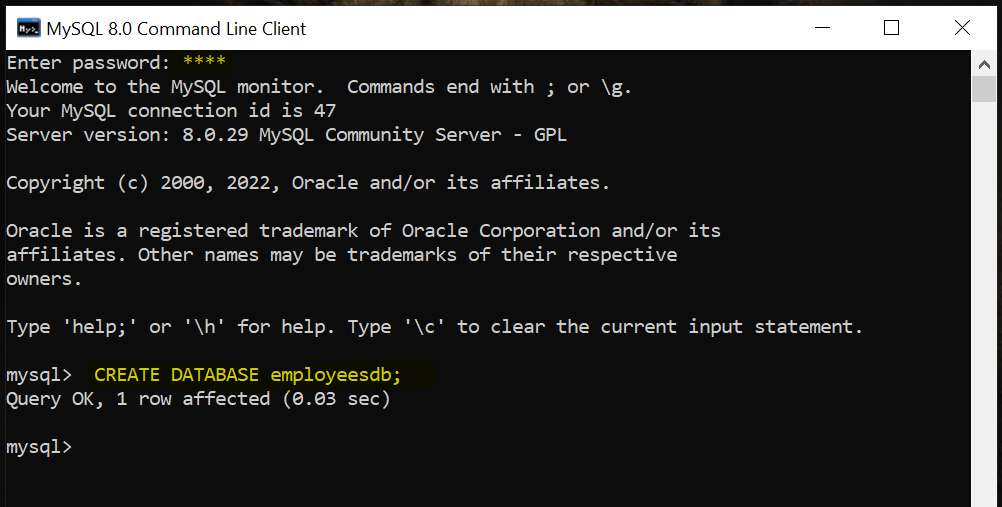
|  |  |
| --- | --- |
| **Parameter** | **Description** |
| database\_name | It is the name of a new database that should be unique in the MySQL server instance. The **IF NOT EXIST** clause avoids an error when we create a database that already exists. |
| charset\_name | It is optional. It is the name of the character set to store every character in a string. MySQL database server supports many character sets. If we do not provide this in the statement, MySQL takes the default character set. |
| collation\_name | It is optional that compares characters in a particular character set. |

**Example**

Let us understand how to create a database in MySQL with the help of an example. Open the MySQL console and write down the password, if we have set during installation. Now we are ready to create a database. Here, we are going to create a database name **"employeedb"** using the following statement:

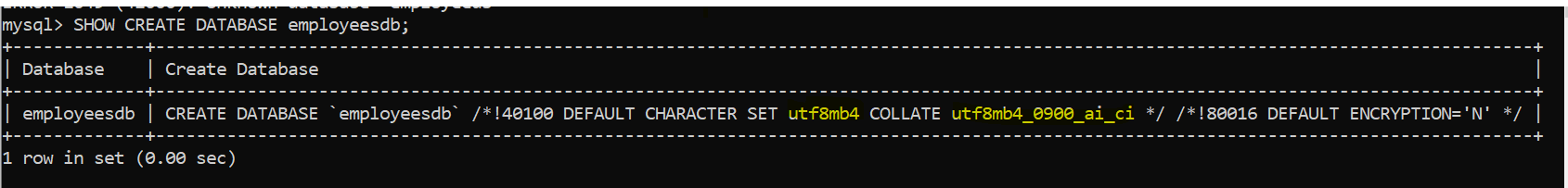
**mysql>**CREATEDATABASE**employeesdb;**

It will look like the below output:



We can review the newly created database using the below query that returns the database name, character set, and collation of the database:

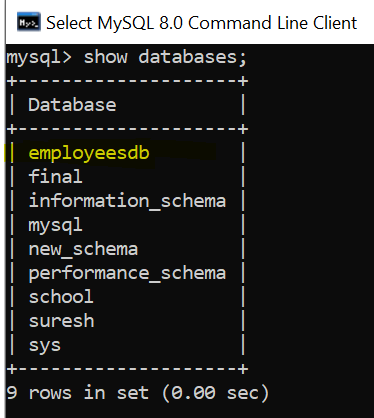
**mysql> SHOW CREATE DATABASE employeesdb;**



We can check the created database using the following query:

**mysql> SHOW DATABASES;**

After executing the above query, we can see all the created databases in the server.



Finally, we can use the below command to access the database that enables us to create a table and other database objects.

**mysql> USE emplyeedb;**

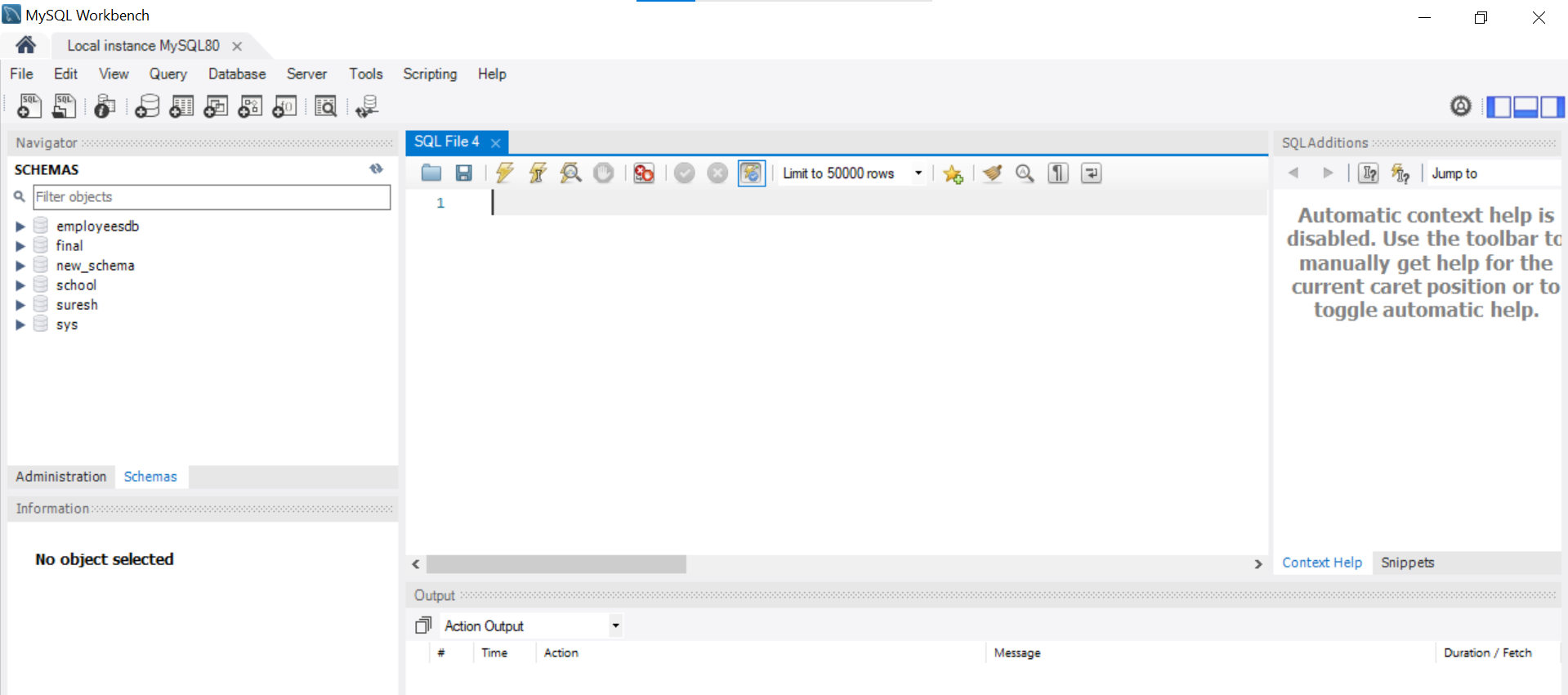
#### **NOTE: All the database names, table names, and table field names are case sensitive. We must have to use proper names while giving any SQL command.**

### ***MySQL Workbench***

It is a visual database designing or GUI tool used to work with database architects, developers, and Database Administrators. This visual tool supports **SQL** development, data modelling, data migration, and comprehensive administration tools for server configuration, user administration, backup, and many more. It allows us to create new physical data models, E-R diagrams, and SQL development (run queries, etc.).

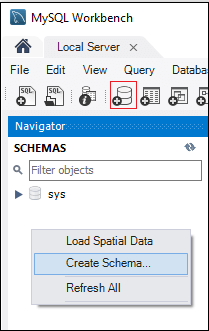
To create a new database using this tool, we first need to launch the **MySQL Workbench**

and log in using the username and password that you want. It will show the following screen

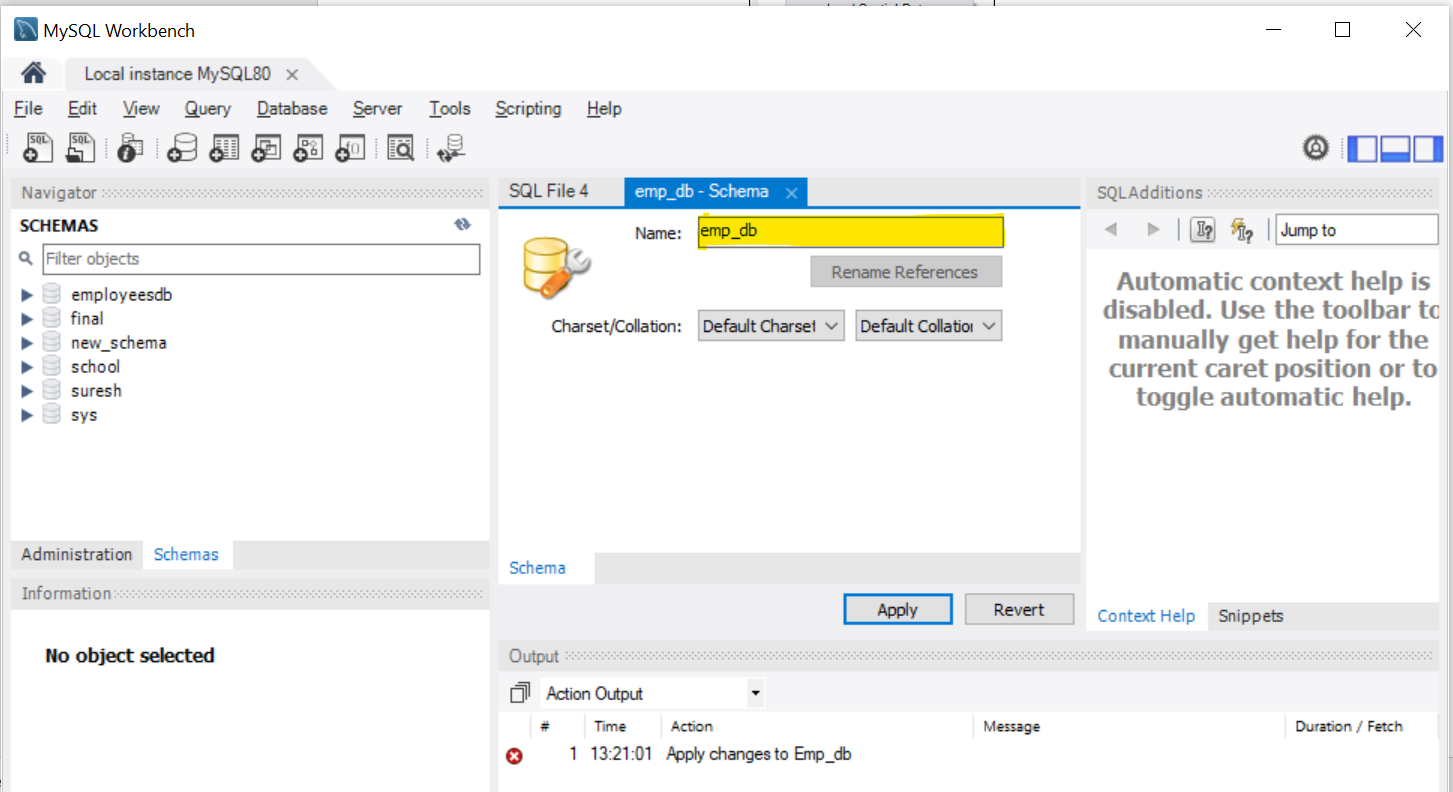


Now do the following steps for database creation:

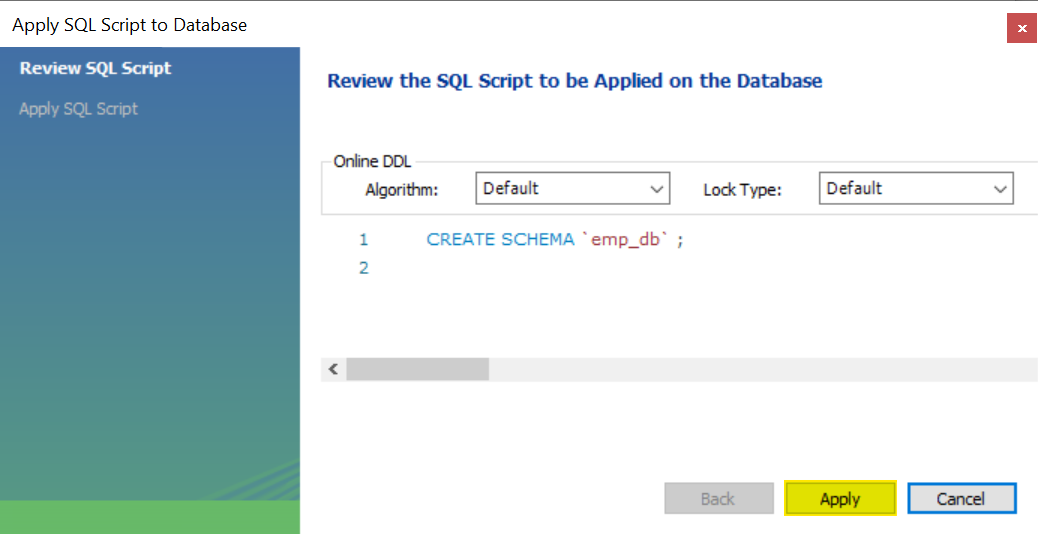
1. Go to the Navigation tab and click on the **Schema menu**. Here, we can see all the previously created databases. If we want to create a new database, right-click under the Schema menu and select Create Schema or click the database **icon (red rectangle)**, as shown in the following screen.



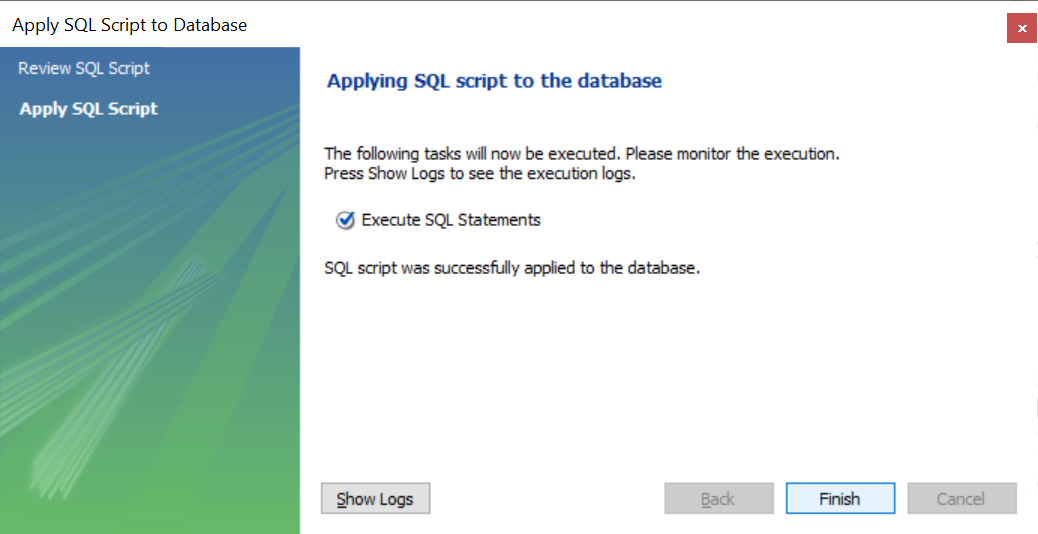
2. The new Schema window screen open. Enter the new database name (for example, **emp\_db**) and use default character set and collation. Now, click on the Apply button as shown in the screen below:



3. A new popup window appears. Click on the **Apply** button.

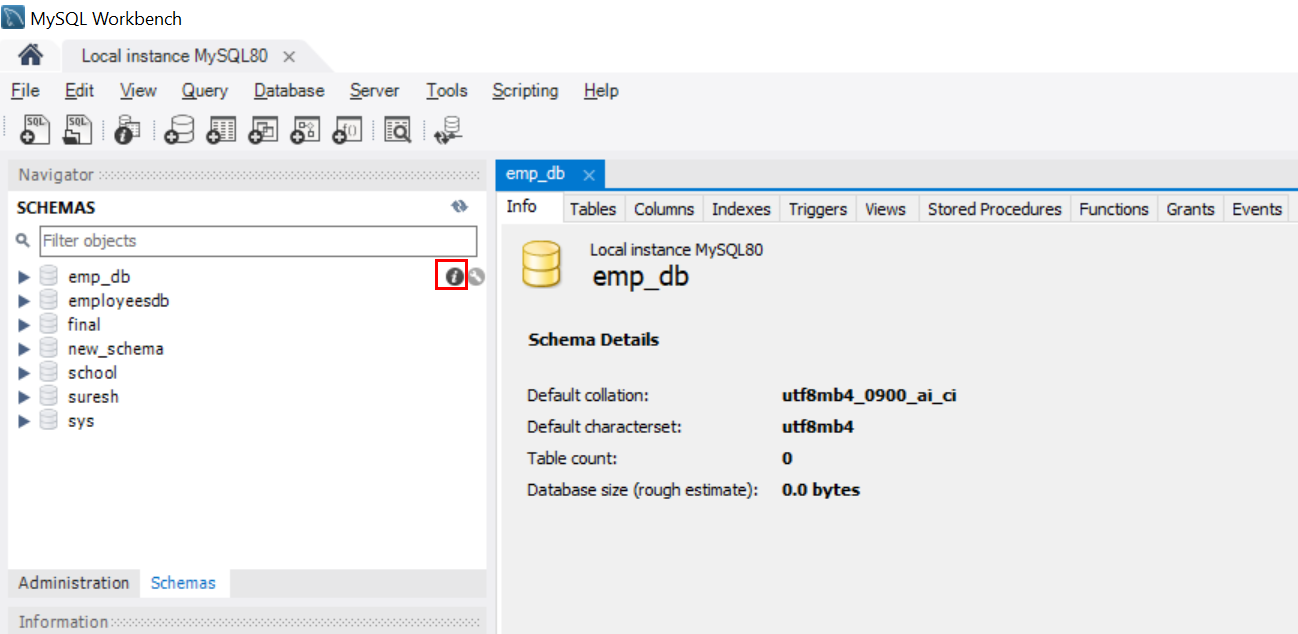


4. A new popup screen appears. Click on the **Finish** button to complete the database creation.



5. After successful database creation, we can see new databases in the Schema menu. If we do not see this, click on the **refresh icon** into the Schema menu.

6. We can see more information about the database by selecting the database and click on the **'i'** icon. The information window displays several options, like Table, Triggers, Indexes, Users, and many more.



7. MySQL Workbench does not provide an option to rename the database name, but we can create, update, and delete the table and data rows from the database.

# **MySQL SELECT Database**

SELECT Database is used in MySQL to select a particular database to work with. This query is used when multiple databases are available with MySQL Server.

You can use SQL command **USE** to select a particular database.

**Syntax:**

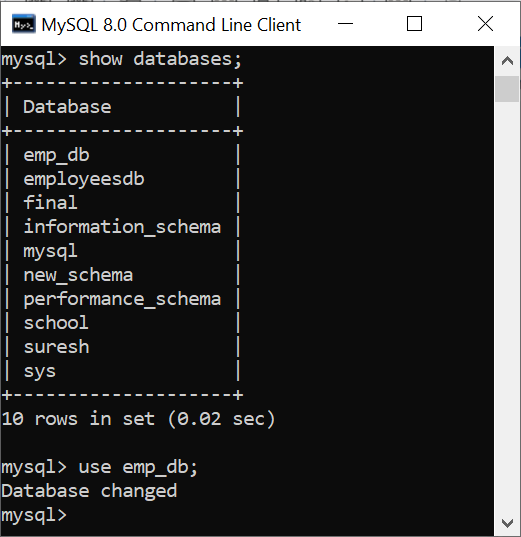
**USE database\_name;**

**Example:**

Let's take an example to use a database name "emp\_db".

**USE emp\_db;**

It will look like this:



#### **Note: All the database names, table names and table fields name are case sensitive. You must have to use proper names while giving any SQL command.**

# MySQL Show/List Databases

When we work with the MySQL server, it is a common task to show or list the databases, displaying the table from a particular database, and information of user accounts and their privileges that reside on the server. we are going to focus on how to list databases in the MySQL server.

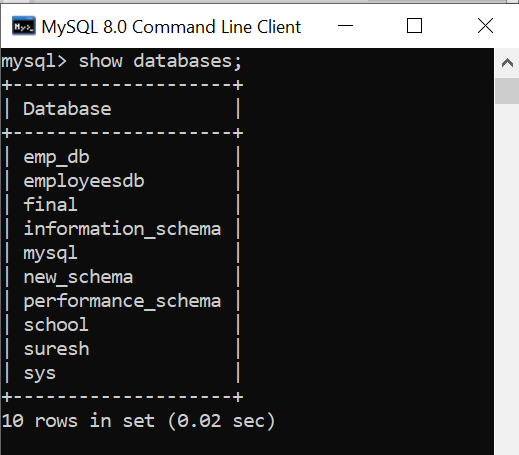
We can list all the databases available on the MySQL server host using the following command, as shown below:

**mysql> SHOW DATABASES;**

Open the MySQL Command Line Client that appeared with a **mysql> prompt**. Next, **log in** to the MySQL database server using the **password** that you have created during the installation of MySQL. Now, you are connected to the MySQL server host, where you can execute all the SQL statements. Finally, run the SHOW Databases command to list/show databases.

We can see the following output that explains it more clearl1.1KOOPs Concepts in

MySQL also allows us another command to list the databases, which is a **SHOW SCHEMAS** statement. This command is the synonyms of the SHOW DATABASES and gives the same result. We can understand it with the following output:



### **List Databases Using Pattern Matching**

Show Databases command in MySQL also provides an option that allows us to **filter** the returned database using different pattern matching with **LIKE** and **WHERE** clause. The LIKE clause lists the database name that matches the specified pattern. The WHERE clause provides more flexibility to list the database that matches the given condition in the SQL statement.

**Syntax**

The following are the syntax to use pattern matching with Show Databases command:

**mysql> SHOW DATABASES LIKE pattern;**

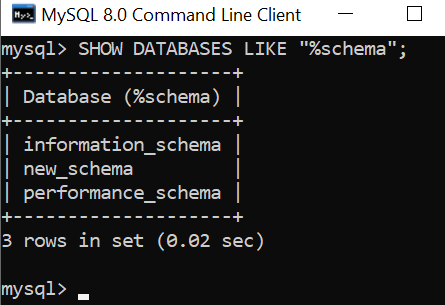
**OR,**

**mysql> SHOW DATABASES**WHERE**expression;**

We can understand it with the example given below where **percent (%) sign** assumes zero, one, or multiple characters:

mysql> SHOW DATABASES LIKE "%schema";

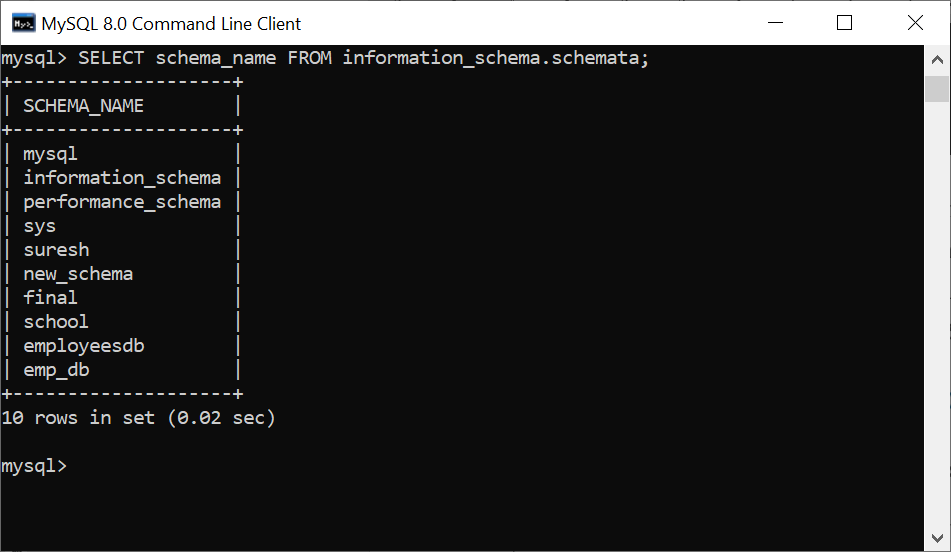
The above statement will give the following output:



Sometimes the LIKE clause is not sufficient; then, we can make a more complex search to query the database information from the schemata table in the information schema. The information schema in MySQL is an information database so that we can use it to get the output using the SHOW DATABASES command.

**mysql>**SELECT**schema\_name**FROM**information\_schema.schemata;**

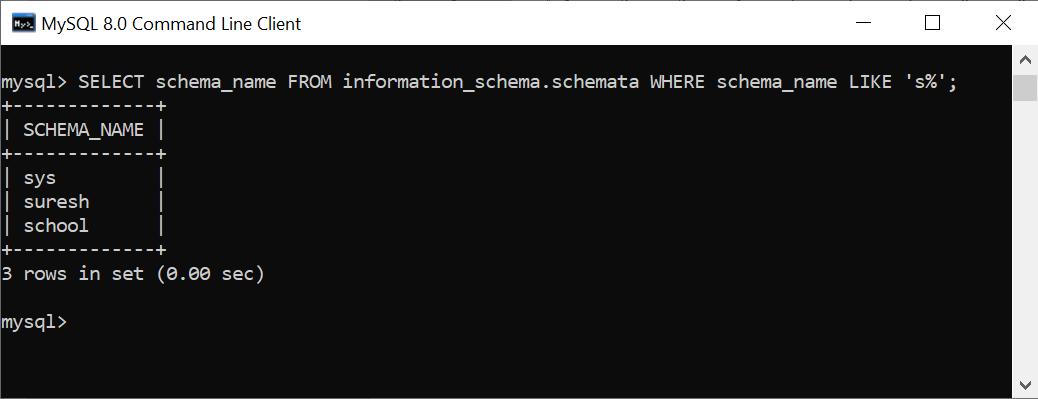
This statement will give the same result as the SHOW DATABASES command:



Now, we are going to see how we can use the WHERE clause with the SHOW DATABASES command. This statement returns the database whose schema name starts with **"s":**

**mysql>**SELECT**schema\_name**FROM**information\_schema.schemata**WHERE**schema\_name LIKE 's%';**

It will give the following output:



#### **NOTE: It is to be noted that if the MySQL server started with the "--skip-show-database" option, we could not use the SHOW DATABASES command unless we have the SHOW DATABASES privilege.**

# **MySQL DROP Database**

We can drop/delete/remove a MySQL database quickly with the MySQL DROP DATABASE command. It will delete the database along with all the tables, indexes, and constraints permanently. Therefore, we should have to be very careful while removing the database in MySQL because we will lose all the data available in the database. If the database is not available in the MySQL server, the DROP DATABASE statement throws an error.

allows us to drop/delete/remove a database mainly in **two ways:**

* MySQL Command Line Client
* MySQL Workbench

### **MySQL Command Line Client**

We can drop an existing database in MySQL by using the DROP DATABASE statement with the below syntax:

DROPDATABASE**[IF EXISTS] database\_name;**

In MySQL, we can also use the below syntax for deleting the database. It is because the **schema** is the synonym for the database, so we can use them interchangeably

DROPSCHEMA**[IF EXISTS] database\_name;**

**Parameter Explanation**

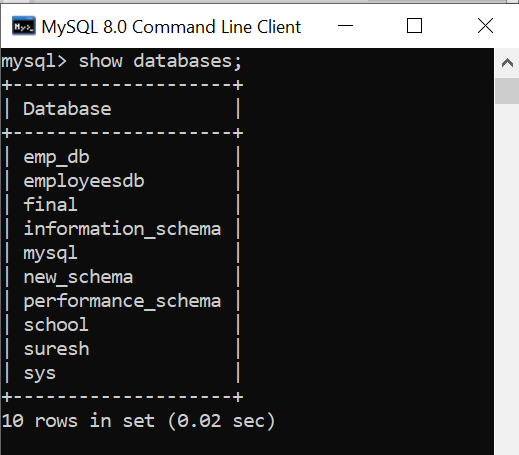
The parameter descriptions of the above syntax are as follows:

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| database\_name | It is the name of an existing database that we want to delete from the server. It should be unique in the MySQL server instance. |
| IF EXISTS | It is optional. It is used to prevent from getting an error while removing a database that does not exist. |

**Example**

Let us understand how to drop a database in MySQL with the help of an example. Open the MySQL console and write down the password, if we have set during installation. Now we are ready to delete a database.

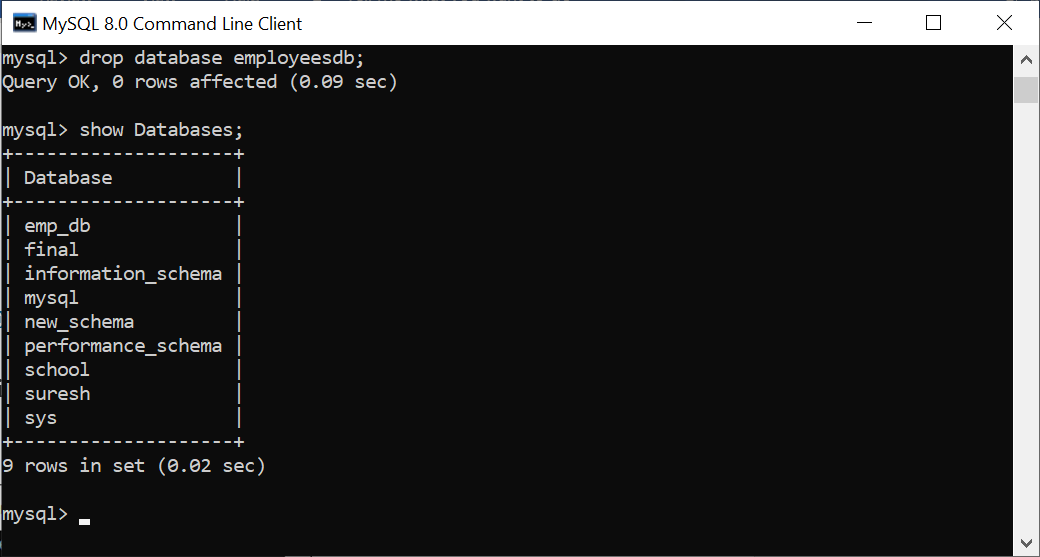
Next, use the **SHOW DATABASES** statement to see all available database in the server:



Suppose we want to remove a database named **"employeesdb"**. Execute the below statement:

DROP DATABASE **employeesdb**;

Now we can verify that either our database is removed or not by executing the following query. It will look like this:



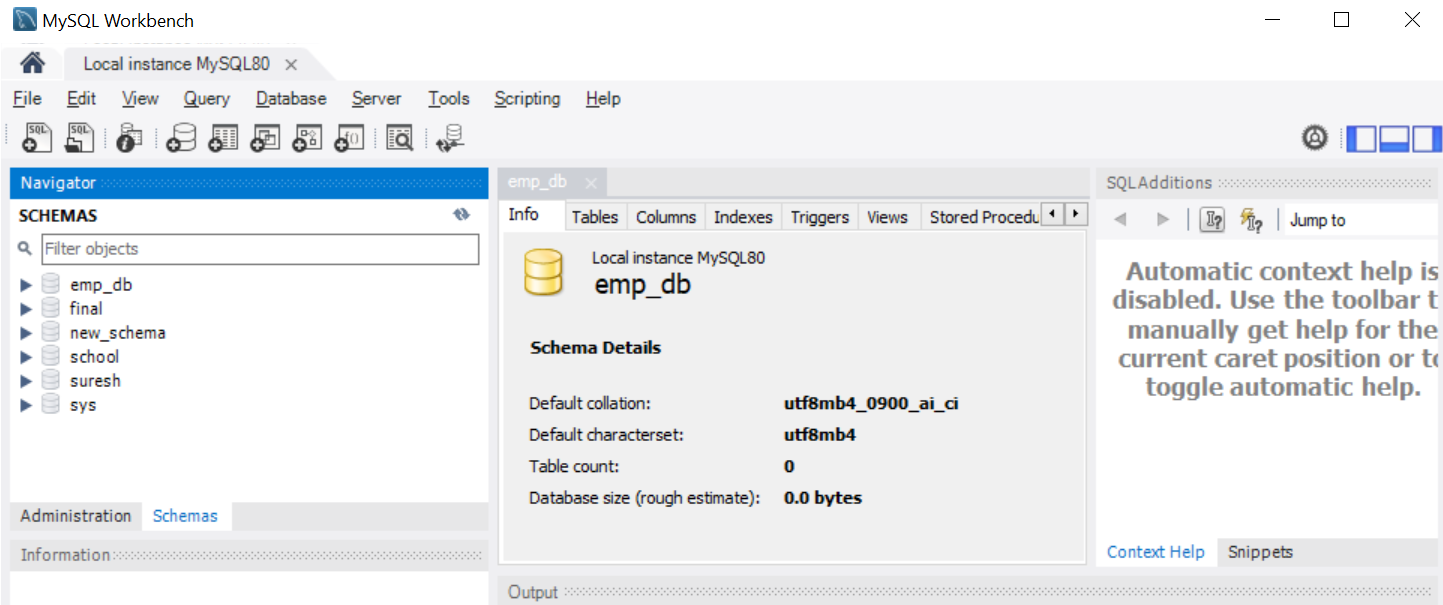
From the above, we can see that the database "**employeesdb** " is removed successfully.

#### **Note: All the database names, table names, and table field names are case sensitive. We must have to use proper names while giving any SQL command.**

### **DROP Database using MySQL Workbench**

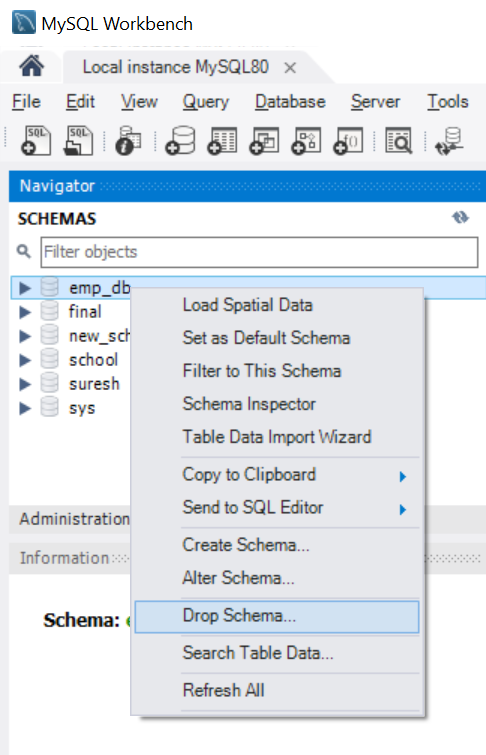
To drop a database using this tool, we first need to launch the [MySQL Workbench](https://www.javatpoint.com/mysql-workbench)

and log in with the **username** and **password** to the MySQL server. It will show the following screen:



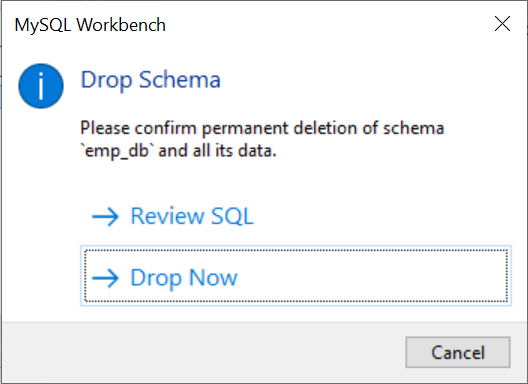
Now do the following steps for database deletion:

1. Go to the Navigation tab and click on the **Schema menu**. Here, we can see all the previously created databases. If we want to delete a database, right-click the database that you want to remove, for example, **emp\_db** under the Schema menu and select **Drop Schema** option, as shown in the following screen.

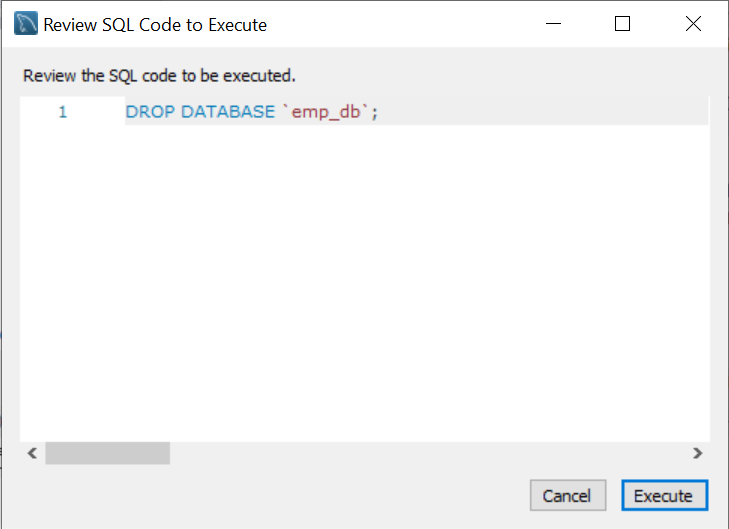


When we click the Drop Schema option, MySQL Workbench displays a dialog box to confirm the deletion process. If we select **Review SQL**, it will produce the **SQL**

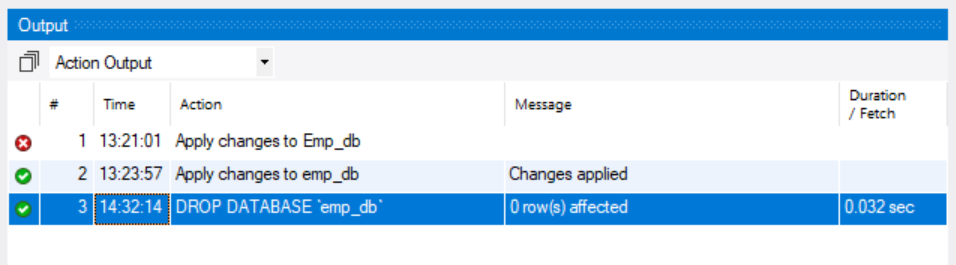
statement that will be executed. And if we choose **Drop Now** option, the database will be deleted permanently.



If we want the safe deletion of the database, it is required to choose the **Review SQL** option. Once we sure, click the Execute button to execute the statement. The below screen explains it more clearly:



Once we click the execute button, MySQL will return the below message indicating that the database is dropped successfully. Since the database **emp\_db** is an empty database, the number of affected rows is zero.



If we verify the schemas tab, we will not find the **emp\_db** database on the list anymore.

# **MySQL COPY Database**

A database is an application used for storing the organized collection of records that can be accessed and manage by the user. It holds the data into tables, rows, columns, and indexes to quickly find the relevant information.

MySQL copy or clone database is a feature that allows us to create a **duplicate copy of an existing database**, including the table structure, indexes, constraints, default values, etc. Making a duplicate copy of an original database into a new database is very useful when accidentally our database is **lost or failure**. The most common use of making a duplicate copy of the database is for data backups. It is also useful when planning the major changes to the structure of the original database.

In **MySQL**, making the clone of an original database is a **three-step process:** First, the original database records are dumped (copied) to a temporary file that holds the SQL commands for reinserting the data into the new database. Second, it is required to create a new database. Finally, the **SQL**file is processed, and the data will be copied into the new database.

We need to follow these steps to copy a database to another database:

1. First, use the **CREATE DATABASE** statement to create a new database.
2. Second, store the data to an **SQL file**. We can give any name to this file, but it must end with a **.sql** extension.
3. Third, export all the database objects along with its data to copy using the **mysqldump** tool and then import this file into the new database.

For the demonstration, we will copy the **emp\_db** database to **emp\_db\_copy** database using the following steps:

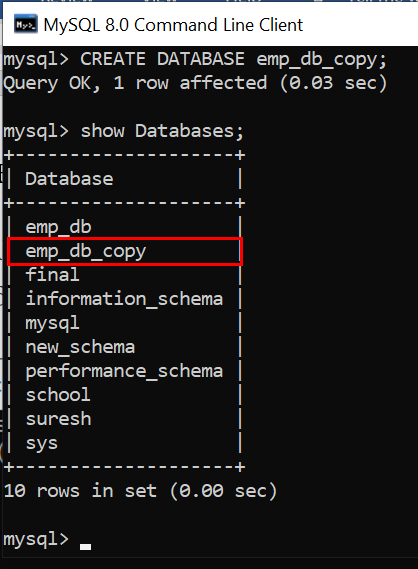
Open the MySQL console and write down the password, if we have set during installation. Now we are ready to create a duplicate database of testdb using the command below:

**mysql>**CREATEDATABASE**emp\_db\_copy;**

Next, use the SHOW DATABASES statement for verification:

**mysql> SHOW DATABASES;**

This command will return all available database in the server where we can see the newly created database in red rectangle box:



Now, open a DOS or terminal window to access the MySQL server on the command line. For example, if we have installed the MySQL in the **C folder**, copy the following folder and paste it in our DOS command. Then, press the **Enter** key.

C:\Windows\system32> CD C:\Program Files\MySQL\MySQL Server 8.0\bin

In the next step, we need to use the mysqldump tool to copy the database objects and data into the SQL file. Suppose we want to dump (copy) the database objects and data of the emp\_db into an SQL file located at **C:\Desktop\Database\_backup folder.** To do this, execute the below statement:

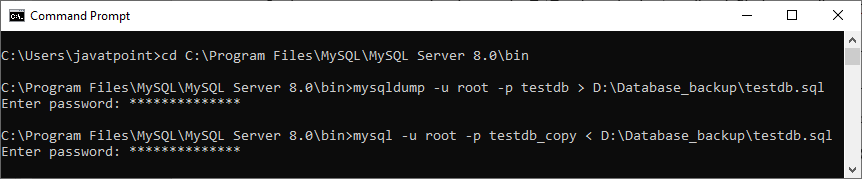
1. mysqldump -u root -p emp\_db> C:\Desktop\Database\_backup\emp\_db.sql
2. Enter password: \*\*\*\*\*\*\*\*\*\*

The above statement instructs mysqldump tool to log in to the MySQL database server using the username and password and then exports the database objects and data of the testdb database to **C:\Desktop\Database\_backup\emp\_db.sql.** It is to note that the operator (>) used for exporting the database from one location to another.

In the next step, we need to import the C:\Desktop\Database\_backup\emp\_db.sql file into emp\_db\_copy database. To do this, execute the below statement:

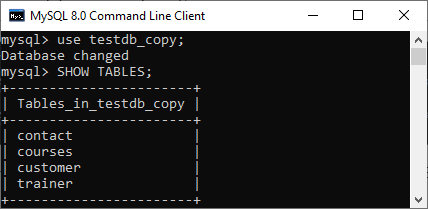
1. mysql -u root -p emp\_db\_copy < c:\desktop\Database\_backup\emp\_db.sql
2. Enter password: \*\*\*\*\*\*\*\*\*\*

It is to note that the operator (<) used for importing the database from one location to another.



Finally, we can verify whether the above operation is successful or not by using the **SHOW TABLES** command in the MySQL command-line tool:

1. mysql> SHOW TABLES;



In this output, we can see that all the objects and data from the testdb database to emp\_db\_copy database have successfully copied.

**MySQL CREATE TABLE**

A table is used to organize data in the form of rows and columns and used for both storing and displaying records in the structure format. It is similar to worksheets in the spreadsheet application. A table creation command requires **three things**:

* Name of the table
* Names of fields
* Definitions for each field

MySQL allows us to create a table into the database mainly in **two ways**:

1. MySQL Command Line Client
2. MySQL Workbench

**MySQL Command Line Client**

**MySQL** allows us to create a table into the database by using the **CREATE TABLE** command. Following is a generic **syntax** for creating a MySQL table in the database.

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

    column\_definition1,

    column\_definition2,

    ........,

    table\_constraints

);

**Parameter Explanation**

11.9K

Eve Cam: P\\\\\rotect Your Home AND Your PRIVACY! Best Smart Home Camera?!

The parameter descriptions of the above syntax are as follows:

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| database\_name | It is the name of a new table. It should be unique in the MySQL database that we have selected. The **IF NOT EXIST** clause avoids an error when we create a table into the selected database that already exists. |
| column\_definition | It specifies the name of the column along with data types for each column. The columns in table definition are separated by the comma operator. The syntax of column definition is as follows: **column\_name1 data\_type(size) [NULL | NOT NULL]** |
| table\_constraints | It specifies the table constraints such as PRIMARY KEY, UNIQUE KEY, FOREIGN KEY, CHECK, etc. |

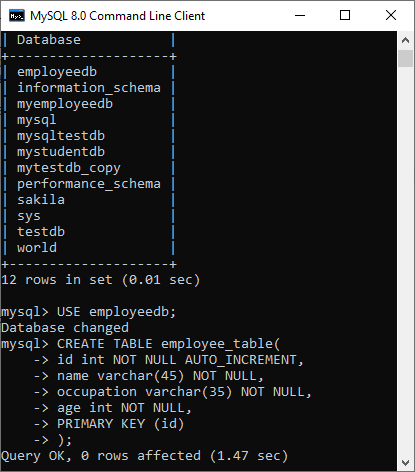
**Example**

Let us understand how to create a table into the database with the help of an example. Open the MySQL console and write down the password, if we have set during installation. Now open the database in which you want to create a table. Here, we are going to create a table name **"employee\_table"** in the database **"employeedb"** using the following statement:

1. mysql> **CREATE** **TABLE** employee\_table(
2. id **int** NOT NULL AUTO\_INCREMENT,
3. **name** **varchar**(45) NOT NULL,
4. occupation **varchar**(35) NOT NULL,
5. age **int** NOT NULL,
6. **PRIMARY** **KEY** (id)
7. );

NOTE:  
1. Here, NOT NULL is a field attribute, and it is used because we don't want this field to be NULL. If we try to create a record with a NULL value, then MySQL will raise an error.  
2. The field attribute AUTO\_INCREMENT specifies MySQL to go ahead and add the next available number to the id field. PRIMARY KEY is used to define a column's uniqueness. We can use multiple columns separated by a comma to define a primary key.

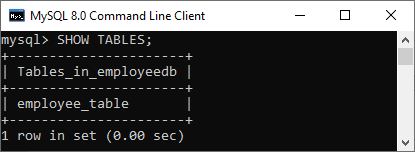
**Visual representation of creating a MySQL table:**



We need to use the following command to see the newly created table:

1. mysql> SHOW TABLES;

It will look like the below output:

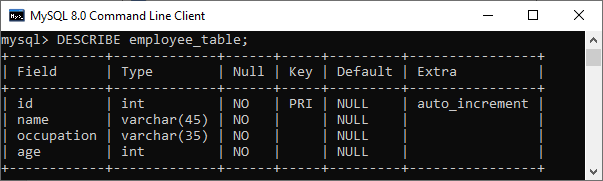


**See the table structure:**

We can use the following command to see the information or structure of the newly created table:

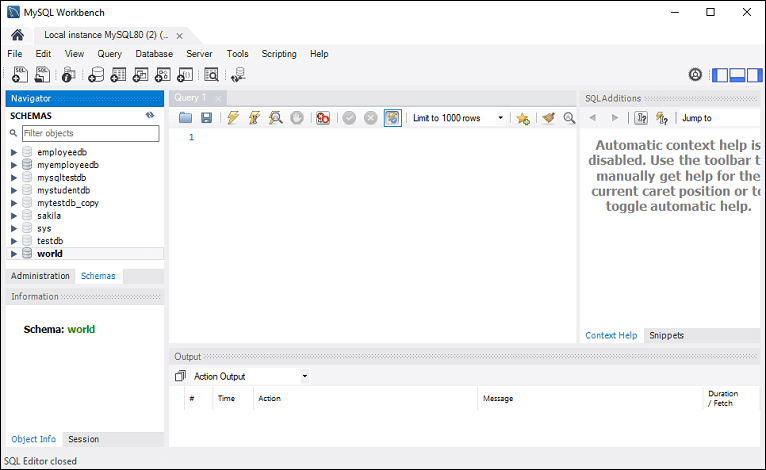
1. mysql> DESCRIBE employee\_table;

It will look like this:



**Create Table Using MySQL Workbench**

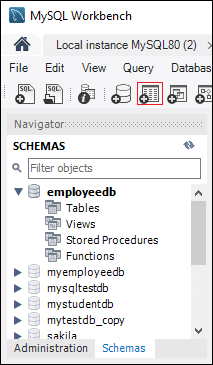
It is a visual GUI tool used to create databases, tables, indexes, views, and stored procedures quickly and efficiently. To create a new database using this tool, we first need to launch the **MySQL Workbench** and log in using the username and password that you want. It will show the following screen:



Now do the following steps for table creation:

1. Go to the Navigation tab and click on the **Schema menu**. Here, we can see all the previously created databases. Now we are ready to select the database in which a table is created.

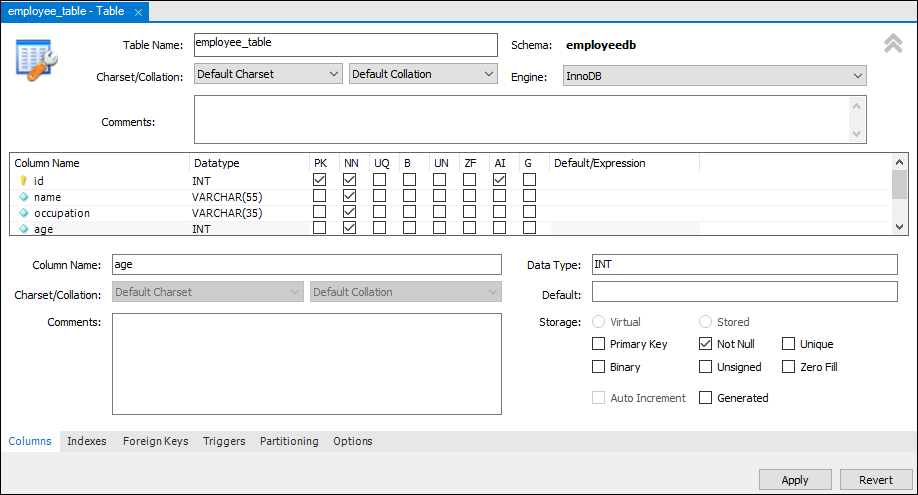
2. Select the database, double click on it, and we will get the sub-menu under the database. These **sub-menus** are Tables, Views, Functions, and Stored Procedures, as shown in the below screen.



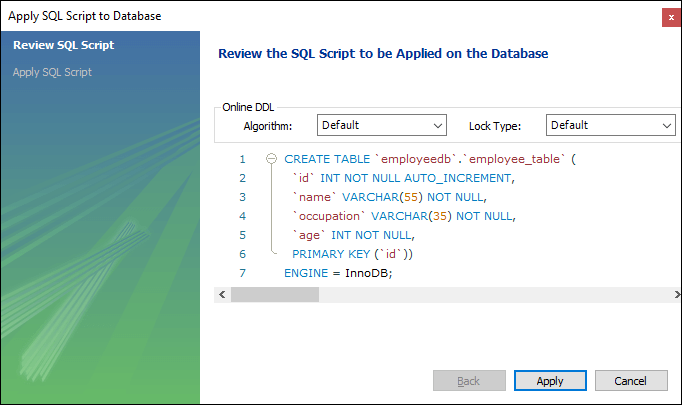
3. Select Tables sub-menu, right-click on it, and select **Create Table** option. We can also click on create a new table icon (shown in red rectangle) to create a table.

4. On the new table screen, we need to fill all the details to create a table. Here, we will enter the table name (**for example**, employee\_table) and use default collation and engine.

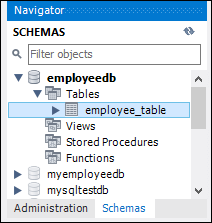
5. Click inside the middle window and fill the column details. Here, the column name contains many attributes such as Primary Key(PK), Not Null (NN), Unique Index (UI), Binary(B), Unsigned Data type(UN), Auto Incremental (AI), etc. The following screen explains it more clearly. After filling all the details, click on the **Apply** button.



6. As soon as you click on the Apply button, it will open the SQL statement window. Again, click on the Apply button to execute the statement and **Finish** button to save the changes.



7. Now, go to the Schema menu and select the database which contains the newly created table, as shown in the screen below.



# MySQL ALTER Table

MySQL ALTER statement is used when you want to change the name of your table or any table field. It is also used to add or delete an existing column in a table.

The ALTER statement is always used with "ADD", "DROP" and "MODIFY" commands according to the situation.

## 1) ADD a column in the table

**Syntax:**

1. ALTER TABLE table\_name
2. ADD new\_column\_name column\_definition
3. [ FIRST | AFTER column\_name ];

## Parameters

**table\_name:** It specifies the name of the table that you want to modify.

Keep Watching

Competitive questions on Structures in Hindi00:00/03:34

**new\_column\_name:** It specifies the name of the new column that you want to add to the table.

**column\_definition:** It specifies the data type and definition of the column (NULL or NOT NULL, etc).

**FIRST | AFTER column\_name:** It is optional. It tells MySQL where in the table to create the column. If this parameter is not specified, the new column will be added to the end of the table.

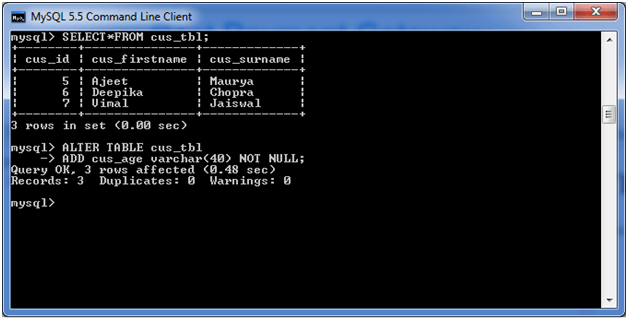
**Example:**

In this example, we add a new column "cus\_age" in the existing table "cus\_tbl".

Use the following query to do this:

1. ALTER TABLE cus\_tbl
2. ADD cus\_age varchar(40) NOT NULL;

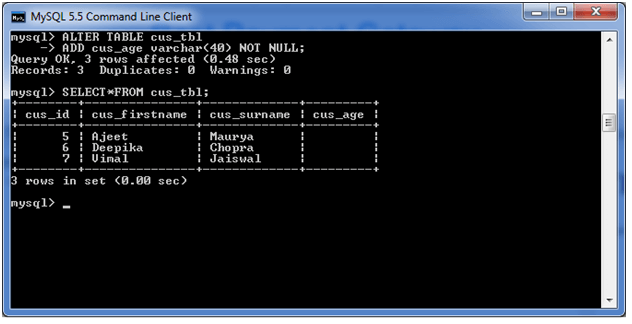
**Output:**



**See the recently added column:**

1. SELECT\* FROM cus\_tbl;

**Output:**



## 2) Add multiple columns in the table

**Syntax:**

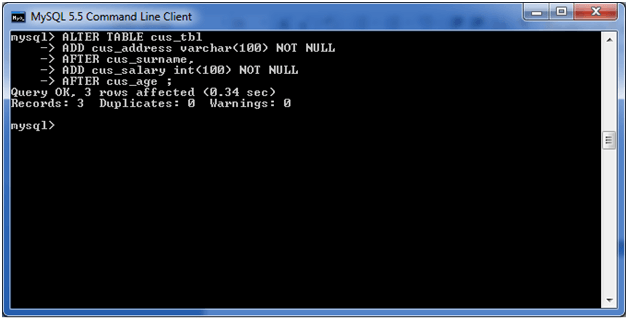
1. ALTER TABLE table\_name
2. ADD new\_column\_name column\_definition
3. [ FIRST | AFTER column\_name ],
4. ADD new\_column\_name column\_definition
5. [ FIRST | AFTER column\_name ],
6. ...
7. ;

**Example:**

In this example, we add two new columns "cus\_address", and cus\_salary in the existing table "cus\_tbl". cus\_address is added after cus\_surname column and cus\_salary is added after cus\_age column.

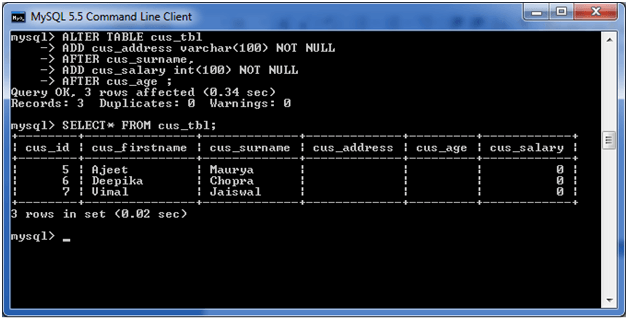
**Use the following query to do this:**

1. ALTER TABLE cus\_tbl
2. ADD cus\_address varchar(100) NOT NULL
3. AFTER cus\_surname,
4. ADD cus\_salary int(100) NOT NULL
5. AFTER cus\_age ;



**See the recently added columns:**

1. SELECT\* FROM cus\_tbl;



## 3) MODIFY column in the table

The MODIFY command is used to change the column definition of the table.

**Syntax:**

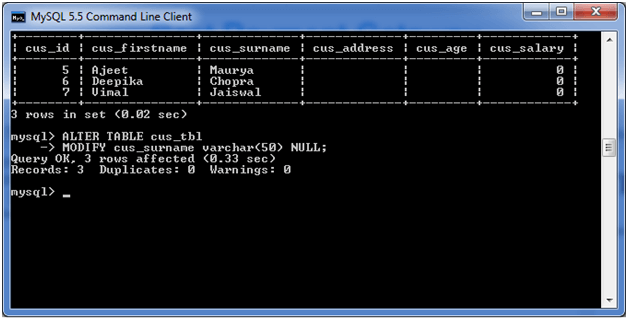
1. ALTER TABLE table\_name
2. MODIFY column\_name column\_definition
3. [ FIRST | AFTER column\_name ];

**Example:**

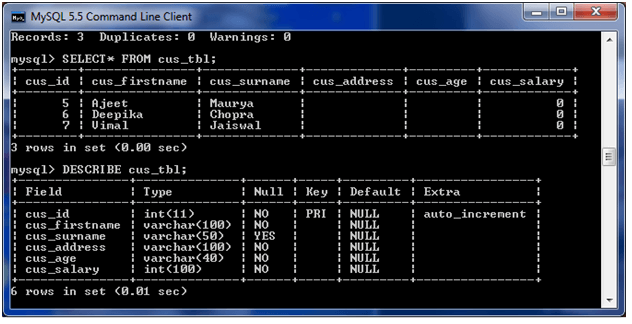
In this example, we modify the column cus\_surname to be a data type of varchar(50) and force the column to allow NULL values.

**Use the following query to do this:**

1. ALTER TABLE cus\_tbl
2. MODIFY cus\_surname varchar(50) NULL;



**See the table structure:**



## 4) DROP column in table

**Syntax:**

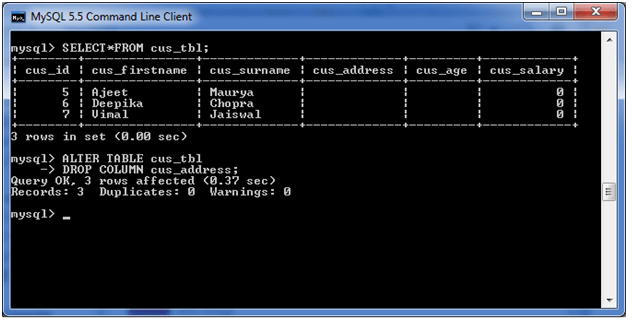
1. ALTER TABLE table\_name
2. DROP COLUMN column\_name;

Let's take an example to drop the column name "cus\_address" from the table "cus\_tbl".

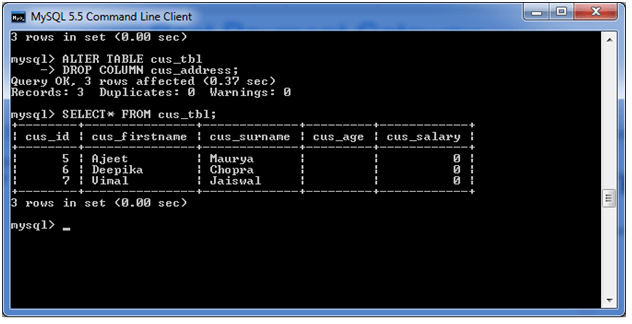
**Use the following query to do this:**

1. ALTER TABLE cus\_tbl
2. DROP COLUMN cus\_address;

**Output:**



**See the table structure:**



## 5) RENAME column in table

**Syntax:**

1. ALTER TABLE table\_name
2. CHANGE COLUMN old\_name new\_name
3. column\_definition
4. [ FIRST | AFTER column\_name ]

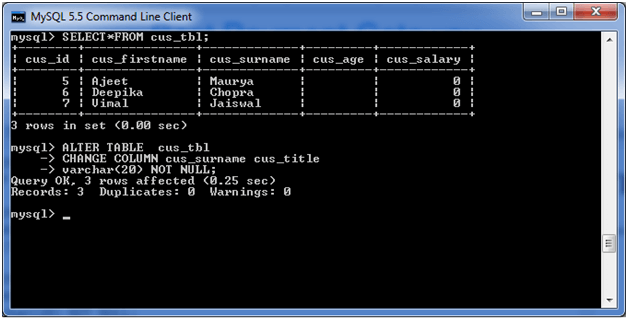
**Example:**

In this example, we will change the column name "cus\_surname" to "cus\_title".

**Use the following query to do this:**

1. ALTER TABLE  cus\_tbl
2. CHANGE COLUMN cus\_surname cus\_title
3. varchar(20) NOT NULL;

**Output:**



## 6) RENAME table

**Syntax:**

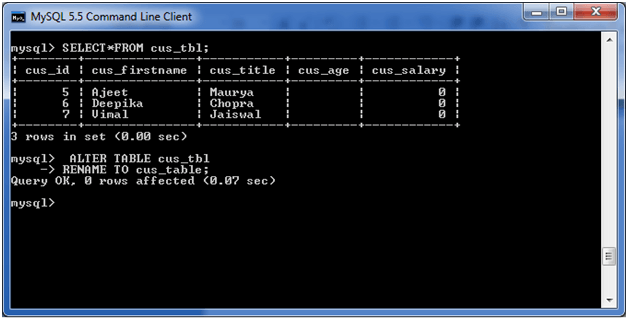
1. ALTER TABLE table\_name
2. RENAME TO new\_table\_name;

**Example:**

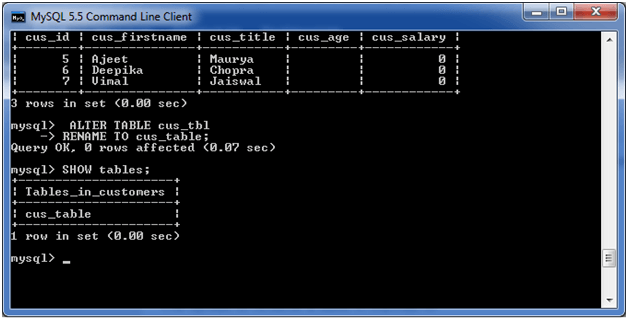
In this example, the table name cus\_tbl is renamed as cus\_table.

1. ALTER TABLE cus\_tbl
2. RENAME TO cus\_table;

**Output:**



**See the renamed table:**



**MySQL Show/List Tables**

The show or list table is very important when we have many databases that contain various tables. Sometimes the table names are the same in many databases; in that case, this query is very useful. We can get the number of table information of a database using the following statement:

**mysql> SHOW TABLES;**

The following steps are necessary to get the list of tables:

**Step 1:** Open the MySQL Command Line Client that appeared with a **mysql> prompt**. Next, **log in** to the MySQL database server using the **password** that you have created during the installation of MySQL. Now, you are connected to the MySQL server, where you can execute all the SQL statements.

**Step 2:** Next, choose the specific database by using the command below:

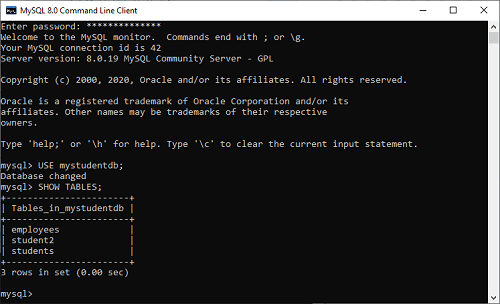
**mysql> USE database\_name;**

**Step 3:** Finally, execute the SHOW TABLES command.

Let us understand it with the example given below. Suppose we have a database name "**mystudentdb**" that contains many tables. Then execute the below statement to list the table it contains:

1. mysql> USE mystudentdb;
2. mysql>SHOW TABLES;

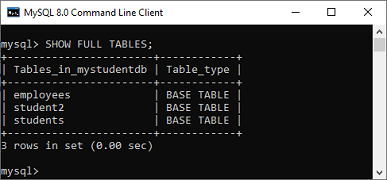
The following output explains it more clearly:



We can also use the **FULL modifier** with the SHOW TABLES query to get the type of table (Base or View) that appears in a second output column.

1. mysql> SHOW **FULL** TABLES;

This statement will give the following output:



If we want to show or list the table name from different databases or database to which you are not connected without switching, MySQL allows us to use the FROM or IN clause followed by the database name. The following statement explains it more clearly:

1. mysql> SHOW TABLES IN database\_name;

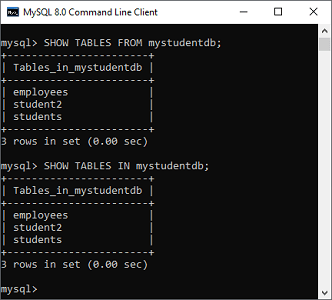
The above statement can also be written as:

1. mysql> SHOW TABLES **FROM** database\_name;

When we execute the below statements, we will get the same result:

1. mysql> SHOW TABLES **FROM** mystudentdb;
2. OR,
3. mysql> SHOW TABLES IN mystudentdb;

**Output:**



**Show Tables Using Pattern Matching**

Show Tables command in MySQL also provides an option that allows us to **filter** the returned table using different pattern matching with LIKE and WHERE clause.

**Syntax**

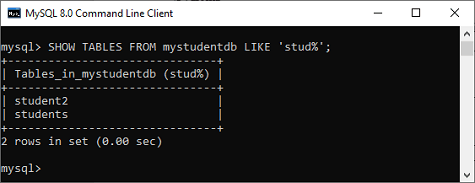
The following are the syntax to use pattern matching with show table command:

1. mysql> SHOW TABLES LIKE pattern;
2. OR,
3. mysql> SHOW TABLES **WHERE** expression;

We can understand it with the example given below where percent (%) sign assumes zero, one, or multiple characters:

1. mysql> SHOW TABLES **FROM** mystudentdb LIKE "stud%";

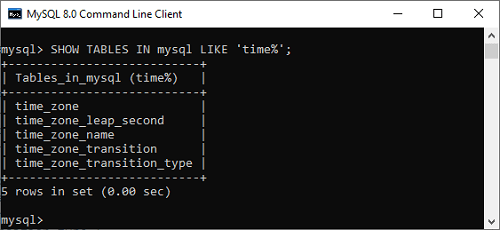
The above statement will give the following output:



Let us see another statement that returned the table names starting with **"time"**:

1. mysql> SHOW TABLES IN mysql LIKE "time%";

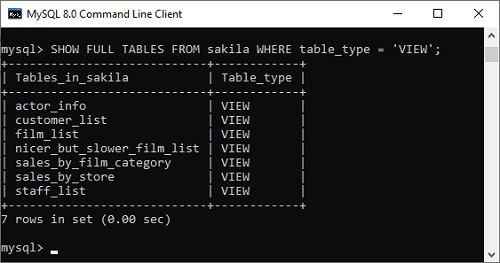
The above query will give the following output:



Now, we are going to see how we can use the **WHERE** clause with the SHOW TABLES command to list different types of tables (either Base or View type) in the selected database:

1. mysql> SHOW TABLES **FROM** sakila **WHERE** table\_type= "VIEW";

This statement gives the below output:

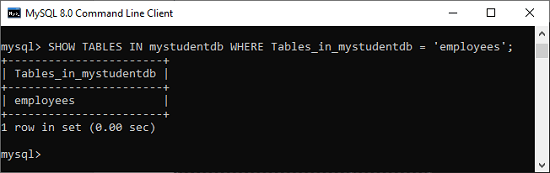


It is noted that if MySQL does not provide the privileges for accessing a Base table or view, then we cannot get the tables in the result set of the SHOW TABLES command.

Here, we can also see another example of Show Tables statement with the WHERE clause:

1. mysql> SHOW TABLES In mystudentdb **WHERE** Tables\_in\_mystudentdb= "employees";

It will give the following output:



MySQL Rename Table

Sometimes our table name is non-meaningful, so it is required to rename or change the name of the table. MySQL provides a useful syntax that can rename one or more tables in the current database.

Syntax

The following are the syntax used to change the name of the table:

1. mysql> RENAME old\_table **TO** new\_table;

Here, we have to make sure that **new\_table\_name** must not exist, and **old\_table\_name** should be present in the database. Otherwise, it will throw an error message. It is to ensure that the table is not locked as well as there are no active transactions before executing this statement.

NOTE: If we use the RENAME TABLE statement, it is required to have ALTER and DROP TABLE privileges to the existing table. Also, this statement cannot change the name of a temporary table.

We can also use the MySQL**RENAME TABLE** statement to change more than one table name with a single statement, as shown below:

40.6M

721

Prime Ministers of India | List of Prime Minister of India (1947-2020

1. RENAME **TABLE** old\_tab1 **TO** new\_tab1,
2. old\_tab2 **TO** new\_tab2, old\_tab3 **TO** new\_tab3;

From the **MySQL 8.0.13** version, we can change the old table name locked with a LOCK statement and also uses the WRITE LOCK clause. For example, following are the valid statement:

mysql> LOCK **TABLE** old\_tab\_name1 WRITE;

RENAME **TABLE** old\_tab\_name1 **TO** new\_tab\_name1,

             new\_tab\_name1 **TO** new\_tab\_name2;

Following statement are not permitted

1. mysql> LOCK **TABLE** old\_tab\_name1 **READ**;
2. RENAME **TABLE** old\_tab\_name1 **TO** new\_tab\_name1,
3. new\_tab\_name1 **TO** new\_tab\_name2;

Before MySQL 8.0.13 version, we cannot change the table name that was locked with the LOCK TABLE statement.

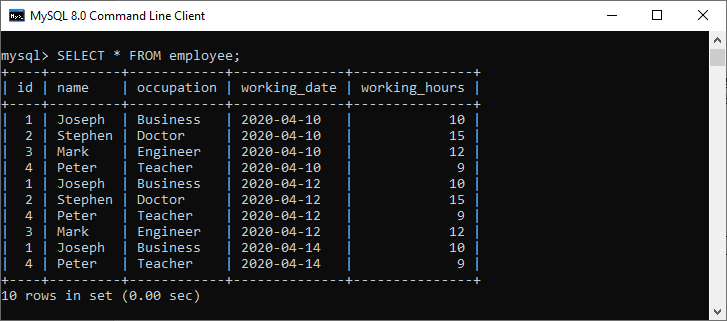
MySQL also use the RENAME TABLE statement for moving a table from one database to other database, which is show below:

1. mysql> RENAME **TABLE** current\_db.tablel\_name **TO** other\_db.tablel\_name;

MySQL RENAME TABLE Example

Let us understand how the RENAME TABLE statement works in MySQL through the various examples. Suppose we have a table named **EMPLOYEE**, and due to some reason, there is a need to change it into the table named **CUSTOMER**.

**Table Name: employee**

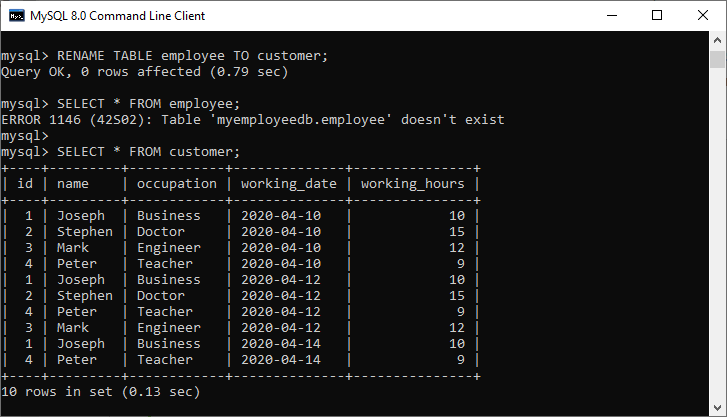


Next, execute the following syntax to change the table name:

1. mysql> RENAME employee **TO** customer;

**Output**

We will see that the table named "employee" will be changed into a new table name "customer":

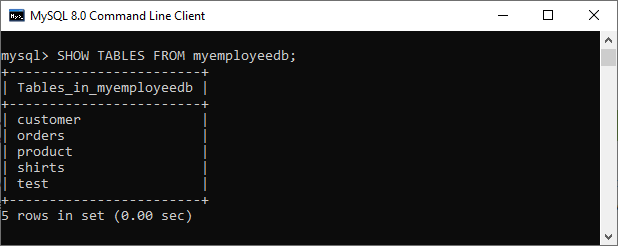


In the above output, we can see that if we use the table name employee after executing a RENAME TABLE statement, it will throw an error message.

**How to RENAME Multiple Tables**

RENAME TABLE statement in MySQL also allows us to change more than one table name within a single statement. See the below statement:

Suppose our database "**myemployeedb**" having the following tables:

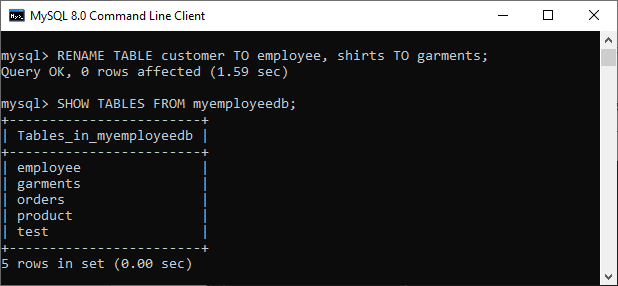


If we want to change the table name customer into employee and table name shirts into garments, execute the following statement:

1. mysql> RENAME **TABLE** customer **TO** employee, shirts **TO** garments;

**Output**

We can see that the table name customer into employee and table name shirts into garments have successfully renamed.



**Rename table using ALTER statement**

The ALTER TABLE statement can also be used to rename the existing table in the current database. The following are the syntax of the ALTER TABLE statement:

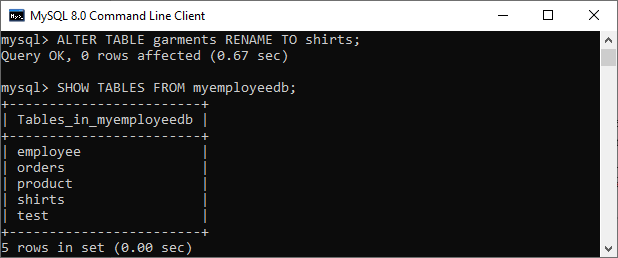
1. **ALTER** **TABLE** old\_table\_name RENAME **TO** new\_table\_name;

See the following query that changes the existing table name garments into new table name shirts:

1. mysql> **ALTER** **TABLE** garments RENAME **TO** shirts:

**Output**

Here, we can see that the table name garments renamed into table name shirts.



**How to RENAME Temporary Table**

A temporary table allows us to keep temporary data, which is visible and accessible in the current session only. So, first, we need to create a temporary table using the following statement:

1. mysql> **CREATE** **TEMPORARY** **TABLE** Students( **name** **VARCHAR**(40) NOT NULL, total\_marks **DECIMAL**(12,2) NOT NULL **DEFAULT** 0.00, total\_subjects **INT** UNSIGNED NOT NULL **DEFAULT** 0);

Next, insert values into this table:

1. mysql> **INSERT** **INTO** Students(**name**, total\_marks, total\_subjects) **VALUES** ('Joseph', 150.75, 2), ('Peter', 180.75, 2);

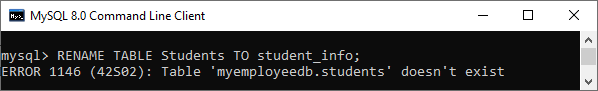
Next, run the show table command to check the temporary table:

1. mysql> **SELECT** \* **FROM** Students;

Now, run the following command to change the name of the temporary table:

1. mysql> RENAME **TABLE** Students **TO** student\_info;

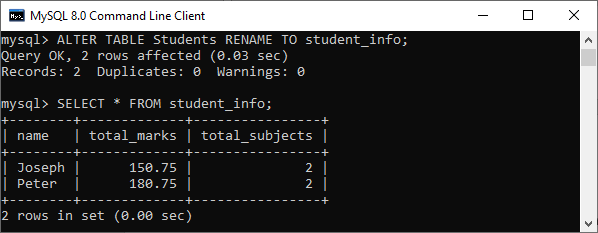
It will throw an error message, as shown below:



Thus, MySQL allows ALTER table statement to rename the temporary table:

1. mysql> **ALTER** **TABLE** Students RENAME **TO** student\_info;

**Output**



**MySQL TRUNCATE Table**

The TRUNCATE statement in MySQL removes the complete data without removing its structure. It is a part of **DDL or data definition language** **command**. Generally, we use this command when we want to delete an entire data from a table without removing the table structure.

The TRUNCATE command works the same as a DELETE command without using a [WHERE clause](https://www.javatpoint.com/mysql-where) that deletes complete rows from a table. However, the TRUNCATE command is more efficient as compared to the [DELETE](https://www.javatpoint.com/mysql-delete) command because it removes and recreates the table instead of deleting single records one at a time. Since this command internally drops the table and recreates it, the number of rows affected by the truncate statement is zero, unlike the delete statement that returns the number of deleted rows.

This command does not maintain the transaction log during the execution. It deallocates the data **pages instead of rows** and makes an entry for the deallocating pages instead of rows in transaction logs. This command also locks the pages instead of rows; thus, it requires fewer locks and resources.

The following points must be considered while using the TRUNCATE command:

2

* We cannot use the **WHERE** clause with this command so that filtering of records is not possible.
* We **cannot rollback the deleted data** after executing this command because the log is not maintained while performing this operation.
* We cannot use the truncate statement when a table is referenced by a **foreign key** or participates in an **indexed view**.
* The TRUNCATE command doesn't fire DELETE **triggers** associated with the table that is being truncated because it does not operate on individual rows.

Syntax

The following syntax explains the TRUNCATE command to remove data from the table:

1. **TRUNCATE** [**TABLE**] table\_name;

In this syntax, first, we will specify the **table name** which data we are going to remove. The TABLE keyword in the syntax is not mandatory. But it's a good practice to use it to distinguish between the **TRUNCATE**() function and the **TRUNCATE TABLE statement**.

MySQL Truncate Table Example

Let us demonstrate how we can truncate the table with the help of an example. First, we are going to create a table named "**customer**" using the below statement:

1. **CREATE** **TABLE** customer (
2. Id **int** **PRIMARY** **KEY** NOT NULL,
3. **Name** **varchar**(45) NOT NULL,
4. Product **varchar**(45) **DEFAULT** NULL,
5. Country **varchar**(25) **DEFAULT** NULL,
6. Year **int** NOT NULL
7. );

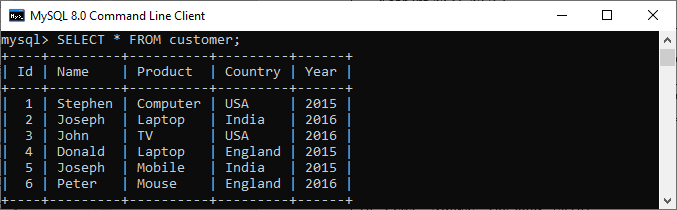
Next, we will add values to this table using the below statement:

1. **INSERT** **INTO** customer ( Id, **Name**, Product, Country, Year)
2. **VALUES** (1, 'Stephen', 'Computer', 'USA', 2015),
3. (2, 'Joseph', 'Laptop', 'India', 2016),
4. (3, 'John', 'TV', 'USA', 2016),
5. (4, 'Donald', 'Laptop', 'England', 2015),
6. (5, 'Joseph', 'Mobile', 'India', 2015),
7. (6, 'Peter', 'Mouse', 'England', 2016);

Now, verify the table by executing the [**SELECT statement**](https://www.javatpoint.com/mysql-select) whether the records inserted or not:

1. mysql> **SELECT** \* **FROM** customer;

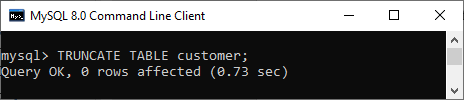
We will get the output, as shown below:



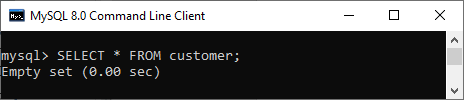
Now, execute the following statement that truncates the table customer using the TRUNCATE syntax discussed above:

1. mysql> **TRUNCATE** **TABLE** customer;

After the successful execution, we will get the following output:



As we can see, this query returns **0 rows are affected** even if all the table records are deleted. We can verify the deletion of the data by executing the SELECT statement again. This command gives the following output that shows none of the records present in the table:



How to Truncate Table with Foreign key?

If we perform the TRUNCATE operation for the table that uses a foreign key constraint, we will get the following error:

1. ERROR 1217 (23000): Cannot **delete** or **update** a parent row: a **foreign** **key** **constraint** fails

In that case, we need to log into the [MySQL](https://www.javatpoint.com/mysql-tutorial) server and **disable foreign key** checks before executing the TRUNCATE statement as below:

1. **SET** FOREIGN\_KEY\_CHECKS=0;

Now, we are able to truncate tables. After execution, **re-enable foreign key** checks as given below:

1. **SET** FOREIGN\_KEY\_CHECKS=1;

How to truncate all tables in MySQL?

The TRUNCATE statement in MySQL will delete only one table at a time. If we want to delete more than one table, we need to execute the separate TRUNCATE statement. The below example shows how to truncate multiple tables in MySQL:

1. **TRUNCATE** **TABLE** table\_name1;
2. **TRUNCATE** **TABLE** table\_name2;
3. **TRUNCATE** **TABLE** table\_name3;

We can also use the below SQL query that generates several TRUNCATE TABLE commands at once using the table names in our database:

1. **SELECT** Concat('TRUNCATE TABLE ', TABLE\_NAME)
2. **FROM** INFORMATION\_SCHEMA.TABLES
3. **WHERE** table\_schema = 'database\_name';

**MySQL DESCRIBE TABLE**

DESCRIBE means to show the information in detail. Since we have tables in MySQL, so we will use the **DESCRIBE command to show the structure of our table**, such as column names, constraints on column names, etc. The **DESC** command is a short form of the DESCRIBE command. Both DESCRIBE and DESC command are equivalent and case sensitive.

Syntax

The following are the syntax to display the table structure:

1. {DESCRIBE | **DESC**} table\_name;

**We can use the following steps to show all columns of the table**:

**Step 1:** Login into the MySQL database server.

**Step 2:** Switch to a specific database.

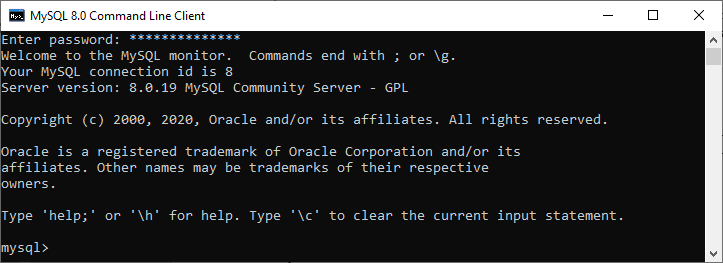
**Step 3:** Execute the DESCRIBE statement.

Let us understand it with the help of an example that explains how to show columns of the table in the selected database.

Login to the MySQL Database

The first step is to login to the database server using the **username** and **password**. We should see the output as below image:

1. >mysql -u root -p
2. Enter **password**: \*\*\*\*\*\*\*\*\*\*
3. mysql>



Switch to a Specific Database

The next step is to open a particular database from which you want to display the table information using the following query. After the execution of a query, we should see the below output:

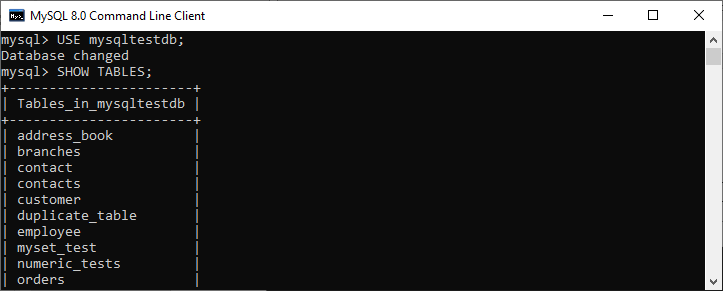
1. mysql> USE mysqltestdb;



Execute DESCRIBE Statement

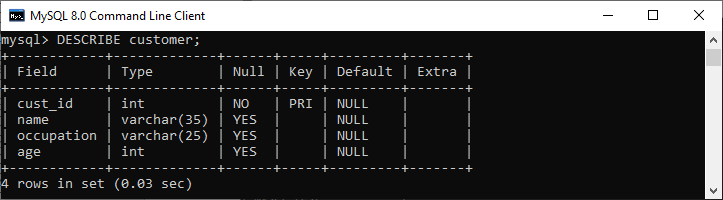
It is the last step to display the table information. Before executing the DESCRIBE statement, we can optionally display all the tables stored in our selected database with the [**SHOW TABLES** statement](https://www.javatpoint.com/mysql-show-list-tables):

1. mysql> SHOW TABLES;

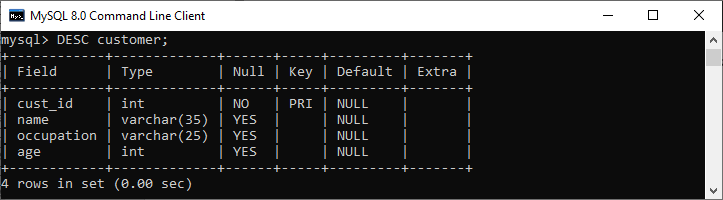


**For example**, if we want to show a **customer table's structure**, execute the below statement. After successful execution, it will give the output as below image:

1. mysql> DESCRIBE customer;

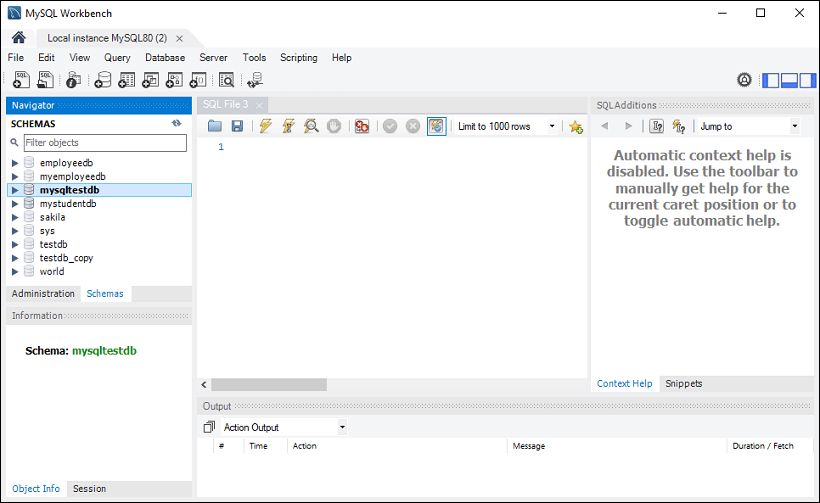


We can also use the DESC statement for practice, which is a shorthand of the DESCRIBE command. See the below output:



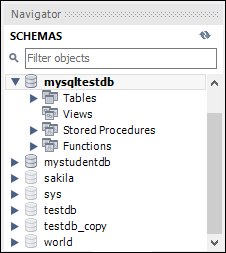
How to display table information in MySQL Workbench?

To display the column information of the table in [MySQL Workbench](https://www.javatpoint.com/mysql-workbench), we first need to launch the Workbench tool and login with the username and password to the [MySQL](https://www.javatpoint.com/mysql-tutorial) database server. We will get the following screen:

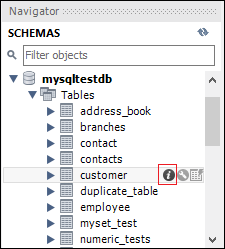


Now do the following steps to show the table information:

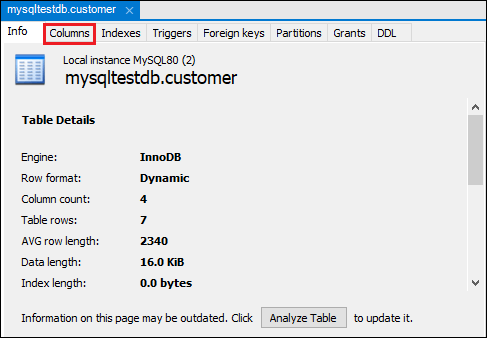
1. Go to the **Navigation tab** and click on the **Schema menu**. Here, we can see all the previously created databases. Select any database under the Schema menu, for example, **mysqltestdb**. It will pop up the multiple options that can be shown in the following image.



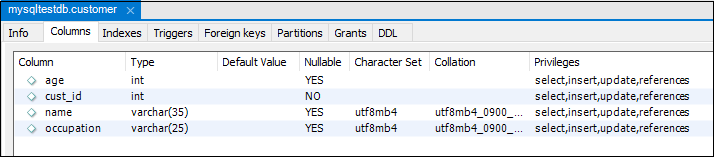
2. Next, click on the "**Tables**" that shows all tables stored in the mysqltestdb database. Select a table whose column information you want to display. Then, mouse hour on that table, it will show **three icons**. See the below image:



Now, click the **icon (i)** shown in the red rectangular box. It will display the following image:



Finally, click on the "**Columns**" menu to display the table structure.



MySQL SHOW COLUMNS Command

MySQL also allows the SHOW COLUMNS command to display table structure. It is a more flexible way to get columns information of a table.

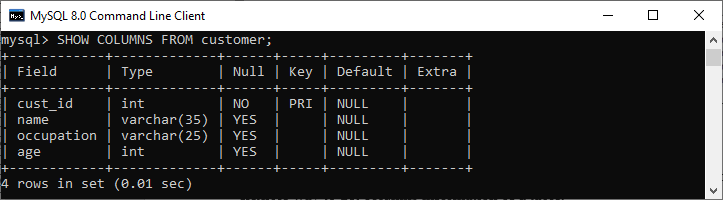
**Syntax:**

The following are the syntax of the SHOW COLUMNS command:

1. mysql> SHOW COLUMNS **FROM** table\_name;

**For example**, if we execute the below query, we will get all columns information of a table in a particular database:

1. mysql> SHOW COLUMNS **FROM** customer;



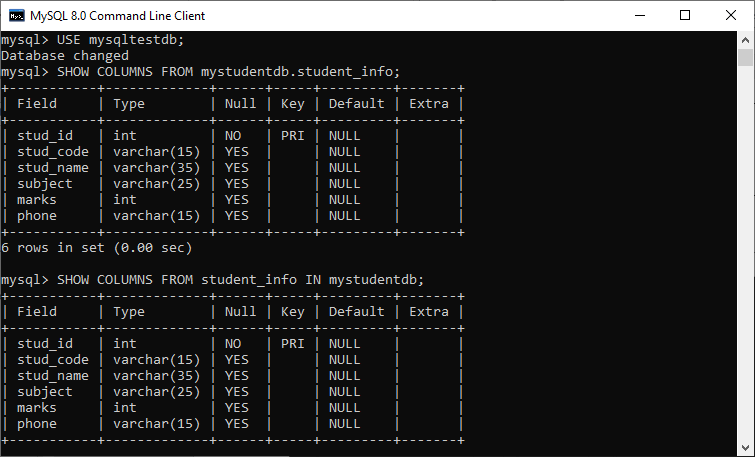
If we want to show the **columns information of a table from another database** or not available in the current database, we can use the following query:

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

OR

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

In the below image, we can see that we had used the mysqltestdb database. But we had displayed the column's information of a table from another database without switching to the current database.



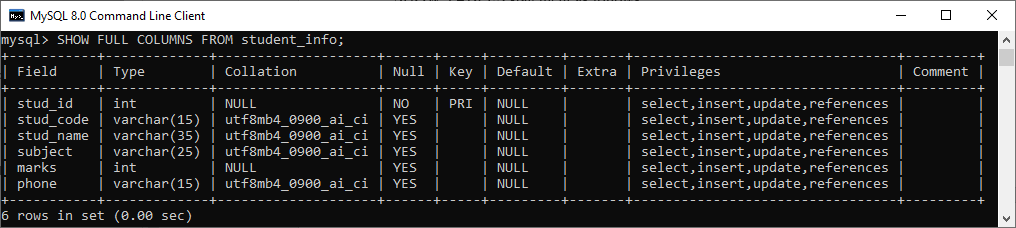
If we want to display the more column information, we need to add **FULL** keyword with the SHOW TABLES statement as follows:

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

**For example**, the below SQL query lists all columns of the **student\_info table** in the **mystudentdb database**:

mysql> SHOW **FULL** COLUMNS **FROM** student\_info;

After execution, we can see that this command adds the **collation, privileges, default**, and **comment** columns to the result set.



MySQL EXPLAIN

The EXPLAIN keyword is synonyms to the DESCRIBE statement, which is **used to obtain information about how MySQL executes the queries**. It can work with **INSERT, SELECT, DELETE, UPDATE, and REPLACE** queries. From **MySQL 8.0.19** and later versions, it can also work with TABLE statements. When we use this keyword in queries, it will process the statement and provide the information about how tables are joined, the order of the table, estimated partitions and rows.

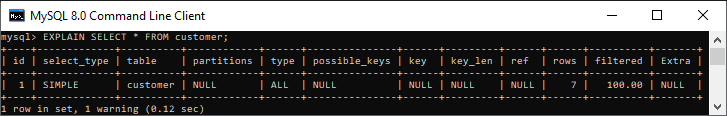
**Example**

If we want to show the execution plan of a **SELECT statement**, we can use the query as below:

1. mysql> EXPLAIN **SELECT** \* **FROM** customer;

**Output:**

This query produces the following information:



MySQL DROP Table

MYSQL uses a Drop Table statement to delete the existing table. This statement removes the complete data of a table along with the whole structure or definition permanently from the database. So, you must be very careful while removing the table because we cannot recover the lost data after deleting it.

Syntax

The following are the syntax to remove the table in MySQL:

1. mysql> **DROP** **TABLE**  table\_name;
2. OR,
3. mysql> **DROP** **TABLE**  schema\_name.table\_name;

The full syntax of DROP TABLE statement in MySQL is:

1. **DROP** [ **TEMPORARY** ] **TABLE** [ IF EXISTS ] table\_name [ **RESTRICT** | **CASCADE** ];

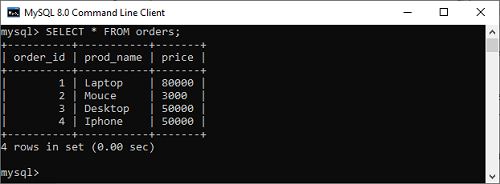
The above syntax used many parameters or arguments. Let us discuss each in detail:

|  |  |
| --- | --- |
| **Parameter Name** | **Description** |
| TEMPORARY | It is an optional parameter that specifies to delete the temporary tables only. |
| table\_name | It specifies the name of the table which we are going to remove from the database. |
| IF EXISTS | It is optional, which is used with the DROP TABLE statement to remove the tables only if it exists in the database. |
| RESTRICT and CASCADE | Both are optional parameters that do not have any impact or effect on this statement. They are included in the syntax for future versions of MySQL. |

NOTE: It is to be noted that you must have a DROP privileges to execute the DROP TABLE statement in the MySQL.

Example

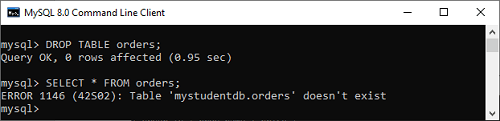
This example specifies how we can drop an existing table from the database. Suppose our database contains a table "orders" as shown in the image below:



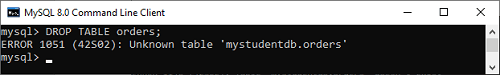
To delete the above table, we need to run the following statement:

1. mysql> **DROP** **TABLE**  orders;

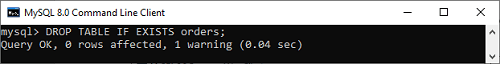
It will remove the table permanently. We can also check the table is present or not as shown in the below output:



If we try to delete a table that does not exist in the database, we will get an error message as given below:

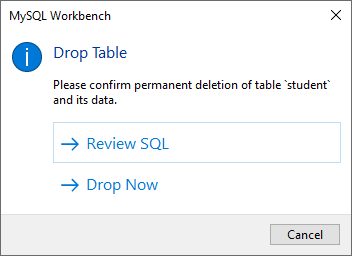


If we use the IF EXISTS clause with the DROP TABLE statement, MySQL gives the warning message which can be shown in the below output:



How to DROP table in Workbench

1. To delete a table, you need to choose the table, right-click on it, and select the Drop Table option. The following screen appears:



2. Select **Drop Now** option in the popup window to delete the table from the database instantly.

**MySQL DROP Multiple Table**

Sometimes we want to delete more than one table from the database. In that case, we have to use the table names and separate them by using the comma operator. The following statement can be used to remove multiple tables:

1. **DROP** **TABLE** IF EXISTS table\_name1, table\_name2, **table**, ......., table\_nameN;

**MySQL TRUNCATE Table vs. DROP Table**

You can also use the DROP TABLE command to delete the complete table, but it will remove complete table data and structure both. You need to re-create the table again if you have to store some data. But in the case of TRUNCATE TABLE, it removes only table data, not structure. You don't need to re-create the table again because the table structure already exists.

***MySQL Queries***

## 1) MySQL Create Database

MySQL create database is used to create database. For example

1. create database db1;

## 2) MySQL Select/Use Database

MySQL use database is used to select database. For example

37.7M

741

How to find Nth Highest Salary in SQL

1. use db1;

## 3) MySQL Create Query

MySQL create query is used to create a table, view, procedure and function. For example:

1. CREATE TABLE customers
2. (id int(10),
3. name varchar(50),
4. city varchar(50),
5. PRIMARY KEY (id )
6. );

## 4) MySQL Alter Query

MySQL alter query is used to add, modify, delete or drop colums of a table. Let's see a query to add column in customers table:

1. ALTER TABLE customers
2. ADD age varchar(50);

## 5) MySQL Insert Query

MySQL insert query is used to insert records into table. For example:

1. insert into customers values(101,'rahul','delhi');

## 6) MySQL Update Query

MySQL update query is used to update records of a table. For example:

1. update customers set name='bob', city='london' where id=101;

## 7) MySQL Delete Query

MySQL update query is used to delete records of a table from database. For example:

1. delete from customers where id=101;

## 8) MySQL Select Query

Oracle select query is used to fetch records from database. For example:

1. SELECT \* from customers;

## 9) MySQL Truncate Table Query

MySQL update query is used to truncate or remove records of a table. It doesn't remove structure. For example:

1. truncate table customers;

## 10) MySQL Drop Query

MySQL drop query is used to drop a table, view or database. It removes structure and data of a table if you drop table. For example:

1. drop table customers;

# MySQL Constraints

The constraint in MySQL is used to specify the rule that allows or restricts what values/data will be stored in the table. They provide a suitable method to ensure data accuracy and integrity inside the table. It also helps to limit the type of data that will be inserted inside the table. If any interruption occurs between the constraint and data action, the action is failed.

### Types of MySQL Constraints

Constraints in MySQL is classified into two types:

1. **Column Level Constraints:** These constraints are applied only to the single column that limits the type of particular column data.
2. **Table Level Constraints:** These constraints are applied to the entire table that limits the type of data for the whole table.

### How to create constraints in MySQL

We can define the constraints during a table created by using the CREATE TABLE statement. MySQL also uses the ALTER TABLE statement to specify the constraints in the case of the existing table schema.

### Syntax

The following are the syntax to create a constraints in table:

51K

C++

1. CREATE TABLE new\_table\_name (
2. col\_name1 datatype constraint,
3. col\_name2 datatype constraint,
4. col\_name3 datatype constraint,
5. .........
6. );

### Constraints used in MySQL

The following are the most common constraints used in the MySQL:

* NOT NULL
* CHECK
* DEFAULT
* PRIMARY KEY
* AUTO\_INCREMENT
* UNIQUE
* INDEX
* ENUM
* FOREIGN KEY

Let us discuss each of these constraints in detail.

### NOT NULL Constraint

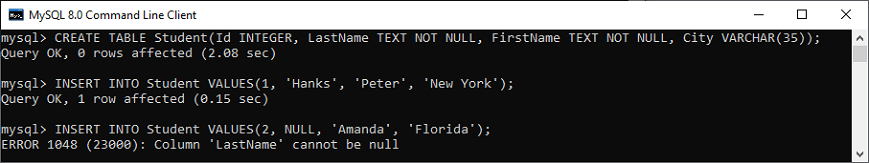
This constraint specifies that the column cannot have NULL or empty values. The below statement creates a table with NOT NULL constraint.

1. mysql> CREATE TABLE Student(Id INTEGER, LastName TEXT NOT NULL, FirstName TEXT NOT NULL, City VARCHAR(35));

Execute the queries listed below to understand how it works:

1. mysql> INSERT INTO Student VALUES(1, 'Hanks', 'Peter', 'New York');
3. mysql> INSERT INTO Student VALUES(2, NULL, 'Amanda', 'Florida');

**Output**



In the above image, we can see that the first INSERT query executes correctly, but the second statement fails and gives an error that says column LastName cannot be null.

### UNIQUE Constraint

This constraint ensures that all values inserted into the column will be unique. It means a column cannot stores duplicate values. MySQL allows us to use more than one column with UNIQUE constraint in a table. The below statement creates a table with a UNIQUE constraint:

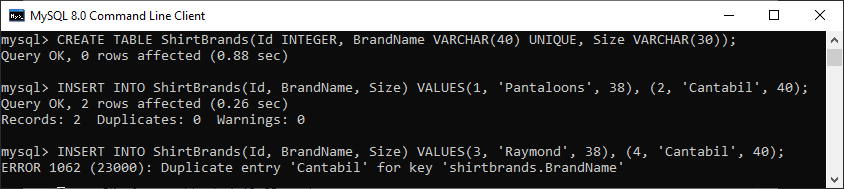
1. mysql> CREATE TABLE ShirtBrands(Id INTEGER, BrandName VARCHAR(40) UNIQUE, Size VARCHAR(30));

Execute the queries listed below to understand how it works:

1. mysql> INSERT INTO ShirtBrands(Id, BrandName, Size) VALUES(1, 'Pantaloons', 38), (2, 'Cantabil', 40);
3. mysql> INSERT INTO ShirtBrands(Id, BrandName, Size) VALUES(1, 'Raymond', 38), (2, 'Cantabil', 40);

**Output**

In the below output, we can see that the first INSERT query executes correctly, but the second statement fails and gives an error that says: Duplicate entry 'Cantabil' for key BrandName.



### CHECK Constraint

It controls the value in a particular column. It ensures that the inserted value in a column must be satisfied with the given condition. In other words, it determines whether the value associated with the column is valid or not with the given condition.

Before the version 8.0.16, MySQL uses the limited version of this constraint syntax, as given below:

1. CHECK (expr)

After the version 8.0.16, MySQL uses the CHECK constraints for all storage engines i.e., table constraint and column constraint, as given below:

1. [CONSTRAINT [symbol]] CHECK (expr) [[NOT] ENFORCED]

Let us understand how a CHECK constraint works in MySQL. For example, the following statement creates a table "Persons" that contains CHECK constraint on the "Age" column. The CHECK constraint ensures that the inserted value in a column must be satisfied with the given condition means the Age of a person should be greater than or equal to 18:

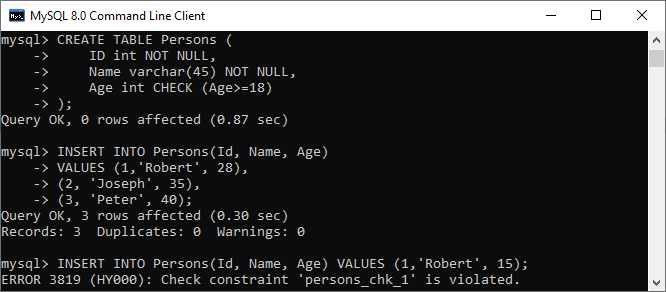
1. mysql> CREATE TABLE Persons (
2. ID int NOT NULL,
3. Name varchar(45) NOT NULL,
4. Age int CHECK (Age>=18)
5. );

Execute the listed queries to insert the values into the table:

1. mysql> INSERT INTO Persons(Id, Name, Age)
2. VALUES (1,'Robert', 28), (2, 'Joseph', 35), (3, 'Peter', 40);
4. mysql> INSERT INTO Persons(Id, Name, Age) VALUES (1,'Robert', 15);

**Output**

In the below output, we can see that the first INSERT query executes successfully, but the second statement fails and gives an error that says: CHECK constraint is violated for key Age.



### DEFAULT Constraint

This constraint is used to set the default value for the particular column where we have not specified any value. It means the column must contain a value, including NULL.

For example, the following statement creates a table "Persons" that contains DEFAULT constraint on the "City" column. If we have not specified any value to the City column, it inserts the default value:

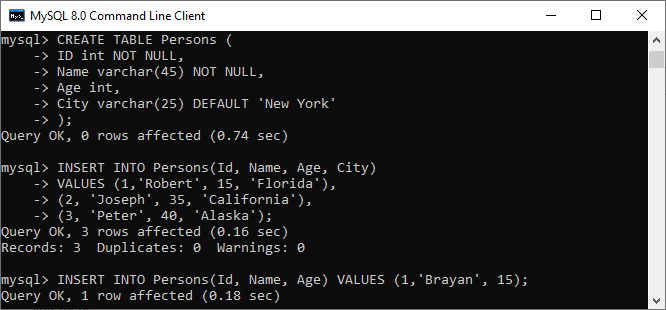
1. mysql> CREATE TABLE Persons (
2. ID int NOT NULL,
3. Name varchar(45) NOT NULL,
4. Age int,
5. City varchar(25) DEFAULT 'New York'
6. );

Execute the listed queries to insert the values into the table:

1. mysql> INSERT INTO Persons(Id, Name, Age, City)
2. VALUES (1,'Robert', 15, 'Florida'),
3. (2, 'Joseph', 35, 'California'),
4. (3, 'Peter', 40, 'Alaska');
6. mysql> INSERT INTO Persons(Id, Name, Age) VALUES (1,'Brayan', 15);

**Output**

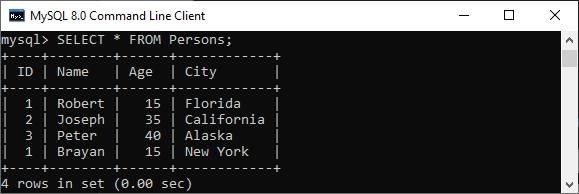
In the below output, we can see that the first insert query that contains all fields executes successfully, while the second insert statement does not contain the "City" column but also executed successfully. It is because it has a default value.



Now, executes the following statement to validate the default value for the 4th column:

1. mysql> SELECT \* FROM Persons;

We can see that it works perfectly. It means default value "New York" stored automatically in the City column.



### PRIMARY KEY Constraint

This constraint is used to identify each record in a table uniquely. If the column contains primary key constraints, then it cannot be null or empty. A table may have duplicate columns, but it can contain only one primary key. It always contains unique value into a column.

The following statement creates a table "Person" and explains the use of this primary key more clearly:

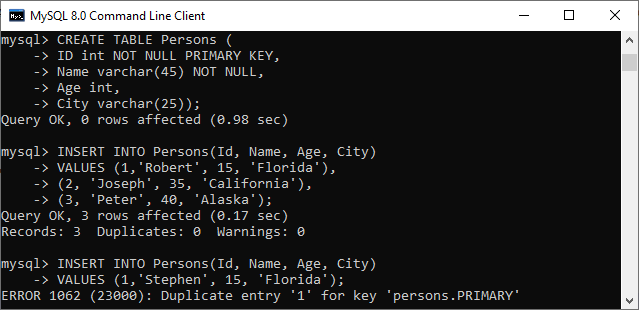
1. CREATE TABLE Persons (
2. ID int NOT NULL PRIMARY KEY,
3. Name varchar(45) NOT NULL,
4. Age int,
5. City varchar(25));

Next, use the insert query to store data into a table:

1. INSERT INTO Persons(Id, Name, Age, City)
2. VALUES (1,'Robert', 15, 'Florida') ,
3. (2, 'Joseph', 35, 'California'),
4. (3, 'Peter', 40, 'Alaska');
6. INSERT INTO Persons(Id, Name, Age, City)
7. VALUES (1,'Stephen', 15, 'Florida');

**Output**

In the below output, we can see that the first insert query executes successfully. While the second insert statement fails and gives an error that says: Duplicate entry for the primary key column.



### AUTO\_INCREMENT Constraint

This constraint automatically generates a unique number whenever we insert a new record into the table. Generally, we use this constraint for the primary key field in a table.

We can understand it with the following example where the id column going to be auto-incremented in the Animal table:

1. mysql> CREATE TABLE Animals(
2. id int NOT NULL AUTO\_INCREMENT,
3. name CHAR(30) NOT NULL,
4. PRIMARY KEY (id));

Next, we need to insert the values into the "Animals" table:

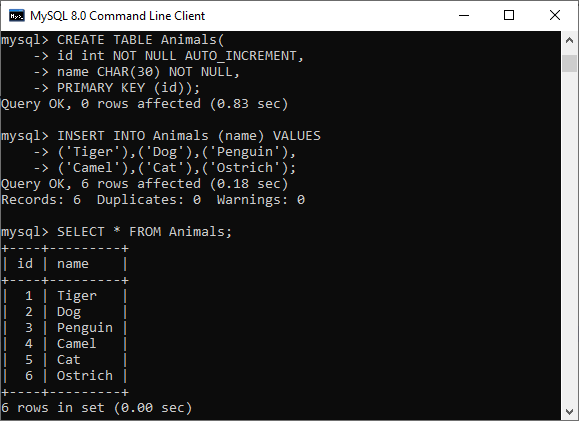
1. mysql> INSERT INTO Animals (name) VALUES
2. ('Tiger'),('Dog'),('Penguin'),
3. ('Camel'),('Cat'),('Ostrich');

Now, execute the below statement to get the table data:

1. mysql> SELECT \* FROM Animals;

**Output**

In the output, we can see that I have not specified any value for the auto-increment column, so MySQL automatically generates a unique number in the sequence order for this field.



### ENUM Constraint

The ENUM data type in MySQL is a string object. It allows us to limit the value chosen from a list of permitted values in the column specification at the time of table creation. It is short for enumeration, which means that each column may have one of the specified possible values. It uses numeric indexes (1, 2, 3…) to represent string values.

The following illustration creates a table named "shirts" that contains three columns: id, name, and size. The column name "size" uses the ENUM data type that contains small, medium, large, and x-large sizes.

1. mysql> CREATE TABLE Shirts (
2. id INT PRIMARY KEY AUTO\_INCREMENT,
3. name VARCHAR(35),
4. size ENUM('small', 'medium', 'large', 'x-large')
5. );

Next, we need to insert the values into the "Shirts" table using the below statements:

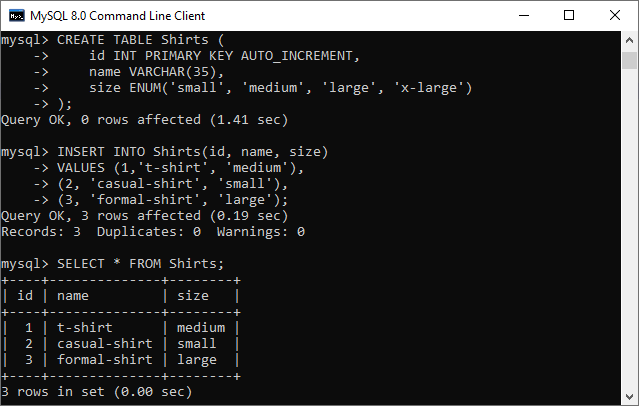
1. mysql> INSERT INTO Shirts(id, name, size)
2. VALUES (1,'t-shirt', 'medium'),
3. (2, 'casual-shirt', 'small'),
4. (3, 'formal-shirt', 'large');

Now, execute the SELECT statement to see the inserted values into the table:

1. mysql> SELECT \* FROM Shirts;

**Output**

We will get the following output:



### INDEX Constraint

This constraint allows us to create and retrieve values from the table very quickly and easily. An index can be created using one or more than one column. It assigns a ROWID for each row in that way they were inserted into the table.

The following illustration creates a table named "shirts" that contains three columns: id, name, and size.

1. mysql> CREATE TABLE Shirts (
2. id INT PRIMARY KEY AUTO\_INCREMENT,
3. name VARCHAR(35),
4. size ENUM('small', 'medium', 'large', 'x-large')
5. );

Next, we need to insert the values into the "Shirts" table using the below statements:

1. mysql> INSERT INTO Shirts(id, name, size)
2. VALUES (1,'t-shirt', 'medium'),
3. (2, 'casual-shirt', 'small'),
4. (3, 'formal-shirt', 'large');

Now, execute this statement for creating index:

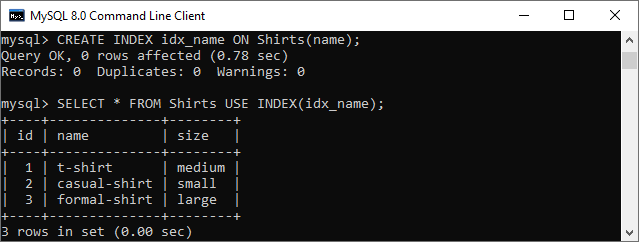
1. mysql> CREATE INDEX idx\_name ON Shirts(name);

We can use the query below to retrieve the data using the index column:

1. mysql> SELECT \* FROM Shirts USE INDEX(idx\_name);

**Output**

The following output appears:



### Foreign Key Constraint

This constraint is used to link two tables together. It is also known as the referencing key. A foreign key column matches the primary key field of another table. It means a foreign key field in one table refers to the primary key field of another table.

Let us consider the structure of these tables: Persons and Orders.

**Table: Persons**

1. CREATE TABLE Persons (
2. Person\_ID int NOT NULL PRIMARY KEY,
3. Name varchar(45) NOT NULL,
4. Age int,
5. City varchar(25)
6. );

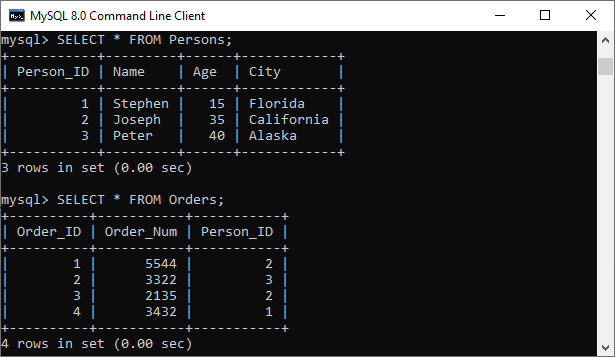
**Table: Orders**

1. CREATE TABLE Orders (
2. Order\_ID int NOT NULL PRIMARY KEY,
3. Order\_Num int NOT NULL,
4. Person\_ID int,
5. FOREIGN KEY (Person\_ID) REFERENCES Persons(Person\_ID)
6. );

In the above table structures, we can see that the "Person\_ID" field in the "Orders" table points to the "Person\_ID" field in the "Persons" table. The "Person\_ID" is the PRIMARY KEY in the "Persons" table, while the "Person\_ID" column of the "Orders" table is a FOREIGN KEY.

**Output**

Our table contains the following data:



# MySQL INSERT Statement

MySQL INSERT statement is used to store or add data in MySQL table within the database. We can perform insertion of records in two ways using a single query in MySQL:

1. Insert record in a single row
2. Insert record in multiple rows

### Syntax:

The below is generic syntax of **SQL INSERT INTO** command to insert a single record in MySQL

 table:

1. INSERT INTO table\_name ( field1, field2,...fieldN )
2. VALUES
3. ( value1, value2,...valueN );

In the above syntax, we first have to specify the table name and list of comma-separated columns. Second, we provide the list of values corresponding to columns name after the VALUES clause.

#### NOTE: Field name is optional. If we want to specify partial values, the field name is mandatory. It also ensures that the column name and values should be the same. Also, the position of columns and corresponding values must be the same.

If we want to insert **multiple records** within a single command, use the following statement:

**00:00/02:28**

1. INSERT INTO table\_name VALUES
2. ( value1, value2,...valueN )
3. ( value1, value2,...valueN )
4. ...........
5. ( value1, value2,...valueN );

In the above syntax, all rows should be separated by commas in the value fields.

### MySQL INSERT Example

Let us understand how **INSERT statements** work in MySQL with the help of multiple examples. First, create a table "**People**" in the database using the following command:

1. CREATE TABLE People(
2. id int NOT NULL AUTO\_INCREMENT,
3. name varchar(45) NOT NULL,
4. occupation varchar(35) NOT NULL,
5. age int,
6. PRIMARY KEY (id)
7. );

**1.** If we want to store single records for all fields, use the syntax as follows:

1. INSERT INTO People (id, name, occupation, age)
2. VALUES (101, 'Peter', 'Engineer', 32);

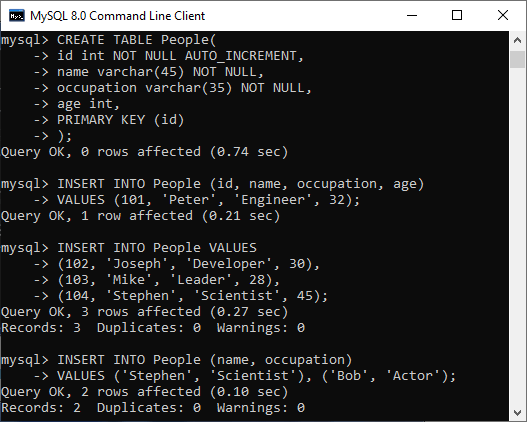
**2.** If we want to store multiple records, use the following statements where we can either specify all field names or don't specify any field.

1. INSERT INTO People VALUES
2. (102, 'Joseph', 'Developer', 30),
3. (103, 'Mike', 'Leader', 28),
4. (104, 'Stephen', 'Scientist', 45);

**3.** If we want to store records without giving all fields, we use the following **partial field** statements. In such case, it is mandatory to specify field names.

1. INSERT INTO People (name, occupation)
2. VALUES ('Stephen', 'Scientist'), ('Bob', 'Actor');

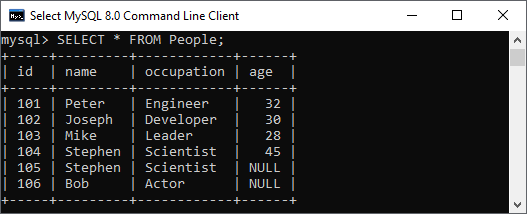
In the below output, we can see that all INSERT statements have successfully executed and stored the value in a table correctly.



We can use the below syntax to show the records of the **People** table:

1. mysql> SELECT \* FROM People;

We will get the output as follows:



### Inserting Date in MySQL Table:

We can also use the INSERT STATEMENT to add the date in MySQL table. MySQL provides several data types for storing dates such as DATE, TIMESTAMP, DATETIME, and YEAR. The **default forma**t of the date in MySQL is **YYYY-MM-DD**.

This format has the below descriptions:

* **YYYY:** It represents the four-digit year, like 2020.
* **MM:** It represents the two-digit month, like 01, 02, 03, and 12.
* **DD:** It represents the two-digit day, like 01, 02, 03, and 31.

Following is the basic syntax to insert date in MySQL table:

1. INSERT INTO table\_name (column\_name, column\_date) VALUES ('DATE: Manual Date', '2008-7-04');

If we want to insert a date in the mm/dd/yyyy format, it is required to use the below statement:

1. INSERT INTO table\_name VALUES (STR\_TO\_DATE(date\_value, format\_specifier));

# MySQL UPDATE Query

MySQL UPDATE query is a DML statement used to modify the data of the MySQL table within the database. In a real-life scenario, records are changed over a period of time. So, we need to make changes in the values of the tables also. To do so, it is required to use the UPDATE query.

The UPDATE statement is used with the **SET** and **WHERE clauses.** The SET clause is used to change the values of the specified column. We can update single or multiple columns at a time.

### Syntax

Following is a generic syntax of UPDATE command to modify data into the **MySQL**

 table:

1. UPDATE table\_name
2. SET column\_name1 = new-value1,
3. column\_name2=new-value2, ...
4. [WHERE Clause]

### Parameter Explanation

The description of parameters used in the syntax of the UPDATE statement is given below:

|  |  |
| --- | --- |
| **Parameter** | **Descriptions** |
| table\_name | It is the name of a table in which we want to perform updation. |
| column\_name | It is the name of a column in which we want to perform updation with the new value using the SET clause. If there is a need to update multiple columns, separate the columns with a comma operator by specifying the value in each column. |
| WHERE Clause | It is optional. It is used to specify the row name in which we are going to perform updation. If we omit this clause, MySQL updates all rows. |

**Note:**

* This statement can update values in a single table at a time.
* We can update single or multiple columns altogether with this statement.
* Any condition can be specified by using the WHERE clause.
* WHERE clause is very important because sometimes we want to update only a single row, and if we omit this clause, it accidentally updates all rows of the table.

The UPDATE command supports these modifiers in MySQL:

**LOW\_PRIORITY:** This modifier instructs the statement to delay the UPDATE command's execution until no other clients reading from the table. It takes effects only for the storage engines that use only table-level locking.

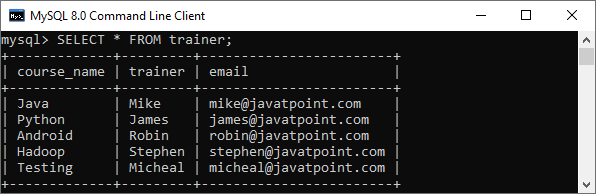
**IGNORE:** This modifier allows the statement to do not abort the execution even if errors occurred. If it finds **duplicate-key** conflicts, the rows are not updated.

Therefore, the full syntax of **UPDATE statement** is given below:

1. UPDATE [LOW\_PRIORITY] [IGNORE] table\_name
2. SET column\_assignment\_list
3. [WHERE condition]

### Example:

Let us understand the UPDATE statement with the help of various examples. Suppose we have a table **"trainer"** within the **"testdb"** database. We are going to update the data within the "trainer" table.



**Update Single Column**

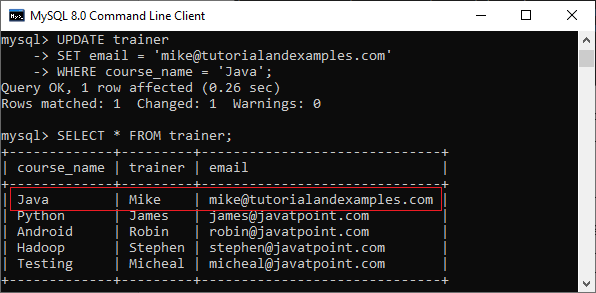
This query will update the **email id of Java** course with the new id as follows:

1. UPDATE trainer
2. SET email = 'mike@tutorialandexamples.com'
3. WHERE course\_name = 'Java';

After successful execution, we will verify the table using the below statement:

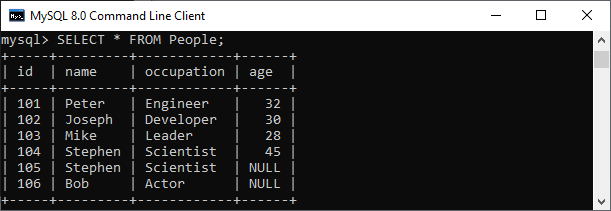
1. SELECT \* FROM trainer;

In the output, we can see that our table is updated as per our conditions.



**Update Multiple Columns**

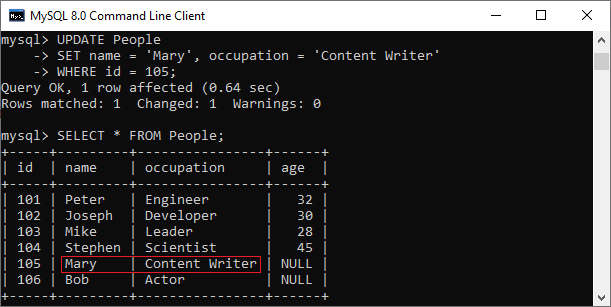
The UPDATE statement can also be used to update multiple columns by specifying a comma-separated list of columns. Suppose we have a table as below:



This statement explains will update the **name** and **occupation** whose **id = 105** in the **People** table as follows:

1. UPDATE People
2. SET name = 'Mary', occupation = 'Content Writer'
3. WHERE id = 105;

We can verify the output below:

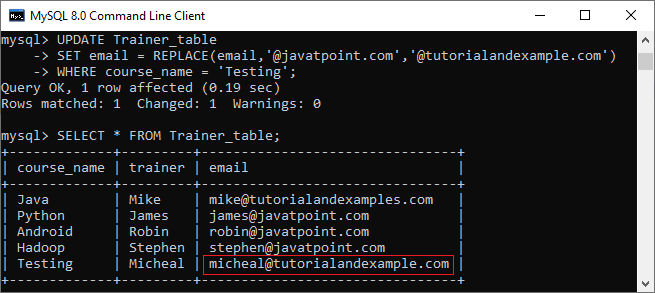


**UPDATE Statement to Replace String**

We can also use the UPDATE statement in MySQL to change the string name in the particular column. The following example updates the domain parts of emails of **Android course:**

1. UPDATE Trainer\_table
2. SET email = REPLACE(email,'@javatpoint.com','@tutorialandexample.com')
3. WHERE course\_name = 'Testing';

It will give the following output:



# **MySQL DELETE Statement**

MySQL DELETE statement is used to remove records from the MySQL table that is no longer required in the database. **This query in MySQL deletes a full row from the table and produces the count of deleted rows**. It also allows us to delete more than one record from the table within a single query, which is beneficial while removing large numbers of records from a table. By using the delete statement, we can also remove data based on conditions.

**Once we delete the records using this query, we cannot recover it**. Therefore before deleting any records from the table, it is recommended to **create a backup of your database**. The database backups allow us to restore the data whenever we need it in the future.

**Syntax:**

The following are the syntax that illustrates how to use the DELETE statement:

1. DELETE FROM table\_name WHERE condition;

In the above statement, we have to first specify the table name from which we want to delete data. Second, we have to specify the condition to delete records in the WHERE clauses, which is optional. If we omit the WHERE clause into the statement, this query will remove whole records from the database table.

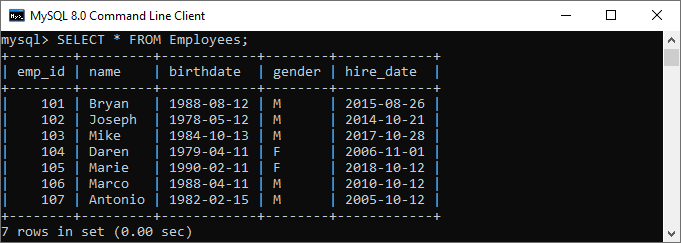
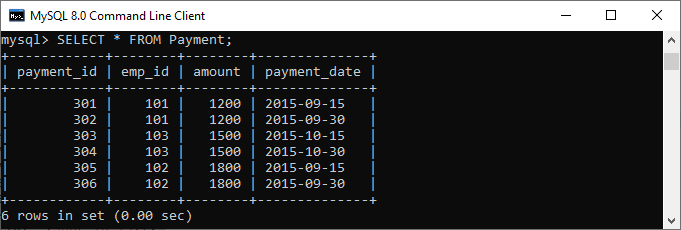
If we want to delete records from multiple tables using a single DELETE query, we must add the **JOIN clause** with the DELETE statement.

If we want to delete all records from a table without knowing the count of deleted rows, we must use the TRUNCATE TABLE statement that gives better performance.

Let us understand how the DELETE statement works in **MySQL** through various examples.

### MySQL DELETE Statement Examples

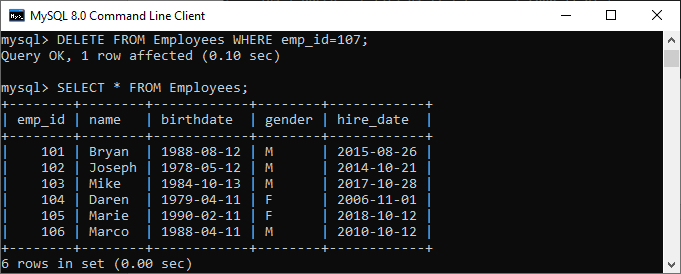
Here, we are going to use the **"Employees"** and **"Payment"** tables for the demonstration of the DELETE statement. Suppose the Employees and Payment tables contain the following data:

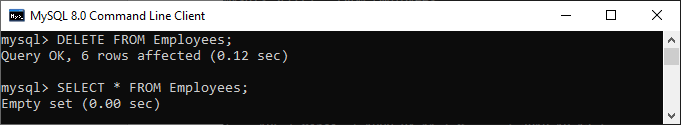
If we want to delete an employee whose **emp\_id is 107**, we should use the DELETE statement with the WHERE clause. See the below query:

1. mysql> DELETE FROM Employees WHERE emp\_id=107;

After the execution of the query, it will return the output as below image. Once the record is deleted, verify the table using the **SELECT statement:**



If we want to delete all records from the table, there is no need to use the WHERE clause with the DELETE statement. See the below code and output:



In the above output, we can see that after removing all rows, the Employees table will be empty. It means no records available in the selected table.

### MySQL DELETE and LIMIT Clause

MySQL Limit clause is used to restrict the count of rows returns from the result set, rather than fetching the whole records in the table. Sometimes we want to limit the number of rows to be deleted from the table; in that case, we will use the LIMIT clause as follows:

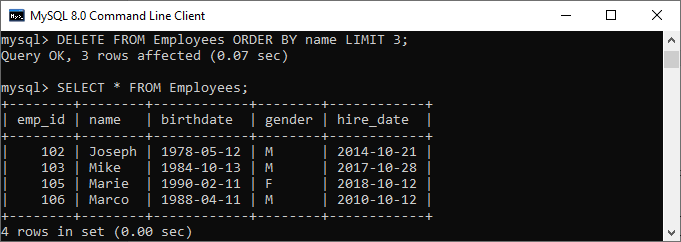
1. DELETE FROM table\_name
2. WHERE condition
3. ORDER BY colm1, colm2, ...
4. LIMIT row\_count;

It is to note that the order of rows in a MySQL table is unspecified. Therefore, we should always use the **ORDER BY** clause while using the LIMIT clause.

**For example**, the following query first sorts the employees according to their names alphabetically and deletes the first three employees from the table:

1. mysql> DELETE FROM Employees ORDER BY name LIMIT 3;

It will give the below output:



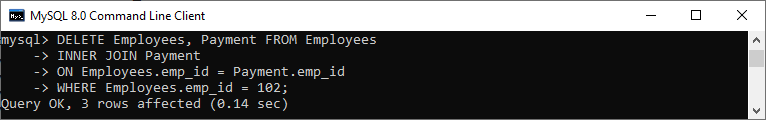
### MySQL DELETE and JOIN Clause

The JOIN clause is used to add the two or more tables in MySQL. We will add the JOIN clause with the DELETE statement whenever we want to delete records from multiple tables within a single query. See the below query:

1. mysql> DELETE Employees, Payment FROM Employees
2. INNER JOIN Payment
3. ON Employees.emp\_id = Payment.emp\_id
4. WHERE Employees.emp\_id = 102;

**Output:**

After execution, we will see the output as below image:



# MySQL SELECT Statement

The SELECT statement in MySQL is used to **fetch data from one or more tables**. We can retrieve records of all fields or specified fields that match specified criteria using this statement. It can also work with various scripting languages such as PHP, Ruby, and many more.

## SELECT Statement Syntax

It is the most commonly used SQL query. The general syntax of this statement to fetch data from tables are as follows:

SELECT field\_name1, field\_name 2,... field\_nameN

FROM table\_name1, table\_name2...

[WHERE condition]

[GROUP BY field\_name(s)]

[HAVING condition]

[ORDER BY field\_name(s)]

[OFFSET M ][LIMIT N];

### Syntax for all fields:

1. SELECT \* FROM tables [WHERE conditions]
2. [GROUP BY fieldName(s)]
3. [HAVING condition]
4. [ORDER BY fieldName(s)]
5. [OFFSET M ][LIMIT N];

## Parameter Explanation

The SELECT statement uses the following parameters:

#### NOTE: It is to note that MySQL always evaluates the FROM clause first, and then the SELECT clause will be evaluated.

|  |  |
| --- | --- |
| **Parameter Name** | **Descriptions** |
| field\_name(s) or \* | It is used to specify one or more columns to returns in the result set. The asterisk (\*) returns all fields of a table. |
| table\_name(s) | It is the name of tables from which we want to fetch data. |
| WHERE | It is an optional clause. It specifies the condition that returned the matched records in the result set. |
| GROUP BY | It is optional. It collects data from multiple records and grouped them by one or more columns. |
| HAVING | It is optional. It works with the GROUP BY clause and returns only those rows whose condition is TRUE. |
| ORDER BY | It is optional. It is used for sorting the records in the result set. |
| OFFSET | It is optional. It specifies to which row returns first. By default, It starts with zero. |
| LIMIT | It is optional. It is used to limit the number of returned records in the result set. |

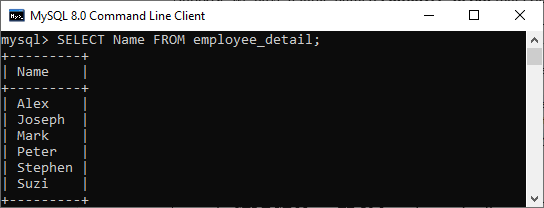
### MySQL SELECT Statement Example:

Let us understand how SELECT command works in mysql with the help of various examples. Suppose we have a table named **employee\_detail** that contains the following data:

**1.** If we want to retrieve a **single column from the table**, we need to execute the below query:

1. mysql> SELECT Name FROM employee\_detail;

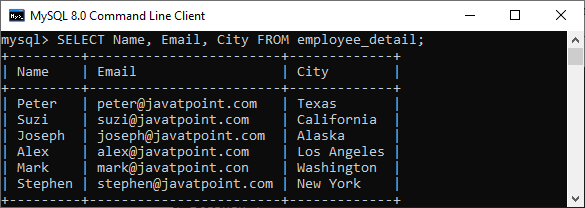
We will get the below output where we can see only one column records.



**2.** If we want to query **multiple columns from the table**, we need to execute the below query:

1. mysql> SELECT Name, Email, City FROM employee\_detail;

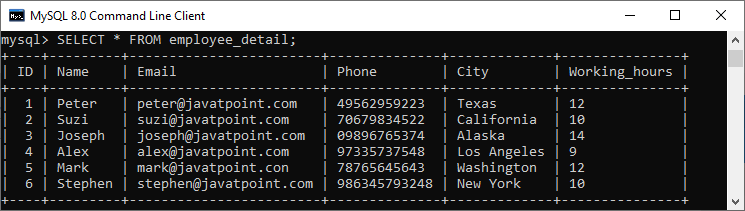
We will get the below output where we can see the name, email, and city of employees.



**3.** If we want to fetch data from **all columns of the table**, we need to use all column's names with the select statement. Specifying all column names is not convenient to the user, so MySQL uses an **asterisk** (\*) to retrieve all column data as follows:

1. mysql> SELECT \* FROM employee\_detail;

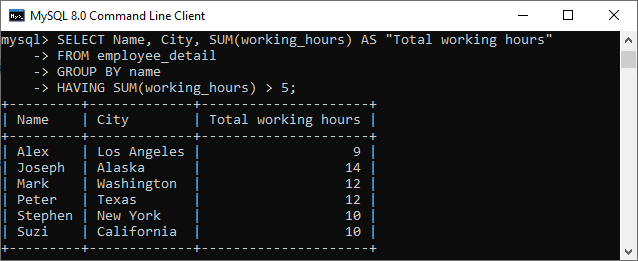
We will get the below output where we can see all columns of the table.



**4.** Here, we use the **SUM function** with the [**HAVING** clause](https://www.javatpoint.com/mysql-having) in the SELECT command to get the employee name, city, and total working hours. Also, it uses the [**GROUP BY** clause](https://www.javatpoint.com/mysql-group-by) to group them by the Name column.

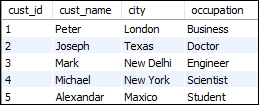
1. SELECT Name, City, SUM(working\_hours) AS "Total working hours"
2. FROM employee\_detail
3. GROUP BY Name
4. HAVING SUM(working\_hours) > 5;

It will give the below output:

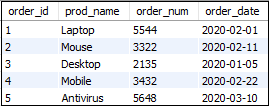


**5.** MySQL SELECT statement can also be used to retrieve records from multiple tables by using a **JOIN statement**. Suppose we have a table named **"customer"** and **"orders"** that contains the following data:

**Table: customer**



**Table: orders**



Execute the following SQL statement that returns the matching records from both tables using the [**INNER JOIN** query](https://www.javatpoint.com/mysql-inner-join):

1. SELECT cust\_name, city, order\_num, order\_date
2. FROM customer INNER JOIN orders
3. ON customer.cust\_id = orders.order\_id
4. WHERE order\_date < '2020-04-30'
5. ORDER BY cust\_name;

After successful execution of the query, we will get the output as follows:

