



How to use Python with MySQL



David Cheah May 28, 2019 · 7 min read



What is Python? Well, it is a high-level and general-purpose programming language that can be used for web development, data science and scripting. According to TIOBE index, Python is ranked top 10 most popular programming language. In this article, I will be sharing how we can use Python to interact with MySQL database.

Python Setup

1. Install Anaconda — <https://www.anaconda.com/distribution/>

2. Select **python version 3** (Version 2 will stop receiving security updates by 2020)

NOTE: Anaconda is a python and R distribution that aims to provide everything you need:

- core python language,
- python packages,
- IDE/editor — Jupyter and Spyder
- Package manager — Conda, for managing your packages)

MySql Setup

1. Download and install **MySql Community Server** — <https://dev.mysql.com/downloads/mysql/>
2. During the installation setup, you will be prompted for a “root” password in the server configuration step.
3. Download and install **MySql Workbench** — <https://dev.mysql.com/downloads/workbench/>
4. Workbench is GUI tool that allows us to manage your MySql database.
5. Launch workbench, at the home page, setup a new connection profile with the configuration (Connection method: Standard (TCP/IP), Hostname: 127.0.0.1, Port: 3306, Username: root, Password: *yourpassword*) and test your connection.
6. Double click on your local instance and it should bring you the schemas view where you can see all your databases and tables.

MySql-Connector-Python Setup

1. This is the python driver for connecting to MySql server.
2. Using terminal and **conda** to download

```
conda install -c anaconda mysql-connector-python
```

Connect to MySql

1. Launch Anaconda-Navigator to bring you a list of applications available to install. The application we are interested in is **Jupyter Notebook** which is a web-based python IDE. Install and launch it.
2. In the notebook, create a new python file. In the first cell, write the following code to test the mysql connection.

```
import mysql.connector

mydb = mysql.connector.connect (
    host="localhost",
    user="root",
    passwd="password"
)

print (mydb)
```

3. If successful, you should get an object returned with its memory address

```
<mysql.connector.connection_cext.CMySQLConnection object at
0x10b4e1320>
```

Create Database

1. Let's create a database called *mydatabase*.

```
import mysql.connector

mydb = mysql.connector.connect (
```

```

        host="localhost",
        user="root",
        passwd="password"
    )

mycursor = mydb.cursor()

mycursor.execute("CREATE DATABASE mydatabase")

```

2. Next, we will try to connect to this new database.

```

import mysql.connector

mydb = mysql.connector.connect(
    host="localhost",
    user="root",
    passwd="password",
    database="mydatabase"
)

```

Create Table

1. In our new database, let's create a new table called customer to hold information related to your customers.
2. In the sql below, we are creating a table with title customers with 3 columns of title id, name and address. Column id stores data of type integer denoted by **INT**. Column id is meant to store unique key for each record. This is done using the **PRIMARY KEY**. It should be also incremented as new record is added by **AUTO_INCREMENT**.
3. For column name and address, it stores data of type character of maximum 255, denoted by **VARCHAR(255)**.

```

import mysql.connector

mydb = mysql.connector.connect(
    host="localhost",
    user="root",
    passwd="password",
    database="mydatabase"
)

```

```
)

mycursor = mydb.cursor()

mycursor.execute("CREATE TABLE customers (id INT AUTO_INCREMENT
PRIMARY KEY, name VARCHAR(255), address VARCHAR(255))")
```

Add record into table

1. Let's add some data into our new table.

```
import mysql.connector

mydb = mysql.connector.connect(
    host="localhost",
    user="root",
    passwd="password",
    database="mydatabase"
)

mycursor = mydb.cursor()

sql = "INSERT INTO customers (name, address) VALUES (%s, %s)"

val = ("David", "California")

mycursor.execute(sql, val)

mydb.commit()

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

2. To insert multiple data, group the data within square brackets, separated by comma. Replace `execute()` with `executemany()`.

```
import mysql.connector

mydb = mysql.connector.connect(
    host="localhost",
    user="root",
    passwd="password",
    database="mydatabase"
)
```

```

mycursor = mydb.cursor()

sql = "INSERT INTO customers (name, address) VALUES (%s, %s)"

val = [
    ("Lily", "California"),
    ("David", "San Francisco"),
    ("Micheal", "Las Vegas"),
    ("Sarah", "New York")
]

mycursor.executemany(sql, val)

mydb.commit()

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

```

Select record from table

1. To fetch all data from table, use **SELECT *** , whereby asterisks means all.

```

import mysql.connector

mydb = mysql.connector.connect(
    host="localhost",
    user="root",
    passwd="password",
    database="mydatabase"
)

mycursor = mydb.cursor()

sql = "SELECT * FROM customers"

mycursor.execute(sql)

myresult = mycursor.fetchall()

for x in myresult:
    print(x)

```

2. To select specific column, use **SELECT <column_title>** instead of *. For

example `SELECT name FROM customers .`

3. Instead of `fetchall()`, we can use `fetchone()` to retrieve first row of result.

4. We can apply filter to our search using **WHERE** keyword. NOTE: Notice how the query value is stored in separate variable. This is a technique called ***parameterized queries*** to prevent SQL injection.

```
import mysql.connector

mydb = mysql.connector.connect (
    host="localhost",
    user="root",
    passwd="password",
    database="mydatabase"
)

mycursor = mydb.cursor()

sql = "SELECT * FROM customers WHERE address = %s"
adr = ("California", )

mycursor.execute(sql, adr)

myresult = mycursor.fetchall()

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

5. We can also limit the number of search result by adding **LIMIT** .

```
import mysql.connector

mydb = mysql.connector.connect (
    host="localhost",
    user="root",
    passwd="password",
    database="mydatabase"
)
```

```
mycursor = mydb.cursor()

mycursor.execute("SELECT * FROM customers LIMIT 5")

myresult = mycursor.fetchall()

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

Sort Records

1. We can retrieve data from table in ascending order. To make it in descending order, add **DESC** after name.

```
import mysql.connector

mydb = mysql.connector.connect(
    host="localhost",
    user="root",
    passwd="password",
    database="mydatabase"
)

mycursor = mydb.cursor()

sql = "SELECT * FROM customers WHERE address = %s ORDER by name"
adr = ("California", )

mycursor.execute(sql, adr)

myresult = mycursor.fetchall()

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

Delete a record

```
import mysql.connector

mydb = mysql.connector.connect(
```



```

    host="localhost",
    user="root",
    passwd="password",
    database="mydatabase"
)

mycursor = mydb.cursor()

sql = "DELETE FROM customers WHERE address = %s"
adr = ("California", )

mycursor.execute(sql, adr)
mydb.commit()
print(mycursor.rowcount, "record(s) deleted.")

```

Drop a table

```

import mysql.connector

mydb = mysql.connector.connect(
    host="localhost",
    user="root",
    passwd="password",
    database="mydatabase"
)

mycursor = mydb.cursor()

sql = "DROP TABLE customers"

mycursor.execute(sql)

```

Update Record

```

import mysql.connector

mydb = mysql.connector.connect(
    host="localhost",
    user="root",
    passwd="password",
    database="mydatabase"
)

```

```

mycursor = mydb.cursor()

sql = "UPDATE customers SET address = %s WHERE name = %s"
val = ("California", "John", )
mycursor.execute(sql, val)

mydb.commit()
print(mycursor.rowcount, "record(s) updated.")

```

Join Table

1. Let's create 2 more tables, one is called ***products*** storing information such as product_id, name of product and their prices. Another table would be ***orders***, storing all orders with each containing order_id, product_id and customer_id. The reason why we do not bound order_id with product name or customer name is because these values might change in future and to prevent updating value in multiple tables, we usually stick to value that is constant, in this case their index number.

2. Create ***products*** table

```

import mysql.connector
mydb = mysql.connector.connect(
    host="localhost",
    user="root",
    passwd="password",
    database="mydatabase"
)
mycursor = mydb.cursor()

mycursor = mydb.cursor()

sql = "CREATE TABLE products (id INT AUTO_INCREMENT PRIMARY KEY, name VARCHAR(255), price VARCHAR(255))"

mycursor.execute(sql)

sql = "INSERT INTO products (name, price) VALUES (%s, %s)"
val = [
    ("macbook", "2000"),
    ("iphone", "1000"),
    ("apple watch", "500")
]

```

```

]
mycursor.executemany(sql, val)
mydb.commit()
print(mycursor.rowcount, "record(s) inserted.")

```

3. Create *orders* table

```

import mysql.connector
mydb = mysql.connector.connect(
    host="localhost",
    user="root",
    passwd="password",
    database="mydatabase"
)
mycursor = mydb.cursor()

mycursor = mydb.cursor()

sql = "CREATE TABLE orders (id INT AUTO_INCREMENT PRIMARY KEY,
customer_id INT, product_id INT)"

mycursor.execute(sql)

sql = "INSERT INTO orders (customer_id, product_id) VALUES (%s, %s)"
val = [
    ("1", "1"),
    ("1", "2"),
    ("2", "3"),
    ("3", "3")
]
mycursor.executemany(sql, val)
mydb.commit()
print(mycursor.rowcount, "record(s) inserted.")

```

4. Next we would like to join 2 tables. There are 2 types of JOIN query which are the INNER JOIN and OUTER JOIN. The former only shows results in which all criteria of joining are met meaning showing records with common column.

NOTE: JOIN keyword by default means INNER JOIN, hence you can use JOIN or INNER JOIN.

5. We are going to show a list of all customers with their respective order ids. The relationship between customer and their orders are stored in orders table. To do this, we are to show customer name and order id from a joined table of customers and orders with criteria on matching customer id.

NOTE: You can add an alias to each table and reference column name from alias.

```
import mysql.connector

mydb = mysql.connector.connect (
    host="localhost",
    user="root",
    passwd="password",
    database="mydatabase"
)

mycursor = mydb.cursor()

sql = "SELECT \
      c.name, o.id\
      FROM customers c\
      JOIN orders o ON c.id = o.customer_id\
      "

mycursor.execute(sql)

myresult = mycursor.fetchall()

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

6. Below shows the result of using JOIN. Notice the table does not show the full list of customers because some customers do not have any order id.

```
('Lily', 1)
('Lily', 2)
('David', 3)
('Micheal', 4)
```

7. To demonstrate OUTER JOIN, this time we are going to join orders to customers table but we are going to SELECT FROM customers instead of orders.

```
import mysql.connector

mydb = mysql.connector.connect(
    host="localhost",
    user="root",
    passwd="password",
    database="mydatabase"
)

mycursor = mydb.cursor()

sql = "SELECT \
      c.name, o.id\
      FROM customers c\
      LEFT JOIN orders o ON c.id = o.customer_id\
      "

mycursor.execute(sql)

myresult = mycursor.fetchall()

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

8. Below shows the result using LEFT JOIN. Notice now we have “Sarah” with no order id in the result. That is the function of LEFT JOIN, meaning to show the full records of the LEFT table (customer) irrespective if the criteria is met. RIGHT JOIN does the same thing except it shows the full records of the right table.

```
('Lily', 1)
('Lily', 2)
('David', 3)
('Micheal', 4)
('Sarah', None)
```

9. Now, let's go back to INNER JOIN but with multiple tables. Since we would like to show a table of orders, containing order id, customer name, product name and product price, our SELECT query should be FROM orders table. Next we would like to join orders with products (by finding matching product id) and customers table (by finding matching customer id).

```
import mysql.connector

mydb = mysql.connector.connect(
    host="localhost",
    user="root",
    passwd="password",
    database="mydatabase"
)

mycursor = mydb.cursor()

sql = "SELECT \
    o.id, c.name, p.name, p.price\
    FROM orders o\
    JOIN products p ON o.product_id = p.id\
    JOIN customers c ON o.customer_id = c.id"


mycursor.execute(sql)

myresult = mycursor.fetchall()

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

10. Below are the results from the JOIN query with multiple tables.

```
(1, 'Lily', 'macbook', '2000')
(2, 'Lily', 'iphone', '1000')
(3, 'David', 'apple watch', '500')
(4, 'Micheal', 'apple watch', '500')
```

No rights reserved by the author. 

11. Other queries include

- NATURAL JOIN — automatic joining by the database

Python MySQL

- CROSS JOIN — list of all possible combinations
- UNION — combination of 2 or more queries.

References

About

Help

Legal

- How SQL injection works — <https://tableplus.io/blog/2018/08/sql-injection-attack-explained-with-example.html>
- SQL Injection Prevention Cheat Sheet — https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.md