INTRODUCTION TO SQL

(STRUCTURAL QUERY LANGUAGE)

Table of Content

Contents

1. Introduction to SQL

SQL stands for Structural Query Language, and SQL is used for storing, manipulation, and retrieving data from the database.

History of SQL

The SQL(Structural Query language) was first created in the 1970s by IBM researchers Raymond Boyce and Donald Chamberlin. The Query language, known then as **SEQUEL**, was created following the publishing of Edgar Frank Todd's paper, In 1970, A Relational Model of Data for Large Shared Data Banks.

In his paper, Todd proposed that all the data in a database be represented in the form of relations. It was based on this theory that Chamberlin and Boyce came up with SQL. The original SQL version was designed to retrieve and manipulate data stored in IBM's original RDBMS known as "**System R**." It wasn't until several years later, however, that the Structural Query language was made available publicly. In 1979, a company named as Relational Software, which later became Oracle, commercially released its version of the SQL language called Oracle V2.

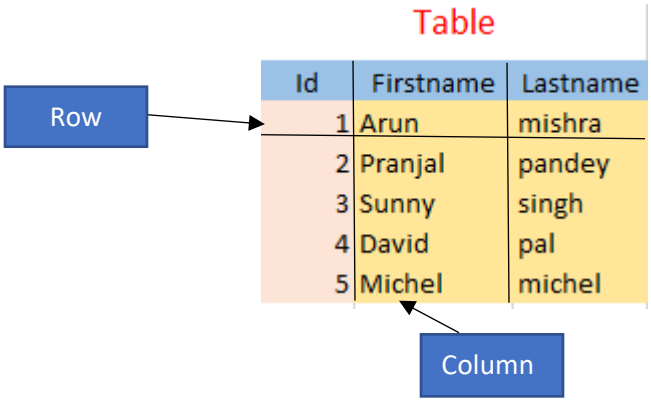
Since that time, the American National Standards Institute (ANSI) and the International Standards Organization have deemed the SQL language as the standard language in relational database communication. While major SQL vendors do modify the language to their desires, most base their SQL programs off of the ANSI approved version.

1.1. What is Database?

A database is a well-ordered collection of data. A database is an electronic system that permits data to be easily manipulated, accessed, and updated, or an organization uses a database as a method of managing, storing, and retrieving information. Modern databases are handled using a database management system (DBMS).

1.2. Relational Database

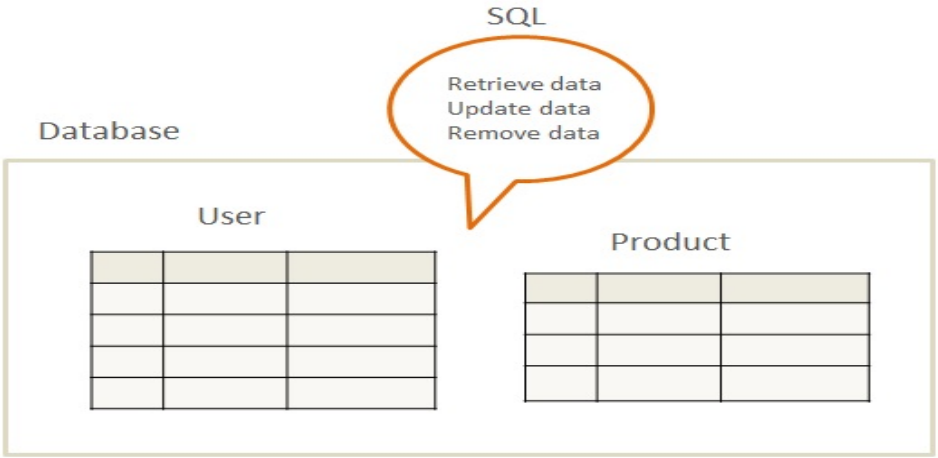
Relational Databases are used to store data in tables (rows and columns). Some common relational **database** management systems that use **SQL** are **Oracle**, **Sybase**, **Microsoft SQL Server**, **Access**, **Ingres**, etc.



Id	Firstname	Lastname
1	Arun	mishra
2	Pranjal	pandey
3	Sunny	singh
4	David	pal
5	Michel	michel

1.3. SQL and Relational Databases

A Relational Database contains tables that store the data that is related in some way. SQL is the query language that allows **retrieval and manipulation** of table data in the relational database. The database below has two tables: one with data on **Users** and another with data on **Products**.



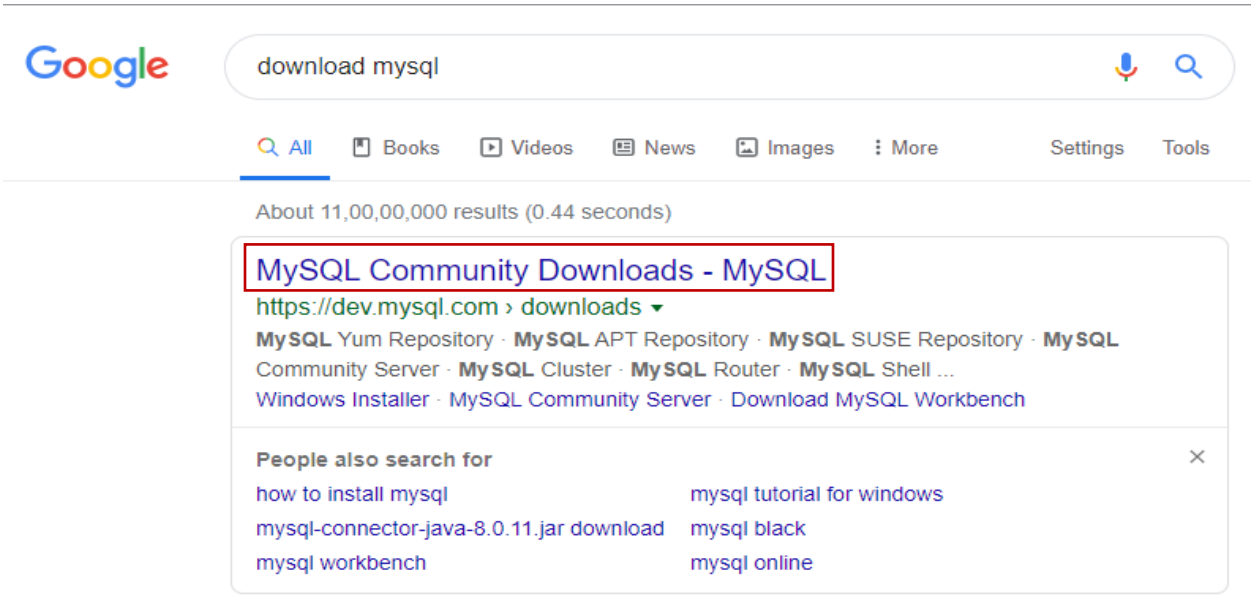
1.4. How to run SQL Query on the local system

To run the SQL query on the local system, we need to install the MYSQL community server on the system. We have given step by step installation process below.

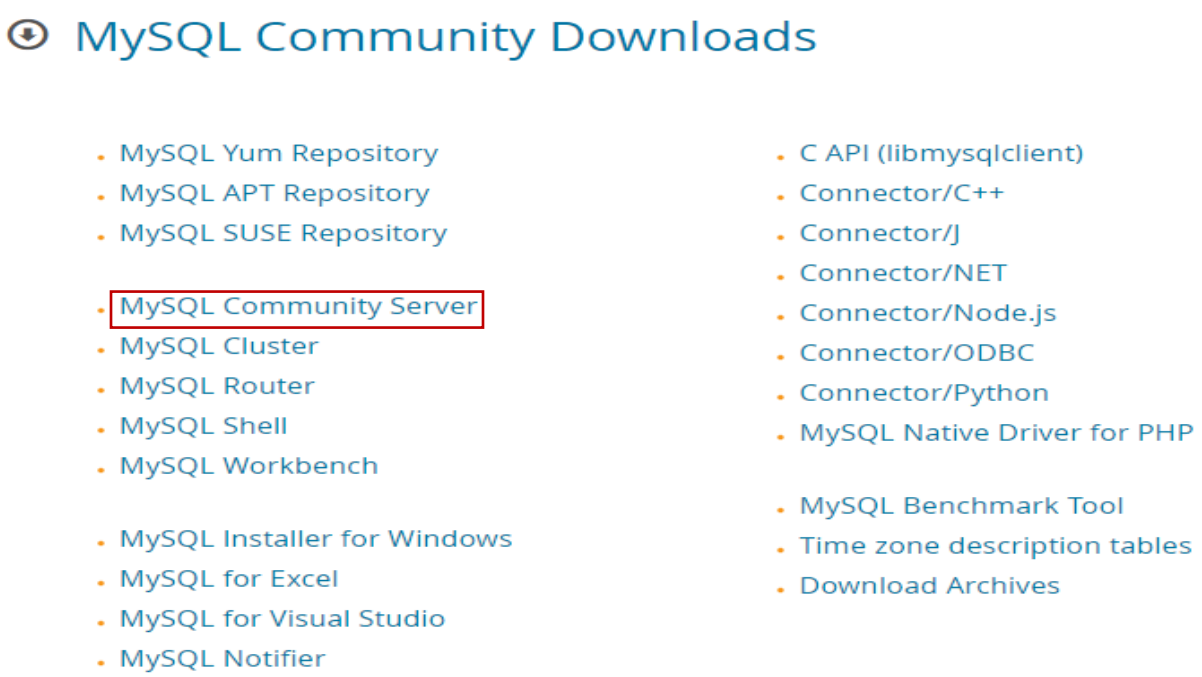
2. Downloading and Installing MySQL

2.1 Downloading MySQL

Step 1: Open Google and type **Download MySQL** and Click on **MySQL Community Downloads**

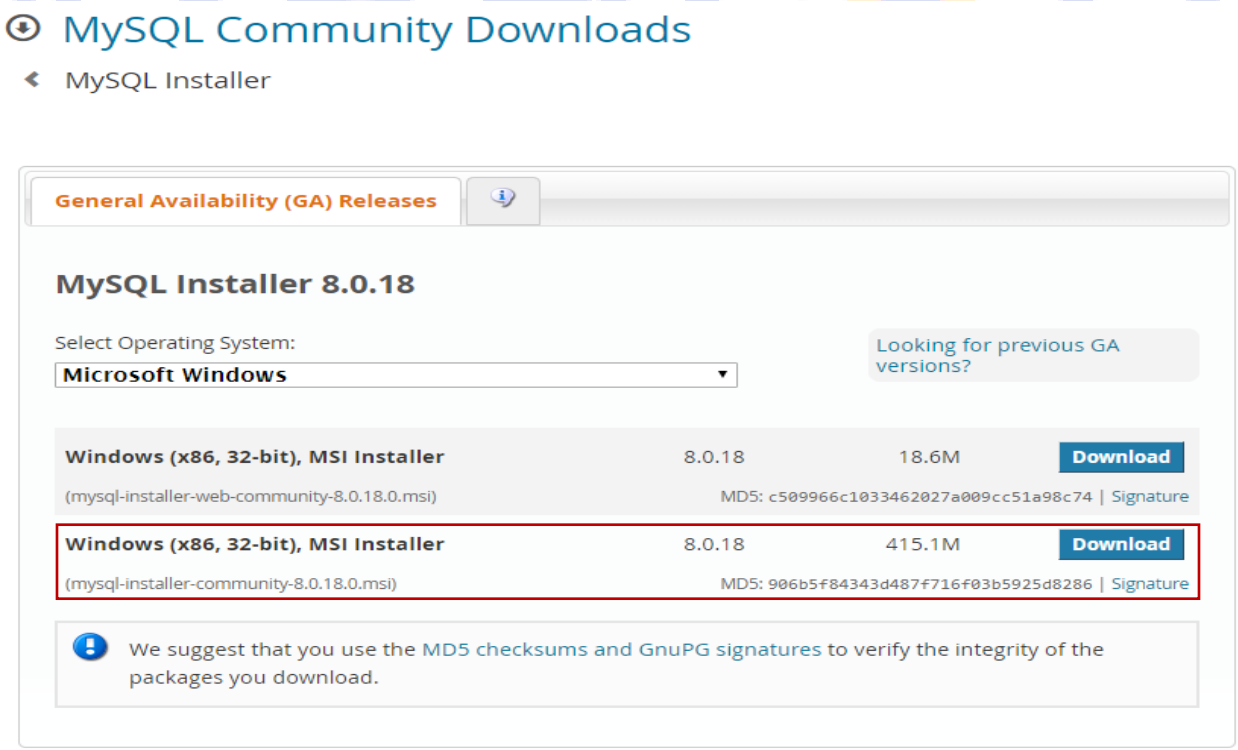


Step 2: Click on **MySQL Community Server**

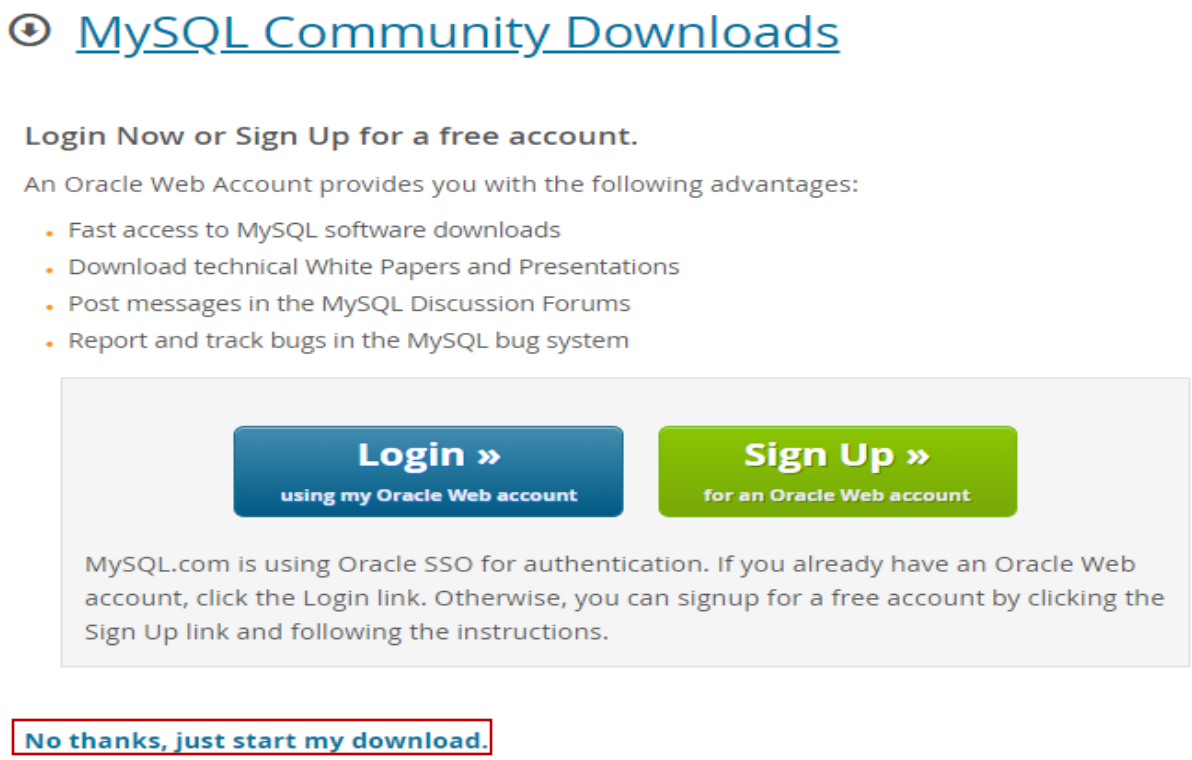


Step 3: Click on the MySQL installer MSI **Go to Download Page >**

Step 4: Select the OS and click on **MSI Installer community**



Step 5: Click on **start my download**

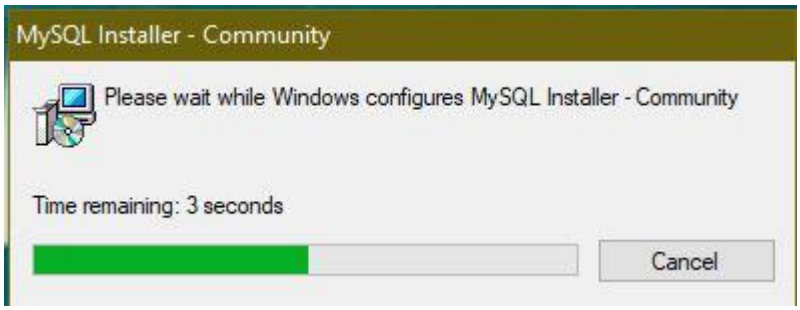


Note: Once the Downloading is completed, then double-click on that and install it on the local system.

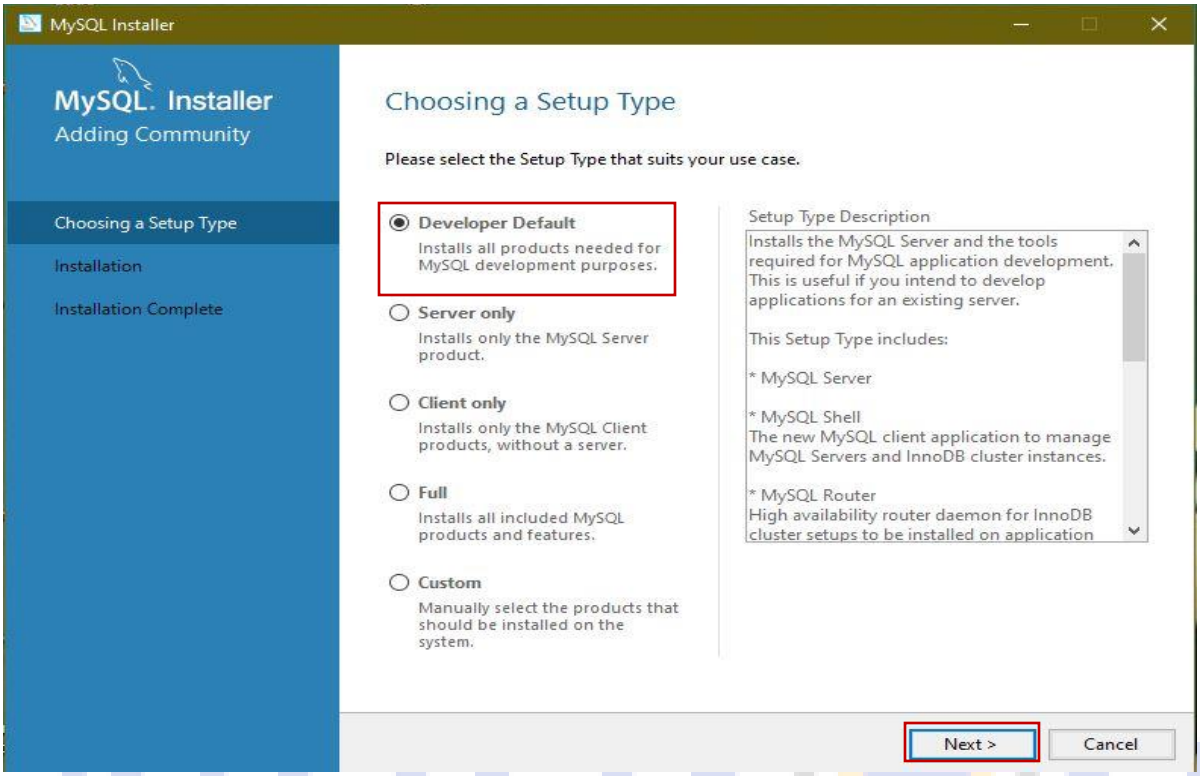
2.2. Installation of MySQL

Step 1: Double-Click on Downloaded Application.

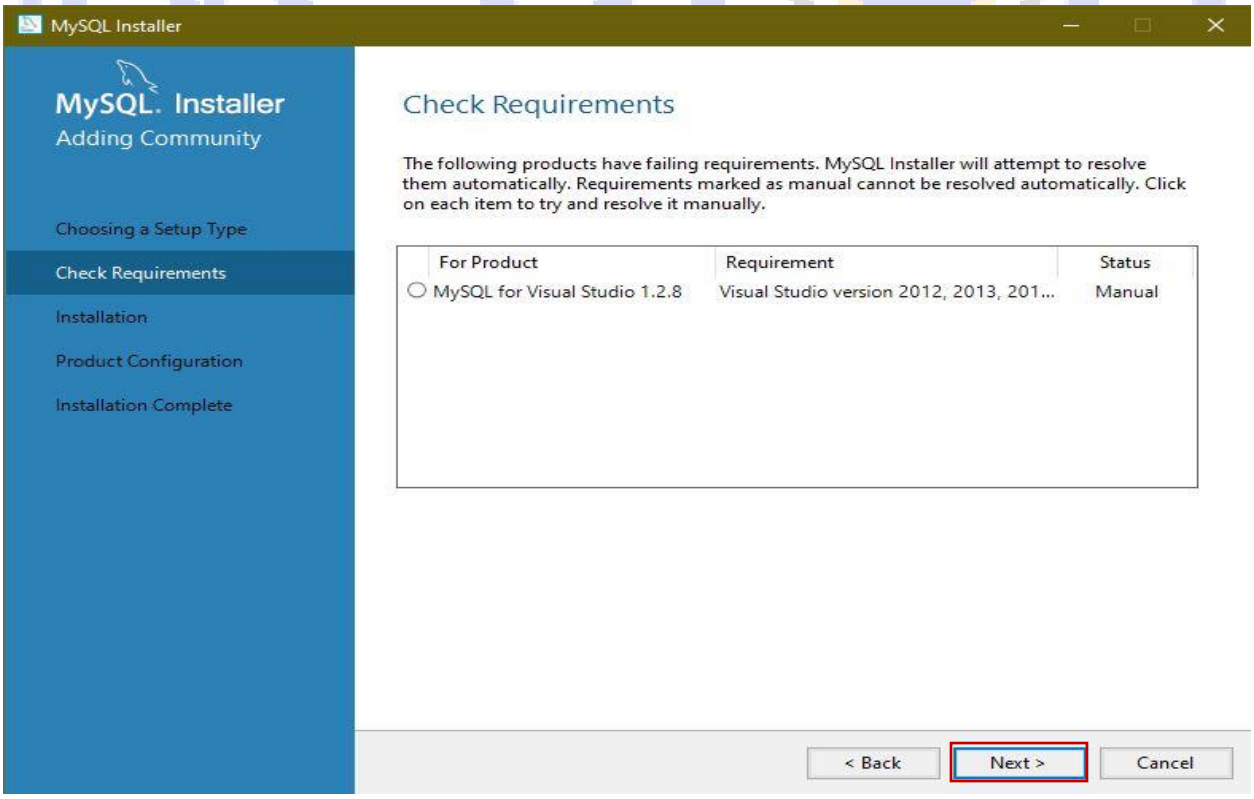
Step 2: After clicking on the application we will get a window like below



Step 3: Choosing the Setup type and click Next.

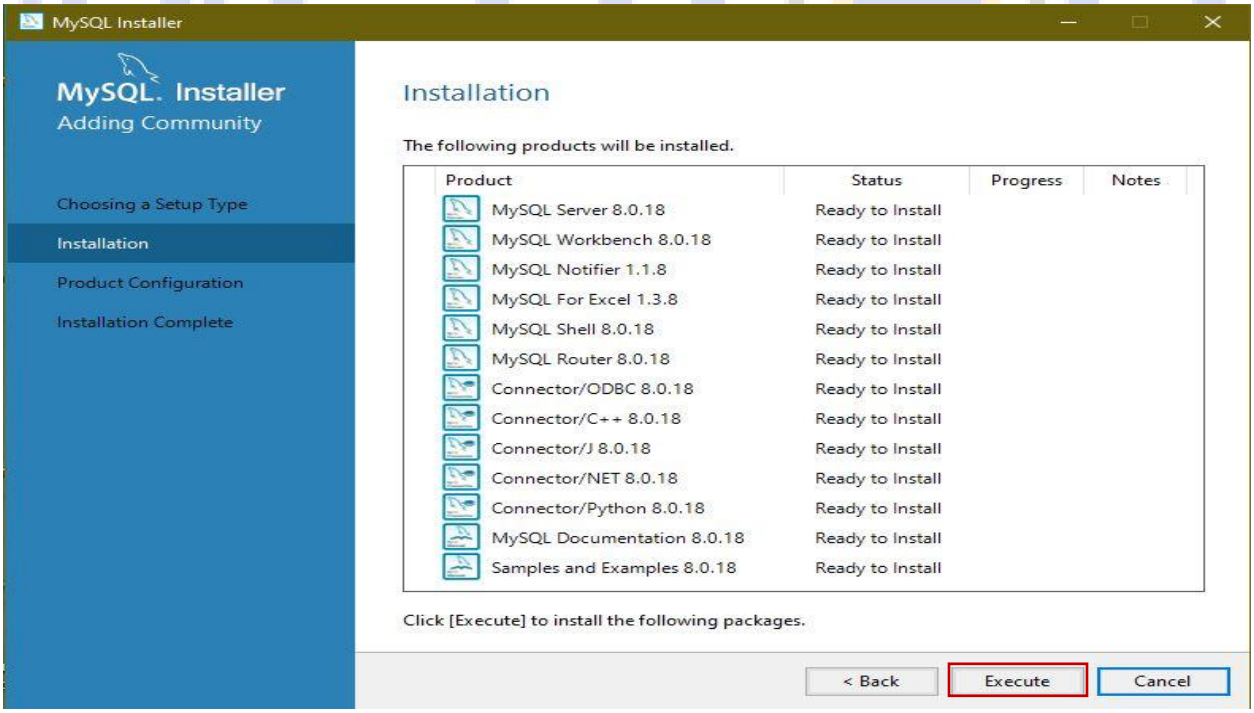


Setup 4: Click Next.

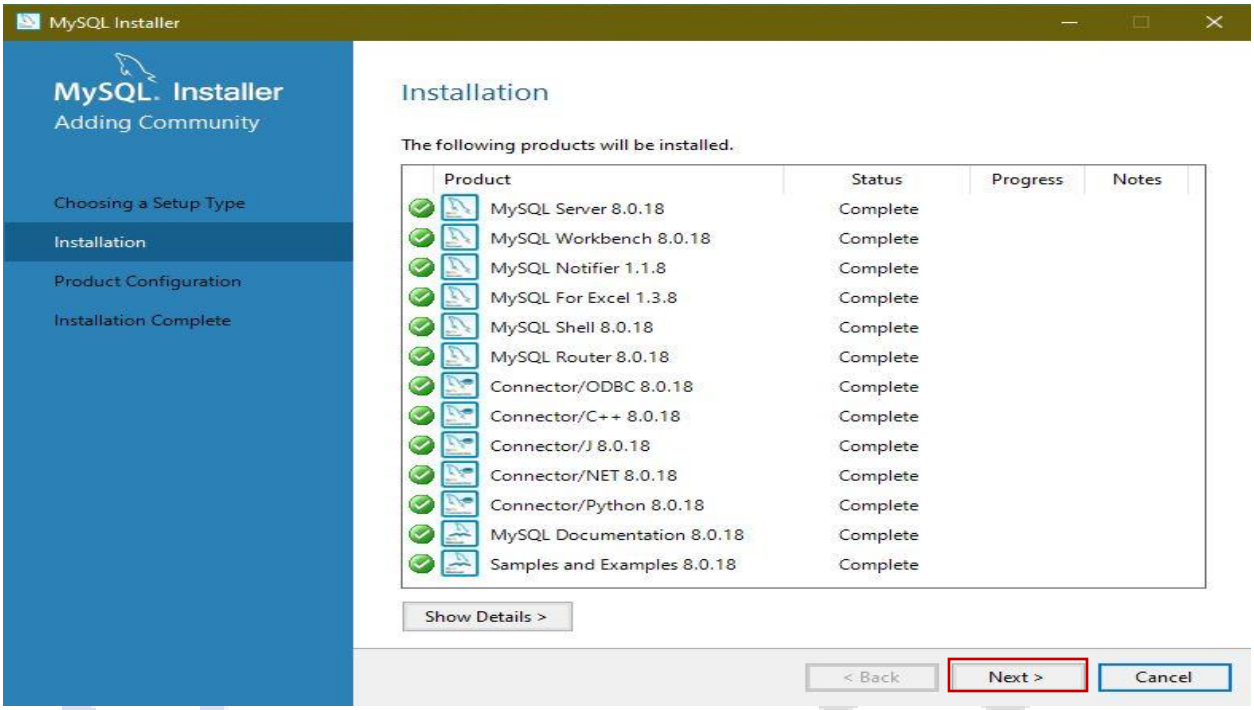


Step 5: Click Yes.

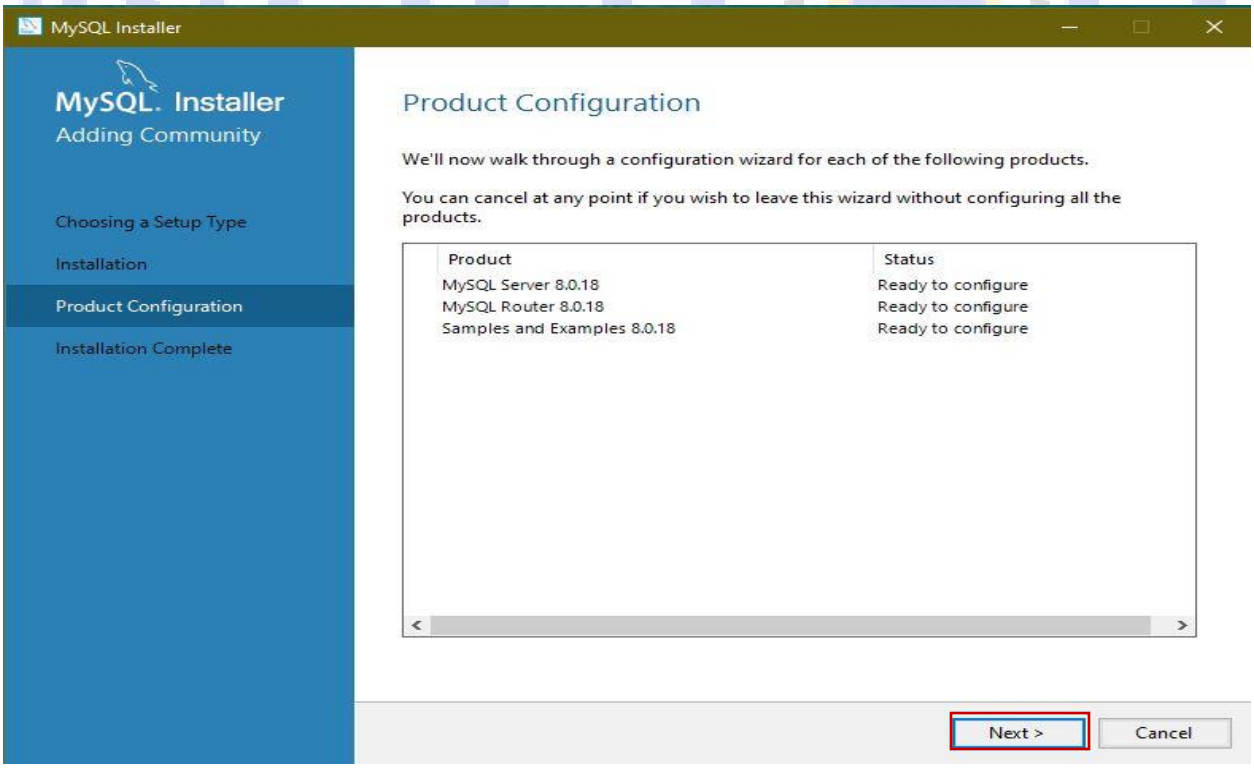
Step 6: Click Execute.



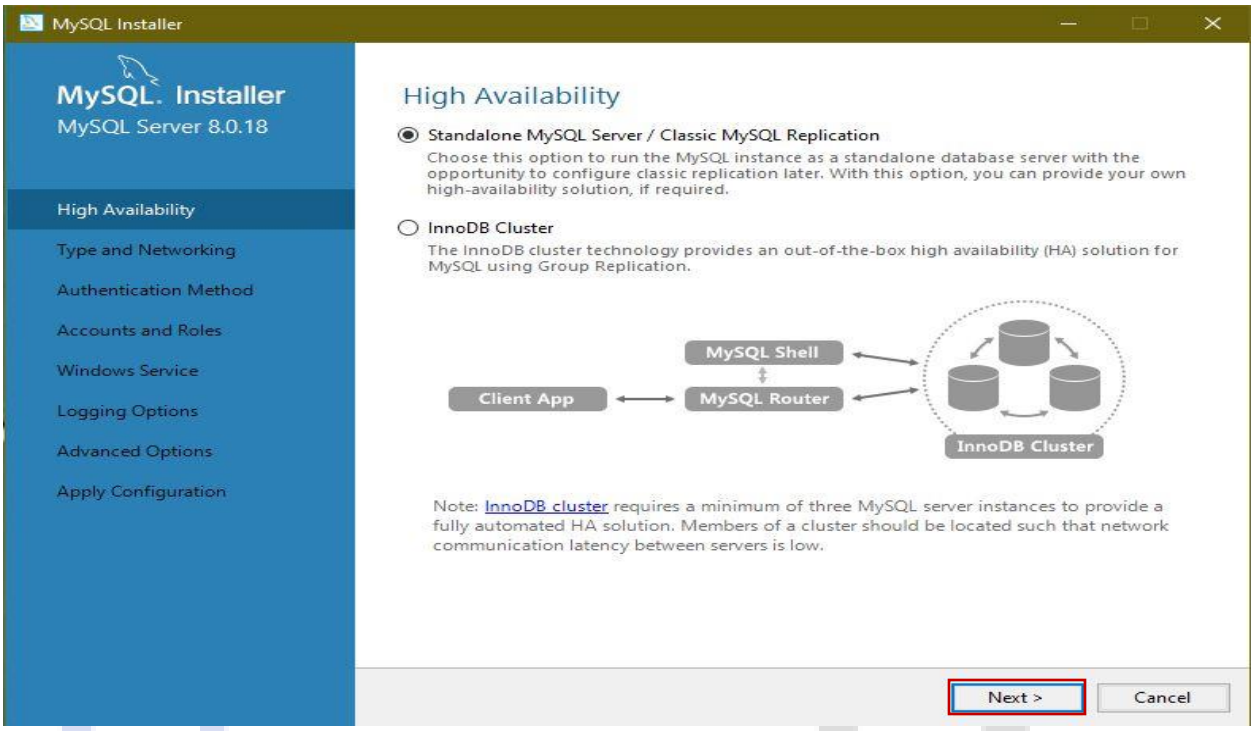
Step 7: After Execution, click on the Next.



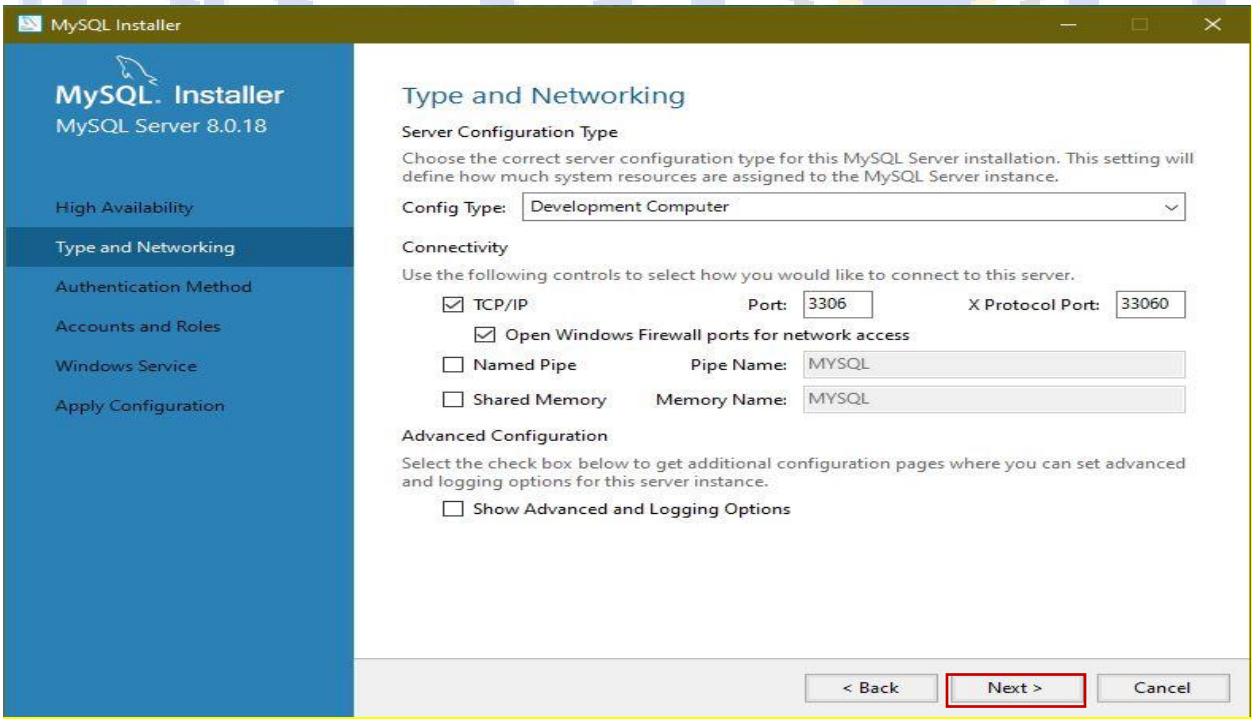
Step 8: Click Next.



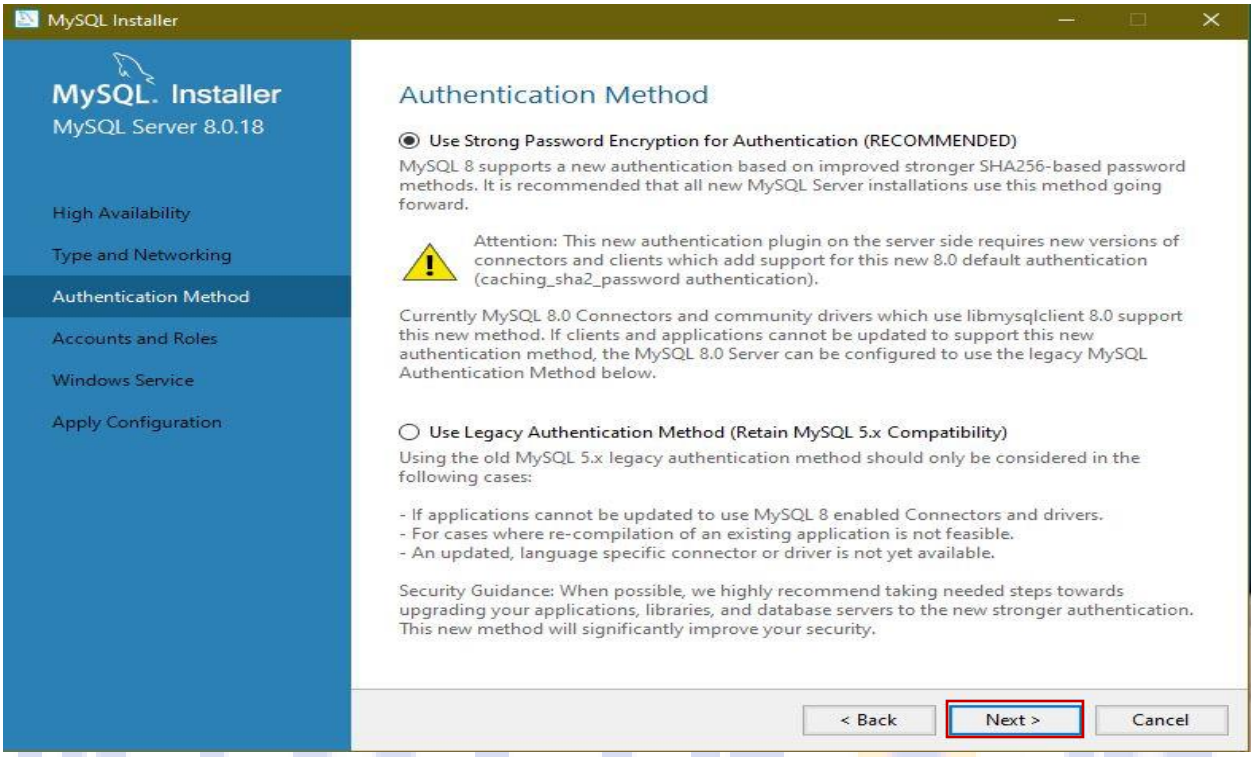
Step 9: Click Next.



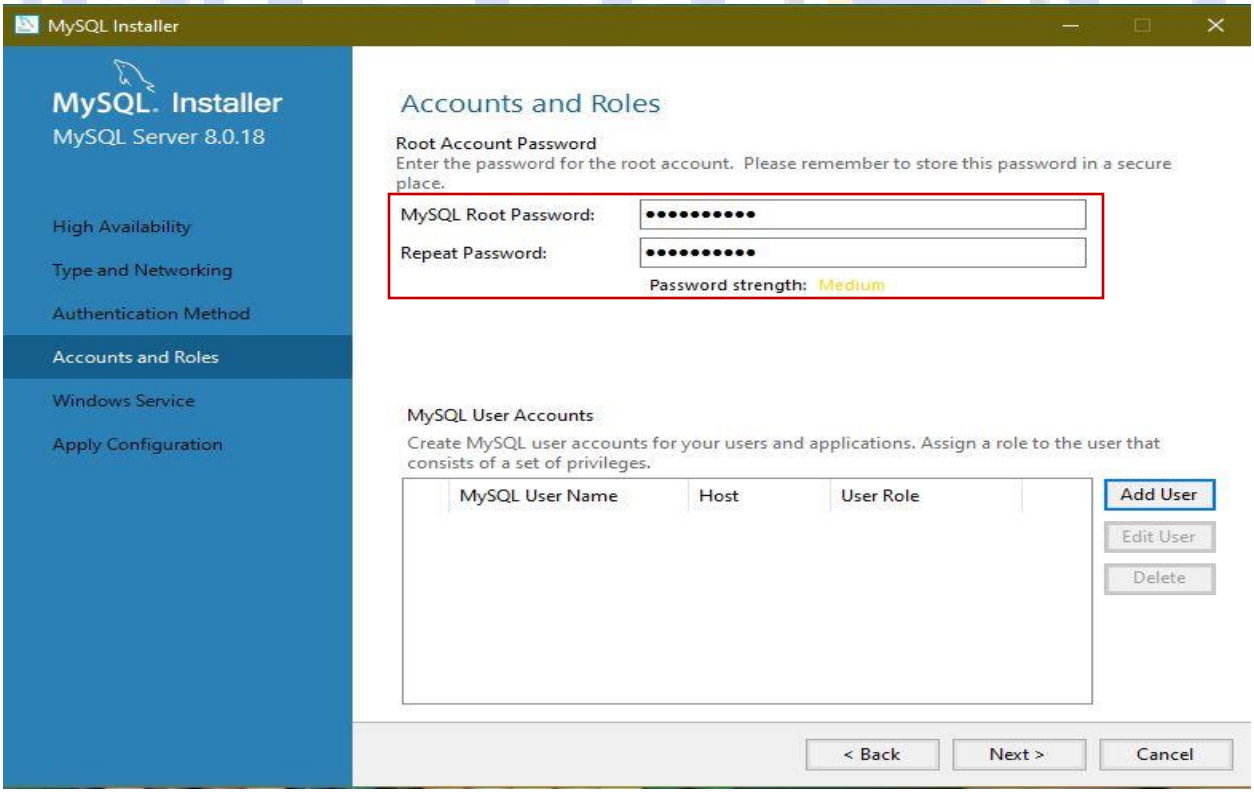
Step 10: Leave it as default and click Next.



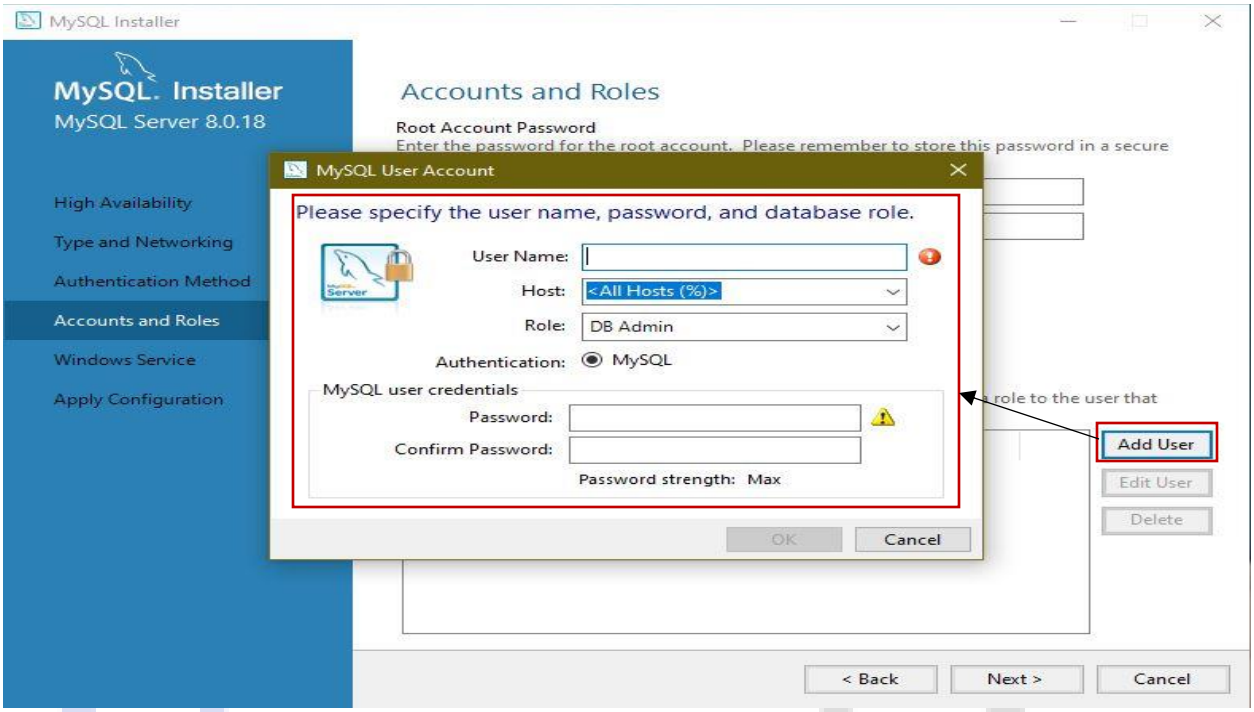
Step 11: Click Next



Step 12: Choose the root password

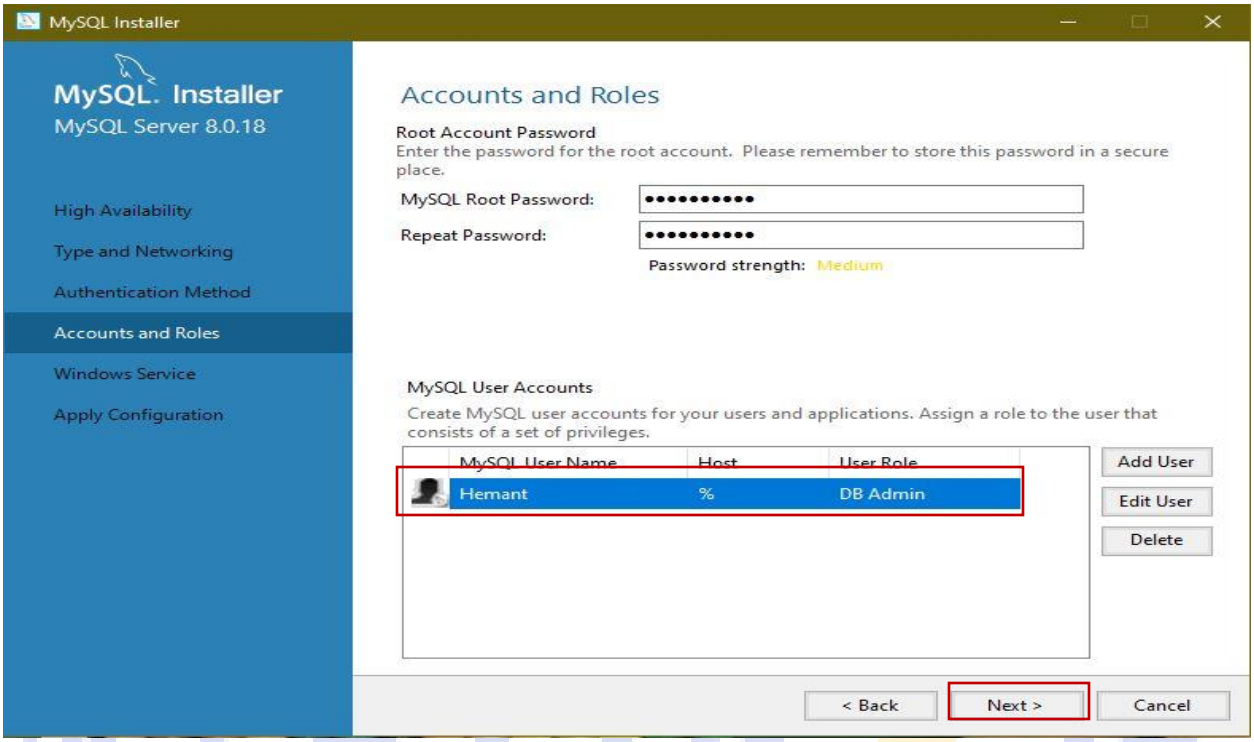


Step 13: Click on Add User and give the username and password.

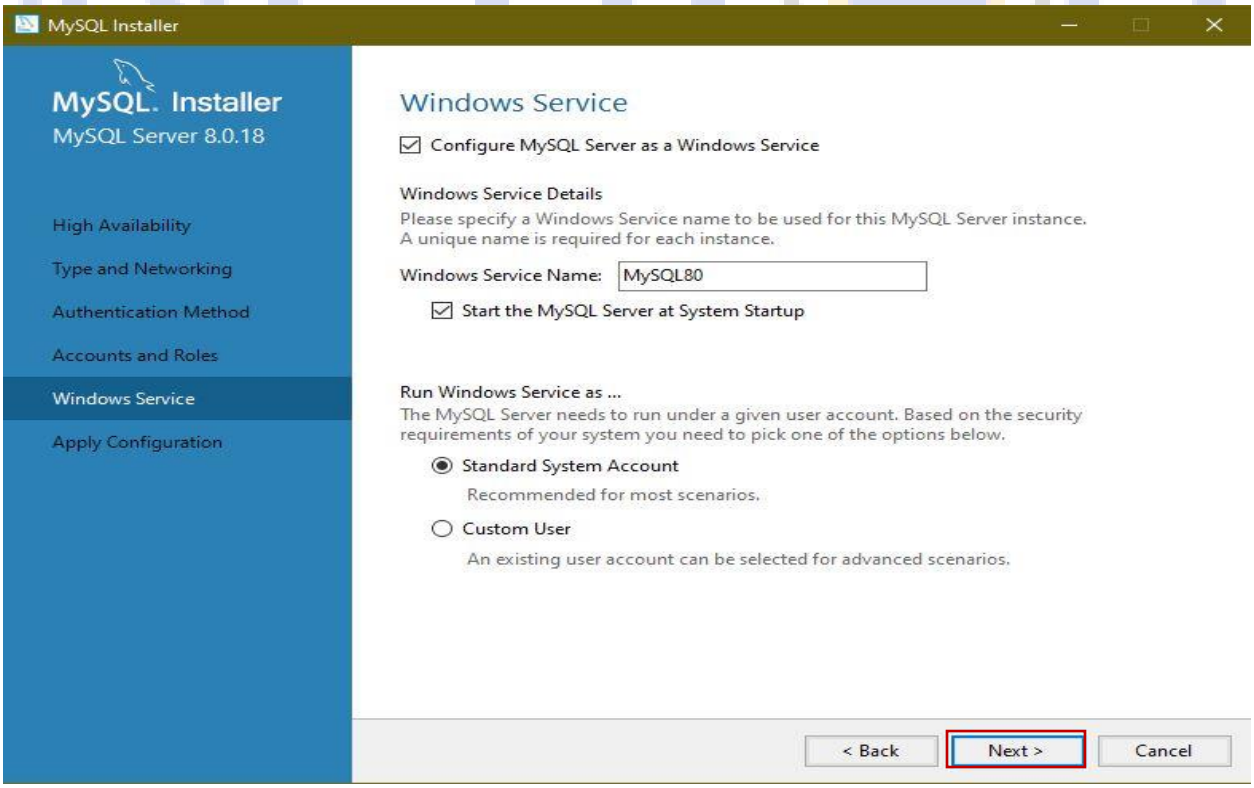


Step 14: After inserting the name and password click OK

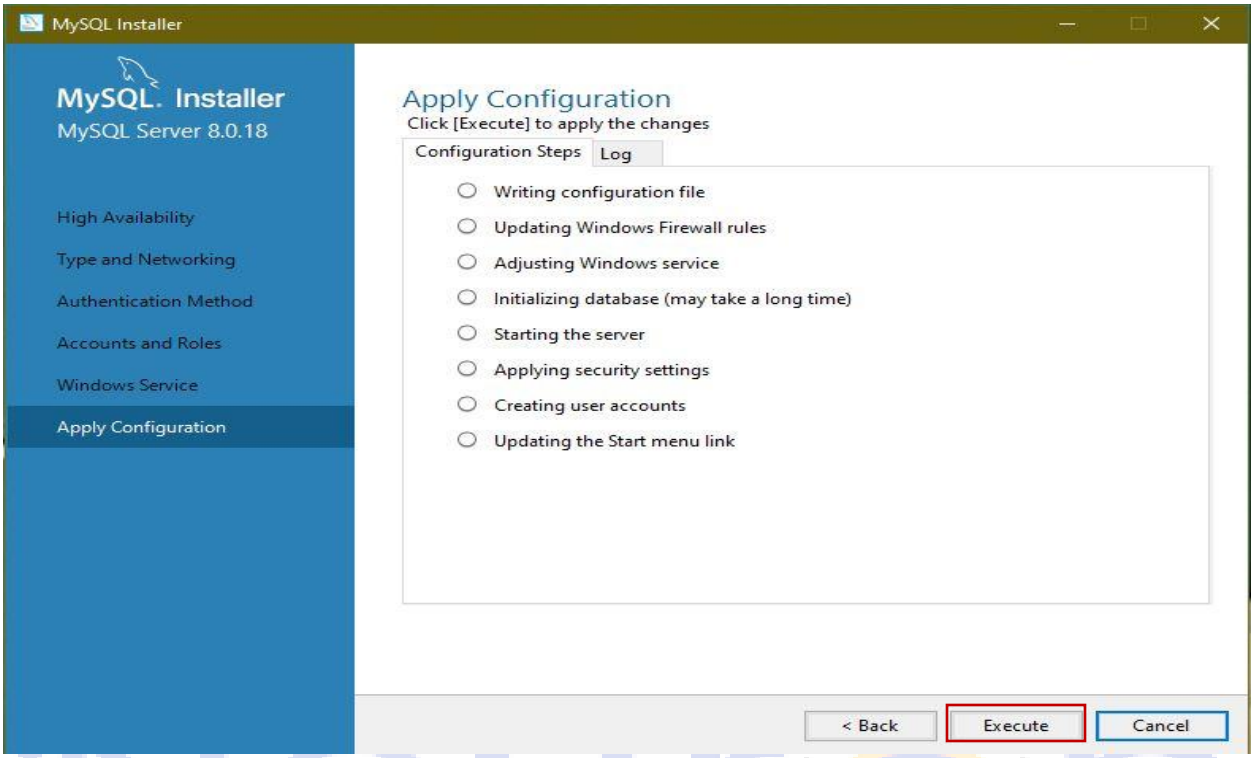
Step 15: After adding the user click Next



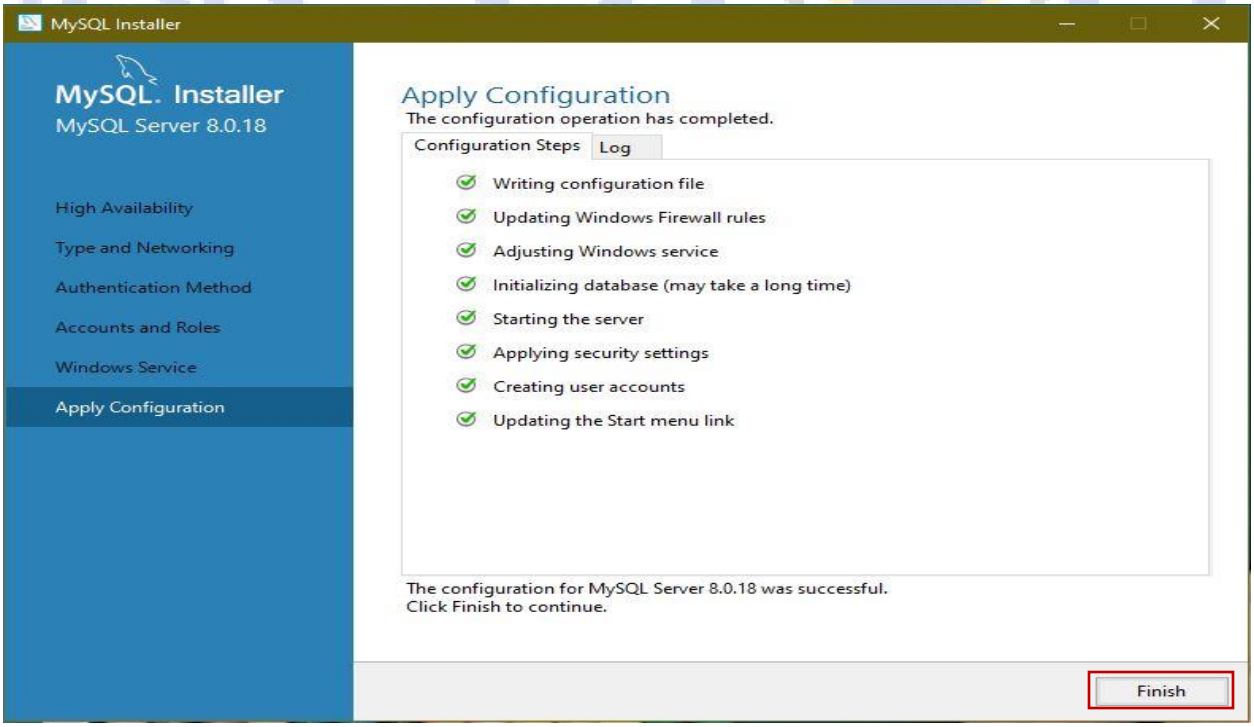
Step 16: Click Next



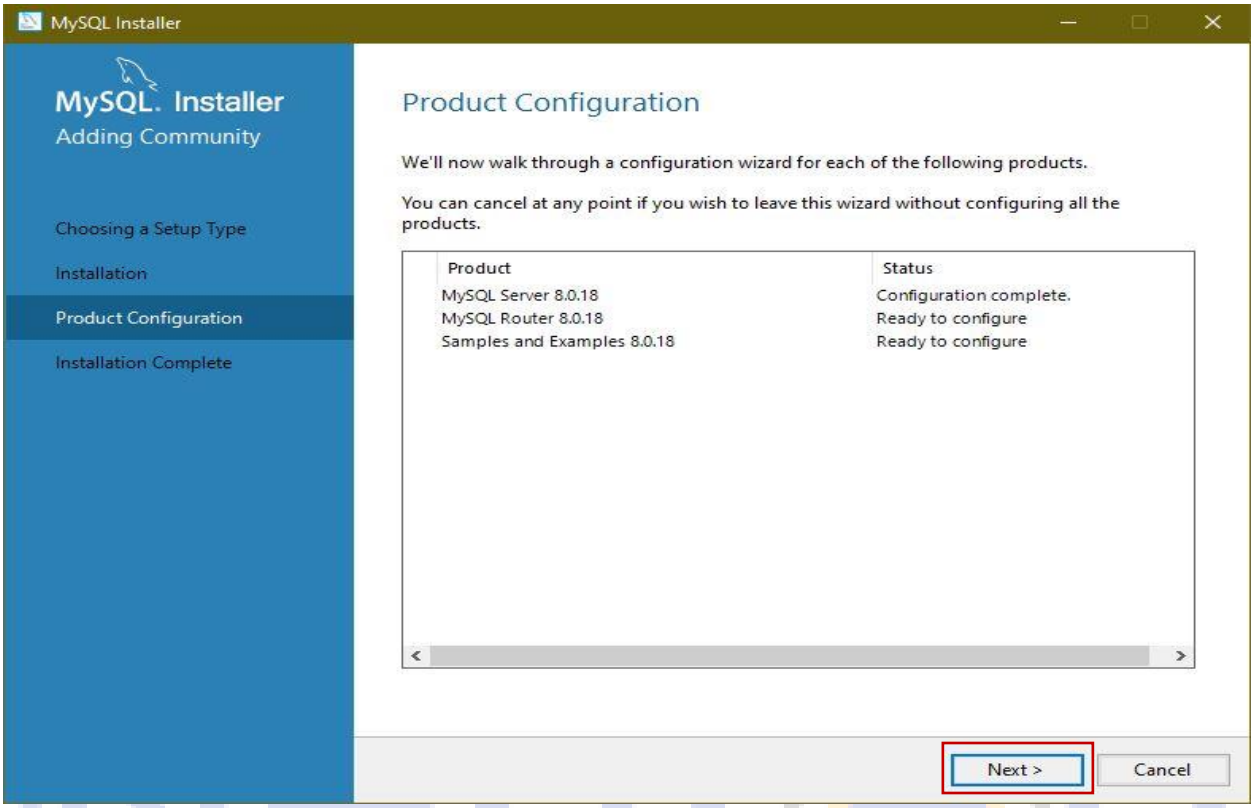
Step 17: Click on Execute



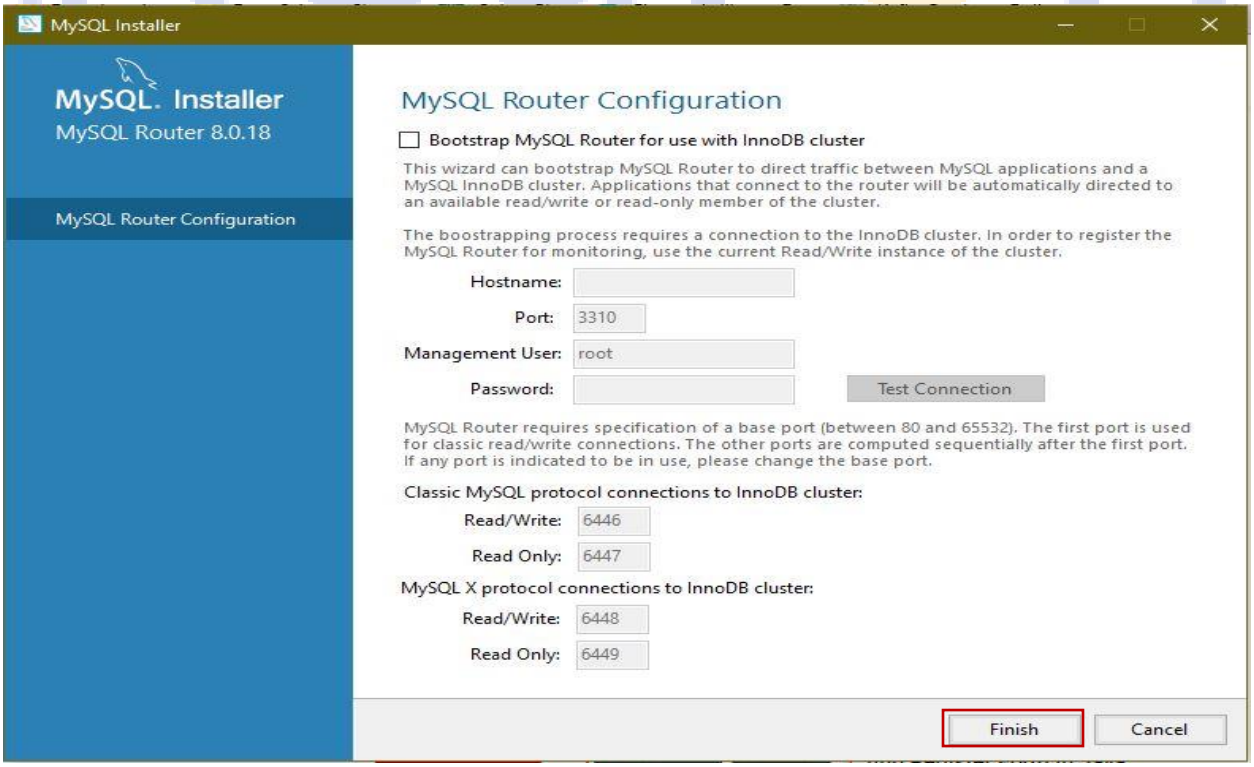
Step 18: After Clicking on execute, click Finish



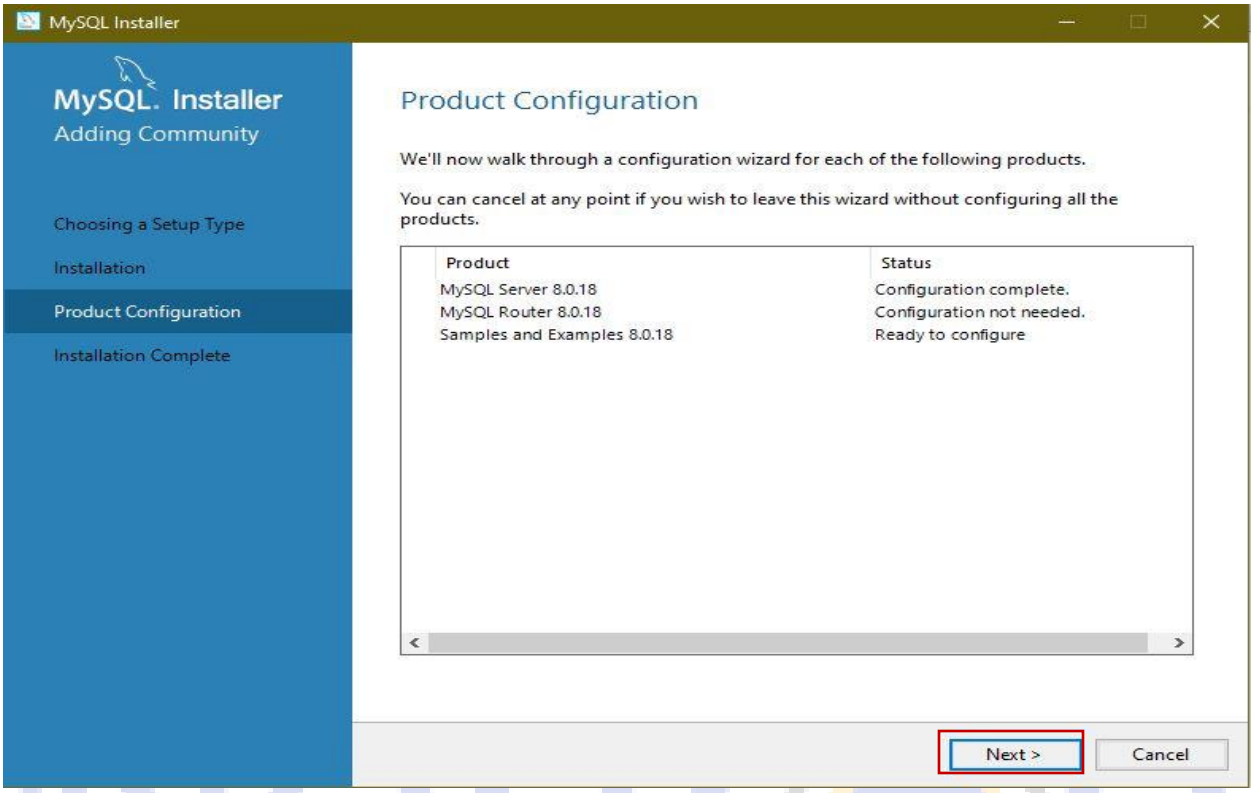
Step 19: Click Next



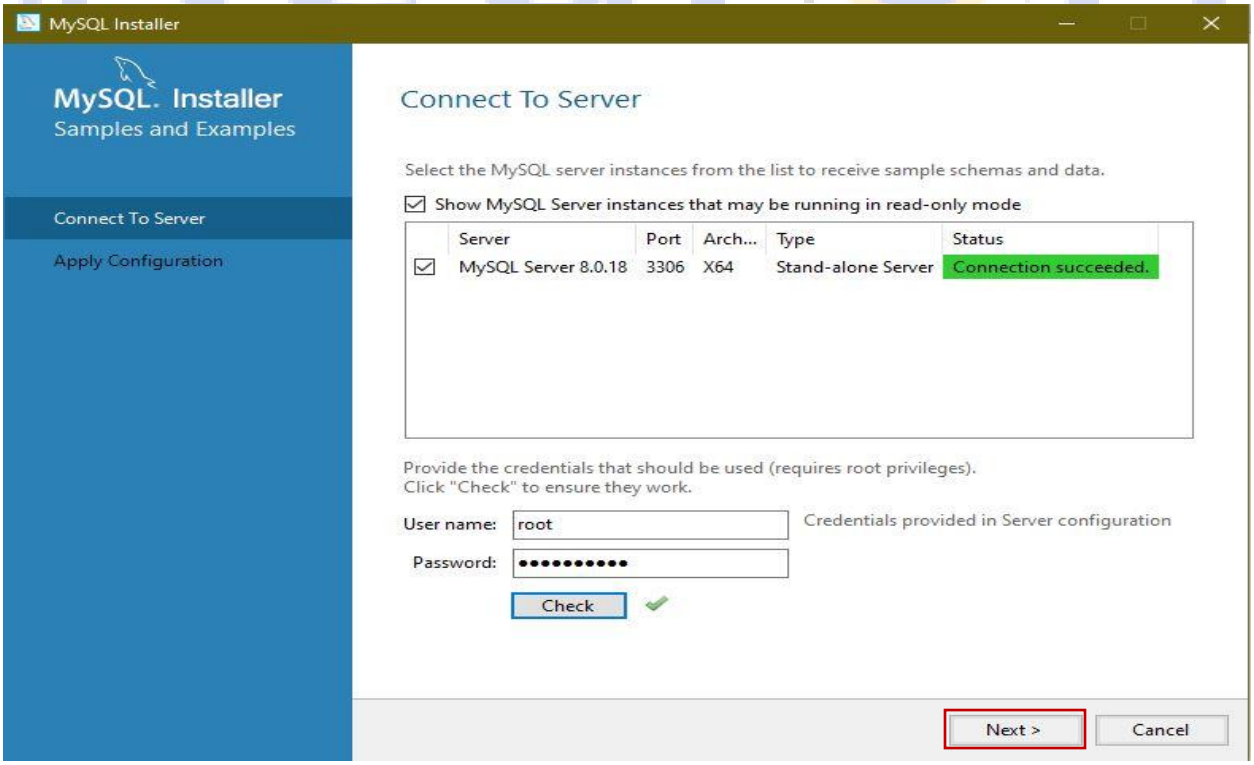
Step 20: Click on Finish



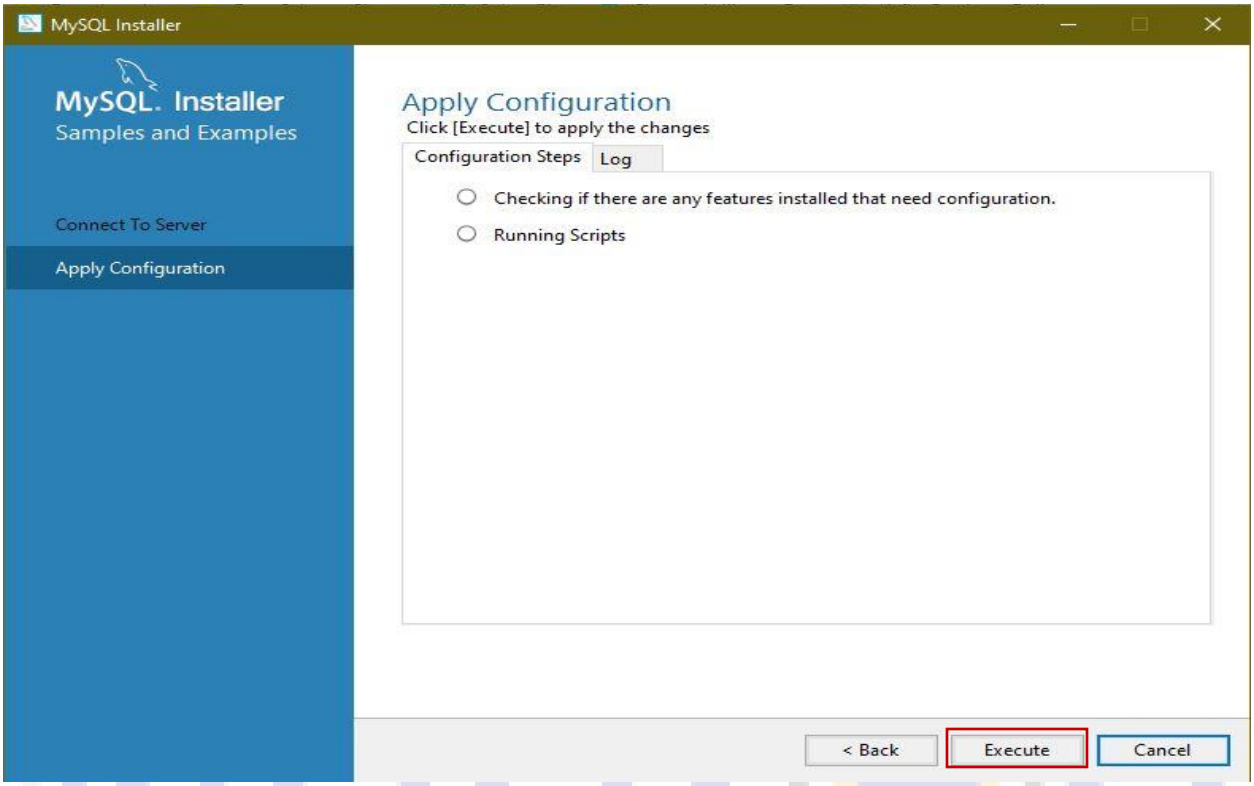
Step 21: Click Next.



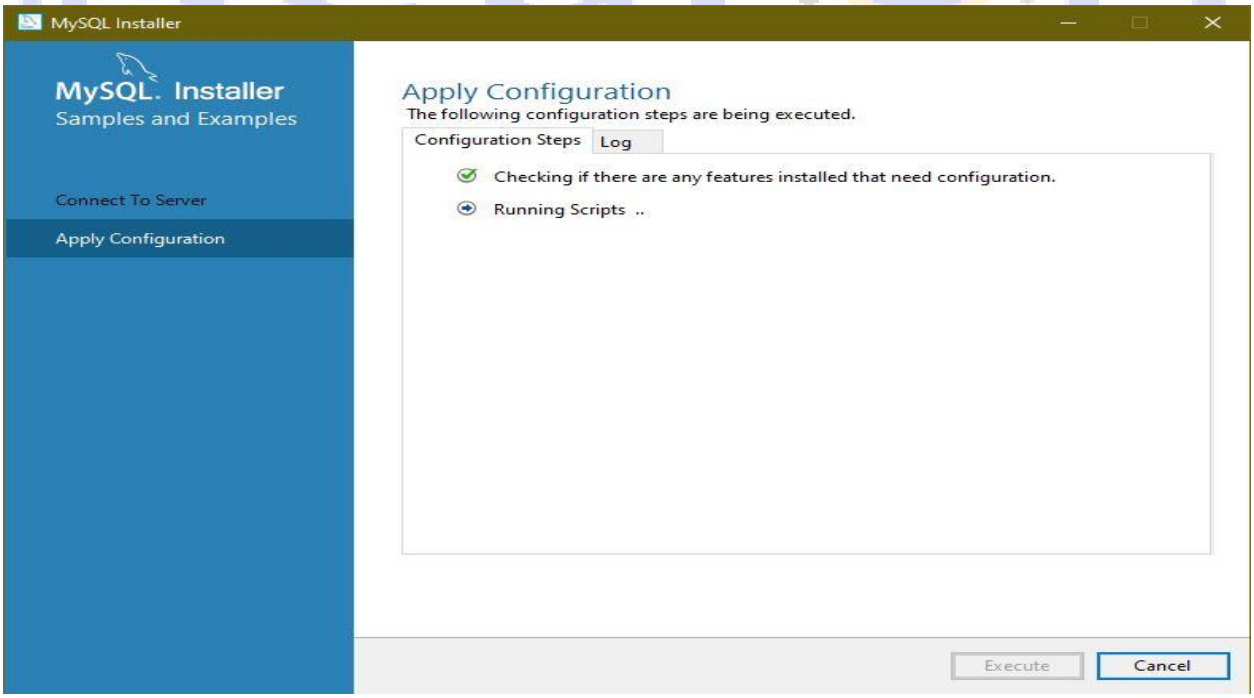
Step 22: Check the password and Click Next



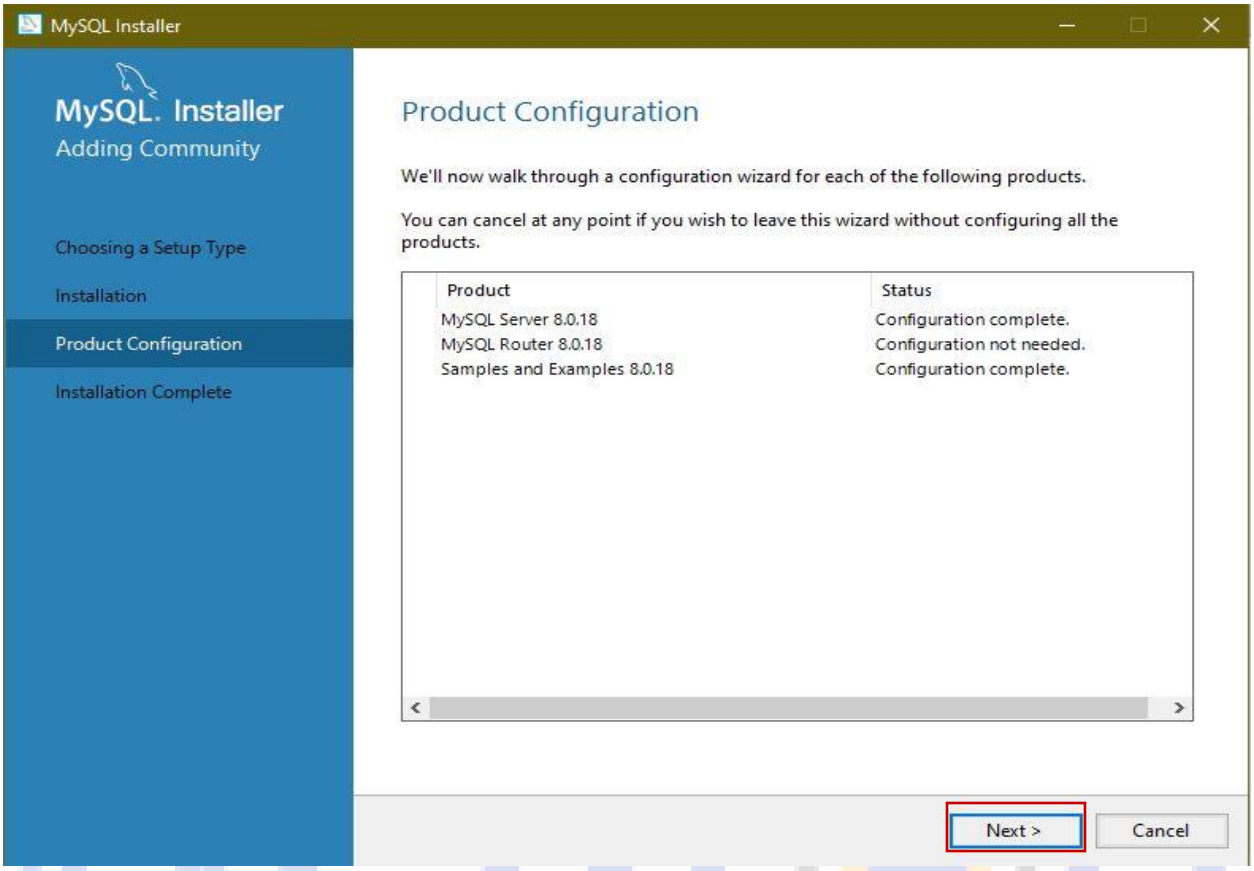
Step 23: Click Execute



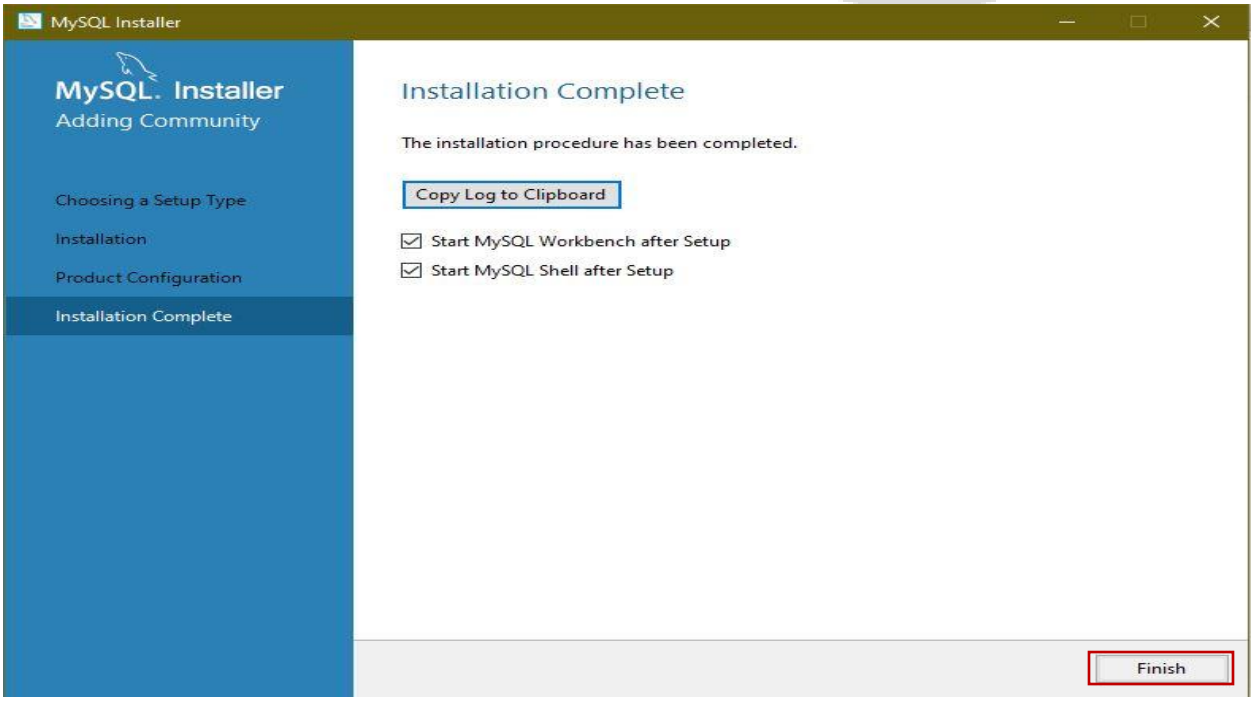
Step 24: After clicking on Execute



Step 25: Click Next

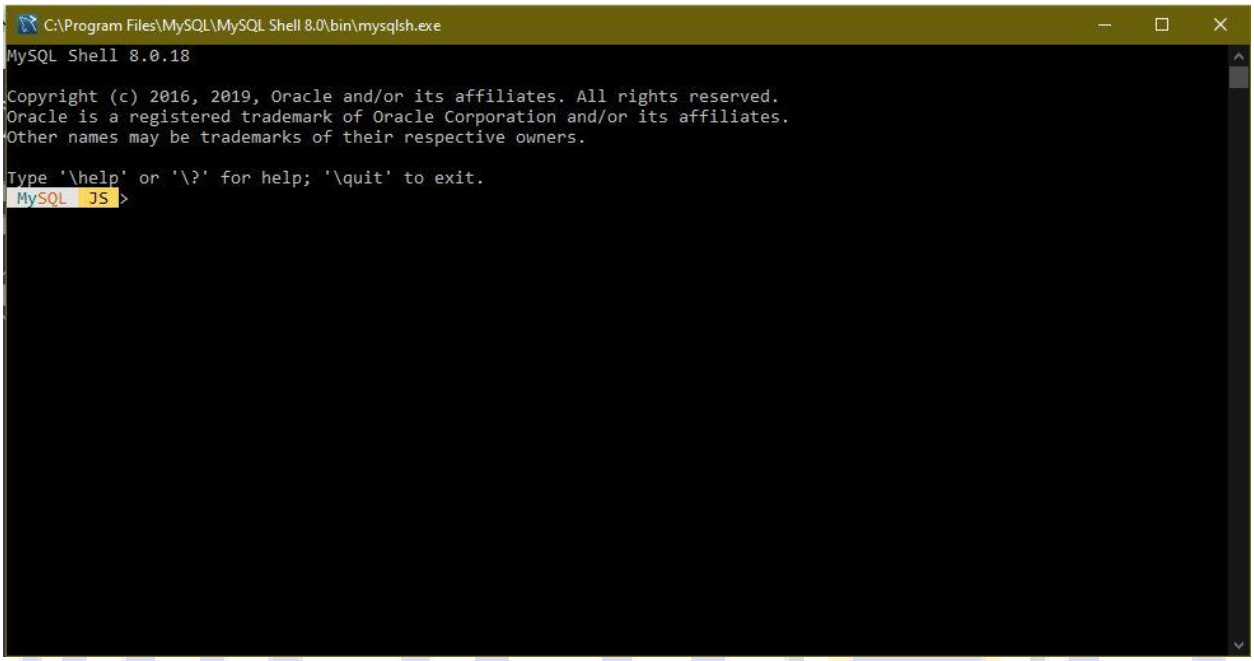


Step 26: Click Finish

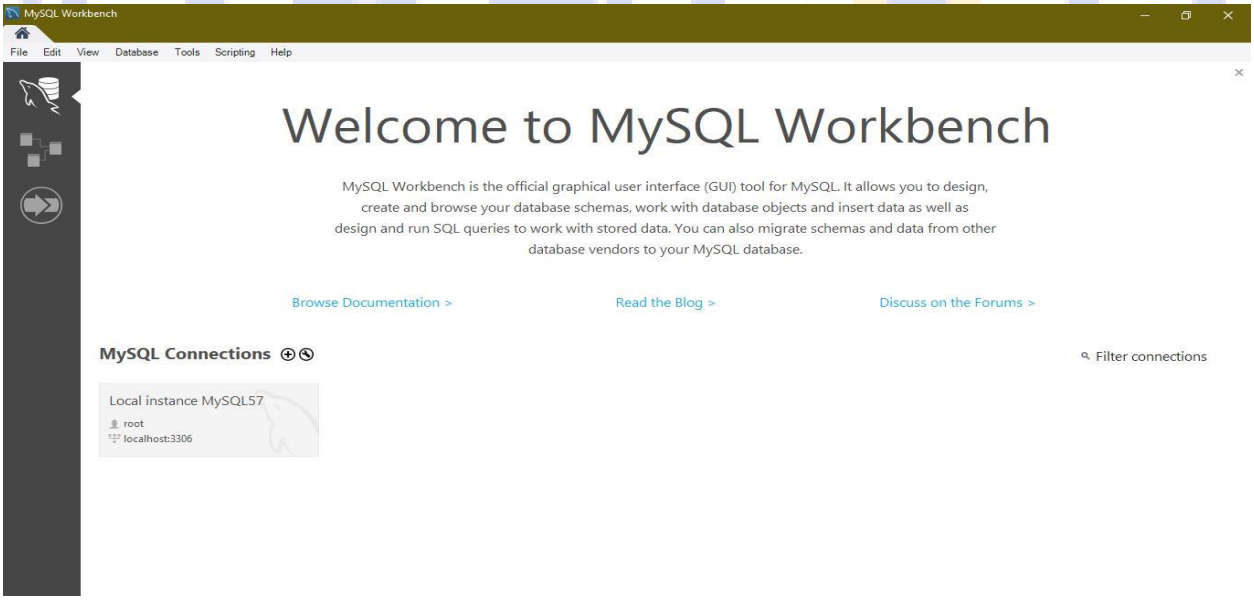


Step 27: After the successful installation of MySQL, two windows will open.

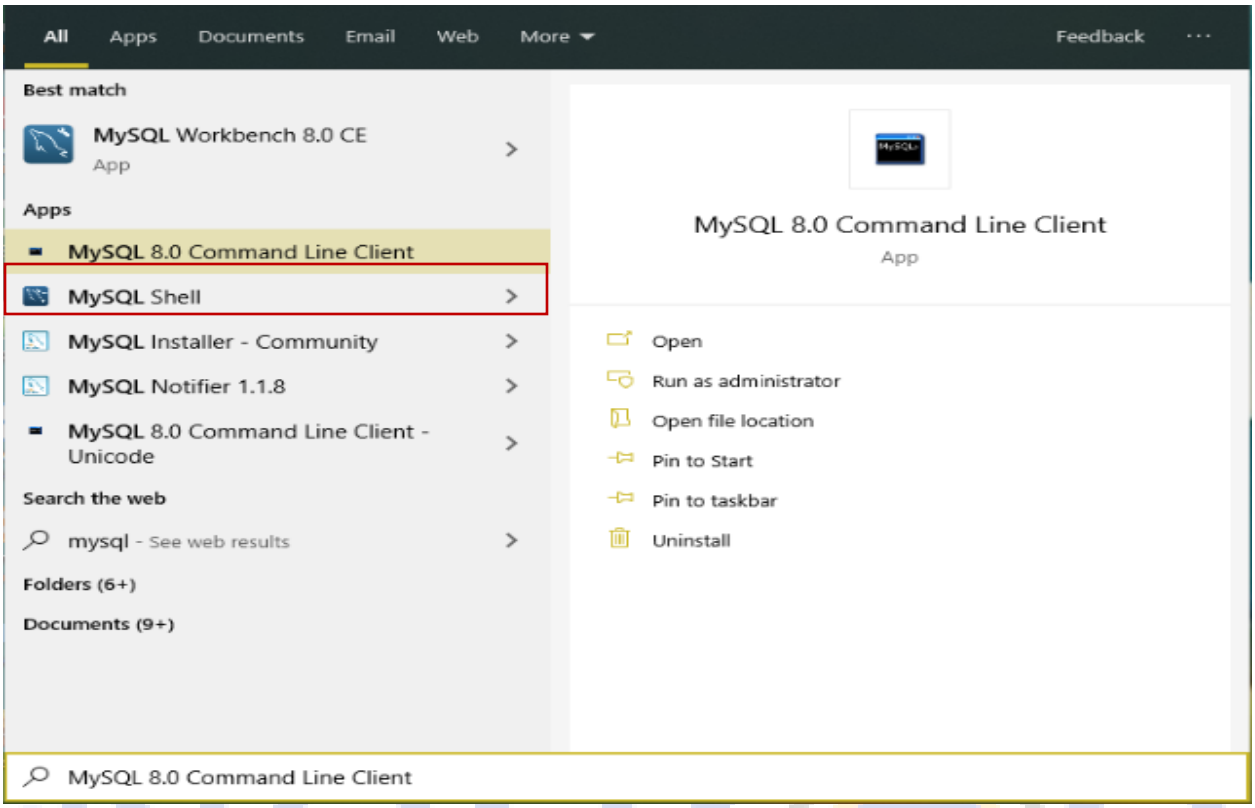
- MySQL Shell
- MySQL WorkBench



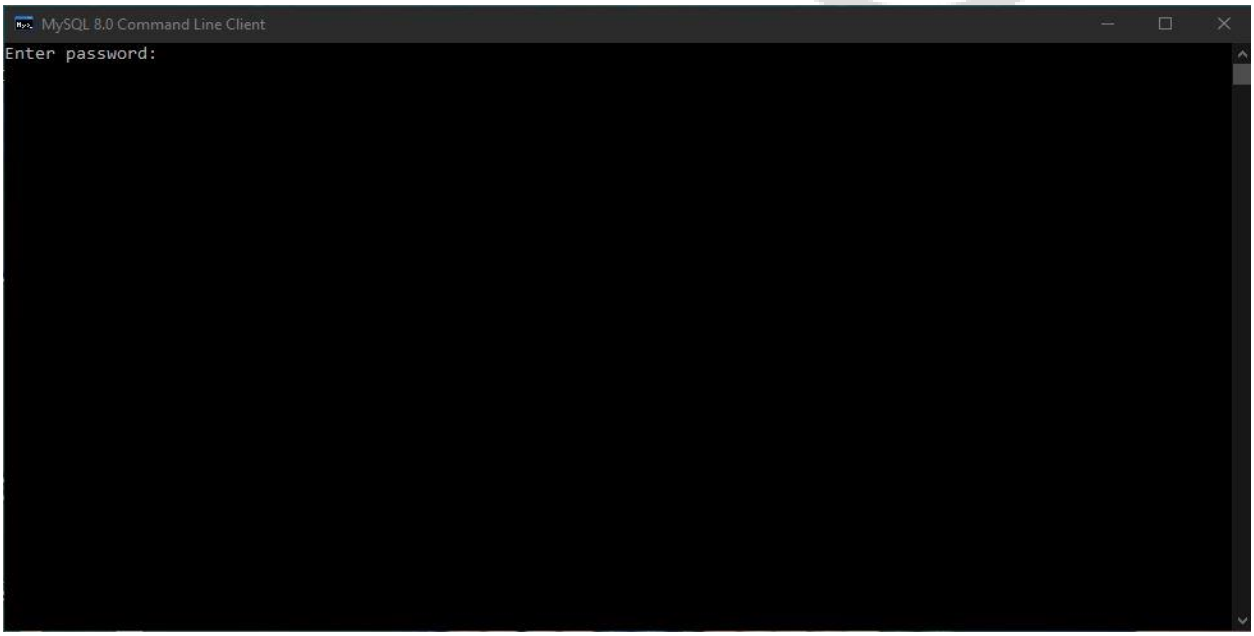
MySQL WorkBench will tell about database connectivity and other features of MySQL.



Step 28: Click on window Button and search for Open MySQL Command.



Step 29: Open MySQL Command-line Client and enter the password.



Step 29: After entering the password, your MySQL client will get connected with MySQL.

```
MySQL 8.0 Command Line Client
Enter password: **********
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 15
Server version: 8.0.18 MySQL Community Server - GPL

Copyright (c) 2000, 2019, Oracle and/or its affiliates. All rights reserved.

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

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

mysql>
```

Step 30: There are many in-build Databases in MySQL; we can type **show database.**

```
MySQL 8.0 Command Line Client
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sakila             |
| sys                |
| world              |
+--------------------+
6 rows in set (0.00 sec)

mysql>
```

Step 31: we can use any of the above databases by just typing **use database_name**

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sakila             |
| sys                |
| world              |
+--------------------+
6 rows in set (0.03 sec)

mysql> use sakila;
Database changed
```

3. SQL QUERY

A database most often contains tables. Some name identifies each table. The table includes records(rows) with Data. To access those records, we need SQL Syntax. Most of the action you need to perform Database by using the SQL Statement.

Note: SQL keywords are not case-sensitive (e.g., select as SELECT)

- The syntax of the language describes the language element.
- SQL syntax is somewhat like simple English sentences.
- Keywords include SELECT, UPDATE, WHERE, ORDER BY ETC.

Four fundamental operations that can apply to any databases are:

1. Read the Data -- **SELECT**
2. Insert the new Data -- **INSERT**
3. Update existing Data -- **UPDATE**
4. Remove Data –**DELETE**

These operations are referred to as the **CRUD** (Create, Read, Update, Delete).

The SQL SELECT QUERY

The SELECT statement permits you to read data from one or more tables.

The general syntax is:

SELECT first_name, last_name

FROM customer;

Example: Read the first_name and last_name from table **customer**.

```
mysql> select first_name,last_name from customer;
+-------------+--------------+
| first_name  | last_name    |
+-------------+--------------+
| MARY        | SMITH        |
| PATRICIA    | JOHNSON      |
| LINDA       | WILLIAMS     |
| BARBARA     | JONES        |
| ELIZABETH   | BROWN        |
| JENNIFER    | DAVIS        |
| MARIA       | MILLER       |
| SUSAN       | WILSON       |
| MARGARET    | MOORE        |
| DOROTHY     | TAYLOR       |
| LISA        | ANDERSON     |
| NANCY       | THOMAS       |
| KAREN       | JACKSON      |
| BETTY       | WHITE        |
| HELEN       | HARRIS       |
| SANDRA      | MARTIN       |
| DONNA       | THOMPSON     |
| CAROL       | GARCIA       |
| RUTH        | MARTINEZ     |
| SHARON      | ROBINSON     |
| MICHELLE    | CLARK        |
| LAURA       | RODRIGUEZ    |
| SARAH       | LEWIS        |
| KIMBERLY    | LEE          |
```

To select all columns, use *****

SELECT *****

FROM customer;

```
mysql> select * from customer;
+-------------+----------+------------+-------------+-----------------------------------------+------------+--------+---------------------+---------------------+
| customer_id | store_id | first_name | last_name   | email                                   | address_id | active | create_date         | last_update         |
+-------------+----------+------------+-------------+-----------------------------------------+------------+--------+---------------------+---------------------+
|           1 |        1 | MARY       | SMITH       | MARY.SMITH@sakilacustomer.org           |          5 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|           2 |        1 | PATRICIA   | JOHNSON     | PATRICIA.JOHNSON@sakilacustomer.org     |          6 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|           3 |        1 | LINDA      | WILLIAMS    | LINDA.WILLIAMS@sakilacustomer.org       |          7 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|           4 |        2 | BARBARA    | JONES       | BARBARA.JONES@sakilacustomer.org        |          8 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|           5 |        1 | ELIZABETH  | BROWN       | ELIZABETH.BROWN@sakilacustomer.org      |          9 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|           6 |        2 | JENNIFER   | DAVIS       | JENNIFER.DAVIS@sakilacustomer.org       |         10 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|           7 |        1 | MARIA      | MILLER      | MARIA.MILLER@sakilacustomer.org         |         11 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|           8 |        2 | SUSAN      | WILSON      | SUSAN.WILSON@sakilacustomer.org         |         12 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|           9 |        2 | MARGARET   | MOORE       | MARGARET.MOORE@sakilacustomer.org       |         13 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          10 |        1 | DOROTHY    | TAYLOR      | DOROTHY.TAYLOR@sakilacustomer.org       |         14 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          11 |        2 | LISA       | ANDERSON    | LISA.ANDERSON@sakilacustomer.org        |         15 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          12 |        1 | NANCY      | THOMAS      | NANCY.THOMAS@sakilacustomer.org         |         16 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          13 |        2 | KAREN      | JACKSON     | KAREN.JACKSON@sakilacustomer.org        |         17 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          14 |        2 | BETTY      | WHITE       | BETTY.WHITE@sakilacustomer.org          |         18 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          15 |        1 | HELEN      | HARRIS      | HELEN.HARRIS@sakilacustomer.org         |         19 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          16 |        2 | SANDRA     | MARTIN      | SANDRA.MARTIN@sakilacustomer.org        |         20 |      0 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          17 |        1 | DONNA      | THOMPSON    | DONNA.THOMPSON@sakilacustomer.org       |         21 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          18 |        2 | CAROL      | GARCIA      | CAROL.GARCIA@sakilacustomer.org         |         22 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          19 |        1 | RUTH       | MARTINEZ    | RUTH.MARTINEZ@sakilacustomer.org        |         23 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          20 |        2 | SHARON     | ROBINSON    | SHARON.ROBINSON@sakilacustomer.org      |         24 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
```

3.1. The SQL SELECT DISTINCT

The SELECT DISTINCT statement is to return the different values.

SELECT DISTINCT first_name

FROM customer;

```
mysql> select distinct first_name from customer;
+-------------+
| first_name  |
+-------------+
| MARY        |
| PATRICIA    |
| LINDA       |
| BARBARA     |
| ELIZABETH   |
| JENNIFER    |
| MARIA       |
| SUSAN       |
| MARGARET    |
| DOROTHY     |
| LISA        |
| NANCY       |
| KAREN       |
| BETTY       |
| HELEN       |
| SANDRA      |
| DONNA       |
```

3.2. The SQL WHERE CLAUSE

The WHERE clause allows the user to filter the data from the table. The WHERE clause allows the user to extract only those records that satisfy a specified condition.

When we access, the Text value

SQL requires single quotes around **text values** (many database systems will also use double quotes). And **numeric fields** should not be enclosed in quotes.

SELECT first_name FROM customer

WHERE last_name = ‘perry’;

```
mysql> select first_name from customer where last_name = 'perry';
+------------+
| first_name |
+------------+
| SARA       |
+------------+
1 row in set (0.03 sec)
```

When we access the Numeric field

SELECT first_name , last_name FROM customer WHERE active = 0;

```
mysql> select first_name, last_name from customer where active = 0;
+------------+-----------+
| first_name | last_name |
+------------+-----------+
| SANDRA     | MARTIN    |
| JUDITH     | COX       |
| SHEILA     | WELLS     |
| ERICA      | MATTHEWS  |
| HEIDI      | LARSON    |
| PENNY      | NEAL      |
| KENNETH    | GOODEN    |
| HARRY      | ARCE      |
| NATHAN     | RUNYON    |
| THEODORE   | CULP      |
| MAURICE    | CRAWLEY   |
| BEN        | EASTER    |
| CHRISTIAN  | JUNG      |
| JIMMIE     | EGGLESTON |
| TERRANCE   | ROUSH     |
+------------+-----------+
15 rows in set (0.00 sec)
```

Operators in where clause

=	Equal
>	Greater than
<	Less than
>=	Greater than equal
<=	Less than equal
< >	Not equal (also written as !=)
BETWEEN	Between a range
LIKE	Search for pattern
IN	Specify multiple possible values for a column

3.3. The SQL WHERE CLAUSE WITH AND, OR & NOT

A WHERE clause with AND:

SELECT first_name, email, address_id

FROM customer

WHERE fisrt_name = ‘IAN’ AND last_name = ‘STILL’

```
mysql> select first_name,        email, address_id from customer where first_name = 'IAN' and last_name = 'STILL';
+------------+-------+------------+
| first_name | email | address_id |
+------------+-------+------------+
| IAN        | STILL |        567 |
+------------+-------+------------+
1 row in set (0.00 sec)
```

A WHERE clause with OR:

UPDATE customer

SET first_name = ‘jingle’

WHERE last_name = ‘ GREY’;

```
mysql> update customer set first_name = 'jingle'
    -> where last_name = 'GREY';
Query OK, 1 row affected (0.30 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from customer where address_id = 586;
+-------------+----------+------------+-----------+-----------------------------+------------+--------+---------------------+---------------------+
| customer_id | store_id | first_name | last_name | email                       | address_id | active | create_date         | last_update         |
+-------------+----------+------------+-----------+-----------------------------+------------+--------+---------------------+---------------------+
|         580 |        1 | jingle     | GREY      | ROSS.GREY@sakilacustomer.org |        586 |      1 | 2006-02-14 22:04:37 | 2019-11-29 10:50:42 |
+-------------+----------+------------+-----------+-----------------------------+------------+--------+---------------------+---------------------+
1 row in set (0.01 sec)
```

A WHERE clause with NOT:

Select store_id, first_name,last_name, email, address_id FROM customer

WHERE NOT store_id = 2;

```
mysql> Select  store_id, first_name,last_name, email, address_id FROM customer
    -> WHERE NOT store_id = 2;
+----------+-------------+-------------+----------------------------------------+------------+
| store_id | first_name  | last_name   | email                                  | address_id |
+----------+-------------+-------------+----------------------------------------+------------+
|        1 | MARY        | SMITH       | MARY.SMITH@sakilacustomer.org          |          5 |
|        1 | PATRICIA    | JOHNSON     | PATRICIA.JOHNSON@sakilacustomer.org    |          6 |
|        1 | LINDA       | WILLIAMS    | LINDA.WILLIAMS@sakilacustomer.org      |          7 |
|        1 | ELIZABETH   | BROWN       | ELIZABETH.BROWN@sakilacustomer.org     |          9 |
|        1 | MARIA       | MILLER      | MARIA.MILLER@sakilacustomer.org        |         11 |
|        1 | DOROTHY     | TAYLOR      | DOROTHY.TAYLOR@sakilacustomer.org      |         14 |
```

3.4. The SQL ORDER BY

Order by is used to print the values from the table in order(ascending or descending)

Order By in Descending order

SELECT first_name, last_name,email

FROM customer

ORDER BY first_name DESC;

```
mysql> select first_name, last_name, email from customer order by first_name desc;
+-------------+--------------+----------------------------------------+
| first_name  | last_name    | email                                  |
+-------------+--------------+----------------------------------------+
| ZACHARY     | HITE         | ZACHARY.HITE@sakilacustomer.org        |
| YVONNE      | WATKINS      | YVONNE.WATKINS@sakilacustomer.org      |
| YOLANDA     | WEAVER       | YOLANDA.WEAVER@sakilacustomer.org      |
| WILMA       | RICHARDS     | WILMA.RICHARDS@sakilacustomer.org      |
| WILLIE      | HOWELL       | WILLIE.HOWELL@sakilacustomer.org       |
| WILLIE      | MARKHAM      | WILLIE.MARKHAM@sakilacustomer.org      |
| WILLIAM     | SATTERFIELD  | WILLIAM.SATTERFIELD@sakilacustomer.org |
| WILLARD     | LUMPKIN      | WILLARD.LUMPKIN@sakilacustomer.org     |
```

Order By in Ascending order

SELECT first_name, last_name,email

FROM customer

ORDER BY first_name ASC;

```
mysql> select first_name, last_name, email from customer order by first_name asc;
+-------------+--------------+-----------------------------------------+
| first_name  | last_name    | email                                   |
+-------------+--------------+-----------------------------------------+
| AARON       | SELBY        | AARON.SELBY@sakilacustomer.org          |
| ADAM        | GOOCH        | ADAM.GOOCH@sakilacustomer.org           |
| ADRIAN      | CLARY        | ADRIAN.CLARY@sakilacustomer.org         |
| AGNES       | BISHOP       | AGNES.BISHOP@sakilacustomer.org         |
| ALAN        | KAHN         | ALAN.KAHN@sakilacustomer.org            |
| ALBERT      | CROUSE       | ALBERT.CROUSE@sakilacustomer.org        |
| ALBERTO     | HENNING      | ALBERTO.HENNING@sakilacustomer.org      |
| ALEX        | GRESHAM      | ALEX.GRESHAM@sakilacustomer.org         |
| ALEXANDER   | FENNELL      | ALEXANDER.FENNELL@sakilacustomer.org    |
```

3.5. The SQL SELECT TOP CLAUSE

The **SELECT TOP** is used to specify the number of records from the to return. The SELECT TOP is useful on large tables with millions of records. It is returning a large number of records that can impact performance.

Note: Not all database systems support the SELECT TOP clause. MySQL supports the LIMIT clause to select a limited number of records, while Oracle uses ROWNUM.

MySQL Syntax:

```
SELECT first_name, last_name,email
    FROM customer WHERE first_name = 'AUSTIN'
    LIMIT 20;
```

```
mysql> select first_name,last_name,email from customer where first_name = 'AUSTIN' limit 20;
+------------+-----------+-----------------------------------+
| first_name | last_name | email                             |
+------------+-----------+-----------------------------------+
| AUSTIN     | CINTRON   | AUSTIN.CINTRON@sakilacustomer.org |
+------------+-----------+-----------------------------------+
1 row in set (0.00 sec)

mysql> select first_name,last_name,email from customer limit 20;
+------------+-----------+-------------------------------------+
| first_name | last_name | email                               |
+------------+-----------+-------------------------------------+
| MARY       | SMITH     | MARY.SMITH@sakilacustomer.org       |
| PATRICIA   | JOHNSON   | PATRICIA.JOHNSON@sakilacustomer.org |
| LINDA      | WILLIAMS  | LINDA.WILLIAMS@sakilacustomer.org   |
| BARBARA    | JONES     | BARBARA.JONES@sakilacustomer.org    |
| ELIZABETH  | BROWN     | ELIZABETH.BROWN@sakilacustomer.org  |
| JENNIFER   | DAVIS     | JENNIFER.DAVIS@sakilacustomer.org   |
| MARIA      | MILLER    | MARIA.MILLER@sakilacustomer.org     |
| SUSAN      | WILSON    | SUSAN.WILSON@sakilacustomer.org     |
| MARGARET   | MOORE     | MARGARET.MOORE@sakilacustomer.org   |
| DOROTHY    | TAYLOR    | DOROTHY.TAYLOR@sakilacustomer.org   |
| LISA       | ANDERSON  | LISA.ANDERSON@sakilacustomer.org    |
| NANCY      | THOMAS    | NANCY.THOMAS@sakilacustomer.org     |
| KAREN      | JACKSON   | KAREN.JACKSON@sakilacustomer.org    |
| BETTY      | WHITE     | BETTY.WHITE@sakilacustomer.org      |
| HELEN      | HARRIS    | HELEN.HARRIS@sakilacustomer.org     |
| SANDRA     | MARTIN    | SANDRA.MARTIN@sakilacustomer.org    |
| DONNA      | THOMPSON  | DONNA.THOMPSON@sakilacustomer.org   |
| CAROL      | GARCIA    | CAROL.GARCIA@sakilacustomer.org     |
| RUTH       | MARTINEZ  | RUTH.MARTINEZ@sakilacustomer.org    |
| SHARON     | ROBINSON  | SHARON.ROBINSON@sakilacustomer.org  |
+------------+-----------+-------------------------------------+
20 rows in set (0.00 sec)
```

3.6. The SQL MIN() AND MAX() FUNCTION

The MIN() function in SQL returns the smallest value of the selected column from the table. The MAX() function in SQL returns the largest value of the selected column from the table.

MIN() Syntax

SELECT MIN(address_id)

FROM customer;

```
mysql> select min(address_id) from customer;
+-----------------+
| min(address_id) |
+-----------------+
|               5 |
+-----------------+
1 row in set (0.03 sec)
```

MAX() Syntax

SELECT MAX(address_id)

FROM customer;

```
mysql> select max(address_id) from customer;
+-----------------+
| max(address_id) |
+-----------------+
|             605 |
+-----------------+
1 row in set (0.00 sec)
```

3.7. The SQL COUNT(), AVG() AND SUM() FUNCTION

The **COUNT()** function gives the number of rows that matches specified conditions. And the **AVG()** function in SQL returns the average value of a numeric column. The **SUM()** function in SQL returns the total sum of a numeric column.

COUNT() Syntax

SELECT COUNT(email)

FROM customer;

```
mysql> select count(email) from customer;
+--------------+
| count(email) |
+--------------+
|          599 |
+--------------+
1 row in set (0.00 sec)
```

AVG() Syntax

SELECT AVG(active)

FROM customer;

```
mysql> select avg(active) from customer;
+-------------+
| avg(active) |
+-------------+
|      0.9750 |
+-------------+
1 row in set (0.03 sec)
```

SUM() Syntax

SELECT SUM(active)

FROM customer

```
mysql> select sum(active) from customer;
+-------------+
| sum(active) |
+-------------+
|         584 |
+-------------+
1 row in set (0.00 sec)
```

3.8. The SQL LIKE-OPERATOR

The **LIKE** operator is used with the WHERE clause to find for a specified pattern in an attribute. The two wildcards are used in conjunction with the LIKE operator:

- **%** - it represents zero, one, or multiple characters
- **_** - it represents a single character

Note: MS Access uses an asterisk (*) in place of the percent sign (%)and a question mark (?) in place of the underscore (_).

The '%' and the '_' can also be used in combinations.

LIKE Syntax

SELECT column1, column2, ...

FROM table_name

WHERE column LIKE pattern;

Selects all columns of the customer with a first_name starting with "D".

```
SELECT * FROM customer

WHERE first_name LIKE 'D%';
```

```
mysql> select * from customer where first_name like 'D%';
+-------------+----------+------------+-----------+-----------------------------------+------------+--------+---------------------+---------------------+
| customer_id | store_id | first_name | last_name | email                             | address_id | active | create_date         | last_update         |
+-------------+----------+------------+-----------+-----------------------------------+------------+--------+---------------------+---------------------+
|          10 |        1 | DOROTHY    | TAYLOR    | DOROTHY.TAYLOR@sakilacustomer.org |         14 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          17 |        1 | DONNA      | THOMPSON  | DONNA.THOMPSON@sakilacustomer.org |         21 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          25 |        1 | DEBORAH    | WALKER    | DEBORAH.WALKER@sakilacustomer.org |         29 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          39 |        1 | DEBRA      | NELSON    | DEBRA.NELSON@sakilacustomer.org   |         43 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          50 |        1 | DIANE      | COLLINS   | DIANE.COLLINS@sakilacustomer.org  |         54 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          55 |        2 | DORIS      | REED      | DORIS.REED@sakilacustomer.org     |         59 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          74 |        1 | DENISE     | KELLY     | DENISE.KELLY@sakilacustomer.org   |         78 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          96 |        1 | DIANA      | ALEXANDER | DIANA.ALEXANDER@sakilacustomer.org |        100 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|         105 |        1 | DAWN       | SULLIVAN  | DAWN.SULLIVAN@sakilacustomer.org  |        109 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|         141 |        1 | DEBBIE     | REYES     | DEBBIE.REYES@sakilacustomer.org   |        145 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|         150 |        2 | DANIELLE   | DANIELS   | DANIELLE.DANIELS@sakilacustomer.org |        154 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|         157 |        2 | DARLENE    | ROSE      | DARLENE.ROSE@sakilacustomer.org   |        161 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|         171 |        2 | DOLORES    | WAGNER    | DOLORES.WAGNER@sakilacustomer.org |        175 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|         179 |        1 | DANA       | HART      | DANA.HART@sakilacustomer.org      |        183 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|         222 |        2 | DELORES    | HANSEN    | DELORES.HANSEN@sakilacustomer.org |        226 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|         249 |        2 | DORA       | MEDINA    | DORA.MEDINA@sakilacustomer.org    |        253 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|         261 |        1 | DEANNA     | BYRD      | DEANNA.BYRD@sakilacustomer.org    |        266 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|         279 |        2 | DIANNE     | SHELTON   | DIANNE.SHELTON@sakilacustomer.org |        284 |      1 | 2006-02-14 22:04:37 | 2006-02-15 04:57:20 |
|         295 |        1 | DAISY      | BATES     | DAISY.BATES@sakilacustomer.org    |        300 |      1 | 2006-02-14 22:04:37 | 2006-02-15 04:57:20 |
|         304 |        2 | DAVID      | ROYAL     | DAVID.ROYAL@sakilacustomer.org    |        309 |      1 | 2006-02-14 22:04:37 | 2006-02-15 04:57:20 |
|         310 |        2 | DANIEL     | CABRAL    | DANIEL.CABRAL@sakilacustomer.org  |        315 |      1 | 2006-02-14 22:04:37 | 2006-02-15 04:57:20 |
```

Selects all columns of the customer with a first_name Ending with "E":

```
SELECT * FROM customer

WHERE first_name LIKE '%E';
```

```
mysql> select * from customer where first_name like '%E';
+-------------+----------+------------+-----------+-----------------------------------+------------+--------+---------------------+---------------------+
| customer_id | store_id | first_name | last_name | email                             | address_id | active | create_date         | last_update         |
+-------------+----------+------------+-----------+-----------------------------------+------------+--------+---------------------+---------------------+
|          21 |        1 | MICHELLE   | CLARK     | MICHELLE.CLARK@sakilacustomer.org |         25 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          41 |        1 | STEPHANIE  | MITCHELL  | STEPHANIE.MITCHELL@sakilacustomer.org |         45 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          43 |        2 | CHRISTINE  | ROBERTS   | CHRISTINE.ROBERTS@sakilacustomer.org |         47 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          44 |        1 | MARIE      | TURNER    | MARIE.TURNER@sakilacustomer.org   |         48 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          46 |        2 | CATHERINE  | CAMPBELL  | CATHERINE.CAMPBELL@sakilacustomer.org |         50 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          49 |        2 | JOYCE      | EDWARDS   | JOYCE.EDWARDS@sakilacustomer.org  |         53 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          50 |        1 | DIANE      | COLLINS   | DIANE.COLLINS@sakilacustomer.org  |         54 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          51 |        1 | ALICE      | STEWART   | ALICE.STEWART@sakilacustomer.org  |         55 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          52 |        1 | JULIE      | SANCHEZ   | JULIE.SANCHEZ@sakilacustomer.org  |         56 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          61 |        2 | KATHERINE  | RIVERA    | KATHERINE.RIVERA@sakilacustomer.org |         65 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          65 |        2 | ROSE       | HOWARD    | ROSE.HOWARD@sakilacustomer.org    |         69 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          66 |        2 | JANICE     | WARD      | JANICE.WARD@sakilacustomer.org    |         70 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          68 |        1 | NICOLE     | PETERSON  | NICOLE.PETERSON@sakilacustomer.org |         72 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          74 |        1 | DENISE     | KELLY     | DENISE.KELLY@sakilacustomer.org   |         78 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          76 |        2 | IRENE      | PRICE     | IRENE.PRICE@sakilacustomer.org    |         80 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          77 |        2 | JANE       | BENNETT   | JANE.BENNETT@sakilacustomer.org   |         81 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          83 |        1 | LOUISE     | JENKINS   | LOUISE.JENKINS@sakilacustomer.org |         87 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          85 |        2 | ANNE       | POWELL    | ANNE.POWELL@sakilacustomer.org    |         89 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          86 |        2 | JACQUELINE | LONG      | JACQUELINE.LONG@sakilacustomer.org |         90 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          88 |        2 | BONNIE     | HUGHES    | BONNIE.HUGHES@sakilacustomer.org  |         92 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          97 |        2 | ANNIE      | RUSSELL   | ANNIE.RUSSELL@sakilacustomer.org  |        101 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|         106 |        1 | CONNIE     | WALLACE   | CONNIE.WALLACE@sakilacustomer.org |        110 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
```

Selects all columns of the customer with a first_name that have "or" in any position.

SELECT * FROM customer

WHERE first_name LIKE '%or%';

```
mysql> select * from customer where first_name like '%or%';
+-------------+----------+------------+------------+-----------------------------------+------------+--------+---------------------+---------------------+
| customer_id | store_id | first_name | last_name  | email                             | address_id | active | create_date         | last_update         |
+-------------+----------+------------+------------+-----------------------------------+------------+--------+---------------------+---------------------+
|          10 |        1 | DOROTHY    | TAYLOR     | DOROTHY.TAYLOR@sakilacustomer.org |         14 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          25 |        1 | DEBORAH    | WALKER     | DEBORAH.WALKER@sakilacustomer.org |         29 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          55 |        2 | DORIS      | REED       | DORIS.REED@sakilacustomer.org     |         59 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          56 |        1 | GLORIA     | COOK       | GLORIA.COOK@sakilacustomer.org    |         60 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          78 |        1 | LORI       | WOOD       | LORI.WOOD@sakilacustomer.org      |         82 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          94 |        1 | NORMA      | GONZALES   | NORMA.GONZALES@sakilacustomer.org |         98 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|         107 |        1 | FLORENCE   | WOODS      | FLORENCE.WOODS@sakilacustomer.org |        111 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|         116 |        1 | VICTORIA   | GIBSON     | VICTORIA.GIBSON@sakilacustomer.org |        120 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|         128 |        1 | MARJORIE   | TUCKER     | MARJORIE.TUCKER@sakilacustomer.org |        132 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|         148 |        1 | ELEANOR    | HUNT       | ELEANOR.HUNT@sakilacustomer.org   |        152 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|         165 |        2 | LORRAINE   | STEPHENS   | LORRAINE.STEPHENS@sakilacustomer.org |        169 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|         171 |        2 | DOLORES    | WAGNER     | DOLORES.WAGNER@sakilacustomer.org |        175 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|         189 |        1 | LORETTA    | CARPENTER  | LORETTA.CARPENTER@sakilacustomer.org |        193 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|         222 |        2 | DELORES    | HANSEN     | DELORES.HANSEN@sakilacustomer.org |        226 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|         231 |        1 | GEORGIA    | JACOBS     | GEORGIA.JACOBS@sakilacustomer.org |        235 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|         249 |        2 | DORA       | MEDINA     | DORA.MEDINA@sakilacustomer.org    |        253 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
```

Selects all columns of the customer with a first_name that starts with "a" and ends with "o":

SELECT * FROM customer

WHERE first_name LIKE 'a%o';

```
mysql> select * from customer where first_name like 'a%o';
+-------------+----------+------------+-----------+-----------------------------------+------------+--------+---------------------+---------------------+
| customer_id | store_id | first_name | last_name | email                             | address_id | active | create_date         | last_update         |
+-------------+----------+------------+-----------+-----------------------------------+------------+--------+---------------------+---------------------+
|         398 |        1 | ANTONIO    | MEEK      | ANTONIO.MEEK@sakilacustomer.org   |        403 |      1 | 2006-02-14 22:04:37 | 2006-02-15 04:57:20 |
|         556 |        2 | ARMANDO    | GRUBER    | ARMANDO.GRUBER@sakilacustomer.org |        562 |      1 | 2006-02-14 22:04:37 | 2006-02-15 04:57:20 |
|         567 |        2 | ALFREDO    | MCADAMS   | ALFREDO.MCADAMS@sakilacustomer.org |        573 |      1 | 2006-02-14 22:04:37 | 2006-02-15 04:57:20 |
|         568 |        2 | ALBERTO    | HENNING   | ALBERTO.HENNING@sakilacustomer.org |        574 |      1 | 2006-02-14 22:04:37 | 2006-02-15 04:57:20 |
+-------------+----------+------------+-----------+-----------------------------------+------------+--------+---------------------+---------------------+
4 rows in set (0.00 sec)
```

Selects all columns of the customer with a first_name that starts with "a" and are at least six characters in length:

SELECT * FROM customer

WHERE first_name LIKE 'a__%';

```
mysql> select * from customer where first_name like 'a_____%';
+-------------+----------+------------+-----------+-------------------------------------+------------+--------+---------------------+---------------------+
| customer_id | store_id | first_name | last_name | email                               | address_id | active | create_date         | last_update         |
+-------------+----------+------------+-----------+-------------------------------------+------------+--------+---------------------+---------------------+
|         175 |        1 | ANNETTE    | OLSON     | ANNETTE.OLSON@sakilacustomer.org    |        179 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|         228 |        2 | ALLISON    | STANLEY   | ALLISON.STANLEY@sakilacustomer.org  |        232 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|         320 |        2 | ANTHONY    | SCHWAB    | ANTHONY.SCHWAB@sakilacustomer.org   |        325 |      1 | 2006-02-14 22:04:37 | 2006-02-15 04:57:20 |
|         398 |        1 | ANTONIO    | MEEK      | ANTONIO.MEEK@sakilacustomer.org     |        403 |      1 | 2006-02-14 22:04:37 | 2006-02-15 04:57:20 |
|         439 |        2 | ALEXANDER  | FENNELL   | ALEXANDER.FENNELL@sakilacustomer.org |        444 |      1 | 2006-02-14 22:04:37 | 2006-02-15 04:57:20 |
|         556 |        2 | ARMANDO    | GRUBER    | ARMANDO.GRUBER@sakilacustomer.org   |        562 |      1 | 2006-02-14 22:04:37 | 2006-02-15 04:57:20 |
|         567 |        2 | ALFREDO    | MCADAMS   | ALFREDO.MCADAMS@sakilacustomer.org  |        573 |      1 | 2006-02-14 22:04:37 | 2006-02-15 04:57:20 |
|         568 |        2 | ALBERTO    | HENNING   | ALBERTO.HENNING@sakilacustomer.org  |        574 |      1 | 2006-02-14 22:04:37 | 2006-02-15 04:57:20 |
+-------------+----------+------------+-----------+-------------------------------------+------------+--------+---------------------+---------------------+
8 rows in set (0.00 sec)
```

3.9. The SQL IN AND NOT IN OPERATORS

The **IN** operator allows users to specify multiple values in a WHERE clause. The IN operator is a shorthand for various **OR** conditions.

IN Syntax

SELECT column_name(s)

FROM table_name

WHERE column_name IN (value1, value2, ...);

OR:

SELECT column_name(s)

FROM table_name

WHERE column_name IN (SELECT STATEMENT);

Selects all the columns of customer whose customer_id in (1,2,3):

SELECT * FROM customer

WHERE cutomer_id IN (1,2,3);

```
mysql> SELECT * FROM customer
    -> where customer_id IN(1,2,3);
+-------------+----------+------------+-----------+-------------------------------------+------------+--------+---------------------+---------------------+
| customer_id | store_id | first_name | last_name | email                               | address_id | active | create_date         | last_update         |
+-------------+----------+------------+-----------+-------------------------------------+------------+--------+---------------------+---------------------+
|           1 |        1 | MARY       | SMITH     | MARY.SMITH@sakilacustomer.org       |          5 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|           2 |        1 | PATRICIA   | JOHNSON   | PATRICIA.JOHNSON@sakilacustomer.org |          6 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|           3 |        1 | LINDA      | WILLIAMS  | LINDA.WILLIAMS@sakilacustomer.org   |          7 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
+-------------+----------+------------+-----------+-------------------------------------+------------+--------+---------------------+---------------------+
3 rows in set (0.00 sec)
```

Selects all the columns of customer whose customer_id in (1,2,3):

```
SELECT * FROM customer

WHERE cutomer_id  NOT IN (1,2,3);
```

```
mysql> SELECT * FROM customer
    -> where customer_id NOT IN(1,2,3);
+-------------+----------+------------+-------------+----------------------------------------+------------+--------+---------------------+---------------------+
| customer_id | store_id | first_name | last_name   | email                                  | address_id | active | create_date         | last_update         |
+-------------+----------+------------+-------------+----------------------------------------+------------+--------+---------------------+---------------------+
|           4 |        2 | BARBARA    | JONES       | BARBARA.JONES@sakilacustomer.org       |          8 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|           5 |        1 | ELIZABETH  | BROWN       | ELIZABETH.BROWN@sakilacustomer.org     |          9 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|           6 |        2 | JENNIFER   | DAVIS       | JENNIFER.DAVIS@sakilacustomer.org      |         10 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|           7 |        1 | MARIA      | MILLER      | MARIA.MILLER@sakilacustomer.org        |         11 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|           8 |        2 | SUSAN      | WILSON      | SUSAN.WILSON@sakilacustomer.org        |         12 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|           9 |        2 | MARGARET   | MOORE       | MARGARET.MOORE@sakilacustomer.org      |         13 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          10 |        1 | DOROTHY    | TAYLOR      | DOROTHY.TAYLOR@sakilacustomer.org      |         14 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          11 |        2 | LISA       | ANDERSON    | LISA.ANDERSON@sakilacustomer.org       |         15 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          12 |        1 | NANCY      | THOMAS      | NANCY.THOMAS@sakilacustomer.org        |         16 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          13 |        2 | KAREN      | JACKSON     | KAREN.JACKSON@sakilacustomer.org       |         17 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          14 |        2 | BETTY      | WHITE       | BETTY.WHITE@sakilacustomer.org         |         18 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          15 |        1 | HELEN      | HARRIS      | HELEN.HARRIS@sakilacustomer.org        |         19 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          16 |        2 | SANDRA     | MARTIN      | SANDRA.MARTIN@sakilacustomer.org       |         20 |      0 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
```

3.10. The SQL BETWEEN OPERATOR

The **BETWEEN** operator retrieves values within the given range. The values can be texts, numbers, or dates. The **BETWEEN** operator is inclusive: begin and end values are included.

BETWEEN Syntax

```
SELECT column_name(s)

FROM table_name

WHERE column_name BETWEEN value1 AND value2;
```

Select all the columns from the customer with customer_id between 1 to 20.

```
SELECT * FROM customer WHERE customer_id BETWEEN 1 AND 20;
```

```
mysql> SELECT * FROM customer
    -> where customer_id between 1 and 20;
+-------------+----------+------------+-----------+----------------------------------+------------+--------+---------------------+---------------------+
| customer_id | store_id | first_name | last_name | email                            | address_id | active | create_date         | last_update         |
+-------------+----------+------------+-----------+----------------------------------+------------+--------+---------------------+---------------------+
|           1 |        1 | MARY       | SMITH     | MARY.SMITH@sakilacustomer.org    |          5 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|           2 |        1 | PATRICIA   | JOHNSON   | PATRICIA.JOHNSON@sakilacustomer.org |        6 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|           3 |        1 | LINDA      | WILLIAMS  | LINDA.WILLIAMS@sakilacustomer.org |          7 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|           4 |        2 | BARBARA    | JONES     | BARBARA.JONES@sakilacustomer.org |          8 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|           5 |        1 | ELIZABETH  | BROWN     | ELIZABETH.BROWN@sakilacustomer.org |        9 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|           6 |        2 | JENNIFER   | DAVIS     | JENNIFER.DAVIS@sakilacustomer.org |         10 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|           7 |        1 | MARIA      | MILLER    | MARIA.MILLER@sakilacustomer.org  |         11 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|           8 |        2 | SUSAN      | WILSON    | SUSAN.WILSON@sakilacustomer.org  |         12 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|           9 |        2 | MARGARET   | MOORE     | MARGARET.MOORE@sakilacustomer.org |         13 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          10 |        1 | DOROTHY    | TAYLOR    | DOROTHY.TAYLOR@sakilacustomer.org |         14 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          11 |        2 | LISA       | ANDERSON  | LISA.ANDERSON@sakilacustomer.org |         15 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          12 |        1 | NANCY      | THOMAS    | NANCY.THOMAS@sakilacustomer.org  |         16 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          13 |        2 | KAREN      | JACKSON   | KAREN.JACKSON@sakilacustomer.org |         17 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          14 |        2 | BETTY      | WHITE     | BETTY.WHITE@sakilacustomer.org   |         18 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          15 |        1 | HELEN      | HARRIS    | HELEN.HARRIS@sakilacustomer.org  |         19 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          16 |        2 | SANDRA     | MARTIN    | SANDRA.MARTIN@sakilacustomer.org |         20 |      0 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          17 |        1 | DONNA      | THOMPSON  | DONNA.THOMPSON@sakilacustomer.org |         21 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          18 |        2 | CAROL      | GARCIA    | CAROL.GARCIA@sakilacustomer.org  |         22 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          19 |        1 | RUTH       | MARTINEZ  | RUTH.MARTINEZ@sakilacustomer.org |         23 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|          20 |        2 | SHARON     | ROBINSON  | SHARON.ROBINSON@sakilacustomer.org |        24 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
+-------------+----------+------------+-----------+----------------------------------+------------+--------+---------------------+---------------------+
20 rows in set (0.00 sec)
```

Select all the columns from the customer with customer_id, not between 1 to 570.

SELECT * FROM customer WHERE customer_id NOT BETWEEN 1 AND 570;

```
mysql> SELECT * FROM customer
    -> where customer_id not between 1 and 570;
+-------------+----------+------------+-----------+----------------------------------+------------+--------+---------------------+---------------------+
| customer_id | store_id | first_name | last_name | email                            | address_id | active | create_date         | last_update         |
+-------------+----------+------------+-----------+----------------------------------+------------+--------+---------------------+---------------------+
|         571 |        2 | JOHNNIE    | CHISHOLM  | JOHNNIE.CHISHOLM@sakilacustomer.org |       577 |      1 | 2006-02-14 22:04:37 | 2006-02-15 04:57:20 |
|         572 |        1 | SIDNEY     | BURLESON  | SIDNEY.BURLESON@sakilacustomer.org |       578 |      1 | 2006-02-14 22:04:37 | 2006-02-15 04:57:20 |
|         573 |        1 | BYRON      | BOX       | BYRON.BOX@sakilacustomer.org     |       579 |      1 | 2006-02-14 22:04:37 | 2006-02-15 04:57:20 |
|         574 |        2 | JULIAN     | VEST      | JULIAN.VEST@sakilacustomer.org   |       580 |      1 | 2006-02-14 22:04:37 | 2006-02-15 04:57:20 |
|         575 |        2 | ISAAC      | OGLESBY   | ISAAC.OGLESBY@sakilacustomer.org |       581 |      1 | 2006-02-14 22:04:37 | 2006-02-15 04:57:20 |
|         576 |        2 | MORRIS     | MCCARTER  | MORRIS.MCCARTER@sakilacustomer.org |       582 |      1 | 2006-02-14 22:04:37 | 2006-02-15 04:57:20 |
|         577 |        2 | CLIFTON    | MALCOLM   | CLIFTON.MALCOLM@sakilacustomer.org |       583 |      1 | 2006-02-14 22:04:37 | 2006-02-15 04:57:20 |
|         578 |        2 | WILLARD    | LUMPKIN   | WILLARD.LUMPKIN@sakilacustomer.org |       584 |      1 | 2006-02-14 22:04:37 | 2006-02-15 04:57:20 |
|         579 |        2 | DARYL      | LARUE     | DARYL.LARUE@sakilacustomer.org   |       585 |      1 | 2006-02-14 22:04:37 | 2006-02-15 04:57:20 |
|         580 |        1 | ROSS       | GREY      | ROSS.GREY@sakilacustomer.org     |       586 |      1 | 2006-02-14 22:04:37 | 2006-02-15 04:57:20 |
|         581 |        1 | VIRGIL     | WOFFORD   | VIRGIL.WOFFORD@sakilacustomer.org |       587 |      1 | 2006-02-14 22:04:37 | 2006-02-15 04:57:20 |
|         582 |        2 | ANDY       | VANHORN   | ANDY.VANHORN@sakilacustomer.org  |       588 |      1 | 2006-02-14 22:04:37 | 2006-02-15 04:57:20 |
|         583 |        1 | MARSHALL   | THORN     | MARSHALL.THORN@sakilacustomer.org |       589 |      1 | 2006-02-14 22:04:37 | 2006-02-15 04:57:20 |
|         584 |        2 | SALVADOR   | TEEL      | SALVADOR.TEEL@sakilacustomer.org |       590 |      1 | 2006-02-14 22:04:37 | 2006-02-15 04:57:20 |
|         585 |        1 | PERRY      | SWAFFORD  | PERRY.SWAFFORD@sakilacustomer.org |       591 |      1 | 2006-02-14 22:04:37 | 2006-02-15 04:57:20 |
|         586 |        1 | KIRK       | STCLAIR   | KIRK.STCLAIR@sakilacustomer.org  |       592 |      1 | 2006-02-14 22:04:37 | 2006-02-15 04:57:20 |
|         587 |        1 | SERGIO     | STANFIELD | SERGIO.STANFIELD@sakilacustomer.org |       593 |      1 | 2006-02-14 22:04:37 | 2006-02-15 04:57:20 |
|         588 |        1 | MARION     | OCAMPO    | MARION.OCAMPO@sakilacustomer.org |       594 |      1 | 2006-02-14 22:04:37 | 2006-02-15 04:57:20 |
|         589 |        1 | TRACY      | HERRMANN  | TRACY.HERRMANN@sakilacustomer.org |       595 |      1 | 2006-02-14 22:04:37 | 2006-02-15 04:57:20 |
|         590 |        2 | SETH       | HANNON    | SETH.HANNON@sakilacustomer.org   |       596 |      1 | 2006-02-14 22:04:37 | 2006-02-15 04:57:20 |
|         591 |        1 | KENT       | ARSENAULT | KENT.ARSENAULT@sakilacustomer.org |       597 |      1 | 2006-02-14 22:04:37 | 2006-02-15 04:57:20 |
|         592 |        1 | TERRANCE   | ROUSH     | TERRANCE.ROUSH@sakilacustomer.org |       598 |      0 | 2006-02-14 22:04:37 | 2006-02-15 04:57:20 |
|         593 |        2 | RENE       | MCALISTER | RENE.MCALISTER@sakilacustomer.org |       599 |      1 | 2006-02-14 22:04:37 | 2006-02-15 04:57:20 |
|         594 |        1 | EDUARDO    | HIATT     | EDUARDO.HIATT@sakilacustomer.org |       600 |      1 | 2006-02-14 22:04:37 | 2006-02-15 04:57:20 |
|         595 |        1 | TERRENCE   | GUNDERSON | TERRENCE.GUNDERSON@sakilacustomer.org |       601 |      1 | 2006-02-14 22:04:37 | 2006-02-15 04:57:20 |
|         596 |        1 | ENRIQUE    | FORSYTHE  | ENRIQUE.FORSYTHE@sakilacustomer.org |       602 |      1 | 2006-02-14 22:04:37 | 2006-02-15 04:57:20 |
|         597 |        1 | FREDDIE    | DUGGAN    | FREDDIE.DUGGAN@sakilacustomer.org |       603 |      1 | 2006-02-14 22:04:37 | 2006-02-15 04:57:20 |
|         598 |        1 | WADE       | DELVALLE  | WADE.DELVALLE@sakilacustomer.org |       604 |      1 | 2006-02-14 22:04:37 | 2006-02-15 04:57:20 |
|         599 |        2 | AUSTIN     | CINTRON   | AUSTIN.CINTRON@sakilacustomer.org |       605 |      1 | 2006-02-14 22:04:37 | 2006-02-15 04:57:20 |
+-------------+----------+------------+-----------+----------------------------------+------------+--------+---------------------+---------------------+
29 rows in set (0.00 sec)
```

3.11. The SQL ALIAS

Aliases are used to give a nickname to a column in a table, a temporary name. Aliases are used to make column names more readable to the user.

Alias Column Syntax

SELECT first_name AS first, last_name AS last

FROM customer;

Creates two aliases, one for the first_name column and one for the last_name column:

```
mysql> select first_name as first, last_name as last from customer;
+-------------+--------------+
| first       | last         |
+-------------+--------------+
| MARY        | SMITH        |
| PATRICIA    | JOHNSON      |
| LINDA       | WILLIAMS     |
| BARBARA     | JONES        |
| ELIZABETH   | BROWN        |
| JENNIFER    | DAVIS        |
| MARIA       | MILLER       |
| SUSAN       | WILSON       |
| MARGARET    | MOORE        |
```

Alias Table Syntax

SELECT c.first_name, c.last_name

FROM customer AS c

Create an alias for the customer table

```
mysql> select c.first_name, c.last_name from customer as c;
+-------------+--------------+
| first_name  | last_name    |
+-------------+--------------+
| MARY        | SMITH        |
| PATRICIA    | JOHNSON      |
| LINDA       | WILLIAMS     |
| BARBARA     | JONES        |
| ELIZABETH   | BROWN        |
| JENNIFER    | DAVIS        |
| MARIA       | MILLER       |
| SUSAN       | WILSON       |
| MARGARET    | MOORE        |
| DOROTHY     | TAYLOR       |
```

3.12. The SQL GROUP BY STATEMENT

The **GROUP BY** used to group rows from the table. And it has the same values as summary rows. For example, find the number of customers in each country, The **GROUP BY** is often used with aggregate functions like (COUNT, MAX, MIN, SUM, AVG) to group the result-set by one or more columns.

GROUP BY Syntax

SELECT column_name(s)

FROM table_name

WHERE condition

GROUP BY column_name(s)

ORDER BY column_name(s);

Count the number of active and non-active customers

SELECT COUNT(customer_id) FROM customer GROUP BY active

```
mysql> select count(customer_id) from customer group by active;
+--------------------+
| count(customer_id) |
+--------------------+
|                584 |
|                 15 |
+--------------------+
2 rows in set (0.04 sec)
```

3.13. The SQL HAVING CLAUSE

The **HAVING** clause is added to SQL because the WHERE keyword can not be used with aggregate functions.

HAVING Syntax

SELECT column_name(s)

FROM table_name

WHERE condition

GROUP BY column_name(s)

HAVING condition

ORDER BY column_name(s);

List the number of continents which has a region more than 6.

SELECT * from country group by(continent) having count(region) >6;

```
mysql> select * from country group by (continent) having count(region) >6;
+------+----------------+---------------+---------------------------+-------------+-----------+------------+-----------------+-----------+-----------+-------------------
----+-----------------------------------------------+---------------------------+---------+---------+-------+
| Code | Name           | Continent     | Region                    | SurfaceArea | IndepYear | Population | LifeExpectancy | GNP       | GNPOld    | LocalName
    | GovernmentForm                                | HeadOfState               | Capital | Code2 |
+------+----------------+---------------+---------------------------+-------------+-----------+------------+-----------------+-----------+-----------+-------------------
----+-----------------------------------------------+---------------------------+---------+---------+-------+
| ABW  | Aruba          | North America | Caribbean                 |      193.00 |      NULL |     103000 |            78.4 |    828.00 |    793.00 | Aruba
    | Nonmetropolitan Territory of The Netherlands | Beatrix                   |     129 | AW    |
| AFG  | Afghanistan    | Asia          | Southern and Central Asia |   652090.00 |      1919 |   22720000 |            45.9 |   5976.00 |      NULL | Afganistan/Afqanes
tan | Islamic Emirate                               | Mohammad Omar             |       1 | AF    |
| AGO  | Angola         | Africa        | Central Africa            |  1246700.00 |      1975 |   12878000 |            38.3 |   6648.00 |   7984.00 | Angola
    | Republic                                      | JosÃ© Eduardo dos Santos |      56 | AO    |
| ALB  | Albania        | Europe        | Southern Europe           |    28748.00 |      1912 |    3401200 |            71.6 |   3205.00 |   2500.00 | ShqipÃ«ria
    | Republic                                      | Rexhep Mejdani            |      34 | AL    |
| ARG  | Argentina      | South America | South America             |  2780400.00 |      1816 |   37032000 |            75.1 | 340238.00 | 323310.00 | Argentina
    | Federal Republic                              | Fernando de la RÃªa      |      69 | AR    |
| ASM  | American Samoa | Oceania       | Polynesia                 |      199.00 |      NULL |      68000 |            75.1 |    334.00 |      NULL | Amerika Samoa
    | US Territory                                  | George W. Bush            |      54 | AS    |
+------+----------------+---------------+---------------------------+-------------+-----------+------------+-----------------+-----------+-----------+-------------------
----+-----------------------------------------------+---------------------------+---------+---------+-------+
6 rows in set (0.03 sec)
```

3.14. The SQL UNION

The UNION operator allows the user to combine the result-set of two or more SELECT statements in SQL. Each SELECT statement within UNION should have the same number of columns. The columns in each SELECT statement should also be in the same order. The columns should also have similar data types.

The SQL UNION

```
Select  column_name(s) from table1
UNION
Select column_name(s) from table2;
```

UNION ALL Query

The UNION operator selects only different values by default. To allow duplicate values, the user can use UNION ALL operator.

```
SELECT column_name(s) FROM table1
UNION ALL
SELECT column_name(s) FROM table2;
```

Note: The column names in the output are usually equal to the column names in the first SELECT statement in the UNION.

3.15. The SQL STORED PROCEDURE

What is a SQL Stored Procedure?

The **stored procedure** is a prepared SQL query that you can save so that the query can be **reused** over and over again. So, if the user has an SQL query that you write over and over again, keep it as a stored procedure and execute it. Users can also pass parameters to a stored procedure so that the stored procedure can act based on the parameter value that is given.

Stored Procedure Syntax

```
CREATE PROCEDURE procedure_name
AS
sql_statement
GO;
```

Execute a Stored Procedure

```
EXEC procedure_name;
```

4. SQL JOIN

The SQL Join help in retrieving data from two or more database tables. The tables are mutually related using primary keys and foreign keys.

Type of Join

4.1. INNER JOIN

The **INNER JOIN** is used to print rows from both tables that satisfy the given condition. For example, the user wants to get a list of users who have rented movies together with titles of movies rented by them. Users can use an INNER JOIN for that, which returns rows from both tables that satisfy with given conditions.

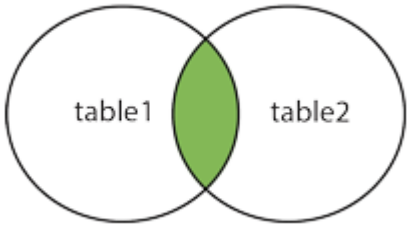


Fig. INNER JOIN

The INNER JOIN keyword selects records that have matching values in both the tables.

INNER JOIN Syntax

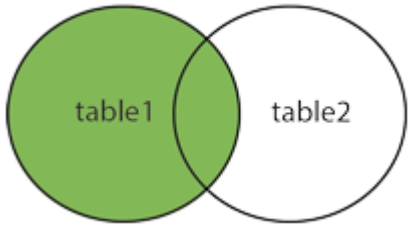
```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

```
SELECT city.city_id, country.country, city.last_update, country.last_update FROM city
INNER JOIN country ON city.country_id = country.country_id
```

```
mysql> SELECT city.city_id, country.country, city.last_update,country.last_update
    -> FROM city
    -> INNER JOIN country ON city.country_id=country.country_id;
+---------+-------------------------------------+---------------------+---------------------+
| city_id | country                             | last_update         | last_update         |
+---------+-------------------------------------+---------------------+---------------------+
|     251 | Afghanistan                         | 2006-02-15 04:45:25 | 2006-02-15 04:44:00 |
|      59 | Algeria                             | 2006-02-15 04:45:25 | 2006-02-15 04:44:00 |
|      63 | Algeria                             | 2006-02-15 04:45:25 | 2006-02-15 04:44:00 |
|     483 | Algeria                             | 2006-02-15 04:45:25 | 2006-02-15 04:44:00 |
|     516 | American Samoa                      | 2006-02-15 04:45:25 | 2006-02-15 04:44:00 |
|      67 | Angola                              | 2006-02-15 04:45:25 | 2006-02-15 04:44:00 |
|     360 | Angola                              | 2006-02-15 04:45:25 | 2006-02-15 04:44:00 |
|     493 | Anguilla                            | 2006-02-15 04:45:25 | 2006-02-15 04:44:00 |
|      20 | Argentina                           | 2006-02-15 04:45:25 | 2006-02-15 04:44:00 |
|      43 | Argentina                           | 2006-02-15 04:45:25 | 2006-02-15 04:44:00 |
|      45 | Argentina                           | 2006-02-15 04:45:25 | 2006-02-15 04:44:00 |
|     128 | Argentina                           | 2006-02-15 04:45:25 | 2006-02-15 04:44:00 |
```

4.2. LEFT JOIN

The **LEFT JOIN** returns all the records from the table1 (left table) and the matched records from the table2 (right table). The output is NULL from the right side if there is no match.



Left Join

LEFT JOIN Syntax

SELECT column_name(s)

FROM table1

LEFT JOIN table2

ON table1.column_name = table2.column_name;

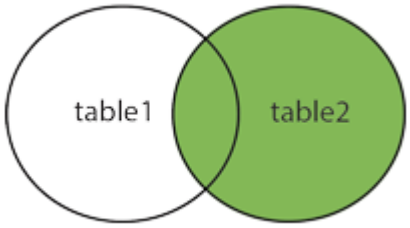
SELECT city.city_id, country.country, city.last_update, country.last_update FROM city

LEFT JOIN country ON city.country_id = country.country_id

```
mysql> SELECT city.city_id, country.country, city.last_update
    -> FROM country
    -> LEFT JOIN city ON city.country_id=country.country_id;
+---------+-------------------------------------+---------------------+
| city_id | country                             | last_update         |
+---------+-------------------------------------+---------------------+
|     251 | Afghanistan                         | 2006-02-15 04:45:25 |
|      59 | Algeria                             | 2006-02-15 04:45:25 |
|      63 | Algeria                             | 2006-02-15 04:45:25 |
|     483 | Algeria                             | 2006-02-15 04:45:25 |
|     516 | American Samoa                      | 2006-02-15 04:45:25 |
|      67 | Angola                              | 2006-02-15 04:45:25 |
|     360 | Angola                              | 2006-02-15 04:45:25 |
|     493 | Anguilla                            | 2006-02-15 04:45:25 |
```

4.3. RIGHT JOIN

The RIGHT JOIN is the opposite of LEFT JOIN. The RIGHT JOIN prints all the columns from the table2(right table) even if there no matching rows have been found in the table1 (left table). If there no matches have been found in the table (left table), NULL is returned.



RIGHT JOIN

RIGHT JOIN Syntax

SELECT column_name(s)

FROM table1

RIGHT JOIN table2 ON table1.column_name = table2.column_name;

SELECT city.city_id, country.country, city.last_update, country.last_update FROM city

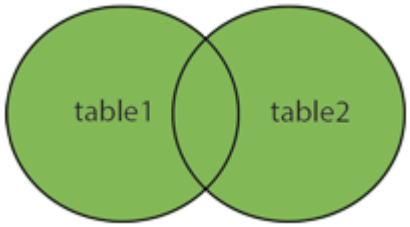
RIGHT JOIN country ON city.country_id = country.country_id

```
mysql> SELECT city.city_id, country.country, city.last_update
    -> FROM city
    -> RIGHT JOIN country ON city.country_id=country.country_id;
+---------+-------------------------------------+---------------------+
| city_id | country                             | last_update         |
+---------+-------------------------------------+---------------------+
|     251 | Afghanistan                         | 2006-02-15 04:45:25 |
|      59 | Algeria                             | 2006-02-15 04:45:25 |
|      63 | Algeria                             | 2006-02-15 04:45:25 |
|     483 | Algeria                             | 2006-02-15 04:45:25 |
|     516 | American Samoa                      | 2006-02-15 04:45:25 |
```

4.4. Full OUTER JOIN

The FULL OUTER JOIN keyword returns all records when there are a match in left (table1) or right (table2) table records.

Note: FULL OUTER JOIN can potentially return very large result-sets!



Full Join

Tip: FULL OUTER JOIN and FULL JOIN are the same.

FULL OUTER JOIN Syntax

SELECT column_name(s)

FROM table1

FULL OUTER JOIN table2

ON table1.column_name = table2.column_name WHERE condition;

Note: MySQL does not support the Full Join, so we can perform left join and right join separately then take the union of them.

```
SELECT * FROM t1
LEFT JOIN t2 ON t1.id = t2.id
UNION
SELECT * FROM t1
RIGHT JOIN t2 ON t1.id = t2.id
```

4.5. SELF-JOIN

A self-JOIN is a regular join, but the table is joined with itself.

Self -JOIN Syntax

```
SELECT column_name(s)
FROM table1 T1, table1 T2
WHERE condition;
```

5. SQL DATABASE

5.1. The SQL CREATE DATABASE STATEMENT

The CREATE DATABASE statement in SQL is used to create a new SQL database.

Syntax

CREATE DATABASE database_name;

Let's create a database and give name as testdb

CREATE database testdb;

```
mysql> create database testdb;
Query OK, 1 row affected (0.28 sec)
```

Now, let's check the databases in MySQL by using **show databases** query.

Show databases;

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sakila             |
| sys                |
| testdb             |
| world              |
+--------------------+
7 rows in set (0.06 sec)
```

5.2. The SQL DROP DATABASE STATEMENT

The DROP DATABASE statement in SQL is used to drop an existing SQL database.

Syntax

DROP DATABASE database_name;

Let's drop the created database by using drop database testdb.

DROP database testdb;

```
mysql> drop database testdb;
Query OK, 0 rows affected (0.78 sec)
```

Now, let's check the databases in MySQL by using **show databases** query after dropping the testdb.

SHOW databases;

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sakila             |
| sys                |
| world              |
+--------------------+
6 rows in set (0.00 sec)
```

The created database(testdb) has been dropped.

5.3. The SQL CREATE TABLE

The CREATE TABLE statement in SQL is used to create a new table in a database.

Syntax

CREATE TABLE table_name (

column1 data_type,

column2 data_type,

column3 data_type,

....

);

The column1, column2, ….., specify the names of the columns of the table. The datatype parameter specifies the type of data the column can hold (e.g., varchar, integer, date, etc.)

Let's create a customer table

CREATE TABLE cutomer(id integer, first_name varchar(10), last_name varchar(10), city varchar(10), country varchar(15), phone varchar(15));

```
mysql> create table customer (id integer, first_name varchar(10),last_name varchar(10),city varchar(10),country varchar(15),phone varchar(15));
Query OK, 0 rows affected (1.85 sec)
```

To check the schema of the table, use desc table_name.

DESC customer;

```
mysql> desc customer;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| id         | int(11)     | YES  |     | NULL    |       |
| first_name | varchar(10) | YES  |     | NULL    |       |
| last_name  | varchar(10) | YES  |     | NULL    |       |
| city       | varchar(10) | YES  |     | NULL    |       |
| country    | varchar(15) | YES  |     | NULL    |       |
| phone      | varchar(15) | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
6 rows in set (0.04 sec)
```

5.4. The SQL DROP TABLE STATEMENT

The DROP TABLE statement in SQL is used to drop an existing table in a database.

DROP TABLE customer;

```
mysql> drop table customer;
Query OK, 0 rows affected (1.24 sec)

mysql> desc customer;
ERROR 1146 (42S02): Table 'testdb.customer' doesn't exist
```

The table has dropped after running the query drop table table_name. As we can see, the table does not exist after dropped.

Now we are going to create the same table again to insert the values in that table.

5.5. The SQL INSERT INTO STATEMENT

The INSERT INTO statement in SQL is used to insert new records in a table.

INSERT INTO query

We can write the INSERT INTO statement in two ways. The first way is to specify both the column names and the values to be inserted:

INSERT INTO customer(id , first_name, last_name ,city ,country,phone)VALUES (2, 'Ana', 'Trujillo', 'Mexico', 'Mexico', (5) 555-4729);

```
mysql> INSERT INTO customer (id,first_name,last_name,city,country,phone) VALUES(2,'Ana','Trujillo','México','Mexico','(5) 555-4729');
Query OK, 1 row affected (0.12 sec)

mysql> select * from customer;
+------+------------+-----------+--------+---------+--------------+
| id   | first_name | last_name | city   | country | phone        |
+------+------------+-----------+--------+---------+--------------+
|    2 | Ana        | Trujillo  | México | Mexico  | (5) 555-4729 |
+------+------------+-----------+--------+---------+--------------+
1 row in set (0.00 sec)
```

If users are adding values for all the columns of the table, you don't need to specify the particular column names in the SQL query. However, ensure the order of the values is in the same order as the columns in the table.

The INSERT INTO query would be as follows:

INSERT INTO customer

VALUES (3, 'Antonio, 'Moreno, 'Mexico', 'Mexico', (5) 555-3932);

```
mysql> select * from customer;
+------+------------+-----------+--------+---------+--------------+
| id   | first_name | last_name | city   | country | phone        |
+------+------------+-----------+--------+---------+--------------+
|    2 | Ana        | Trujillo  | México | Mexico  | (5) 555-4729 |
|    3 | Antonio    | Moreno    | México | Mexico  | (5) 555-3932 |
+------+------------+-----------+--------+---------+--------------+
2 rows in set (0.00 sec)
```

We have inserted two rows yet. Similarly, we can insert many rows in the table. Finally, we have added ten rows as we can see in the picture below.

SELECT * FROM customer;

```
mysql> select * from customer;
+------+------------+-----------+------------+---------+----------------+
| id   | first_name | last_name | city       | country | phone          |
+------+------------+-----------+------------+---------+----------------+
|    2 | Ana        | Trujillo  | México     | Mexico  | (5) 555-4729   |
|    3 | Antonio    | Moreno    | México     | Mexico  | (5) 555-3932   |
|    4 | Thomas     | Hardy     | London     | UK      | (171) 555-7788 |
|    5 | Christina  | Berglund  | Luleå      | Sweden  | 0921-12 34 65  |
|    6 | Hanna      | Moos      | Mannheim   | Germany | 0621-08460     |
|    7 | Frédérique | Citeaux   | Strasbourg | France  | 88.60.15.31    |
|    8 | Martín     | Sommer    | Madrid     | Spain   | (91) 555 22 82 |
|    9 | Laurence   | Lebihan   | Marseille  | France  | 91.24.45.40    |
|   10 | Elizabeth  | Lincoln   | Tsawassen  | Canada  | (604) 555-4729 |
|   11 | Victoria   | Ashworth  | London     | UK      | (171) 555-1212 |
+------+------------+-----------+------------+---------+----------------+
10 rows in set (0.00 sec)
```

5.6. The SQL NULL VALUES

What is a NULL Value?

The field with a NULL value is a field with no value. If the field in a table is optional, to insert new data or update data without adding a value to this field and Then, the field will be saved as a NULL value.

Note: A NULL value is not the same as a zero value, or we can say a field that holds spaces. The field with a NULL value is one that has been left blank during record creation!

Insert the NULL values in tables

INSERT INTO customer VALUES(11, ‘Victoria’, ‘Ashworth’, ‘London’, NULL, ‘(171) 555-1212’)

```
mysql> INSERT INTO customer VALUES(11,'Victoria','Ashworth','London',NULL,'(171) 555-1212');
Query OK, 1 row affected (0.16 sec)

mysql> select * from customer;
+------+------------+-----------+------------+---------+----------------+
| id   | first_name | last_name | city       | country | phone          |
+------+------------+-----------+------------+---------+----------------+
|    2 | Ana        | Trujillo  | México     | Mexico  | (5) 555-4729   |
|    3 | Antonio    | Moreno    | México     | Mexico  | (5) 555-3932   |
|    4 | Thomas     | Hardy     | London     | UK      | (171) 555-7788 |
|    5 | Christina  | Berglund  | Luleå      | Sweden  | 0921-12 34 65  |
|    6 | Hanna      | Moos      | Mannheim   | Germany | 0621-08460     |
|    7 | Frédérique | Citeaux   | Strasbourg | France  | 88.60.15.31    |
|    8 | Martín     | Sommer    | Madrid     | Spain   | (91) 555 22 82 |
|    9 | Laurence   | Lebihan   | Marseille  | France  | 91.24.45.40    |
|   10 | Elizabeth  | Lincoln   | Tsawassen  | Canada  | (604) 555-4729 |
|   11 | Victoria   | Ashworth  | London     | NULL    | (171) 555-1212 |
+------+------------+-----------+------------+---------+----------------+
10 rows in set (0.00 sec)
```

As we can able to see, the last row contains one NULL value.

How to check for NULL Values?

To test for NULL values in the table has to use the **IS NULL** and **IS NOT NULL** operators instead.

IS NULL Syntax

```
SELECT *

FROM customer  WHERE country IS NULL;
```

```
mysql> select * from customer where country IS NULL;
+------+------------+-----------+--------+---------+----------------+
| id   | first_name | last_name | city   | country | phone          |
+------+------------+-----------+--------+---------+----------------+
|   11 | Victoria   | Ashworth  | London | NULL    | (171) 555-1212 |
+------+------------+-----------+--------+---------+----------------+
1 row in set (0.00 sec)
```

IS NOT NULL Syntax

```
SELECT *  FROM customer

WHERE country IS NOT NULL;
```

```
mysql> select * from customer where country IS NOT NULL;
+------+------------+-----------+------------+---------+----------------+
| id   | first_name | last_name | city       | country | phone          |
+------+------------+-----------+------------+---------+----------------+
|    2 | Ana        | Trujillo  | México     | Mexico  | (5) 555-4729   |
|    3 | Antonio    | Moreno    | México     | Mexico  | (5) 555-3932   |
|    4 | Thomas     | Hardy     | London     | UK      | (171) 555-7788 |
|    5 | Christina  | Berglund  | Luleå      | Sweden  | 0921-12 34 65  |
|    6 | Hanna      | Moos      | Mannheim   | Germany | 0621-08460     |
|    7 | Frédérique | Citeaux   | Strasbourg | France  | 88.60.15.31    |
|    8 | Martín     | Sommer    | Madrid     | Spain   | (91) 555 22 82 |
|    9 | Laurence   | Lebihan   | Marseille  | France  | 91.24.45.40    |
|   10 | Elizabeth  | Lincoln   | Tsawassen  | Canada  | (604) 555-4729 |
+------+------------+-----------+------------+---------+----------------+
9 rows in set (0.00 sec)
```

It will return those countries which have some values(expect Null values).

5.7. The SQL UPDATE STATEMENT

The UPDATE statement in SQL is used to modify the existing records in a table.

UPDATE Syntax

UPDATE customer

SET country = ‘Mexico’ WHERE id = 11;

```
mysql> update customer set country = 'maxico' where id = 11;
Query OK, 1 row affected (0.12 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from customer;
+------+------------+-----------+------------+---------+----------------+
| id   | first_name | last_name | city       | country | phone          |
+------+------------+-----------+------------+---------+----------------+
|    2 | Ana        | Trujillo  | México     | Mexico  | (5) 555-4729   |
|    3 | Antonio    | Moreno    | México     | Mexico  | (5) 555-3932   |
|    4 | Thomas     | Hardy     | London     | UK      | (171) 555-7788 |
|    5 | Christina  | Berglund  | Luleå      | Sweden  | 0921-12 34 65  |
|    6 | Hanna      | Moos      | Mannheim   | Germany | 0621-08460     |
|    7 | Frédérique | Citeaux   | Strasbourg | France  | 88.60.15.31    |
|    8 | Martín     | Sommer    | Madrid     | Spain   | (91) 555 22 82 |
|    9 | Laurence   | Lebihan   | Marseille  | France  | 91.24.45.40    |
|   10 | Elizabeth  | Lincoln   | Tsawassen  | Canada  | (604) 555-4729 |
|   11 | Victoria   | Ashworth  | London     | maxico  | (171) 555-1212 |
+------+------------+-----------+------------+---------+----------------+
10 rows in set (0.00 sec)
```

We have updated the null value of the country with Mexico.

5.8. The SQL DELETE STATEMENT

The DELETE statement in SQL is used to delete existing records in a table.

DELETE Syntax

DELETE FROM customer WHERE id = 11;

```
mysql> delete from customer where id = 11;
Query OK, 1 row affected (0.15 sec)

mysql> select * from customer;
+------+------------+-----------+------------+---------+----------------+
| id   | first_name | last_name | city       | country | phone          |
+------+------------+-----------+------------+---------+----------------+
|    2 | Ana        | Trujillo  | México     | Mexico  | (5) 555-4729   |
|    3 | Antonio    | Moreno    | México     | Mexico  | (5) 555-3932   |
|    4 | Thomas     | Hardy     | London     | UK      | (171) 555-7788 |
|    5 | Christina  | Berglund  | Luleå      | Sweden  | 0921-12 34 65  |
|    6 | Hanna      | Moos      | Mannheim   | Germany | 0621-08460     |
|    7 | Frédérique | Citeaux   | Strasbourg | France  | 88.60.15.31    |
|    8 | Martín     | Sommer    | Madrid     | Spain   | (91) 555 22 82 |
|    9 | Laurence   | Lebihan   | Marseille  | France  | 91.24.45.40    |
|   10 | Elizabeth  | Lincoln   | Tsawassen  | Canada  | (604) 555-4729 |
+------+------------+-----------+------------+---------+----------------+
9 rows in set (0.00 sec)
```

We have deleted one row, which contains id = 11.

5.9. The SQL ALTER TABLE STATEMENT

The ALTER TABLE statement in SQL is used to add, modify, or delete columns in an existing table. And it also used to add and drop various constraints on a current table.

5.9.1. ALTER TABLE - ADD COLUMN IN EXISTING TABLE

To add a new column in a table, use the SQL query

ALTER TABLE customer

ADD email varchar(25);

```
mysql> alter table customer add email varchar(25);
Query OK, 0 rows affected (2.12 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> select * from customer;
+------+------------+-----------+------------+---------+----------------+-------+
| id   | first_name | last_name | city       | country | phone          | email |
+------+------------+-----------+------------+---------+----------------+-------+
|    2 | Ana        | Trujillo  | México     | Mexico  | (5) 555-4729   | NULL  |
|    3 | Antonio    | Moreno    | México     | Mexico  | (5) 555-3932   | NULL  |
|    4 | Thomas     | Hardy     | London     | UK      | (171) 555-7788 | NULL  |
|    5 | Christina  | Berglund  | Luleå      | Sweden  | 0921-12 34 65  | NULL  |
|    6 | Hanna      | Moos      | Mannheim   | Germany | 0621-08460     | NULL  |
|    7 | Frédérique | Citeaux   | Strasbourg | France  | 88.60.15.31    | NULL  |
|    8 | Martín     | Sommer    | Madrid     | Spain   | (91) 555 22 82 | NULL  |
|    9 | Laurence   | Lebihan   | Marseille  | France  | 91.24.45.40    | NULL  |
|   10 | Elizabeth  | Lincoln   | Tsawassen  | Canada  | (604) 555-4729 | NULL  |
+------+------------+-----------+------------+---------+----------------+-------+
9 rows in set (0.00 sec)
```

5.9.2. ALTER TABLE – MODIFY/ALTER COLUMN

To change the data type of column values in a table, use the following syntax:

ALTER TABLE customer ADD COLUMN dob date;

```
mysql> alter table customer add dob date;
Query OK, 0 rows affected (1.83 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

We have assigned the dob with the datatype date. But now we want to change the datatype from date to year.

ALTER TABLE customer MODIFY dob year;

```
mysql> alter table customer modify dob year;
Query OK, 9 rows affected (3.68 sec)
Records: 9  Duplicates: 0  Warnings: 0
```

5.9.3. ALTER TABLE - DROP COLUMN

To delete a specific column in a table, use the following syntax (notice that some database systems don't allow deleting a column):

Syntax:

ALTER TABLE customer

DROP COLUMN email;

```
mysql> alter table customer drop column email;
Query OK, 0 rows affected (2.40 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> select * from customer;
+------+------------+-----------+------------+---------+----------------+
| id   | first_name | last_name | city       | country | phone          |
+------+------------+-----------+------------+---------+----------------+
|    2 | Ana        | Trujillo  | México     | Mexico  | (5) 555-4729   |
|    3 | Antonio    | Moreno    | México     | Mexico  | (5) 555-3932   |
|    4 | Thomas     | Hardy     | London     | UK      | (171) 555-7788 |
|    5 | Christina  | Berglund  | Luleå      | Sweden  | 0921-12 34 65  |
|    6 | Hanna      | Moos      | Mannheim   | Germany | 0621-08460     |
|    7 | Frédérique | Citeaux   | Strasbourg | France  | 88.60.15.31    |
|    8 | Martín     | Sommer    | Madrid     | Spain   | (91) 555 22 82 |
|    9 | Laurence   | Lebihan   | Marseille  | France  | 91.24.45.40    |
|   10 | Elizabeth  | Lincoln   | Tsawassen  | Canada  | (604) 555-4729 |
+------+------------+-----------+------------+---------+----------------+
9 rows in set (0.00 sec)
```

6. The SQL CONSTRAINTS

The Constraints in SQL can be specified when the table is created with the CREATE TABLE statement, or after the table is altered with the ALTER TABLE statement.

Syntax:

```
CREATE TABLE table_name (
 column1 datatype constraint,
 column2 datatype constraint,
column3 datatype constraint,
 ....
);
```

SQL Constraints

SQL constraints are used to specify any rules for the records in a table. Constraints can be used to limit the type of data that can go into a table. It ensures the accuracy and reliability of the records in the table, and if there is any violation between the constraint and the record action, the action is aborted. Constraints can be column level or table level. Column level constraints apply to a column, and table-level constraints apply to the whole table.

The constraints are commonly used in SQL

CONSTRAINTS	DESCRIPTION
Not Null	It Ensures that a column cannot have a NULL value.
Unique	It Ensures that all the values in a column are unique.
Primary Key	It is a combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table.
Foreign Key	Uniquely identifies a record /row in another table
Check	It checks that all values in a column satisfy a specific condition
Default	It gives a default value for a column when no value is specified
Index	It is Used to create and retrieve data from the database quickly.

6.1. NOT NULL CONSTRAINTS

The NOT NULL constraint enforces a column NOT to accept NULL values. This imposes a field always to contain a value, which means that the user cannot insert a new record in a table or update a record without adding a value to this field.

NOTE: By default, a column can hold NULL values.

Create a table using SQL not null constraints

The following SQL ensures that the "id", "First_name" and "Last_name" columns will NOT accept NULL values when the "student" table is created:

Example

```
CREATE TABLE student(
id int NOT NULL,
first_name varchar(25) NOT NULL,
last_name varchar(25) NOT NULL,
age int
);
```

```
mysql> create table student (id int not null, first_name varchar(25) not null, last_name varchar(25) not null, age int);
Query OK, 0 rows affected (1.16 sec)

mysql> desc student;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| id         | int(11)     | NO   |     | NULL    |       |
| first_name | varchar(25) | NO   |     | NULL    |       |
| last_name  | varchar(25) | NO   |     | NULL    |       |
| age        | int(11)     | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
4 rows in set (0.24 sec)
```

In the above table, it has specified the id, first_name, and last_name as not null and age as null.

SQL NOT NULL on ALTER table Statement

To make a NOT NULL constraint on the "age" column when the "student" table is already created, use the following SQL:

Example:

```
ALTER TABLE student

MODIFY age int NOT NULL;
```

```
mysql> alter table student modify age int not null;
Query OK, 0 rows affected (1.93 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc customer;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| id         | int(11)     | YES  |     | NULL    |       |
| first_name | varchar(10) | YES  |     | NULL    |       |
| last_name  | varchar(10) | YES  |     | NULL    |       |
| city       | varchar(10) | YES  |     | NULL    |       |
| country    | varchar(15) | YES  |     | NULL    |       |
| phone      | varchar(15) | YES  |     | NULL    |       |
| dob        | year(4)     | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
7 rows in set (0.00 sec)
```

In the above table, it has specified the id, first_name,last_name, and age as not null.

6.2. SQL UNIQUE CONSTRAINT

The **UNIQUE** constraint in SQL ensures that all values in a column are distinct. UNIQUE and PRIMARY KEY constraints both provides a guarantee for **uniqueness** for a column or group of columns. A PRIMARY KEY constraint, by default, has a UNIQUE constraint. However, the user can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

Creates UNIQUE constraint on the "id" column when the "person" table is created

```
CREATE TABLE person (

id int NOT NULL,
```

last_name varchar(255) NOT NULL,

first_name varchar(255),

age int,

UNIQUE (ID)

);

```
mysql> create table person(id int not null, first_name varchar(25) not null, last_name varchar(25) not null,age int, unique(id));
Query OK, 0 rows affected (1.29 sec)

mysql> desc person;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| id         | int(11)     | NO   | PRI | NULL    |       |
| first_name | varchar(25) | NO   |     | NULL    |       |
| last_name  | varchar(25) | NO   |     | NULL    |       |
| age        | int(11)     | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
4 rows in set (0.03 sec)
```

We have applied unique constraints on id, and as we can see, it is showing as the primary key.

Create a UNIQUE constraint on the "first_name" column when the "persons" table already exists.

ALTER TABLE persons

ADD UNIQUE (first_name);

```
mysql> alter table person add unique(first_name);
Query OK, 0 rows affected (0.76 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc person;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| id         | int(11)     | NO   | PRI | NULL    |       |
| first_name | varchar(25) | NO   | UNI | NULL    |       |
| last_name  | varchar(25) | NO   |     | NULL    |       |
| age        | int(11)     | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

Now we have two unique constraints(id and first_name) in the person table.

To name the UNIQUE constraint, and to define a UNIQUE constraint on multiple columns, use the following SQL syntax:

ALTER TABLE person

ADD CONSTRAINT UC_person UNIQUE (age, last_name);

```
mysql> ALTER TABLE person
    -> ADD CONSTRAINT UC_Persons UNIQUE (age,last_name);
Query OK, 0 rows affected (1.65 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc person;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| id         | int(11)     | NO   | PRI | NULL    |       |
| first_name | varchar(25) | NO   | UNI | NULL    |       |
| last_name  | varchar(25) | NO   |     | NULL    |       |
| age        | int(11)     | YES  | MUL | NULL    |       |
+------------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

Here the age and last_name are converted as unique constraints.

6.3. DROP A UNIQUE CONSTRAINT

To drop a UNIQUE constraint, use the SQL query

ALTER TABLE person

DROP INDEX UC_Person;

```
mysql> ALTER TABLE person
    -> drop index UC_Persons;
Query OK, 0 rows affected (0.38 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc person;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| id         | int(11)     | NO   | PRI | NULL    |       |
| first_name | varchar(25) | NO   | UNI | NULL    |       |
| last_name  | varchar(25) | NO   |     | NULL    |       |
| age        | int(11)     | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
4 rows in set (0.07 sec)
```

As we can see in the person table The unique constraint(UC_Persons) has been dropped.

6.4. SQL PRIMARY KEY CONSTRAINTS

The PRIMARY KEY constraint uniquely identifies each of the records in a table. Only ONE primary key can have in a table. And also, in the table, this primary key can consist of single or multiple columns (fields). Primary keys should contain UNIQUE values, and cannot contain **NULL** values.

CREATE TABLE person(ID int NOT NULL, last_name varchar(255) NOT NULL, first_name varchar(255), age int, PRIMARY KEY(ID));

```
mysql> CREATE TABLE person(ID int NOT NULL,
    ->     last_name varchar(255) NOT NULL,
    ->     first_name varchar(255),
    ->     age int,
    ->     PRIMARY KEY (ID)
    -> );
Query OK, 0 rows affected (0.61 sec)

mysql> desc person;
+------------+--------------+------+-----+---------+-------+
| Field      | Type         | Null | Key | Default | Extra |
+------------+--------------+------+-----+---------+-------+
| ID         | int(11)      | NO   | PRI | NULL    |       |
| last_name  | varchar(255) | NO   |     | NULL    |       |
| first_name | varchar(255) | YES  |     | NULL    |       |
| age        | int(11)      | YES  |     | NULL    |       |
+------------+--------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

To allow the naming of a PRIMARY KEY constraint, and for defining a PRIMARY KEY constraint on multiple columns, use the SQL syntax.

```
CREATE TABLE person (
  id int NOT NULL,
 last_name varchar(255) NOT NULL,
 first_name varchar(255),
age int,
 CONSTRAINT PK_person PRIMARY KEY (id,last_name)
);
```

```
mysql> CREATE TABLE Person1 (
    ->     id int NOT NULL,
    ->     last_name varchar(25) NOT NULL,
    ->     first_name varchar(25),
    ->     age int,
    ->     CONSTRAINT PK_Person PRIMARY KEY (id,last_name)
    -> );
Query OK, 0 rows affected (0.94 sec)

mysql> desc Person1
    -> ;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| id         | int(11)     | NO   | PRI | NULL    |       |
| last_name  | varchar(25) | NO   | PRI | NULL    |       |
| first_name | varchar(25) | YES  |     | NULL    |       |
| age        | int(11)     | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

Note: In this example, there is only ONE PRIMARY KEY as PK_Person. And the VALUE of the primary key is made up of **two columns** (id+ last_name).

SQL PRIMARY KEY on ALTER TABLE

Create a PRIMARY KEY constraint on the column_name "**id**" when the table_name(student) is already created, use the following SQL:

ALTER TABLE student

ADD PRIMARY KEY (id);

```
mysql> alter table student add primary key(id);
Query OK, 0 rows affected (1.71 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc student;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| id         | int(11)     | NO   | PRI | NULL    |       |
| first_name | varchar(25) | NO   |     | NULL    |       |
| last_name  | varchar(25) | NO   |     | NULL    |       |
| age        | int(11)     | NO   |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

Here we have assigned the primary key as “id” on the student table.

Allow the naming of a PRIMARY KEY constraint, and for defining a PRIMARY KEY constraint on multiple columns, use the SQL query:

ALTER TABLE student

ADD CONSTRAINT PK_student PRIMARY KEY (id,first_name);

```
mysql> desc student;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| id         | int(11)     | NO   |     | NULL    |       |
| first_name | varchar(25) | NO   |     | NULL    |       |
| last_name  | varchar(25) | NO   |     | NULL    |       |
| age        | int(11)     | NO   |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)

mysql> alter table student
    -> ADD CONSTRAINT PK_student PRIMARY KEY (id,first_name);
Query OK, 0 rows affected (1.38 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

6.5. DROP PRIMARY KEY CONSTRAINTS

To drop the PRIMARY KEY constraint from the table, use the SQL Query:

ALTER TABLE student

DROP PRIMARY KEY;

```
mysql> alter table student
    -> drop primary key;
Query OK, 0 rows affected (2.44 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc student;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| id         | int(11)     | NO   |     | NULL    |       |
| first_name | varchar(25) | NO   |     | NULL    |       |
| last_name  | varchar(25) | NO   |     | NULL    |       |
| age        | int(11)     | NO   |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
4 rows in set (0.05 sec)
```

As we can see from the student table, the primary key has been dropped from the table.

6.6. SQL FOREIGN KEY CONSTRAINT

A FOREIGN KEY is used to link two tables together. It is sometimes also called a referencing key. Foreign Key is a combination of columns (can be single column) whose value matches a Primary Key in the different tables. The relationship between two tables matches the Primary Key in one of the tables with a Foreign Key in the second table. If the table contains a primary key defined on any field, then the user should not have two records having the equal value of that field.

Let's create two tables using the foreign key.

CUSTOMER table

```
CREATE TABLE customer(
 Id int NOT NULL,
 Name varchar(20)   NOT NULL,
Age int NOT NULL,
Address varchar(25) ,
 Salary decimal (18, 2),
  PRIMARY KEY (id)
);
```

```
mysql> CREATE TABLE customer(
    ->    Id int NOT NULL,
    ->    Name varchar(20)   NOT NULL,
    ->    Age int NOT NULL,
    ->    Address varchar(25) ,
    ->    Salary decimal (18, 2),
    ->    PRIMARY KEY (id)
    -> );
Query OK, 0 rows affected (1.05 sec)

mysql>
mysql> desc customer;
+---------+---------------+------+-----+---------+-------+
| Field   | Type          | Null | Key | Default | Extra |
+---------+---------------+------+-----+---------+-------+
| Id      | int(11)       | NO   | PRI | NULL    |       |
| Name    | varchar(20)   | NO   |     | NULL    |       |
| Age     | int(11)       | NO   |     | NULL    |       |
| Address | varchar(25)   | YES  |     | NULL    |       |
| Salary  | decimal(18,2) | YES  |     | NULL    |       |
+---------+---------------+------+-----+---------+-------+
5 rows in set (0.08 sec)
```

Order Table with Foreign key

CREATE TABLE Orders (OrderID int NOT NULL, OrderNumber int NOT NULL, Id int,

PRIMARY KEY(OrderID), CONSTRAINT FK_customerOrder FOREIGN KEY(Id));

```
mysql> CREATE TABLE Orders (
    ->     OrderID int NOT NULL,
    ->     OrderNumber int NOT NULL,
    ->     Id int,
    ->     PRIMARY KEY (OrderID),
    ->     CONSTRAINT FK_customerOrder FOREIGN KEY (Id)
    ->     REFERENCES customer(Id)
    -> );
Query OK, 0 rows affected (1.08 sec)

mysql> desc orders;
+-------------+---------+------+-----+---------+-------+
| Field       | Type    | Null | Key | Default | Extra |
+-------------+---------+------+-----+---------+-------+
| OrderID     | int(11) | NO   | PRI | NULL    |       |
| OrderNumber | int(11) | NO   |     | NULL    |       |
| Id          | int(11) | YES  | MUL | NULL    |       |
+-------------+---------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

Here the Id is the primary key for the customer table and foreign key for orders table.

FOREIGN KEY on ALTER TABLE

To create the FOREIGN KEY constraint on the "PersonID" column when the "Orders" table is already created, use the SQL query:

ALTER TABLE Orders

ADD FOREIGN KEY (ID) REFERENCES customer(id);

```
mysql> ALTER TABLE Orders
    -> ADD FOREIGN KEY (ID) REFERENCES customer(id);
Query OK, 0 rows affected (2.38 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql>
mysql> desc orders;
+-------------+---------+------+-----+---------+-------+
| Field       | Type    | Null | Key | Default | Extra |
+-------------+---------+------+-----+---------+-------+
| OrderID     | int(11) | NO   | PRI | NULL    |       |
| OrderNumber | int(11) | NO   |     | NULL    |       |
| Id          | int(11) | YES  | MUL | NULL    |       |
+-------------+---------+------+-----+---------+-------+
3 rows in set (0.03 sec)
```

6.7. DROP A FOREIGN KEY CONSTRAINT

To drop a FOREIGN KEY constraint from the table, use the SQL query:

```
ALTER TABLE Orders

DROP FOREIGN KEY FK_PersonOrder;
```

```
mysql> ALTER TABLE Orders
    -> DROP FOREIGN KEY FK_customerOrder;
Query OK, 0 rows affected (0.19 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

6.8. SQL CHECK CONSTRAINTS

The CHECK CONSTRAINTS is used to limit the range of value that can be placed in a column if the user defines a CHECK constraint on a single column, it allows only specific values for the column. If the user defines a CHECK constraint on a table, it can limit the values in particular columns based on values in another column in the row.

SQL CHECK on CREATE TABLE

SQL Query to creates a CHECK constraint on the column "Age" when the table "Persons" is created. The CHECK constraint makes sure that the user can not have any person below 18 years:

```
CREATE TABLE Persons (

 ID int NOT NULL,

 LastName varchar(255) NOT NULL,

 FirstName varchar(255),

Age int,

 CHECK (Age>=18)

);
```

```
mysql> CREATE TABLE Persons (
    ->      ID int NOT NULL,
    ->      LastName varchar(255) NOT NULL,
    ->      FirstName varchar(255),
    ->      Age int,
    ->      CHECK (Age>=18)
    -> );
Query OK, 0 rows affected (1.19 sec)

mysql> insert into Persons values(1, 'abc', 'aaa', 17);
ERROR 3819 (HY000): Check constraint 'persons_chk_1' is violated.
```

Here we have created the Persons table and given a check constraint on the Age column. If the Age<18, then it will throw an error, as shown below.

INSERT INTO Persons VALUES(1, ‘abc’, ‘aaa’, 17);

```
mysql> insert into Persons values(1, 'abc', 'aaa', 17);
ERROR 3819 (HY000): Check constraint 'persons_chk_1' is violated.
```

For creating a CHECK constraint on multiple columns in the table, use the SQL syntax:

```
mysql> CREATE TABLE Persons (
    ->      ID int NOT NULL,
    ->      LastName varchar(255) NOT NULL,
    ->      FirstName varchar(255),
    ->      Age int,
    ->      City varchar(255),
    ->      CONSTRAINT CHK_Person CHECK (Age>=18 AND City='Sandnes')
    -> );
Query OK, 0 rows affected (1.20 sec)
```

CHECK on ALTER TABLE

Create a CHECK constraint on the column "Age" when the table is already created, use the following SQL:

ALTER TABLE Persons

ADD CHECK (Age >= 18)

```
mysql> ALTER TABLE Persons
    -> ADD CHECK (Age>=18)
    -> ;
Query OK, 0 rows affected (2.58 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Defining CHECK constraint on multiple columns of a table, use the SQL query:

ALTER TABLE Persons

ADD CONSTRAINT CHK_PersonAge CHECK (Age>=18 AND City='Sandnes');

```
mysql> ALTER TABLE Persons
    -> ADD CONSTRAINT CHK_PersonAge CHECK (Age>=18 AND City='Sandnes');
Query OK, 0 rows affected (2.31 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

6.9. DROP A CHECK CONSTRAINT

To drop a CHECK constraint from the table, use the following SQL:

ALTER TABLE Persons

DROP CHECK CHK_PersonAge;

```
mysql> ALTER TABLE Persons
    -> DROP CHECK CHK_PersonAge;
Query OK, 0 rows affected (0.38 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Here we have dropped the CHK_PersonAge constraints by using the drop statement.

6.10. SQL DEFAULT CONSTRAINT

The DEFAULT constraint in SQL is used to provide a default value for a column of the table. The default value will be added to every new record if no other value is mentioned.

SQL DEFAULT on CREATE TABLE

The SQL query to sets a DEFAULT value for the "City" column when the "Persons" table is created

CREATE TABLE Persons (

ID int NOT NULL,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Age int,

City varchar(255) DEFAULT 'Sandnes'

);

```
mysql> CREATE TABLE Persons (
    ->     ID int NOT NULL,
    ->     LastName varchar(255) NOT NULL,
    ->     FirstName varchar(255),
    ->     Age int,
    ->     City varchar(255) DEFAULT 'Sandnes'
    -> );
Query OK, 0 rows affected (1.06 sec)

mysql> desc persons
    -> ;
+-----------+--------------+------+-----+---------+-------+
| Field     | Type         | Null | Key | Default | Extra |
+-----------+--------------+------+-----+---------+-------+
| ID        | int(11)      | NO   |     | NULL    |       |
| LastName  | varchar(255) | NO   |     | NULL    |       |
| FirstName | varchar(255) | YES  |     | NULL    |       |
| Age       | int(11)      | YES  |     | NULL    |       |
| City      | varchar(255) | YES  |     | Sandnes |       |
+-----------+--------------+------+-----+---------+-------+
5 rows in set (0.06 sec)
```

As we can see in the Persons table, the city name is written as Sandnes by Default.

SQL DEFAULT on ALTER TABLE

To create a DEFAULT constraint on the column "City" when the table is already created, use the following SQL:

ALTER TABLE Persons

ALTER Age SET DEFAULT 20;

```
mysql> ALTER TABLE Persons
    -> ALTER Age SET DEFAULT 20;
Query OK, 0 rows affected (0.44 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc persons;
+-----------+--------------+------+-----+---------+-------+
| Field     | Type         | Null | Key | Default | Extra |
+-----------+--------------+------+-----+---------+-------+
| ID        | int(11)      | NO   |     | NULL    |       |
| LastName  | varchar(255) | NO   |     | NULL    |       |
| FirstName | varchar(255) | YES  |     | NULL    |       |
| Age       | int(11)      | YES  |     | 20      |       |
| City      | varchar(255) | YES  |     | Sandnes |       |
+-----------+--------------+------+-----+---------+-------+
5 rows in set (0.04 sec)
```

6.11. DROP A DEFAULT CONSTRAINT

To drop a DEFAULT constraint from the table, use the SQL query:

ALTER TABLE Persons

ALTER City DROP DEFAULT;

```
mysql> ALTER TABLE Persons
    -> ALTER City DROP DEFAULT;
Query OK, 0 rows affected (0.18 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc persons;
+-----------+--------------+------+-----+---------+-------+
| Field     | Type         | Null | Key | Default | Extra |
+-----------+--------------+------+-----+---------+-------+
| ID        | int(11)      | NO   |     | NULL    |       |
| LastName  | varchar(255) | NO   |     | NULL    |       |
| FirstName | varchar(255) | YES  |     | NULL    |       |
| Age       | int(11)      | YES  |     | 20      |       |
| City      | varchar(255) | YES  |     | NULL    |       |
+-----------+--------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

As we can see in the Persons table, the default value of the city has been removed.

7. SQL CREATE INDEX STATEMENT

CREATE INDEX statement in SQL is used to create indexes in tables. The indexes are used to retrieve data from the database more quickly than others. The user can not see the indexes, and they are just used to speed up queries /searches.

Note: Updating the table with indexes takes a lot of time than updating a table without indexes. It is because the indexes also need an update. So, only create indexes on those columns that will be frequently searched against.

CREATE INDEX Syntax

It creates an index on a table. Duplicate values are allowed:

```
CREATE INDEX index_name
ON table_name (column1, column2, ...);
```

Example:

Creates an index named "idex_lastname" on the "LastName" column in the "Persons" table:

```
CREATE INDEX idex_lastname
on Persons (LastName)
```

```
mysql> CREATE INDEX idex_lastname
    -> ON Persons (LastName);
Query OK, 0 rows affected (0.86 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc persons;
+-----------+--------------+------+-----+---------+-------+
| Field     | Type         | Null | Key | Default | Extra |
+-----------+--------------+------+-----+---------+-------+
| ID        | int(11)      | NO   |     | NULL    |       |
| LastName  | varchar(255) | NO   | MUL | NULL    |       |
| FirstName | varchar(255) | YES  |     | NULL    |       |
| Age       | int(11)      | YES  |     | 20      |       |
| City      | varchar(255) | YES  |     | NULL    |       |
+-----------+--------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

If a user wants to create an index on a combination of columns, you can list the column names within the parentheses, separated by commas:

```
CREATE INDEX idex_pname
ON Persons (LastName, FirstName);
```

CREATE UNIQUE INDEX

It creates a unique index on a table and Duplicate values are not allowed.

Syntax:

```
Create UNIQUE INDEX index_name
on table_name (column1, column2, ...);
```

Note: The query for creating indexes varies among different databases. Therefore, Check the query for creating indexes in your database.

7.1. DROP INDEX STATEMENT

The DROP INDEX statement in SQL is used to delete an index in a table.

```
ALTER TABLE table_name
DROP INDEX index_name;
```

8. SQL VIEWS STATEMENT

In SQL, the view is a virtual table based on the result-set of an SQL statement. A view holds rows and columns, similar to a real table. The fields in a view are fields from one or more real tables in the database. You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

CREATE VIEW Syntax

```
CREATE VIEW view_name AS

SELECT column1, column2, ...

FROM table_name

WHERE condition;
```

Note: A view always shows up-to-date data! The database engine recreates the data, using the view's SQL statement, every time a user queries a view.

Create a table customer

```
mysql> select * from customers;
+------+--------+-----------+--------+------+
| id   | name   | address   | salary | age  |
+------+--------+-----------+--------+------+
|    1 | ramesh | ahamdabad |  35000 |   25 |
|    2 | khilan | dubai     |  45000 |   35 |
|    3 | delhi  | ram       |  44500 |   32 |
|    4 | patna  | komal     |  50500 |   27 |
+------+--------+-----------+--------+------+
4 rows in set (0.00 sec)
```

Create a view on the table **customers**. Here, the view would be used to have a customer name and age from the **customers** table.

```
CREATE VIEW CUSTOMERS_VIEW AS

SELECT name, age

FROM customers;
```

```
mysql> CREATE VIEW CUSTOMERS_VIEW AS
    -> SELECT name, age
    -> FROM customers;
Query OK, 0 rows affected (0.22 sec)

mysql> select * from CUSTOMERS_VIEW;
+--------+------+
| name   | age  |
+--------+------+
| ramesh |   25 |
| khilan |   35 |
| delhi  |   32 |
| patna  |   27 |
+--------+------+
4 rows in set (0.13 sec)
```

8.1. The WITH CHECK OPTION

The **WITH CHECK OPTION** in SQL is a CREATE VIEW statement option. The objective of the WITH CHECK OPTION is to make sure that all UPDATE and INSERTs satisfy the condition(s) in the view definition.

If they do not satisfy the condition(s), the UPDATE or INSERT returns an error.

The following code block has an example of creating the same view CUSTOMERS_VIEW with the WITH CHECK OPTION.

```
CREATE VIEW CUSTOMER_VIEW AS
SELECT name, age
FROM  customers
WHERE age IS NOT NULL
WITH CHECK OPTION;
```

```
mysql> CREATE VIEW CUSTOMER_VIEW AS
    -> SELECT name, age
    -> FROM  customers
    -> WHERE age IS NOT NULL
    -> WITH CHECK OPTION;
Query OK, 0 rows affected (0.17 sec)

mysql> select * from CUSTOMER_VIEW;
+--------+------+
| name   | age  |
+--------+------+
| ramesh |   25 |
| khilan |   35 |
| delhi  |   32 |
| patna  |   27 |
+--------+------+
4 rows in set (0.00 sec)
```

Here we have created a view(CUSTOMER_VIEW) **with the check** option.

8.2. DELETING ROWS INTO A VIEW

Rows of data can be deleted from a view. The same rules that apply to the UPDATE and INSERT commands apply to the DELETE command.

Example Delete a record having AGE = 25.

DELETE FROM CUSTOMER_VIEW

WHERE age = 25;

```
mysql> DELETE FROM CUSTOMER_VIEW
    ->    WHERE age = 25;
Query OK, 1 row affected (0.16 sec)

mysql> select * from CUSTOMER_VIEW;
+--------+------+
| name   | age  |
+--------+------+
| khilan |   35 |
| delhi  |   32 |
| patna  |   27 |
+--------+------+
3 rows in set (0.00 sec)
```

Here we have deleted the row, which contains the age = 25.

8.3. DROPPING VIEWS

Where the user has a view, you need a method to drop the view if it is no longer needed. The query is straightforward and is given below:

DROP VIEW view_name;

```
mysql> drop view customer_view;
Query OK, 0 rows affected (0.19 sec)

mysql> select * from CUSTOMER_VIEW;
ERROR 1146 (42S02): Table 'testdb.customer_view' doesn't exist
```

It's similar to the other dropping option, as we have done yet for tables. As we can see, the view is not available in the database after dropping the view.