Python MySQL

> **Python MySQL Connector:** Driver that helps to integrate Python and MySQL database.

> **Install MySQL Connector using PIP:** Use the following command:

```
pip install mysql-connector-python
```

> **Import mysql Module:**

```
import mysql.connector
```

Connect to Server

> **Connect to the Database Server:** We can connect to MySQL server using the **connect()** method from mysql.connector module.

> **Syntax:**

```
import mysql.connector
my_db = mysql.connector.connect(
  host = 'host_name',
  user = 'user_name',
  password = 'your_password',
  database = 'database_name'
)
```

```
import mysql.connector

mydb = mysql.connector.connect(
    host="host_name",
    user="user_name",
    password="your_password",
    database="database_name"
)

print(mydb)
```

CREATE Database and Table

> **Create a database:** First, have to create cursor object & pass SQL commands to execute() method, as string arguments.

> SQL command to create database :

```
CREATE DATABASE database_name
```

```
# Preparing cursor object
mycursor = mydb.cursor()

mycursor.execute("CREATE DATABASE py_db")
```

> **List databases in the system:**

- SQL Query to list database names:

```
SHOW DATABASES
```

```
mycursor = mydb.cursor()

mycursor.execute("SHOW DATABASES")

for x in mycursor:
  print(x)
```

> **USE database:**

- SQL query to select database:

```
USE database_name
```

```
use_q = 'USE py_db'

mycursor.execute(use_q)
```

> **CREATE table:**

- SQL query to Create database:

```
CREATE TABLE (
  col_1_name col_1_data_type,
          col_2_name col_2_data_type,
          .
          .
          .
          col_n_name col_n_data_type,
        )
```

```
create_q = 'CREATE TABLE emp(id INT AUTO_INCREMENT PRIMARY KEY, name VARCHAR(20), addr VARCHAR(20))'
```

```
mycursor.execute(create_q)
```

```
mycursor.execute('SHOW TABLES')
```

```
mycursor.execute('DESC emp')
```

SELECT

> **SELECT Query:**

- SQL Query to select all data from the table:

```
    SELECT * FROM table_name;
```

```
mycursor.execute('SELECT * FROM emp')

for x in mycursor:
  print(x)
```

> **db_obj.fetchall():** Method fetches all the records from the table.

```
mycursor.execute('SELECT * FROM emp')

myresult = mycursor.fetchall()

for x in myresult:
  print(x)
```

> **db_obj.fetchone():** Method fetches first row of the result obtained.

```
mycursor.execute('SELECT * FROM emp')

myresult = mycursor.fetchone()

for x in myresult:
  print(x)
```

> **Selecting particular columns from the table:**

- To select only some of the columns in a table, use the "SELECT" statement followed by the column_names_list to be selected.
- SQL Query to select columns:

```
SELECT col_names_list FROM table_name;
```

```
mycursor.execute('SELECT id, name FROM emp')

for x in mycursor:
  print(x)
```

> **WHERE Clause:** Used to select records by applying filter.

- When selecting records from a table, you can filter selection by using the "WHERE" statement:

```
SELECT * FROM table_name WHERE <condition to be applied>;
```

```
mycursor.execute('SELECT id, name FROM emp WHERE id = 5')
print(mycursor)
```

```
mycursor.execute('SELECT id, name FROM emp WHERE name = "kop"')
print(mycursor)
```

> **Wildcard Character:** Can select records that starts, includes or ends given letter or phrase.

```
sql = "SELECT * FROM emp WHERE name LIKE 's%'"
mycursor.execute(sql)

for x in mycursor:
  print(x)
```

> **SQL Injections:**

```
sql = "SELECT * FROM customers WHERE address = %s"
adr = ("Yellow Garden 2", )

mycursor.execute(sql, adr)

myresult = mycursor.fetchall()

for x in myresult:
  print(x)
```

INSERT

> **Note:** my_db.commit() is required to make permanent changes, otherwise changes are not reflected permanently to the database table.

> **Simple Query to INSERT INTO the table:**

- SQL Query to insert data into table using the **INSERT INTO** Query:

```
INSERT INTO table_name (col_names_list) VALUES (data)
```

```
q = 'INSERT INTO emp(id, name, addr) VALUES(1, "snehal", "kop")'
mycursor.execute(q)

# Number of rows inserted
print(mycursor.rowcount)

# Id of the row inserted
print(mycursor.lastrowid)
```

> **INSERT INTO table:**

```
sql = "INSERT INTO emp (id, name, addr) VALUES (%s, %s, %s)"

val = (2, 'shiv', 'kop')

mycursor.execute(sql, val)

mydb.commit()

print(mycursor.rowcount, "record inserted.")
```

> **INSERT multiple values from the list:**

```
vals = [
    (3, 'shubh', 'panhala'),
    (4, 'kal', 'kop'),
    (5, 'mankar', 'sarud')
]

mycursor.executemany(sql, vals)

mydb.commit()

print(mycursor.rowcount, "record inserted.")
```

Sort

- We can use "ORDER BY" statement to sort result in ascending / descending order.
- **ORDER BY** sorts result in ascending order, by default.
- SQL Query to select results using "ORDER BY" Clause:

```
SELECT * FROM table_name ORDER BY col_name
```

> **Sort the Result:**

```
sql = "SELECT * FROM customers ORDER BY age"

mycursor.execute(sql)

myresult = mycursor.fetchall()

for x in myresult:
  print(x)
```

> **ORDER BY DESC:**

- We can use **DESC** keyword, to sort the results in descending order.
- SQL Query to select results using "ORDER BY DESC" Clause:

```
SELECT * FROM table_name ORDER BY col_name DESC
```

```
sql = "SELECT * FROM customers ORDER BY age DESC"

mycursor.execute(sql)

myresult = mycursor.fetchall()

for x in myresult:
  print(x)
```

DELETE

> **DELETE a Record:** We can delete selected records from the table using "WHERE" clause.

- SQL Query to delete selective records from the table:

```
          DELETE FROM <table_name> WHERE <condition>;
```

```
del_q = 'DELETE FROM emp WHERE id = 4'
mycursor.execute(del_q)
mydb.commit()

print(mycursor.rowcount, "record(s) deleted")
```

> **DELETE all the Records from Table:** If we not specify any condition, it deletes all the records from the table.

- SQL Query to delete all records from the table:

```
        DELETE FROM <table_name> WHERE <condition>;
```

```
del_q = 'DELETE FROM emp'
mycursor.execute(del_q)
mydb.commit()

print(mycursor.rowcount, "record(s) deleted")
```

> **DROP a table:**

- Delete the table from database using the following SQL Command:

```
        DROP TABLE <table_name>
```

```
sql = "DROP TABLE emp"

mycursor.execute(sql)
```

> **Drop Table Only if Exist:**

- SQL Query to delete table, only if table exist:

```
        DROP TABLE IF EXISTS <table_name>
```

```
sql = "DROP TABLE IF EXISTS emp"
mycursor.execute(sql)
```

UPDATE

> **Update Table:**

- We can update selective records using SET keyword.
- SQL Query to Update records:

```
UPDATE <table_name> SET <col_name> = <val> WHERE <condition>;
```

```
sql = 'UPDATE emp SET addr="pune" WHERE addr="kop"'
mydb.commit()
mycursor.execute(sql)
print(mycursor.rowcount, "record(s) updated..")
```

> **Update table using SLQ Injection:**

```
sql = 'UPDATE emp SET addr=%s WHERE addr=%s'
val = ('pune','kop')

mycursor.execute(sql)
print(mycursor.rowcount, "record(s) updated..")
```

Other SQL Queries

> **Limit the Result:** Can limit number of records returned from the query.

- SQL Query to Limit results:

```
SELECT <col_names_list> FROM <table_name> LIMIT <num_of_records_to_fetch>
```

```
mycursor.execute("SELECT * FROM customers LIMIT 5")
myresult = mycursor.fetchall()

for x in myresult:
  print(x)
```

> **Start From Another Position:** OFFSET keyword can be used to start position from a different record.

- SQL Query for OFFSET:

```
SELECT <col_names_list> FROM <table_name> OFFSET <start_index>;
```

```
mycursor.execute("SELECT * FROM emp LIMIT 5 OFFSET 2")

myresult = mycursor.fetchall()

for x in myresult:
  print(x)
```

Disconnect from Server

> **Disconnecting from Database Server:** We have to disconnect from the database server, so that have to release the resources.

```
mydb.close()
```

Joins

- **JOIN** operation is performed on 2 or more tables, to fetch their records simultaneously.
- SQL Query for JOIN:

```
    SELECT <col_names_list> AS <alias_names> FROM <table_1_name> JOIN <table_2_name>
            ON <table_1_col_name> = <table_2_col_name>
```

```
# Create another table called skills
sql = 'CREATE TABLE skills(id int primary key auto_increment, name varchar(20), skill varchar(20))'
mycursor.execute(sql)


# Insert into the new table
sql = 'INSERT INTO skills(id, name, skill, emp_id) VALUES (%s, %s, %s, %s)'
vals = [
    (1, 'snehal', 'C', 1),
    (2, 'shubh', 'C++', 2),
    (3, 'snehal', 'Java', 1),
    (4, 'shubh', 'PHP', 2),
    (5, 'kal', 'Angular', 4),
    (6, 'snehal', 'Git', 1),
    (7, 'kal', 'Python', 4)
]
mycursor.executemany(sql, vals)
mydb.commit()
print(mycursor.rowcount, "record(s) inserted..")
```

> **INNER Join:** Fetching all the records which are matching in both the tables.

- SQL Query for JOIN:

```
SELECT <col_names_list> AS <alias_names> FROM <table_1_name> INNER JOIN <table_2_name>
        ON <table_1_col_name> = <table_2_col_name>
```

```
sql = "SELECT \
  skills.id as id, \
  emp.id AS emp_id, \
  emp.name AS emp_name, \
  skills.skill as skill \
  FROM emp \
  INNER JOIN skills ON emp.id = skills.emp_id"
```

> **LEFT Join:** Fetch all the records from left table & only matching entries from the right table.

- SQL Query for JOIN:

```
SELECT <col_names_list> AS <alias_names> FROM <table_1_name> LEFT JOIN <table_2_name>
        ON <table_1_col_name> = <table_2_col_name>
```

```
sql = "SELECT \
  skills.id as id, \
  emp.id AS emp_id, \
  emp.name AS emp_name, \
  skills.skill as skill \
  FROM emp \
  LEFT JOIN skills ON emp.id = skills.emp_id"
```

> **RIGHT Join:** Fetch all the records from right table & only matching entries from the left table.

- SQL Query for JOIN:

```
SELECT <col_names_list> AS <alias_names> FROM <table_1_name> RIGHT JOIN <table_2_name>
        ON <table_1_col_name> = <table_2_col_name>
```

```
sql = "SELECT \
  skills.id as id, \
  emp.id AS emp_id, \
  emp.name AS emp_name, \
  skills.skill as skill \
  FROM emp \
  RIGHT JOIN skills ON emp.id = skills.emp_id"
```