**INTRODUCTION TO SQL**

**(Structural Query Language)**

**Table of Content**

Contents

[1. Introduction to SQL](#_Toc33719294)

[History of SQL](#_Toc33719295)

[What is Database?](#_Toc33719296)

[Relational Database](#_Toc33719297)

[SQL and Relational Databases](#_Toc33719298)

[How to run SQL Query on the local system](#_Toc33719299)

[2. Downloading and Installing MySQL](#_Toc33719300)

[2.1 Downloading MySQL](#_Toc33719301)

[2.2. Installation of MySQL](#_Toc33719302)

[3. SQL QUERY](#SQLQUERY)

[The SQL SELECT DISTINCT](#TheSQLSELECTDISTINCT)

[The SQL WHERE CLAUSE](#TheSQLWHERECLAUSE)

[The SQL WHERE CLAUSE WITH AND, OR & NOT](#TheSQLWHERECLAUSEWITHANDORNOT)

[The SQL ORDER BY](#TheSQLORDERBY)

[The SQL SELECT TOP CLAUSE](#TheSQLSELECTTOPCLAUSE)

[The SQL MIN() AND MAX() FUNCTION](#TheANDMAXFUNCTION)

[The SQL COUNT(), AVG() AND SUM() FUNCTION](#TheSQCOUNAVGSUMFUNCTIO)

[The SQL LIKE-OPERATOR](#TheQLLIKEOPERATOR)

[The SQL IN AND NOT IN OPERATORS](#TheSQLINDNOtINOPERATORS)

[The SQL BETWEEN OPERATOR](#The)

[The SQL ALIAS](#TheSQLALIAS)

[The SQL GROUP BY STATEMENT](#TheSQLGROUPBYSTATEMENT)

[The SQL HAVING CLAUSE](#TheSQLHAVINGCLAUSE)

[The SQL UNION](#TheSQLUNION)

[The SQL STORED PROCEDURE](#TheSQLSTOREDPROCEDURE)

[5. SQL DATABASE](#SQLDATABASE)

[The SQL CREATE DATABASE STATEMENT](#TheSQLCREATEDATABASESTATEMENT)

[The SQL DROP DATABASE STATEMENT](#TheSQLCREATEDATABASESTATEMENT)

[The SQL CREATE TABLE](#_Toc72219325)

[The SQL DROP TABLE STATEMENT](#TheSQLDROPTABLESTATEMENT)

[The SQL INSERT INTO STATEMENT](#_Toc72219327)

[The SQL NULL VALUES](#TheSQlNULLVALUES)

[The SQL UPDATE STATEMENT](#_The_SQL_UPDATE)

[The SQL DELETE STATEMENT](#_The_SQL_DELETE)

[The SQL ALTER TABLE STATEMENT](#TheSQLALTERTABLESTATEMENT)

[ALTER TABLE - ADD COLUMN IN EXISTING TABLE](#_ALTER_TABLE_-)

[ALTER TABLE – MODIFY/ALTER COLUMN](#_ALTER_TABLE_–)

[ALTER TABLE - DROP COLUMN](#ALTERTABLEDROPCOLUMN)

# 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.

## 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).

## 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.

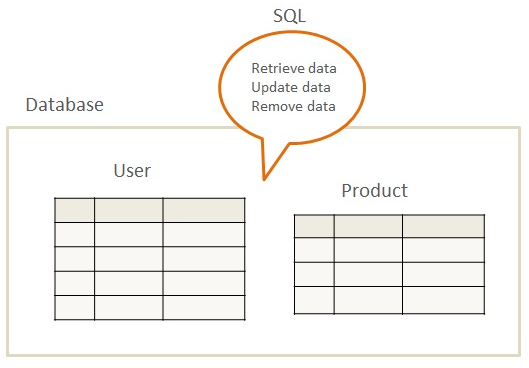


Row

Column

## 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**.



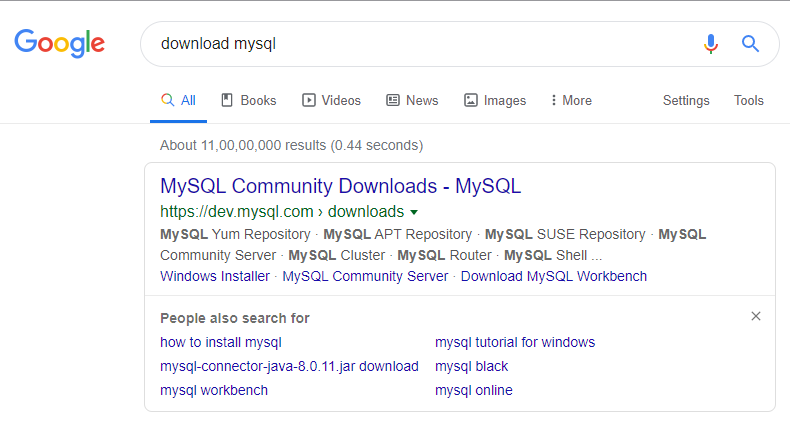
## How to run SQL Query on the local system

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.

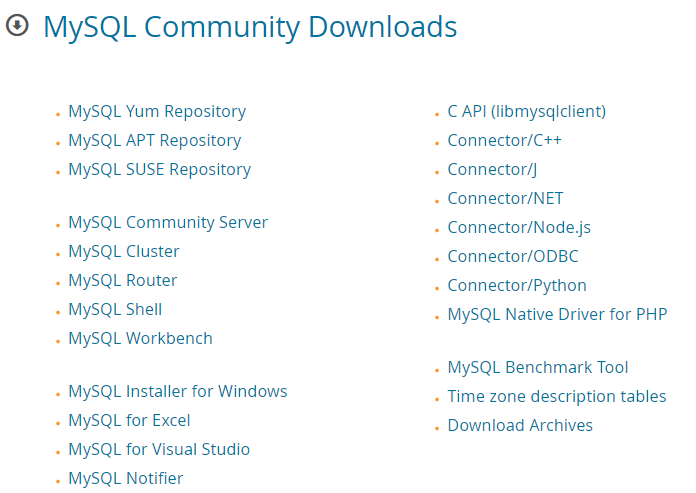
# Downloading and Installing MySQL

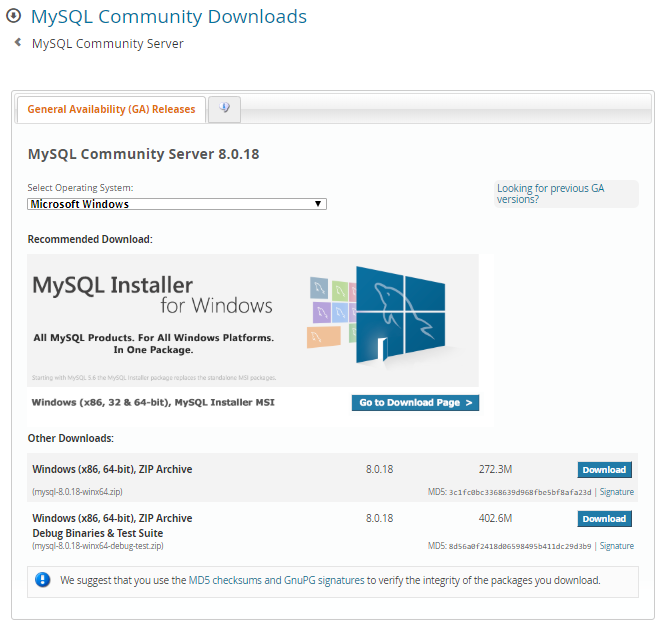
## 2.1 Downloading MySQL

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

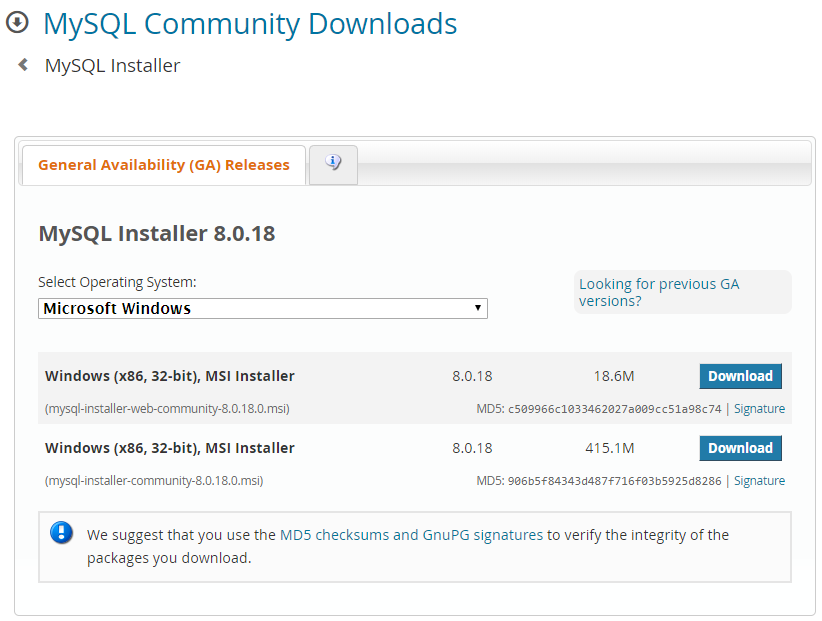


**Step 2:** Click on **MySQL Community Server**

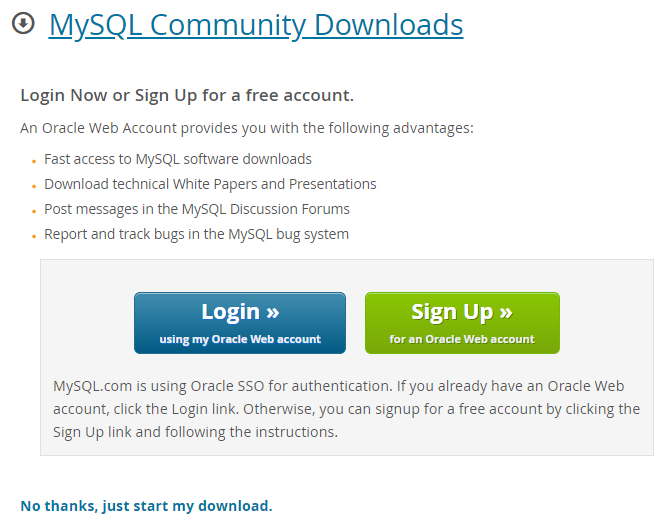


**Step 3:** Click on theMySQL installer MSI **Go to Download Page >**

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



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

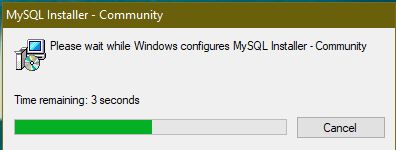


**Note:** Once the Downloading is completed, then double-click on that and install it on the local system.

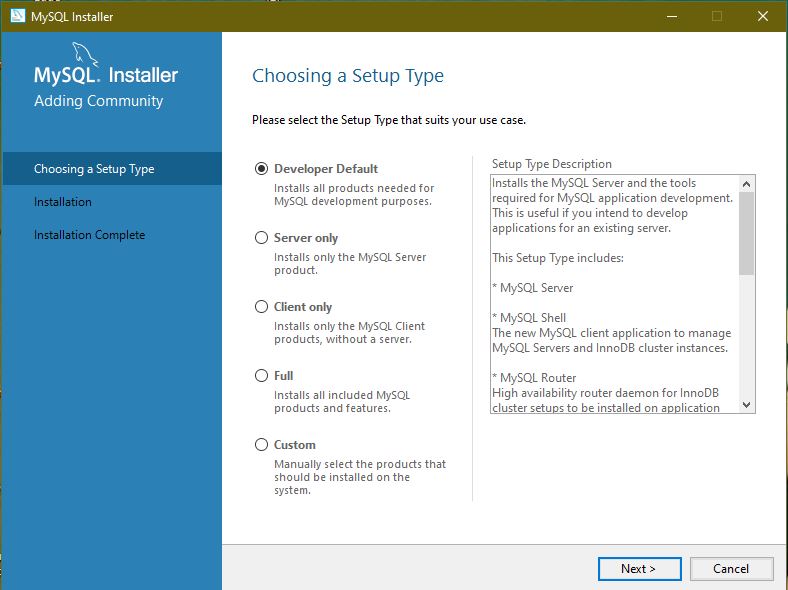
## 2.2. Installation of MySQL

**Step 1:** **Double-Click** on Downloaded Application.

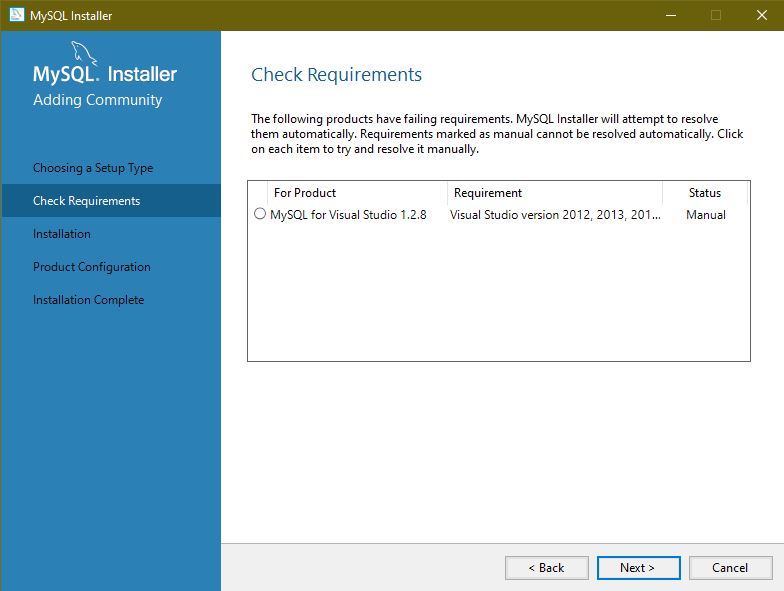
**Step 2:** After clicking on the application we will get a window like below



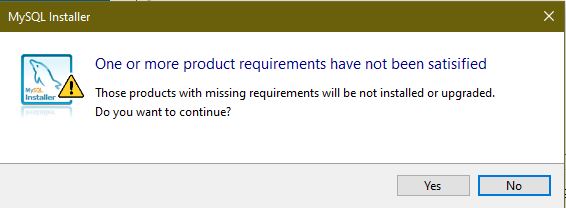
**Step 3:** Choosing the Setup type and click Next.



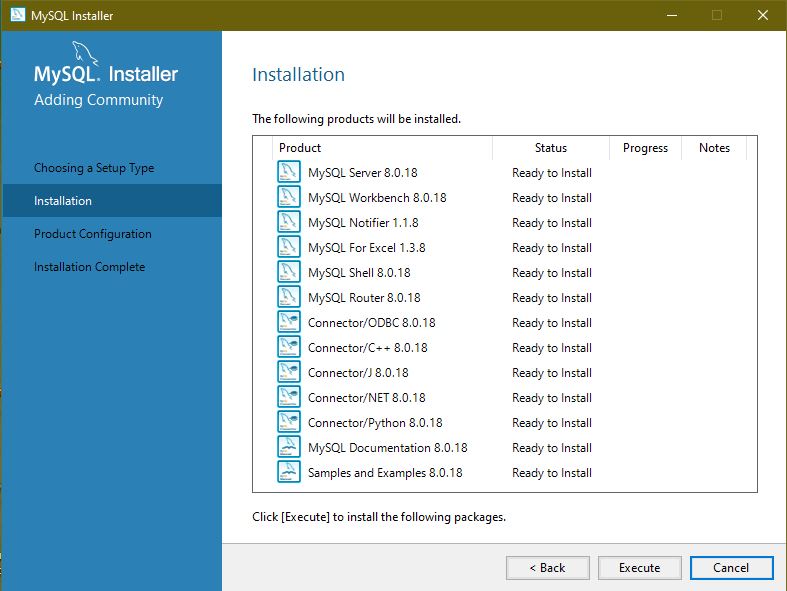
**Setup 4:** Click Next.

****

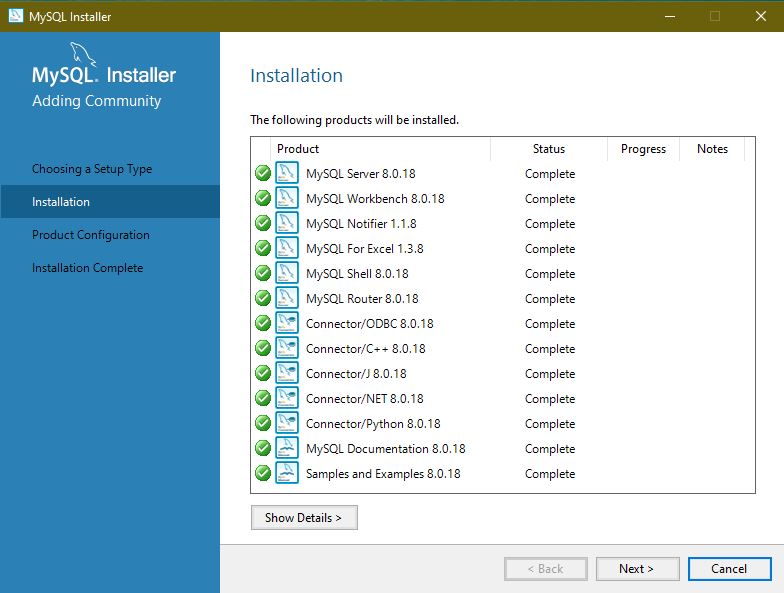
**Step 5:** Click Yes.

****

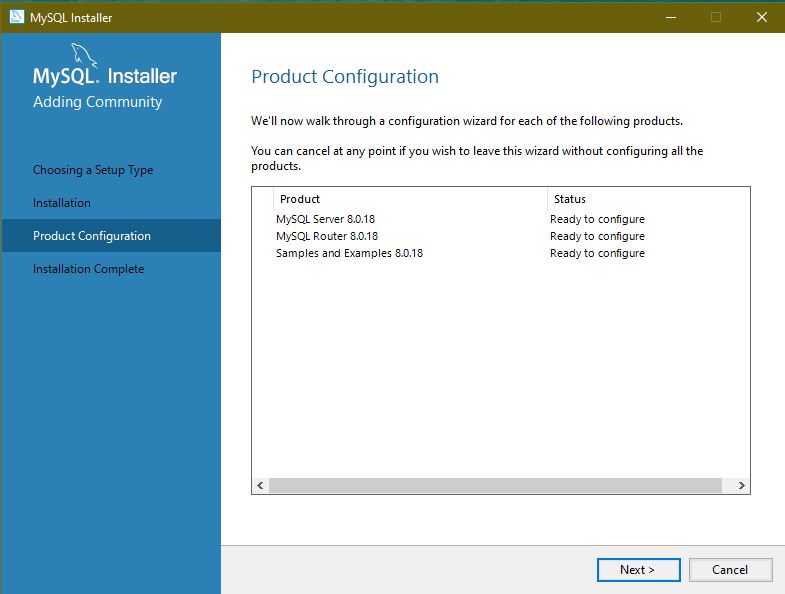
**Step 6:** Click Execute.

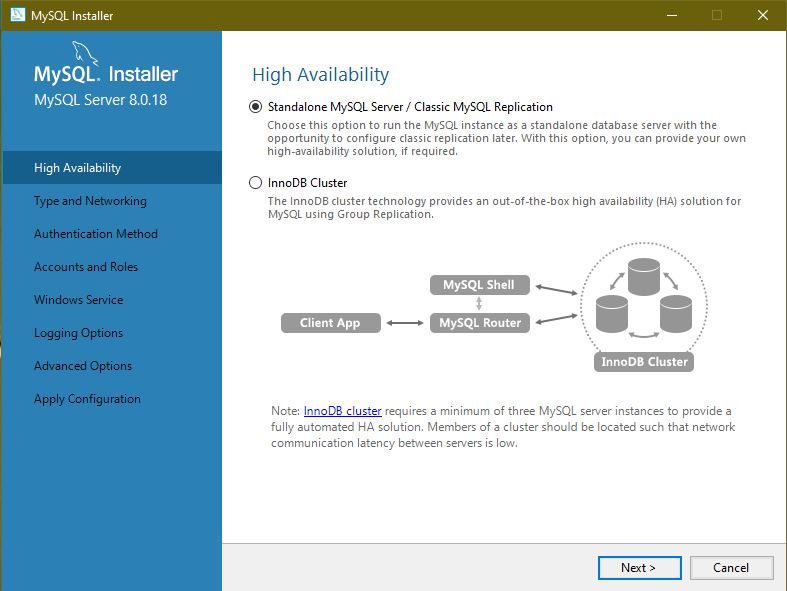
****

**Step 7:** After Execution, click on the Next.

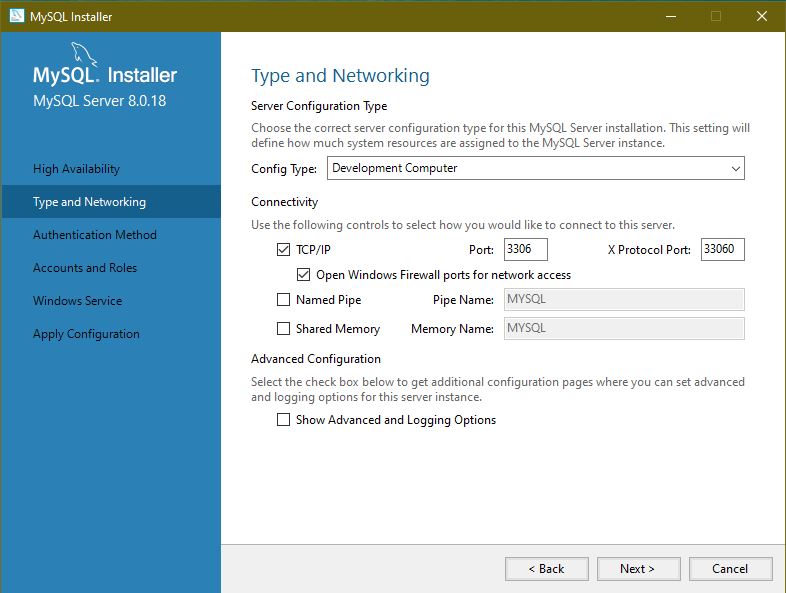
****

**Step 8:** Click Next.

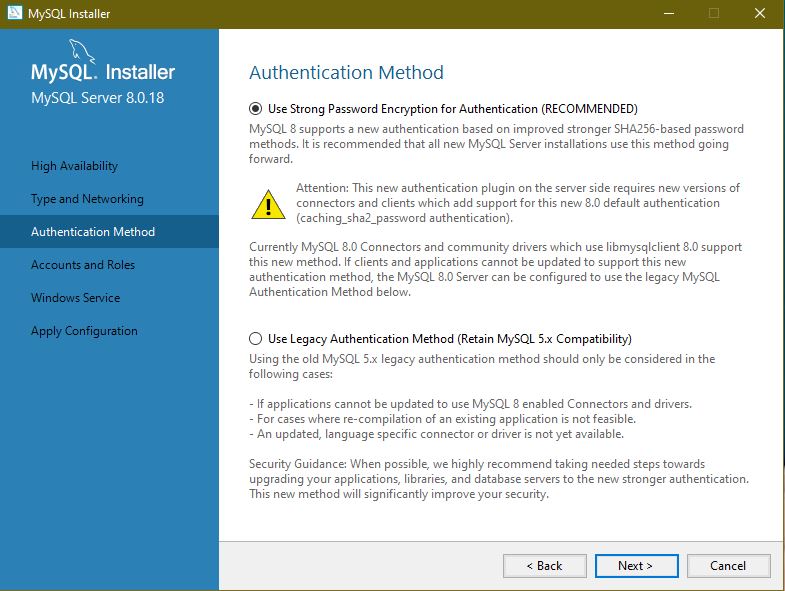
****

**Step 9:** Click Next.****

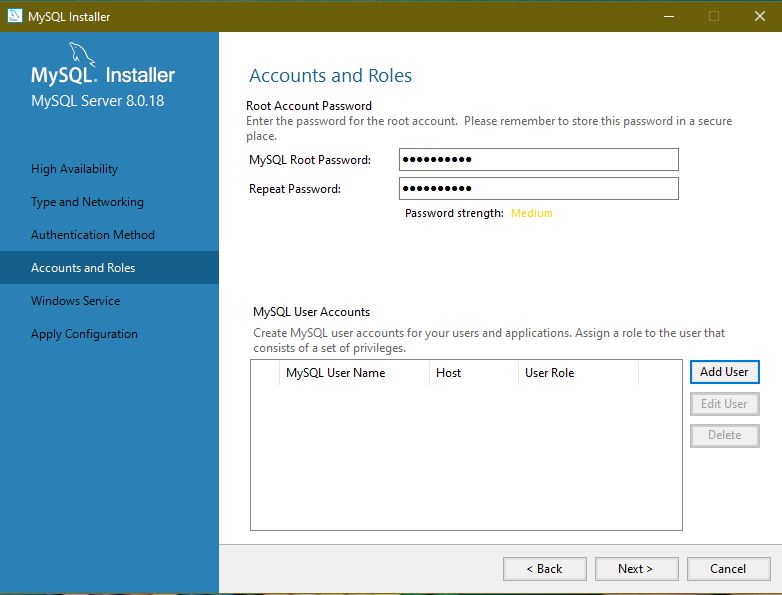
**Step 10:** Leave it as default and click Next.



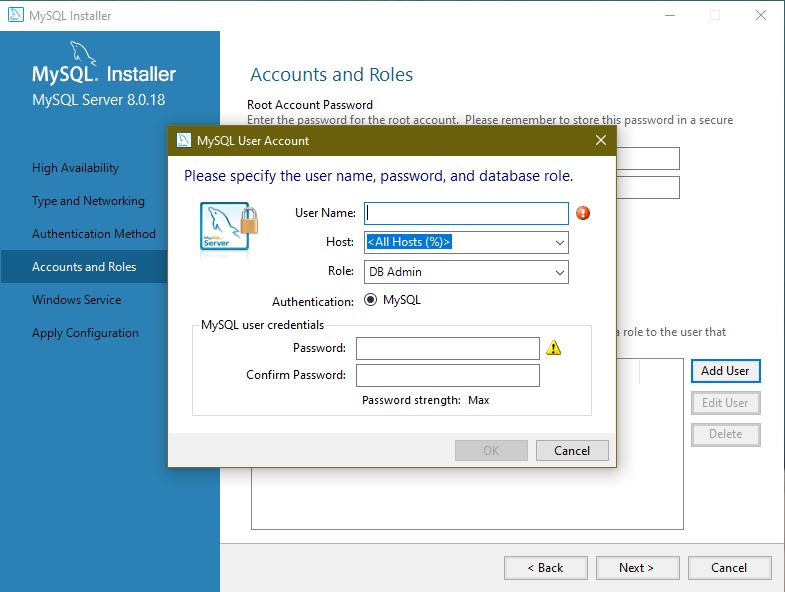
**Step 11:** Click Next

****

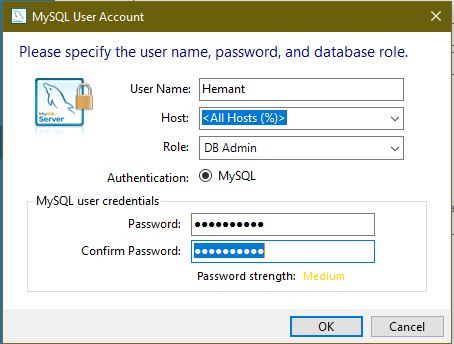
**Step 12:** Choose the root password

****

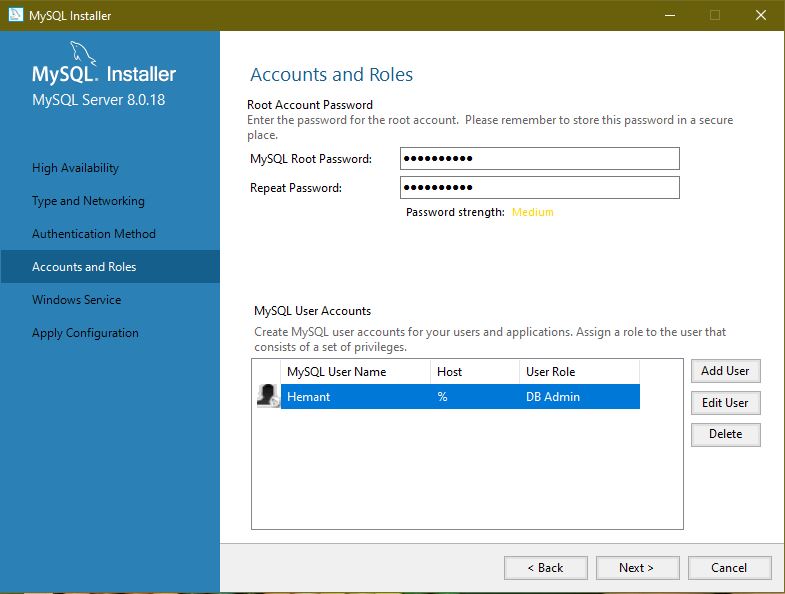
**Step 13:** Click on Add User and give the username and password.

****

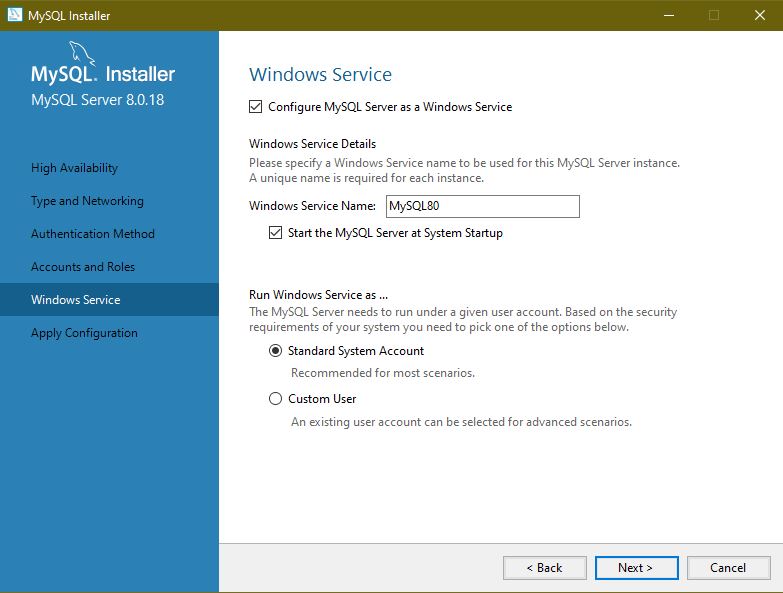
**Step 14:** After inserting the name and password click OK

****

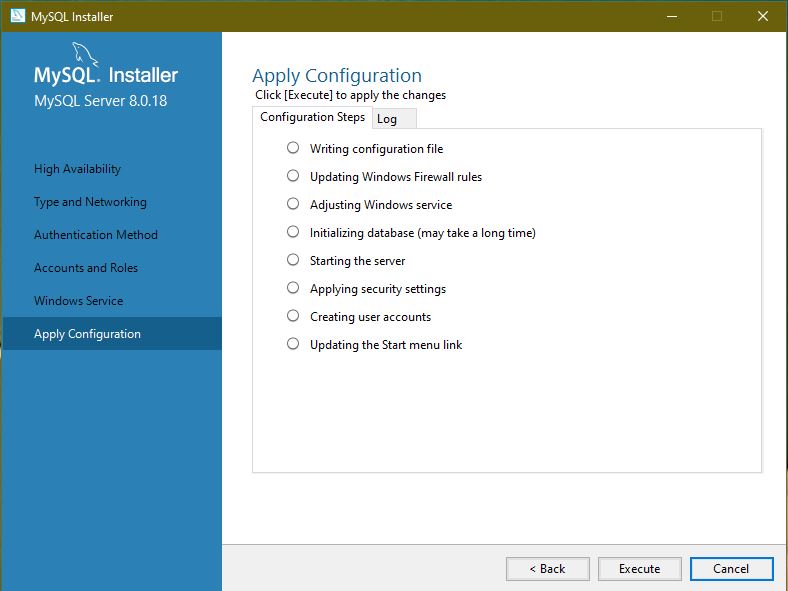
**Step 15:** After adding the user click Next

****

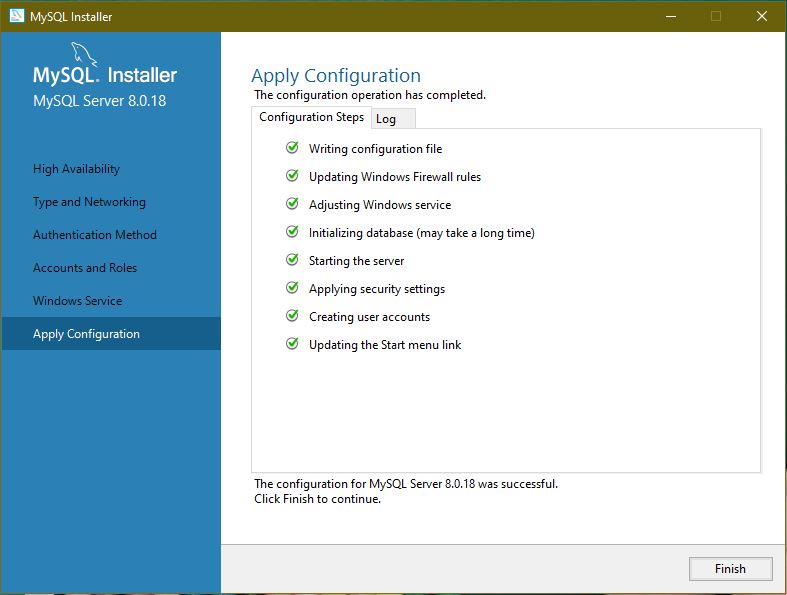
**Step 16:** Click Next

****

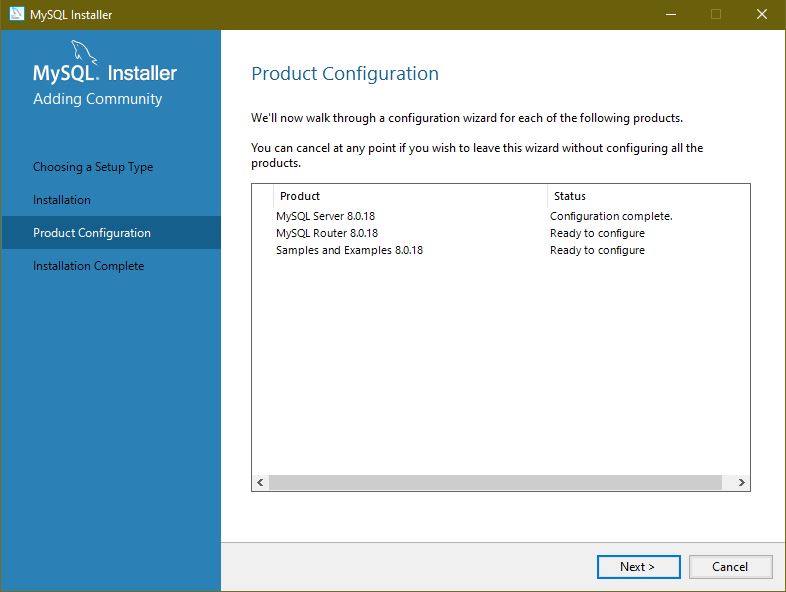
**Step 17:** Click on Execute

****

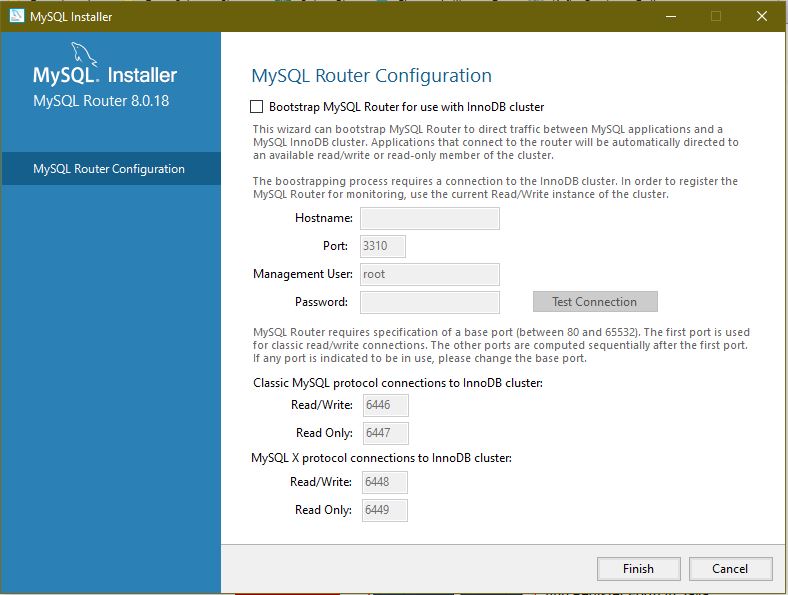
**Step 18:** After Clicking on execute,click Finish

****

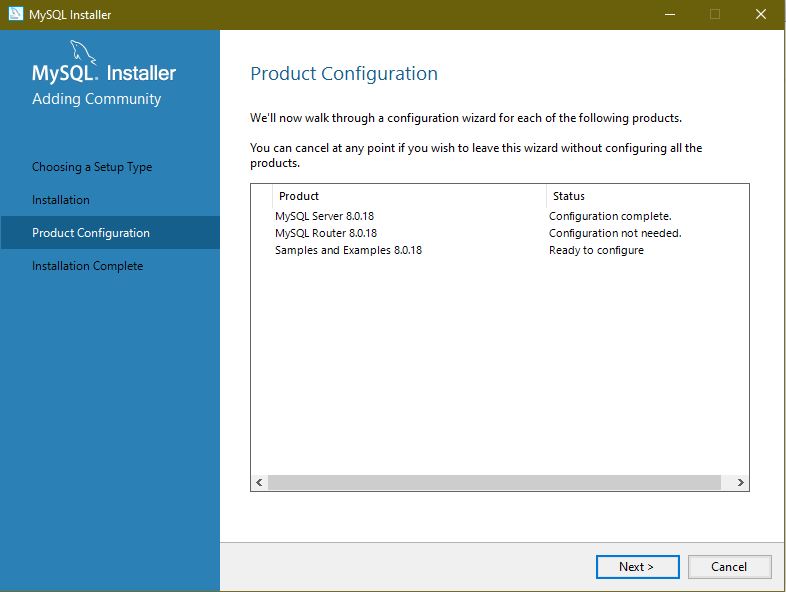
**Step 19:** Click Next

****

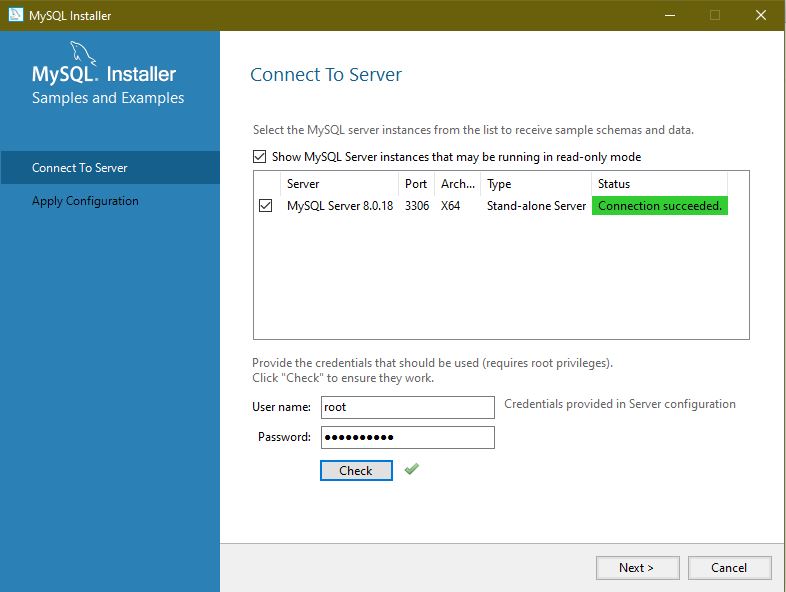
**Step 20:** Click on Finish

****

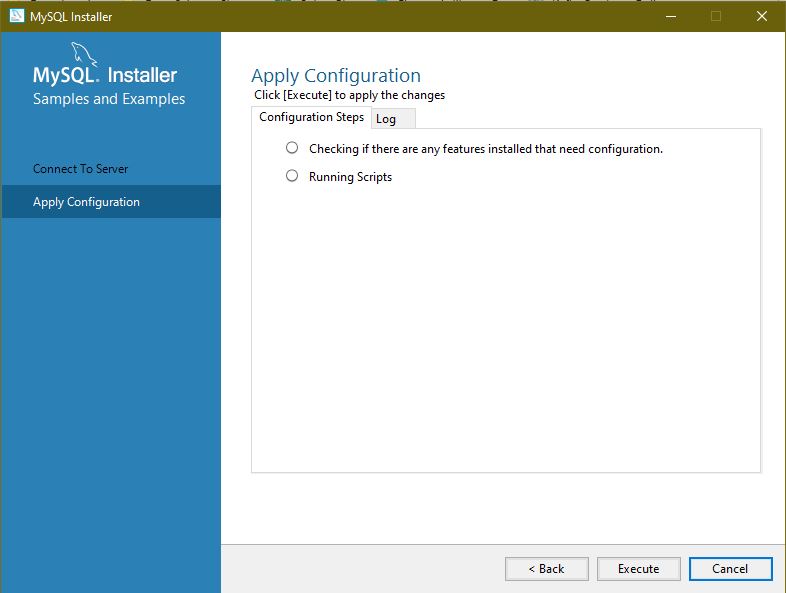
**Step 21:** Click Next.

****

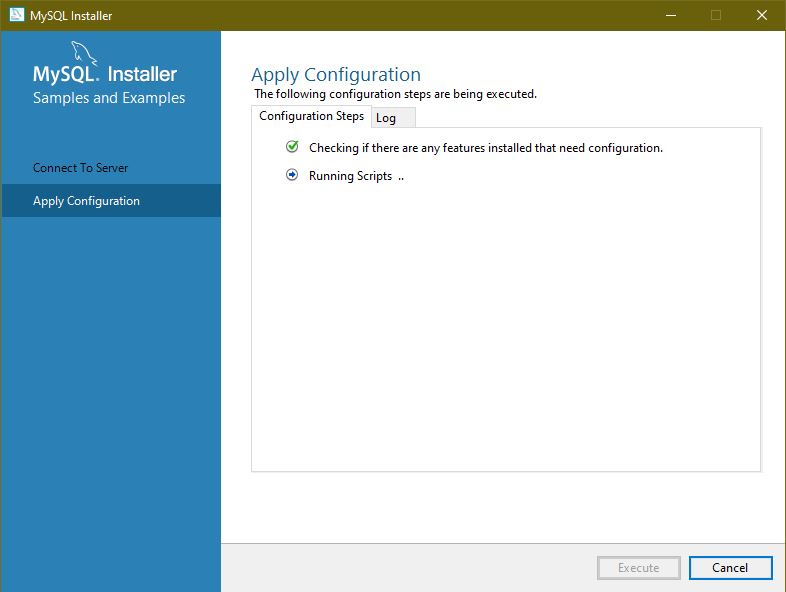
**Step 22:** Check the passwordand Click Next

****

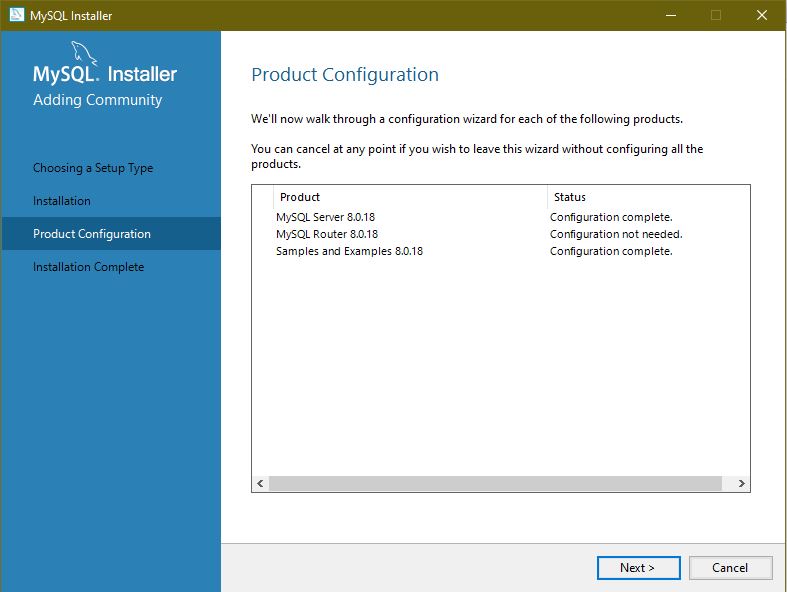
**Step 23:** Click Execute

****

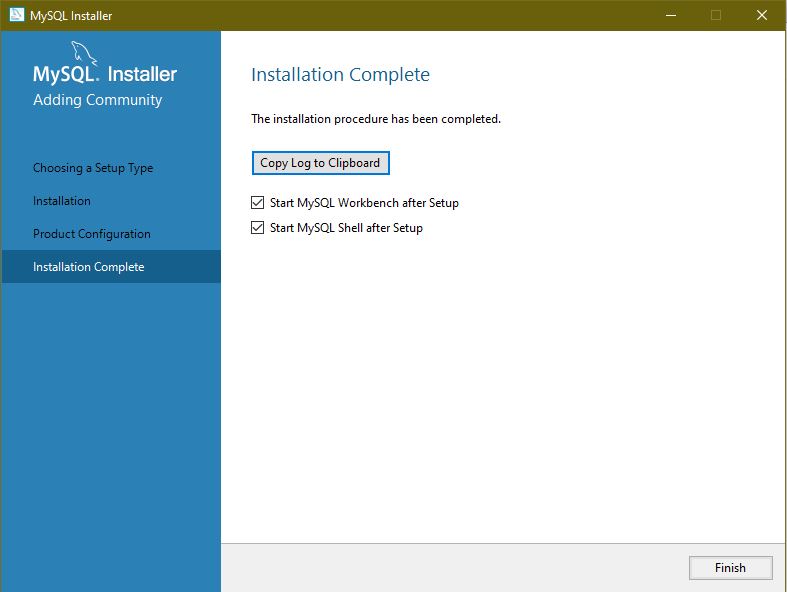
**Step 24:** After clicking on Execute

****

**Step 25:** Click Next

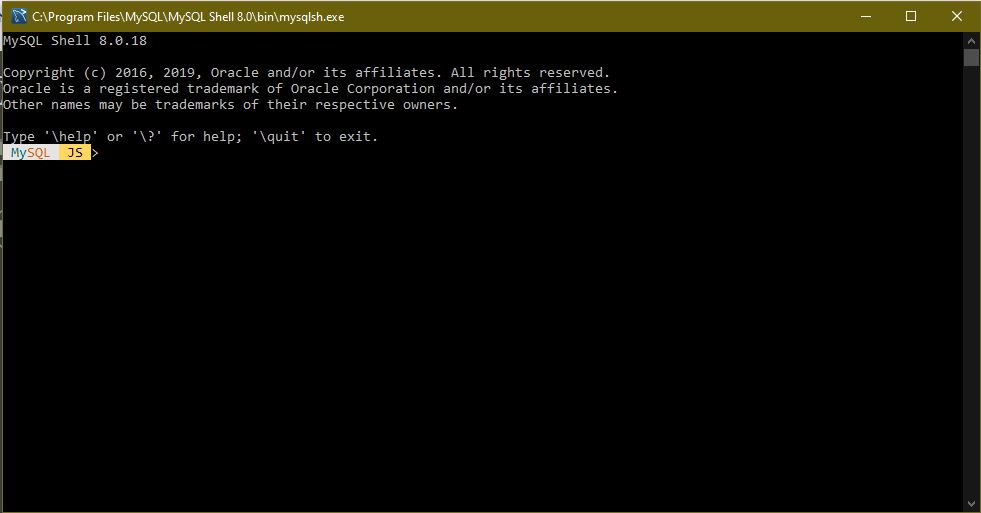


**Step 26:** Click Finish

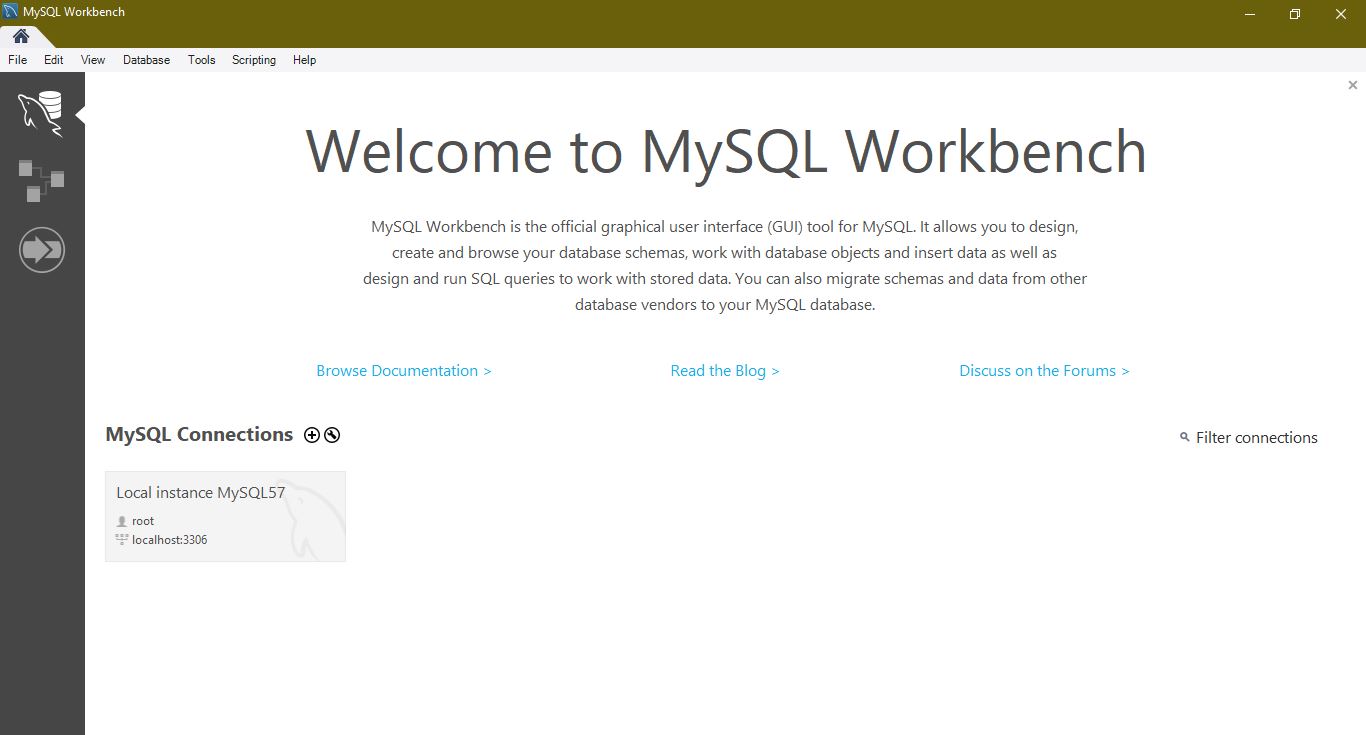
****

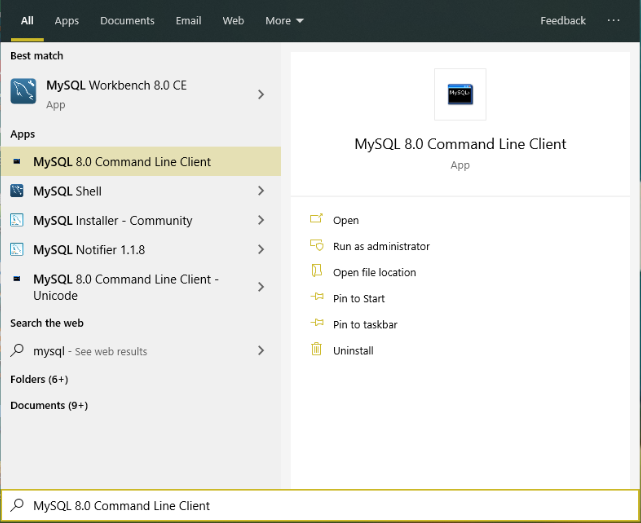
**Step 27:** After the successful installation of MySQL, two windows will open.

* MySQL Shell
* MySQL WorkBench



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

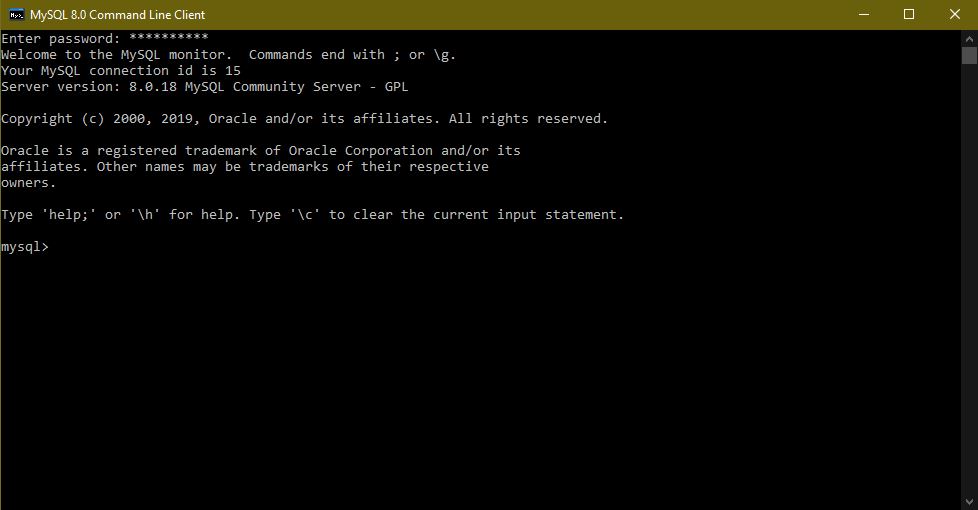


**Step 28:** Click on window Button and search forOpen MySQL Command. 

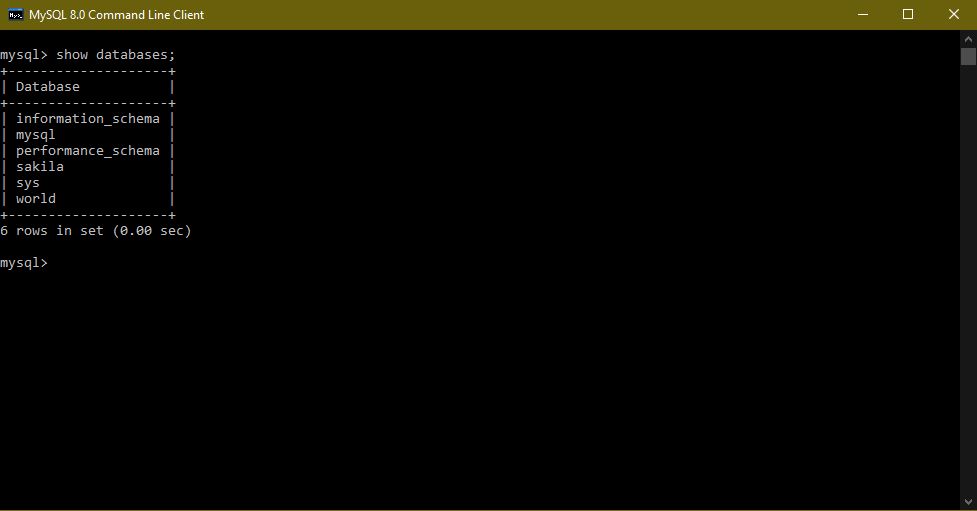
**Step 29:** Open MySQL Command-line Client and enter the password.



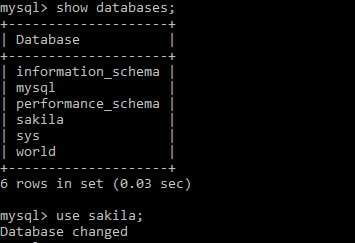
**Step 29:** After entering the password, your MySQL client will get connected with MySQL.



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



**Step 31**: we can use any of the above databases by just typing **use database\_name**

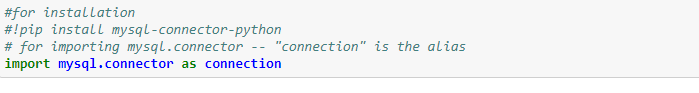


**MYSQL with PYTHON:**

# Integrating SQL with PYTHON

##### ***step1 : installing and importing mysql.connector module***

* to work in real world scenerio we must know how to integrate SQL with Python , since python allows us to connect with varioues databases like MYSQL,MongoDB , etc and to do this we must need to install and import mysql.connector module

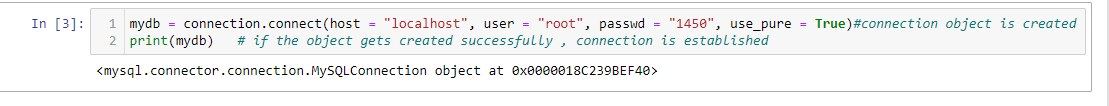


##### ***step2 : creating an object of mysql.connector.connect() --- connection object***

* to establish connection between python (application) and database MYSQL we will create an object of mysql.connector.connect(), connect method of mysql.connector is used

###### we need to pass following argument:

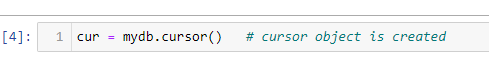
* host = "localhost" (database is available on local system, not on cloud)
* user = "root"
* passwd = "yourpassword" (keep it safe)
* use\_pure = True
* database = "optional- name of the database "



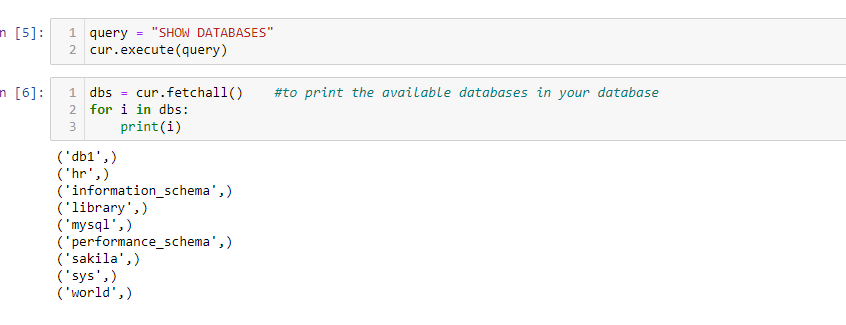
***step3 : Create an object of mysql.connector.cursor()-----cursor object***

* object of mysql.connector.cursor() or connection object ,in other words, iis created to execute operations like SQL statements
* once this cursor object is created it will interact with MYSQL server (local host in this case )using object of mysql.connector.connect() or connection object

##### *Note: GOOD PRACTICE: Everytime you execute an sql statement(query) create new cursor object(name can be same) of the same connection*



##### ***step4 : Execute the query using cursor object***



***step5 : Close the connection using connection object***



Link to this complete is given below: or click [here](https://github.com/swatishayna/SQL_Python/blob/main/tut1-Connecting%20with%20database.ipynb)

# https://github.com/swatishayna/SQL\_Python/blob/main/tut1-Connecting%20with%20database.ipynb

# 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).

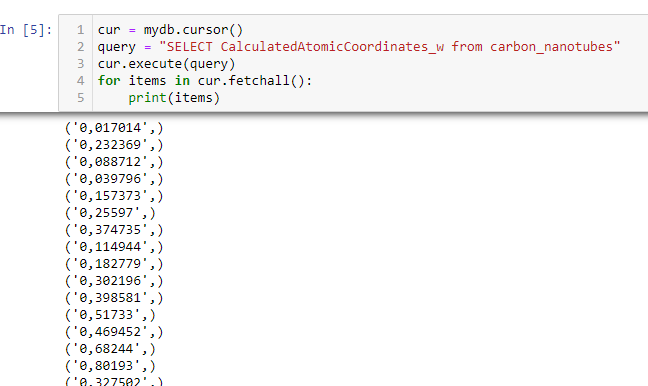
**The SQL SELECT QUERY**

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

The general syntax is:

SELECT CalculatedAtomicCoordinates\_w from carbon\_nanotubes

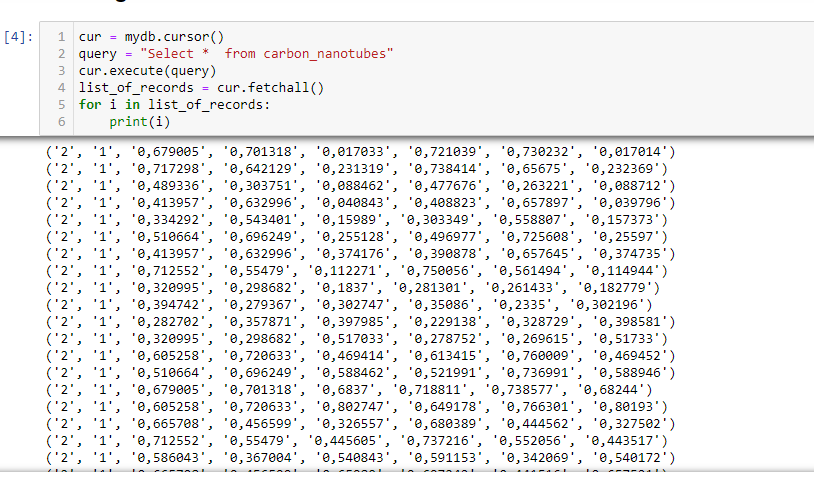
**Example**: Read the CalculatedAtomicCoordinates\_w from carbon\_nanotubes.



To select all columns, use **\***

SELECT **\***

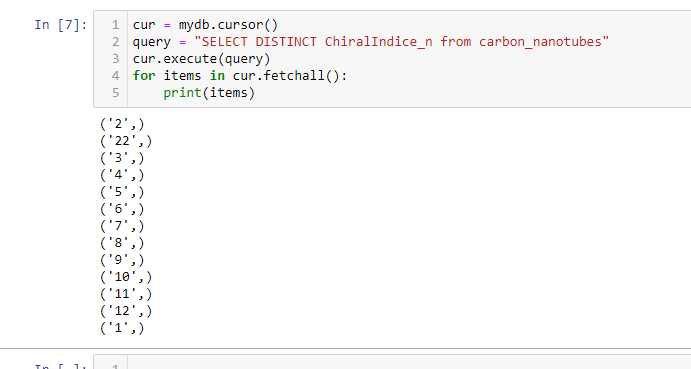
FROM carbon\_nanotubes



## The SQL SELECT DISTINCT

The SELECT DISTINCT statement is to return the different values.

SELECT DISTINCT ChiralIndice\_n from carbon\_nanotubes



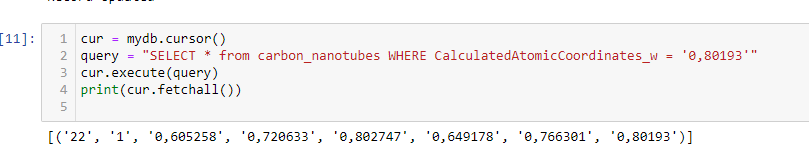
## The SQL WHERE CLAUSE

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 \* from carbon\_nanotubes WHERE CalculatedAtomicCoordinates\_w = '0,80193'

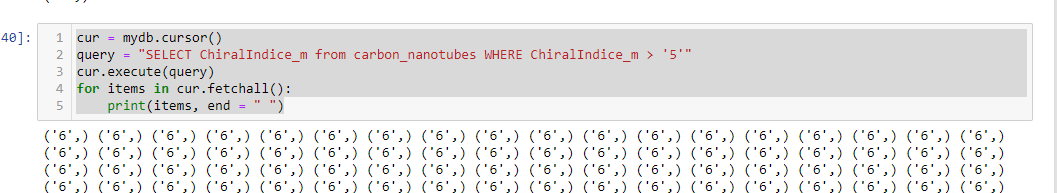


**Operators in where clause**

|  |  |
| --- | --- |
| **=** | 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 |

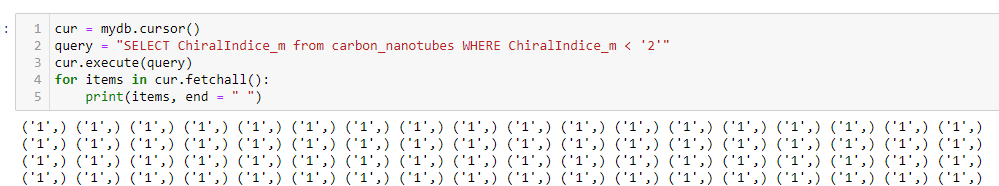
**Greater than**

SELECT ChiralIndice\_m from carbon\_nanotubes WHERE ChiralIndice\_m < '5’



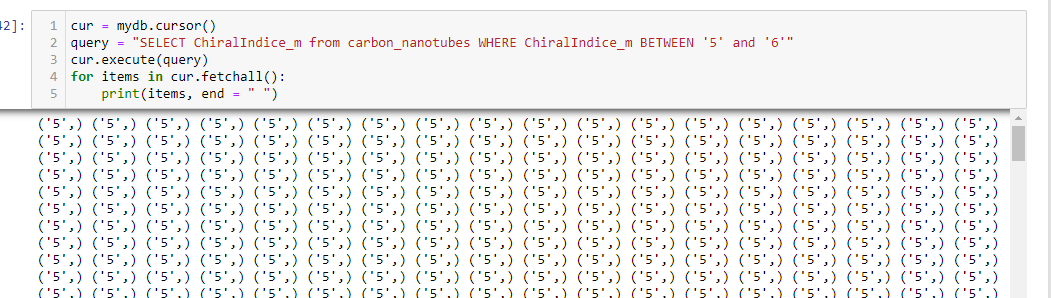
**Less than**

SELECT ChiralIndice\_m from carbon\_nanotubes WHERE ChiralIndice\_m < '2'



**BETWEEN**

**SELECT ChiralIndice\_m from carbon\_nanotubes WHERE ChiralIndice\_m BETWEEN '5' and '6'**

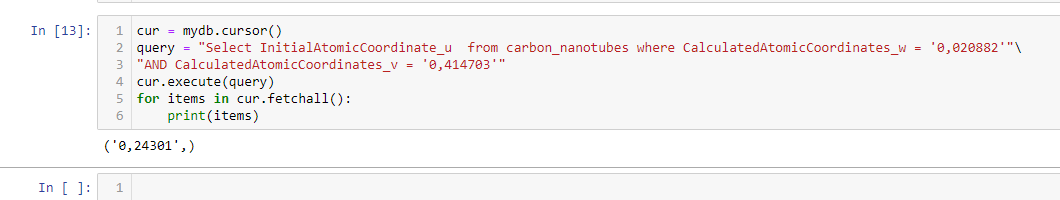


## The SQL WHERE CLAUSE WITH AND, OR & NOT

**A WHERE clause with AND**:

Select InitialAtomicCoordinate\_u from carbon\_nanotubes where CalculatedAtomicCoordinates\_w = '0,020882'

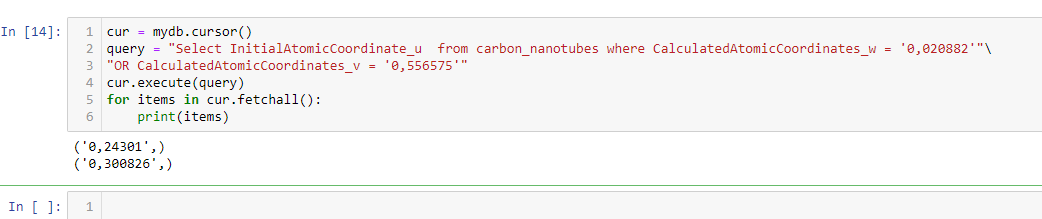
"AND CalculatedAtomicCoordinates\_v = '0,414703'



**A WHERE clause with OR:**

**Select InitialAtomicCoordinate\_u from carbon\_nanotubes where CalculatedAtomicCoordinates\_w = '0,020882'**

**"OR CalculatedAtomicCoordinates\_v = '0,556575'**



**A WHERE clause with NOT:**

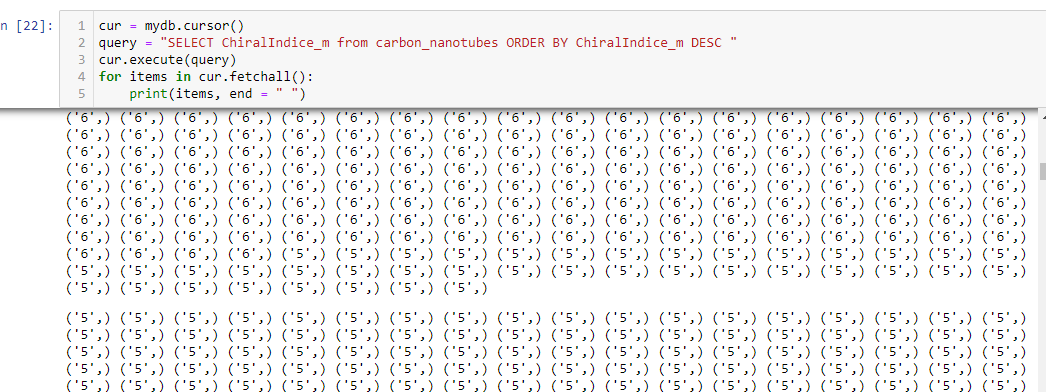
SELECT InitialAtomicCoordinate\_u ,ChiralIndice\_m from carbon\_nanotubes where NOT ChiralIndice\_m = '1'

## The SQL ORDER BY

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

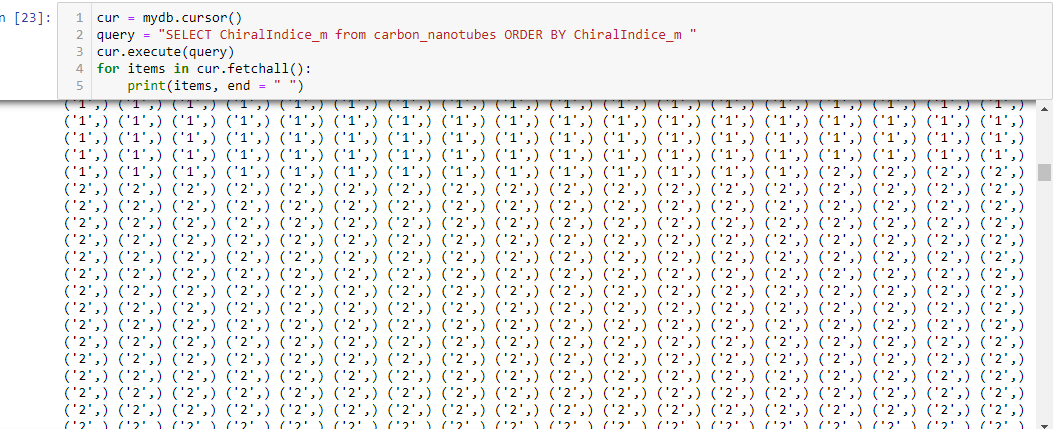
**Order By in Descending order**

**SELECT ChiralIndice\_m from carbon\_nanotubes ORDER BY ChiralIndice\_m DESC**



**Order By in Ascending order**

**SELECT ChiralIndice\_m from carbon\_nanotubes ORDER BY ChiralIndice\_m**



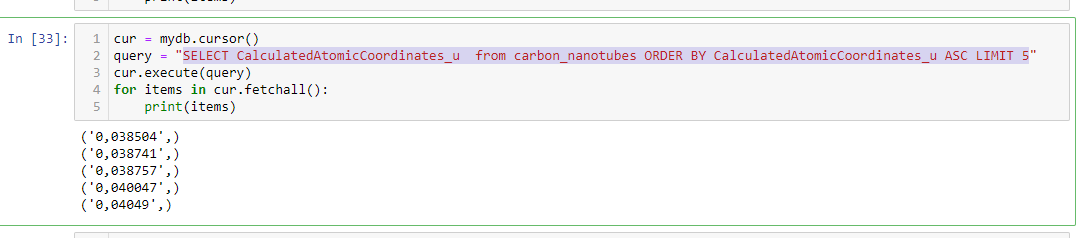
## The SQL SELECT TOP CLAUSE

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 CalculatedAtomicCoordinates\_u from carbon\_nanotubes ORDER BY CalculatedAtomicCoordinates\_u ASC LIMIT 5**



## The SQL MIN() AND MAX() FUNCTION

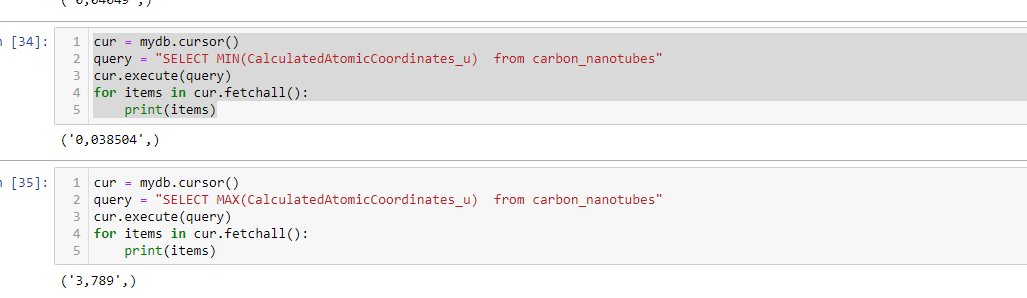
The MIN() function in SQL returns the smallest value of the selected column from the table. The MAX() function in SQL returns the largest value of the selected column from the table.

**MIN() Syntax**

**SELECT MIN(CalculatedAtomicCoordinates\_u) from carbon\_nanotubes**

**MAX() Syntax**

**SELECT MAX(CalculatedAtomicCoordinates\_u) from carbon\_nanotubes**



## 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(CalculatedAtomicCoordinates\_u) from carbon\_nanotubes

**AVG() Syntax**

SELECT AVG(CalculatedAtomicCoordinates\_u) from carbon\_nanotubes

**SUM() Syntax**

SELECT SUM(CalculatedAtomicCoordinates\_u) from carbon\_nanotubes



## 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**

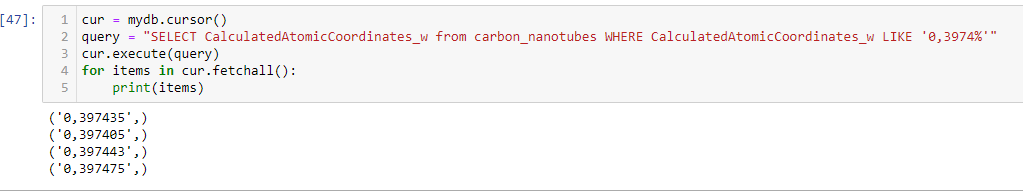
SELECT column1, column2, ...

FROM table\_name

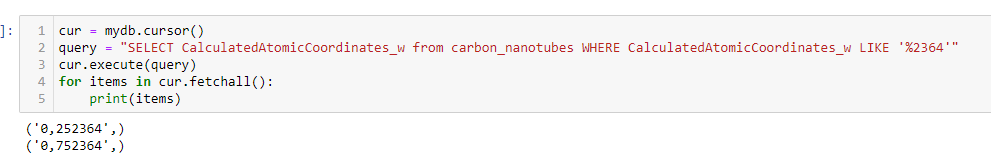
WHERE column LIKE pattern;

**Selects CalculatedAtomicCoordinates\_w of the carbon\_nanotubes with a CalculatedAtomicCoordinates\_w starting with '0,3974’:**

SELECT CalculatedAtomicCoordinates\_w from carbon\_nanotubes WHERE CalculatedAtomicCoordinates\_w LIKE '0,3974%'

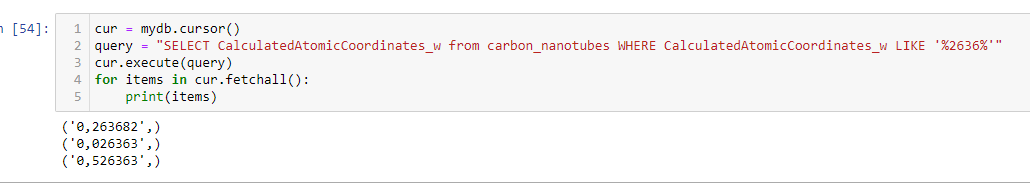


**Selects CalculatedAtomicCoordinates\_w of the carbon\_nanotubes with a CalculatedAtomicCoordinates\_w ending with '2364’:**

SELECT CalculatedAtomicCoordinates\_w from carbon\_nanotubes WHERE CalculatedAtomicCoordinates\_w LIKE '%2364'

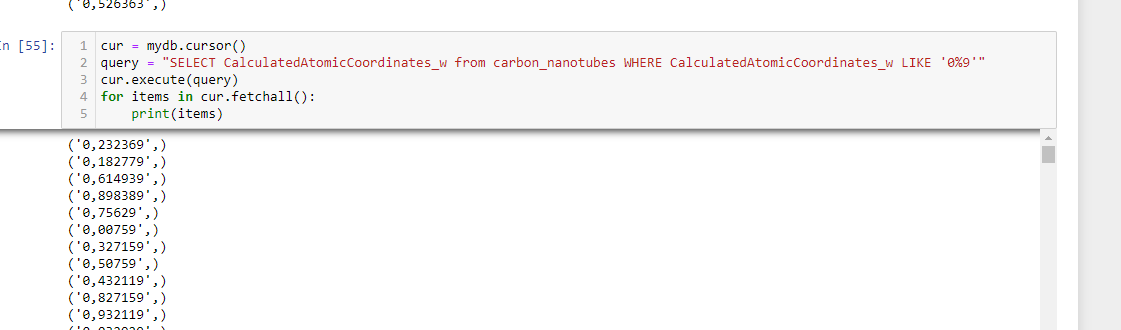
**Selects CalculatedAtomicCoordinates\_w of the carbon\_nanotubes with a CalculatedAtomicCoordinates\_w ‘2636’ in between :**

SELECT CalculatedAtomicCoordinates\_w from carbon\_nanotubes WHERE CalculatedAtomicCoordinates\_w LIKE '%2636%'



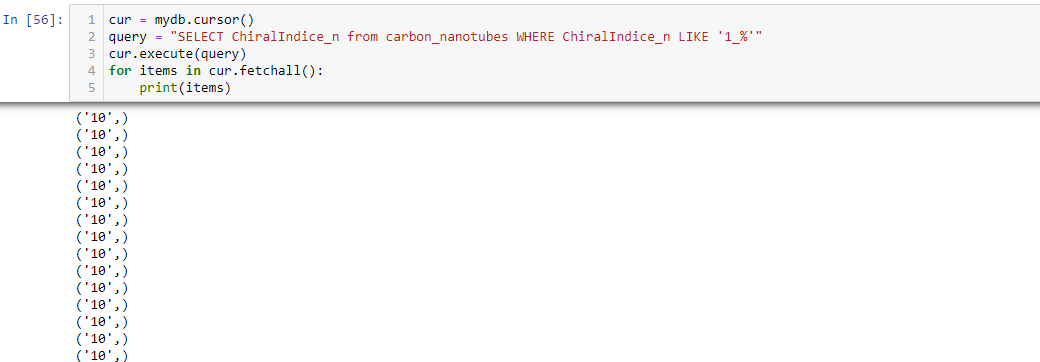
**Selects CalculatedAtomicCoordinates\_w of the carbon\_nanotubes with a CalculatedAtomicCoordinates\_w that starts with "0" and ends with "9":**

**SELECT CalculatedAtomicCoordinates\_w from carbon\_nanotubes WHERE CalculatedAtomicCoordinates\_w LIKE '0%9'**



**Selects ChiralIndice\_n from carbon\_nanotubes with a ChiralIndice\_n that starts with "1" and are at least 2 characters in length:**

SELECT ChiralIndice\_n from carbon\_nanotubes WHERE ChiralIndice\_n LIKE '1\_%

'

## 

## 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:**

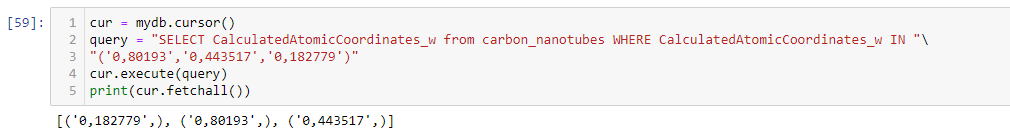
SELECT column\_name(s)

FROM table\_name

WHERE column\_name IN (SELECT STATEMENT);

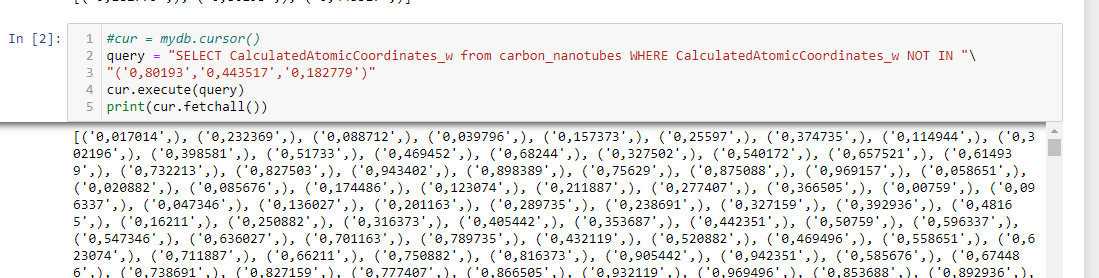
**Selects CalculatedAtomicCoordinates\_w from carbon\_nanotubes WHERE CalculatedAtomicCoordinates\_w IN ('0,80193','0,443517','0,182779'):**

**SELECT CalculatedAtomicCoordinates\_w from carbon\_nanotubes WHERE CalculatedAtomicCoordinates\_w IN ('0,80193','0,443517','0,182779')**



**Selects CalculatedAtomicCoordinates\_w from carbon\_nanotubes WHERE CalculatedAtomicCoordinates\_w NOT IN ('0,80193','0,443517','0,182779'):**

**SELECT CalculatedAtomicCoordinates\_w from carbon\_nanotubes WHERE CalculatedAtomicCoordinates\_w NOT IN ('0,80193','0,443517','0,182779')**

****

## 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)

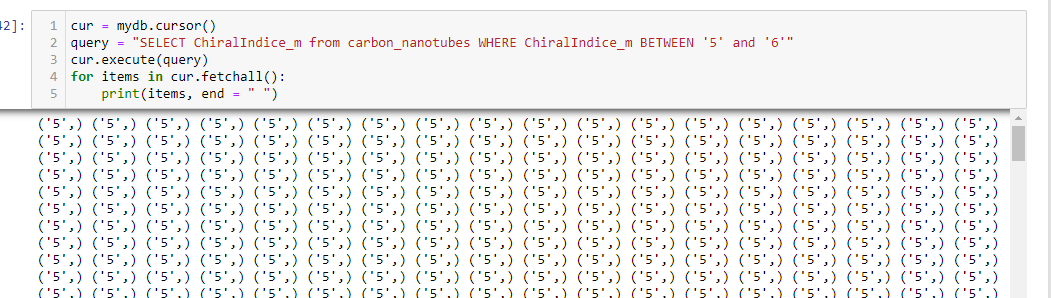
FROM table\_name

WHERE column\_name BETWEEN value1 AND value2;

**SELECT ChiralIndice\_m from carbon\_nanotubes WHERE ChiralIndice\_m BETWEEN '5' and '6':**

**BETWEEN**

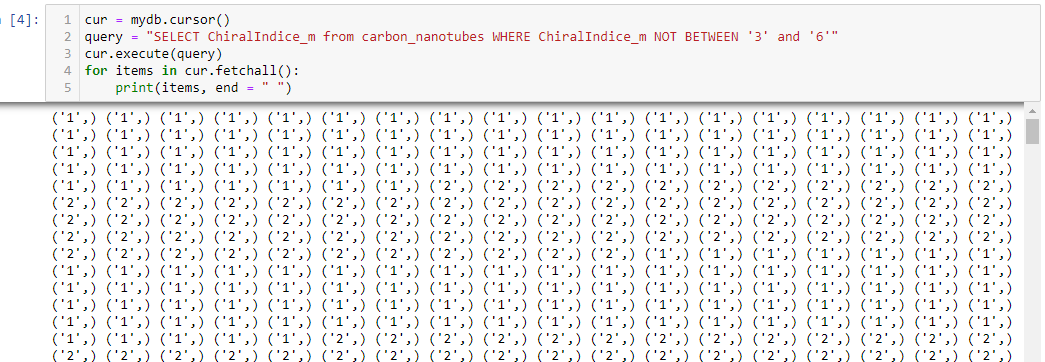
**SELECT ChiralIndice\_m from carbon\_nanotubes WHERE ChiralIndice\_m BETWEEN '5' and '6'**



**SELECT ChiralIndice\_m from carbon\_nanotubes WHERE ChiralIndice\_m NOT BETWEEN '5' and '6'**

**NOT BETWEEN**

**SELECT ChiralIndice\_m from carbon\_nanotubes WHERE ChiralIndice\_m NOT BETWEEN '5' and '6'**



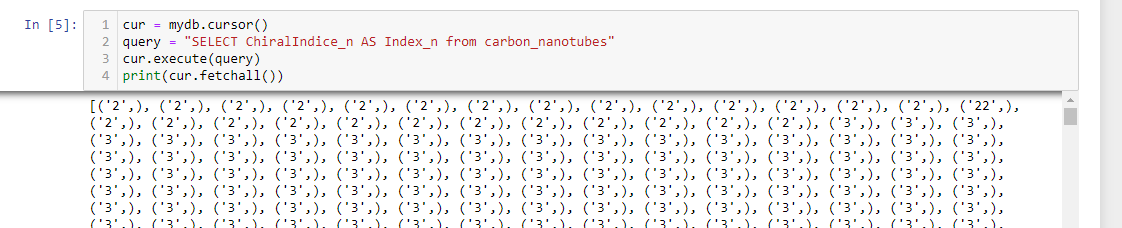
## The SQL ALIAS

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 ChiralIndice\_n AS Index\_n from carbon\_nanotubes

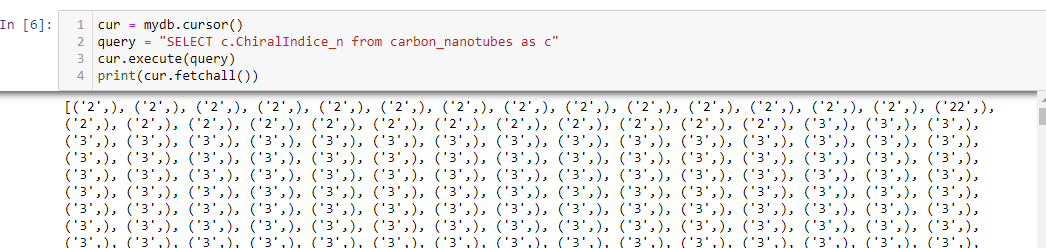
**Creates aliase for the ChiralIndice :**



**Alias Table Syntax**

**SELECT c.ChiralIndice\_n from carbon\_nanotubes as c**

**Create an alias for the carnon\_nanotubes table:**

****

## 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

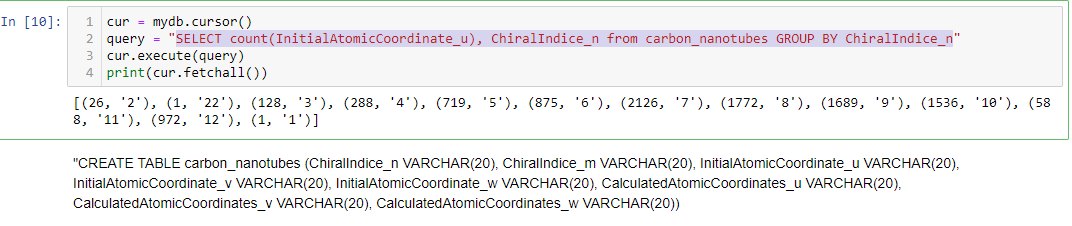
WHERE condition

GROUP BY column\_name(s)

ORDER BY column\_name(s);

**Count the number of InitialAtomicCoordinate\_u group by ChiralIndice\_n:**

**SELECT count(InitialAtomicCoordinate\_u), ChiralIndice\_n from carbon\_nanotubes GROUP BY ChiralIndice\_n**

****

## 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

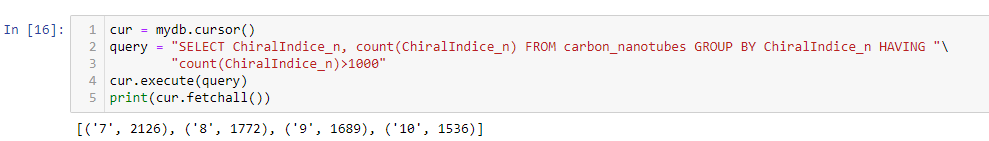
GROUP BY column\_name(s)

HAVING condition

ORDER BY column\_name(s);

**List the ChiralIndice\_n which has total count more than 1000.**

**SELECT ChiralIndice\_n, count(ChiralIndice\_n) FROM carbon\_nanotubes GROUP BY ChiralIndice\_n HAVING count(ChiralIndice\_n)>1000**



## 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.

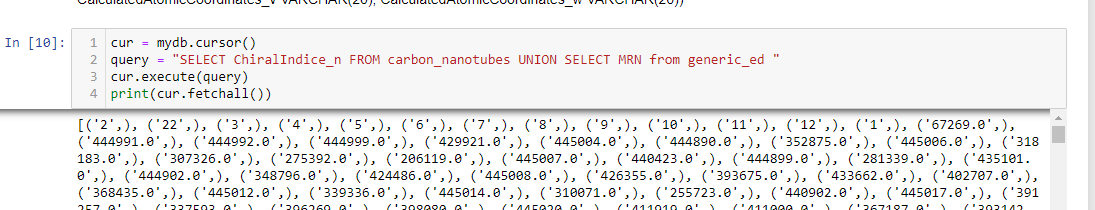
**The SQL UNION**

Select column\_name(s) from table1

UNION

Select column\_name(s) from table2;

**SELECT ChiralIndice\_n FROM carbon\_nanotubes UNION SELECT MRN from generic\_ed**



**UNION ALL Query**

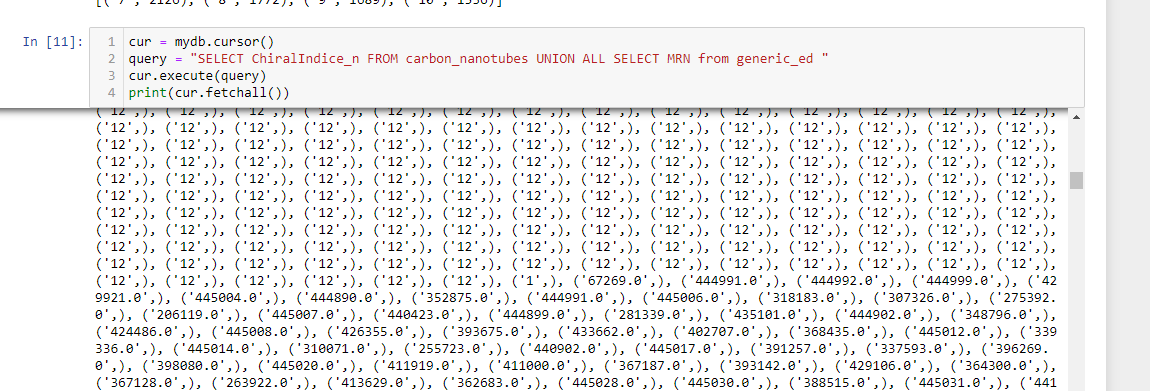
The UNION operator selects only different values by default. To allow duplicate values, the user can use UNION ALL operator.

SELECT column\_name(s) FROM table1

UNION ALL

SELECT column\_name(s) FROM table2;

**SELECT ChiralIndice\_n FROM carbon\_nanotubes UNION ALL SELECT MRN from generic\_ed**



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

## 

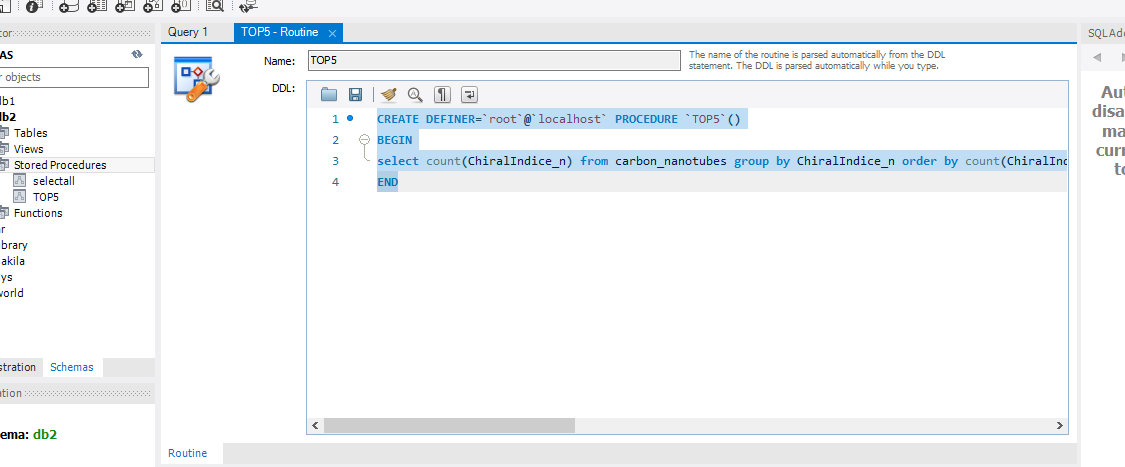
## The SQL STORED PROCEDURE

**What is a SQL Stored Procedure?**

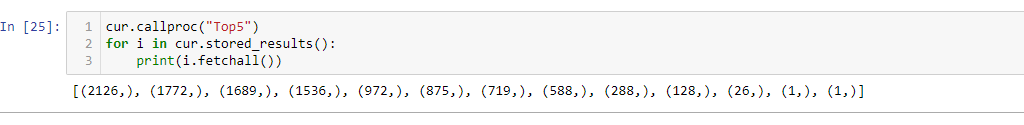
The **stored procedure** is a prepared SQL query that you can save so that the query can be **reused** over and over again. So, if the user has an SQL query that you write over and over again, keep it as a stored procedure and execute it. Users can also pass parameters to a stored procedure so that the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

**CREATE A PROCEDURE**

**select count(ChiralIndice\_n) from carbon\_nanotubes group by ChiralIndice\_n order by count(ChiralIndice\_n) DESC**

****

**Execute a Stored Procedure**



# 

# 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**

## 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.

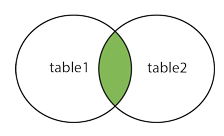


Fig. INNER JOIN

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

**INNER JOIN Syntax**

SELECT column\_name(s)

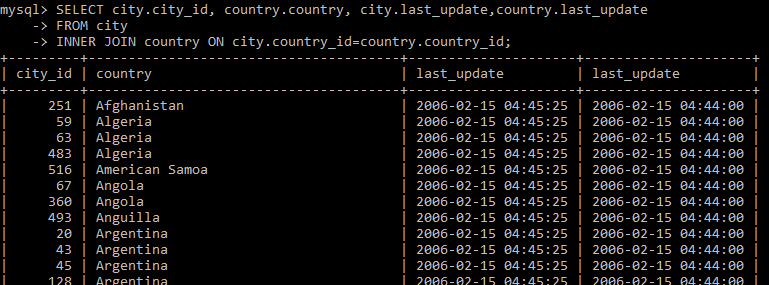
FROM table1

INNER JOIN table2

ON table1.column\_name = table2.column\_name;

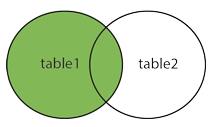
SELECT city.city\_id, country.country, city.last\_update, country.last\_update FROM city

INNER JOIN country ON city.country\_id = country.country\_id



## 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)

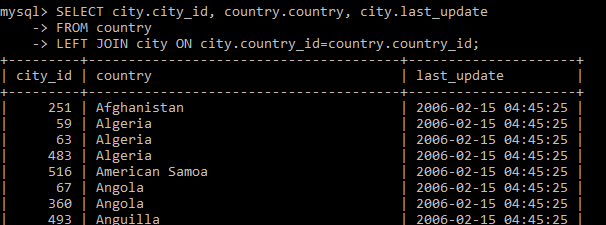
FROM table1

LEFT JOIN table2

ON table1.column\_name = table2.column\_name;

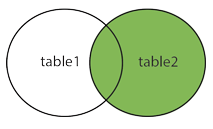
SELECT city.city\_id, country.country, city.last\_update, country.last\_update FROM city

LEFT JOIN country ON city.country\_id = country.country\_id



## 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**

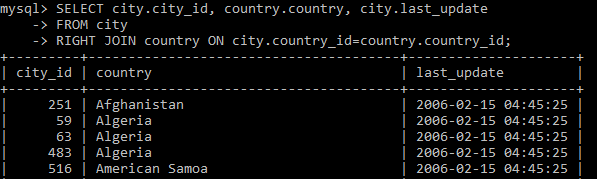
SELECT column\_name(s)

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

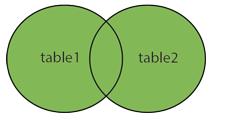
RIGHT JOIN country ON city.country\_id = country.country\_id



## 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

## SELF-JOIN

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

**Self -JOIN Syntax**

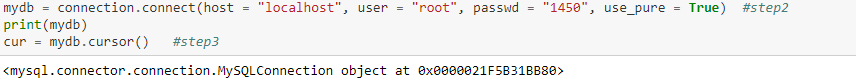
SELECT column\_name(s)

FROM table1 T1, table1 T2

WHERE condition;

# SQL DATABASE

Establish the connection



## The SQL CREATE DATABASE STATEMENT

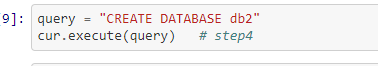
The CREATE DATABASE statement in SQL is used to create a new SQL database.

**Syntax**

CREATE DATABASE database\_name;

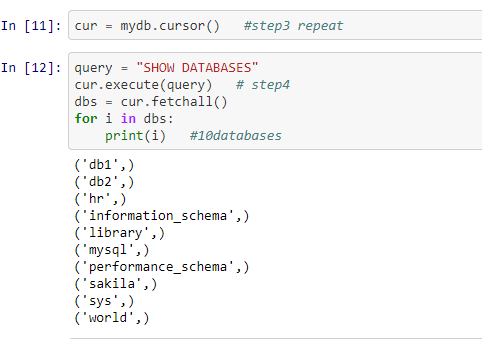
**Let’s create a database and give name as db2**

CREATE database db2;



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

Show databases;



## The SQL DROP DATABASE STATEMENT

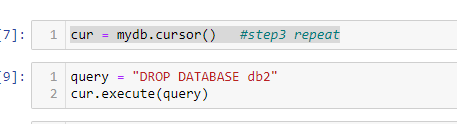
The DROP DATABASE statement in SQL is used to drop an existing SQL database.

**Syntax**

DROP DATABASE database\_name;

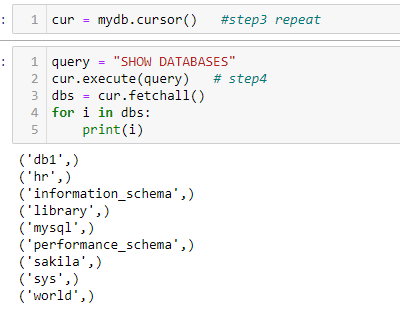
**Let’s drop the created database by using drop database testdb.**

DROP database db2;



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

SHOW databases;



The created database(db2) has been dropped.

## [The SQL CREATE TABLE](#TheSQLCREATETABLE)

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,

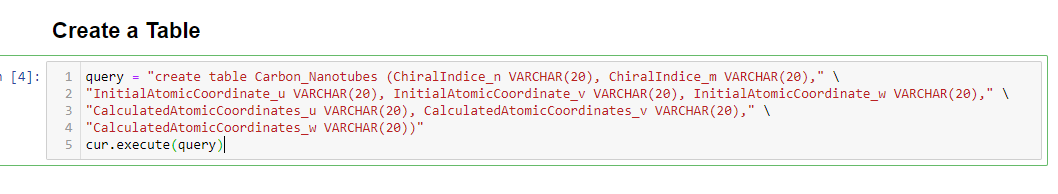
....

);

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.)

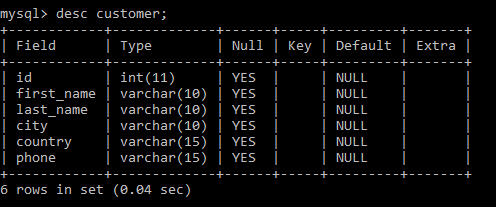
**Let’s create a carbon\_nanotubes table**

**CREATE TABLE carbon\_nanotubes (ChiralIndice\_n VARCHAR(20), ChiralIndice\_m VARCHAR(20), InitialAtomicCoordinate\_u VARCHAR(20), InitialAtomicCoordinate\_v VARCHAR(20), InitialAtomicCoordinate\_w VARCHAR(20), CalculatedAtomicCoordinates\_u VARCHAR(20), CalculatedAtomicCoordinates\_v VARCHAR(20), CalculatedAtomicCoordinates\_w VARCHAR(20))**

****

**To check the schema of the table, use desc table\_name.**

DESC customer;

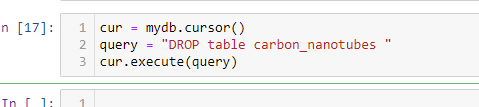


## 

## The SQL DROP TABLE STATEMENT

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

DROP TABLE carbon\_nanotubes;



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.

[TheSQLINSERTINTOSTATEMENT](#TheSQLINSERTINTOSTATEMENT)

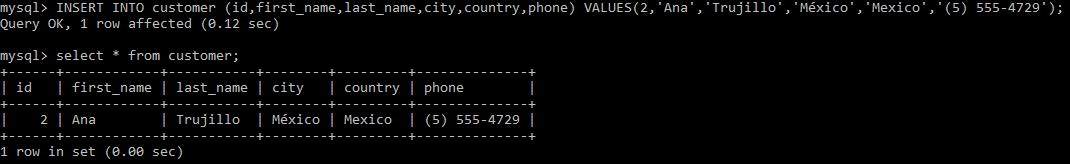
## The SQL INSERT INTO STATEMENT

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);



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**:

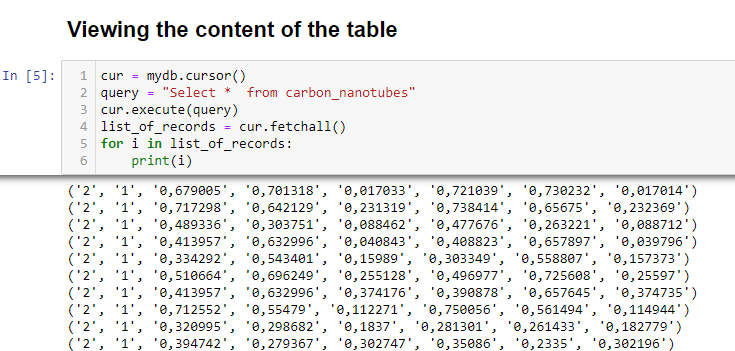
* filename from where data has to be imported = carbon\_nanotubes.csv
* number fields/attributes present in file = 8
* since it is downloaded and it is a raw data it can have missing values (assuming all information present is true)
* it is csv file we need a csv reader - we will install and import csv module
* There are 10000+ Records

query = "INSERT INTO Carbon\_Nanotubes VALUES(%s,%s,%s,%s,%s,%s,%s,%s)"



Finally, we have added many rows as we can see in the picture below.

SELECT \* FROM carbon\_nanotubes



## The SQL NULL VALUES

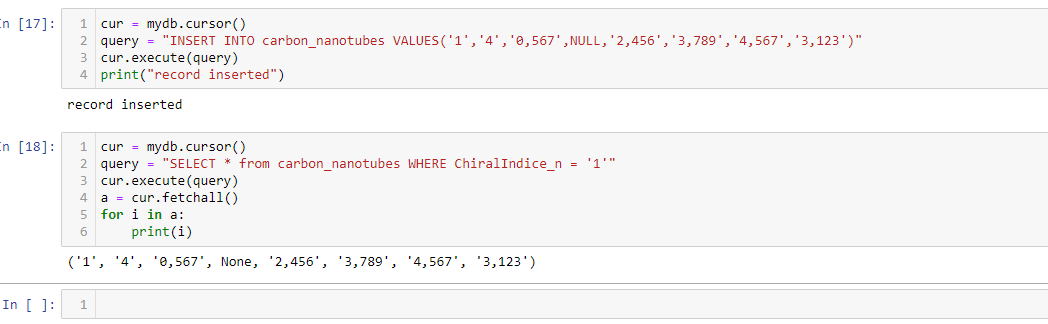
**What is a NULL Value?**

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 carbon\_nanotubes VALUES('1','4','0,567',NULL,'2,456','3,789','4,567','3,123')**

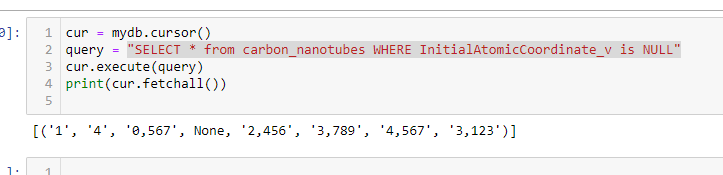
****

**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.

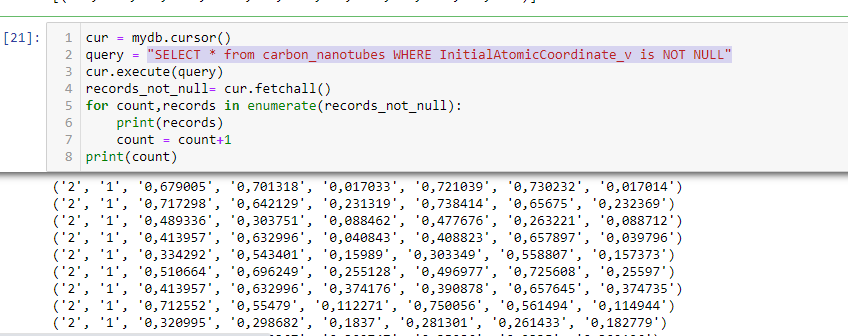
**IS NULL Syntax**

**"SELECT \* from carbon\_nanotubes WHERE InitialAtomicCoordinate\_v is NULL"**



**IS NOT NULL Syntax**

SELECT \* from carbon\_nanotubes WHERE InitialAtomicCoordinate\_v is NOT NULL



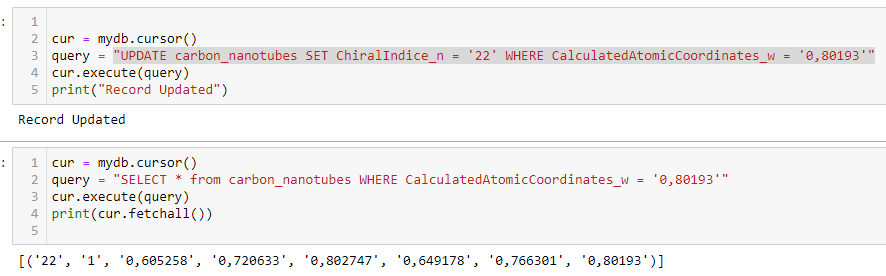
It will return those records which have some values(expect Null values).

## The SQL UPDATE STATEMENT

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

**UPDATE Syntax**

**UPDATE carbon\_nanotubes SET ChiralIndice\_n = '22' WHERE CalculatedAtomicCoordinates\_w = '0,80193'**

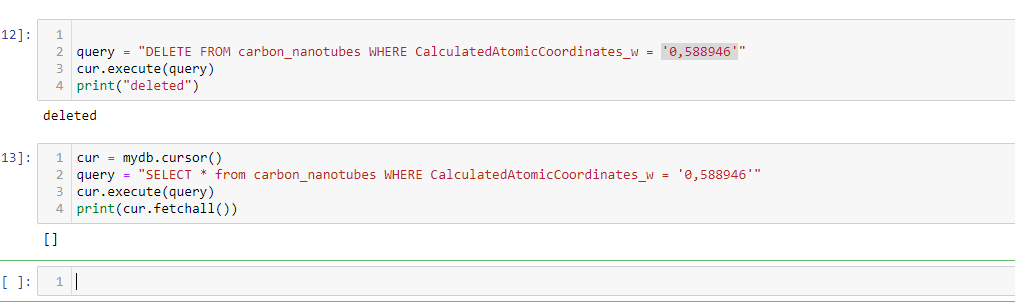
****

## The SQL DELETE STATEMENT

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

**DELETE Syntax**

**DELETE FROM carbon\_nanotubes WHERE CalculatedAtomicCoordinates\_w = '0,588946'**



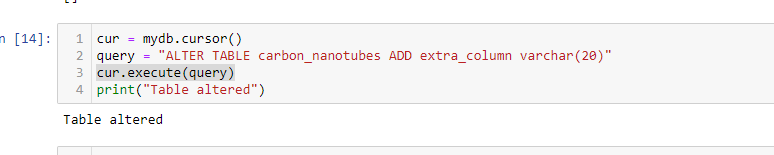
We have deleted one row, which contains CalculatedAtomicCoordinates\_w = '0,588946'

## 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.

### ALTER TABLE - ADD COLUMN IN EXISTING TABLE

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

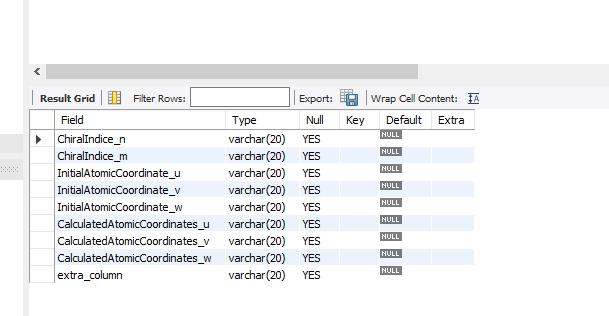
ALTER TABLE carbon\_nanotubes ADD extra\_column varchar(20)

### ALTER TABLE – MODIFY/ALTER COLUMN

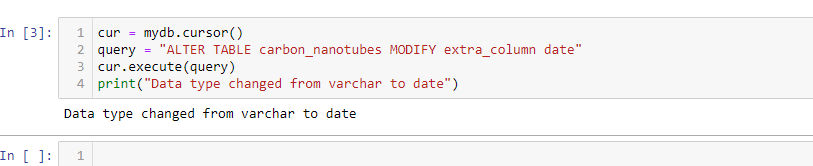
To change the data type of column values in a table, use the following syntax:

We have assigned the extra\_column with the datatype varchar. But now we want to change the datatype from varchar to date.

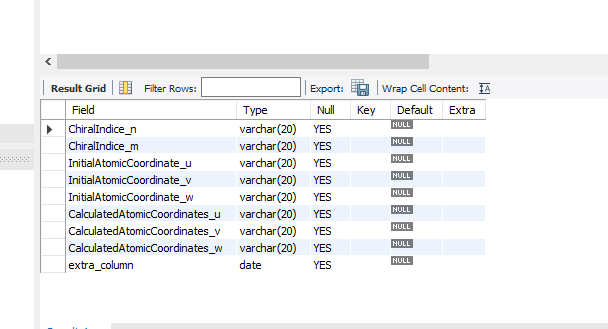
MYSQL SNAP:



ALTER TABLE carbon\_nanotubes MODIFY extra\_column date



MYSQL SNAP:



### 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 carbon\_nanotubes DROP COLUMN extra\_column "**

