Python MySQL

# Get Started

Python can be used in database applications.

One of the most popular databases is MySQL.

## Install MySQL Driver

Python needs a MySQL driver to access the MySQL database.

We will use the driver "MySQL Connector".

We recommend that you use PIP to install "MySQL Connector".

PIP is most likely already installed in your Python environment.

* Create a folder name like **MySql**.
* Open this folder in any code editor like VS Code

Navigate your command line to the location of PIP, and type the following:

C:\S.Maheshwaran\Python\MySql>pip install mysql-connector-python

# **pip**: Python's package installer. It is used to install and manage Python packages.

# **install**: The command to install a package.

# **mysql-connector-python**: The name of the package you want to install. It is the official MySQL driver for Python.

Now you have downloaded and installed a MySQL driver.

## Test MySQL Connector

To test if the installation was successful, or if you already have "MySQL Connector" installed,

* Create a Python file name like **myfirstdb** in MySql folder with the following content

MySql / myfirstdb.py:

import mysql.connector

# **import mysql.connector**: is used in Python to import the mysql.connector

If the above code was executed with no errors, "MySQL Connector" is installed and ready to be used.

## Create Connection

Start by creating a connection to the database.

Use the username and password from your MySQL database:

myfirstdb.py:

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

# **mysql.connector.connect()**: This function establishes a connection to the MySQL server.

# **host="localhost"**: Specifies the hostname of the MySQL server. localhost indicates that the server is running on the same machine.

# **user="root"**: The username to authenticate with the MySQL server. root is the default superuser account.

# **password=""**: The password for the root user. In this case, it is empty, which is common for local development environments but not recommended for production.

Now you can start querying the database using SQL statements.

# Python MySQL Create Database

## Creating a Database

To create a database in MySQL, use the "**CREATE DATABASE**" statement:

create a database named "mydatabase":

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

# **mydb.cursor()**: Creates a cursor object from the connection object mydb. The cursor is used to execute SQL queries and fetch results from the database.  
  
mycursor.execute("CREATE DATABASE mydatabase")

# **mycursor.execute()**: Executes the SQL command provided as a string argument.

# **"CREATE DATABASE mydatabase"**: This SQL command creates a new database named mydatabase.

If the above code was executed with no errors, you have successfully created a database.

## Check if Database Exists

You can check if a database exist by listing all databases in your system by using the "**SHOW DATABASES**" statement:

### Example: 1

Return a list of your system's databases:

import mysql.connector  
  
mydb = mysql.connector.connect(  
  host="localhost",  
  user="root",  
  password=""  
)  
  
mycursor = mydb.cursor()  
  
mycursor.execute("SHOW DATABASES")

# **mycursor.execute()**: Executes the SQL command provided as a string argument.

# **"SHOW DATABASES"**: This SQL command lists all the databases available on the MySQL server.  
  
for x in mycursor:  
  print(x)

# **for x in mycursor**: Iterates over the result set returned by the SHOW DATABASES command.

# **print(x)**: Prints each item in the result set.

### Example: 2

Try connecting to the database "mydatabase":

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

# **database="mydatabase"**: Specifies the database to connect to. This parameter sets the default database for the connection, so you can start executing queries against this database immediately.

If the database does not exist, you will get an error.

# Python MySQL Create Table

## Creating a Table

To create a table in MySQL, use the "CREATE TABLE" statement.

Make sure you define the name of the database when you create the connection

import mysql.connector  
  
mydb = mysql.connector.connect(  
  host="localhost",  
  user="root",  
  password="",  
  database="mydatabase"  
)  
  
mycursor = mydb.cursor()  
  
mycursor.execute("CREATE TABLE customers (name VARCHAR(255), address VARCHAR(255))")  
# **Creating a Table**:

* **CREATE TABLE**: This SQL statement is used to create a new table in the database.
* **customers**: The name of the table being created.

# **Table Schema**:

* **name VARCHAR(255)**: Defines a column named name with a data type of VARCHAR and a maximum length of 255 characters. VARCHAR is used for variable-length strings.
* **address VARCHAR(255)**: Defines a column named address with a data type of VARCHAR and a maximum length of 255 characters.

If the above code was executed with no errors, you have now successfully created a table.

## Check if Table Exists

You can check if a table exist by listing all tables in your database with the "**SHOW TABLES**" statement:

Return a list of your system's databases:

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

# **mycursor.execute("SHOW TABLES")**: Executes the SQL command SHOW TABLES, which retrieves a list of all the tables in the currently selected database.  
  
for x in mycursor:  
  print(x)

## Primary Key

When creating a table, you should also create a column with a unique key for each record.

This can be done by defining a PRIMARY KEY.

We use the statement "**INT AUTO\_INCREMENT PRIMARY KEY**" which will insert a unique number for each record. Starting at 1, and increased by one for each record.

### Example: 1

Create primary key when creating the table:

import mysql.connector  
  
mydb = mysql.connector.connect(  
  host="localhost",  
  user="root",  
  password="",  
  database="mydatabase"  
)  
  
mycursor = mydb.cursor()  
  
mycursor.execute("CREATE TABLE customers (id INT AUTO\_INCREMENT PRIMARY KEY, name VARCHAR(255), address VARCHAR(255))")

‘‘‘

1. **CREATE TABLE customers**:
   * This creates a new table named customers in the selected database.
2. **Column Definitions**:
   * **id INT AUTO\_INCREMENT PRIMARY KEY**:
     + **id**: The name of the column.
     + **INT**: The data type for this column, which is an integer.
     + **AUTO\_INCREMENT**: This automatically generates a unique number for each new record inserted into the table. The first record will get an id of 1, the second will get 2, and so on.
     + **PRIMARY KEY**: This makes the id column the primary key of the table. The primary key uniquely identifies each record in the table, ensuring that each id is unique and not NULL.
   * **name VARCHAR(255)**:
     + **name**: The name of the column.
     + **VARCHAR(255)**: This data type allows storing variable-length strings up to 255 characters.
   * **address VARCHAR(255)**:
     + **address**: The name of the column.
     + **VARCHAR(255)**: This allows storing variable-length strings up to 255 characters for the address.

**’’’**

If the table already exists, use the ALTER TABLE keyword

### Example: 2

Create primary key on an existing table:

import mysql.connector  
  
mydb = mysql.connector.connect(  
  host="localhost",  
  user="root",  
  password="",  
  database="mydatabase"  
)  
  
mycursor = mydb.cursor()  
  
mycursor.execute("ALTER TABLE customers ADD COLUMN id INT AUTO\_INCREMENT PRIMARY KEY")

**‘‘‘**

**ALTER TABLE customers**:

* This part of the command indicates that you are making changes to the existing table named customers.

**ADD COLUMN id INT AUTO\_INCREMENT PRIMARY KEY**:

* **ADD COLUMN**: Specifies that you are adding a new column to the table.
* **id INT**: Adds a column named id with a data type of INT (integer).
* **AUTO\_INCREMENT**: Automatically generates a unique value for each new record. The first row might be 1, the second 2, and so on.
* **PRIMARY KEY**: Sets the id column as the primary key for the table, ensuring that each value in this column is unique and not NULL.

**’’’**

# Python MySQL Insert Into Table

## Insert Into Table

To fill a table in MySQL, use the "INSERT INTO" statement.

Insert a record in the "customers" table:

import mysql.connector  
  
mydb = mysql.connector.connect(  
  host="localhost",  
  user="root",  
  password="",  
  database="mydatabase"  
)  
  
mycursor = mydb.cursor()  
  
sql = "INSERT INTO customers (name, address) VALUES (%s, %s)"

# sql is a string containing the SQL query to insert a new row into the customers table.

# %s placeholders are used for parameterized queries to avoid SQL injection attacks and to handle data safely.  
val = ("Mahesh", "Main road, Tenkasi")

# val is a tuple containing the values to be inserted into the name and address columns of the customers table.

# The values in val correspond to the %s placeholders in the sql string.

mycursor.execute(sql, val)  
# mycursor is a cursor object obtained from a connection to the MySQL database.

# execute() method is used to run the SQL query, with val being passed as the parameters for the placeholders.

**mydb.commit()**

# mydb is the connection object to the MySQL database.

# commit() method is used to save the changes made by the execute() method to the database.

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

# mycursor.rowcount provides the number of rows affected by the last executed query, which should be 1 for a successful insert.

# This line prints a message indicating that one record has been inserted.

**Important!:** Notice the statement: mydb.commit(). It is required to make the changes, otherwise no changes are made to the table.

## Insert Multiple Rows

To insert multiple rows into a table, use the **executemany()** method.

The second parameter of the executemany() method is a list of tuples, containing the data you want to insert:

Fill the "customers" table with data:

import mysql.connector  
  
mydb = mysql.connector.connect(  
  host="localhost",  
  user="root",  
  password="",  
  database="mydatabase"  
)  
  
mycursor = mydb.cursor()  
  
sql = "INSERT INTO customers (name, address) VALUES (%s, %s)"  
val = [  
  ('Gopi', ' MG Road, Chennai'),  
  ('Kannan', 'Anna Nagar, Vellore'),  
  ('Manoj', 'Mount Road, Virudhunagar'),  
  ('Vicky', 'Gandhi Nagar, Madurai'),  
  ('Bala', 'Race Course, Coimbatore'),  
  ('Raj', 'Mahesh Street, Tenkasi'),  
  ('Mani', 'Kamarajar Nagar Nellai'),  
  ('Muruga', 'Anna Salai, Kanyakumari')   
]  
# **val**: This list contains tuples, where each tuple represents a set of values to be inserted into the customers table.

**# Each tuple has two elements**: the first is the name, and the second is the address.  
mycursor.executemany(sql, val)

# **executemany()**: This method executes the given SQL command for each tuple in the val list.

mydb.commit()  
  
print(mycursor.rowcount, "record inserted.")

# **mycursor.rowcount**: This attribute returns the number of rows affected by the last executed operation. In this case, it will return the number of records that were inserted.

# The print statement will output something like 8 record inserted., indicating that eight records were successfully inserted into the customers table.

## Get Inserted ID

You can get the id of the row you just inserted by asking the cursor object.

**Note:** If you insert more than one row, the id of the last inserted row is returned.

Insert one row, and return the ID:

import mysql.connector  
  
mydb = mysql.connector.connect(  
  host="localhost",  
  user="root",  
  password="",  
  database="mydatabase"  
)  
  
mycursor = mydb.cursor()  
  
sql = "INSERT INTO customers (name, address) VALUES (%s, %s)"  
val = ("Siva", "South Street, Namakkal")

mycursor.execute(sql, val)  
  
mydb.commit()  
  
print("1 record inserted, ID:", mycursor.lastrowid)

# **"1 record inserted, ID:"**: This is a string literal that will be printed as is. It informs the user that one record was successfully inserted and will also display the ID of that record.

# **mycursor.lastrowid**: This is an attribute of the mycursor object that returns the ID of the last row that was inserted into the table.

# Python MySQL Select From

## Select From a Table

To select from a table in MySQL, use the "SELECT" statement:

### Example

Select all records from the "customers" table, and display the result:

import mysql.connector  
  
mydb = mysql.connector.connect(  
  host="localhost",  
  user="root",  
  password="",  
  database="mydatabase"  
)  
  
mycursor = mydb.cursor()  
  
mycursor.execute("SELECT \* FROM customers")

# **SELECT \***: The SELECT statement is used to fetch data from a database. The \* wildcard selects all columns from the specified table

# **FROM customers**: Specifies the table from which the data is being retrieved. Here, the data is being fetched from the customers table.  
  
myresult = mycursor.fetchall()

# **myresult**: This variable will store the results of the query.

# **mycursor.fetchall()**: This method retrieves all rows from the result of the executed query and stores them in myresult.

* The data is stored as a list of tuples, where each tuple represents a row from the customers table. Each element of the tuple corresponds to a column in the table.

for x in myresult:  
  print(x)

**‘‘‘**  
**for x in myresult:**: This for loop iterates over each tuple in myresult.

* **x**: In each iteration, x represents one tuple, which corresponds to one row of data from the customers table.
* **print(x)**: This line prints the current tuple (x) to the console. Each tuple contains the values of all the columns for a specific row in the table.

**’’’**

**Note:** We use the fetchall() method, which fetches all rows from the last executed statement.

## Selecting Columns

To select only some of the columns in a table, use the "SELECT" statement followed by the column name(s):

### Example

Select only the name and address columns:

import mysql.connector  
  
mydb = mysql.connector.connect(  
  host="localhost",  
  user="root",  
  password="",  
  database="mydatabase"  
)  
  
mycursor = mydb.cursor()  
  
mycursor.execute("SELECT name, address FROM customers")

# **SELECT**: Specifies that you want to retrieve data from the database.

**# name, address**: Specifies the columns to retrieve. In this case, only the name and address columns will be selected.

**# FROM customers**: Specifies the table from which the data will be retrieved. The table here is customers.  
  
myresult = mycursor.fetchall()  
  
for x in myresult:  
  print(x)

## Using the fetchone() Method

If you are only interested in one row, you can use the fetchone() method.

The fetchone() method will return the first row of the result:

### Example

Fetch only one row:

import mysql.connector  
  
mydb = mysql.connector.connect(  
  host="localhost",  
  user="root",  
  password="",  
  database="mydatabase"  
)  
  
mycursor = mydb.cursor()  
  
mycursor.execute("SELECT \* FROM customers")  
  
myresult = mycursor.fetchone()

# **fetchone()**: is used when you are only interested in retrieving one row from the result set. This is useful if you expect the query to return a single row or if you want to process rows one at a time in a loop.  
  
print(myresult)

# Python MySQL Where

## Select With a Filter

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

### Example

import mysql.connector  
  
mydb = mysql.connector.connect(  
  host="localhost",  
  user="root ",  
  password="",  
  database="mydatabase"  
)  
mycursor = mydb.cursor()  
  
sql = "SELECT \* FROM customers WHERE address ='Anna Salai, Kanyakumari'"

# **WHERE**: The WHERE clause is used to filter records. It specifies the condition that must be met for a row to be included in the result set.

# **address = 'Anna Salai, Kanyakumari'**: This is the condition applied to filter the rows. It checks that the address column exactly matches 'Anna Salai, Kanyakumari'.

mycursor.execute(sql)  
  
myresult = mycursor.fetchall()  
  
for x in myresult:  
  print(x)

## Wildcard Characters

You can also select the records that starts, includes, or ends with a given letter or phrase.

Use the % to represent wildcard characters:

### Example

Select records where the address contains the word "street":

import mysql.connector  
  
mydb = mysql.connector.connect(  
  host="localhost",  
  user="root",  
  password="",  
  database="mydatabase"  
)  
  
mycursor = mydb.cursor()  
  
sql = "SELECT \* FROM customers WHERE address LIKE '%street%'"

# **address LIKE '%street%'**: This is the condition applied to filter the rows.

* **LIKE**: This operator is used in SQL to search for a specified pattern in a column.
* **'%street%'**: The % symbols are wildcards in SQL. They allow for matching any sequence of characters.
  + The % before "street" means that there can be any number of characters before the word "street".
  + The % after "street" means that there can be any number of characters after the word "street".
* **Overall Meaning**: The condition checks if the address column contains the word "street" anywhere in the text.

mycursor.execute(sql)  
  
myresult = mycursor.fetchall()  
  
for x in myresult:  
  print(x)

## Prevent SQL Injection

When query values are provided by the user, you should escape the values.

This is to prevent SQL injections, which is a common web hacking technique to destroy or misuse your database.

The mysql.connector module has methods to escape query values:

## Example

Escape query values by using the placholder %s method:

import mysql.connector  
  
mydb = mysql.connector.connect(  
  host="localhost",  
  user="root",  
  password="",  
  database="mydatabase"  
)  
  
mycursor = mydb.cursor()  
  
sql = "SELECT \* FROM customers WHERE address = %s"  
adr = ("Anna Salai, Kanyakumari", )

# **sql**: This variable stores the SQL query as a string. The query is structured to retrieve all columns (\*) from the customers table where the address column matches a specific value.

# **%s**: This is a placeholder for the value that will be provided later. It is a way to safely include user input or variable data into an SQL query, protecting against SQL injection attacks.

mycursor.execute(sql, adr)  
  
myresult = mycursor.fetchall()  
  
for x in myresult:  
  print(x)

# Python MySQL Order By

## Sort the Result

Use the ORDER BY statement to sort the result in ascending or descending order.

The ORDER BY keyword sorts the result ascending by default. To sort the result in descending order, use the DESC keyword.

### Example

Sort the result alphabetically by name: result:

import mysql.connector  
  
mydb = mysql.connector.connect(  
  host="localhost",  
  user="root",  
  password="",  
  database="mydatabase"  
)  
  
mycursor = mydb.cursor()  
  
sql = "SELECT \* FROM customers ORDER BY name"

**# ORDER BY name**: This clause is used to sort the result set by one or more columns.

* **name**: Specifies the column by which the data should be ordered.
* **Default Sorting**: By default, the ORDER BY clause sorts the data in ascending order (from A to Z for text, and from lowest to highest for numbers).

mycursor.execute(sql)  
  
myresult = mycursor.fetchall()  
  
for x in myresult:  
  print(x)

## ORDER BY DESC

Use the DESC keyword to sort the result in a descending order.

## Example

Sort the result reverse alphabetically by name:

import mysql.connector  
  
mydb = mysql.connector.connect(  
  host="localhost",  
  user="root",  
  password="",  
  database="mydatabase"  
)  
  
mycursor = mydb.cursor()  
  
sql = "SELECT \* FROM customers ORDER BY name DESC"

# **DESC**: The DESC keyword stands for "descending." It indicates that the sorting should be done in descending order (from Z to A for text, and from highest to lowest for numbers).  
  
mycursor.execute(sql)  
  
myresult = mycursor.fetchall()  
  
for x in myresult:  
  print(x)

# Python MySQL Delete From By

## Delete Record

You can delete records from an existing table by using the "DELETE FROM" statement:

### Example

Delete any record where the address is "Anna Salai, Kanyakumari":

import mysql.connector  
  
mydb = mysql.connector.connect(  
  host="localhost",  
  user="root",  
  password="",  
  database="mydatabase"  
)  
  
mycursor = mydb.cursor()  
  
sql = "DELETE FROM customers WHERE address = 'Anna Salai, Kanyakumari'"

# **DELETE FROM customers**: The DELETE statement is used to remove rows from a table. In this case, it specifies that rows should be deleted from the customers table.

# **WHERE address = 'Anna Salai, Kanyakumari'**: The WHERE clause specifies the condition that must be met for a row to be deleted. In this query, only rows where the address column matches 'Anna Salai, Kanyakumari' will be deleted.  
  
mycursor.execute(sql)  
  
mydb.commit()  
  
print(mycursor.rowcount, "record(s) deleted")

**Important!:** Notice the statement: mydb.commit(). It is required to make the changes, otherwise no changes are made to the table.

**Notice the WHERE clause in the DELETE syntax:** The WHERE clause specifies which record(s) that should be deleted. If you omit the WHERE clause, all records will be deleted!

## Prevent SQL Injection

It is considered a good practice to escape the values of any query, also in delete statements.

This is to prevent SQL injections, which is a common web hacking technique to destroy or misuse your database.

The mysql.connector module uses the placeholder %s to escape values in the delete statement:

### Example

Escape values by using the placeholder %s method:

import mysql.connector  
  
mydb = mysql.connector.connect(  
  host="localhost",  
  user="root",  
  password="",  
  database="mydatabase"  
)  
  
mycursor = mydb.cursor()  
  
sql = "DELETE FROM customers WHERE address = %s"  
adr = ("Kamarajar Nagar Nellai", )  
  
mycursor.execute(sql, adr)  
  
mydb.commit()  
  
print(mycursor.rowcount, "record(s) deleted")

# Python MySQL Update Table

## Update Table

You can update existing records in a table by using the "UPDATE" statement:

### Example

Overwrite the address column from "Valley 345" to "Canyon 123":

import mysql.connector  
  
mydb = mysql.connector.connect(  
  host="localhost",  
  user="root",  
  password="",  
  database="mydatabase"  
)  
  
mycursor = mydb.cursor()  
  
sql = "UPDATE customers SET address = 'North Street, Namakkal' WHERE address = 'South Street, Namakkal'"

# **UPDATE customers**: The UPDATE statement is used to modify existing records in a table. Here, it targets the customers table.

# **SET address = 'North Street, Namakkal'**: This specifies that the address field of the records should be updated to 'North Street, Namakkal'.

# **WHERE address = 'South Street, Namakkal'**: The WHERE clause filters the records that should be updated. Only the records where the address is currently 'South Street, Namakkal' will be modified.  
  
mycursor.execute(sql)  
  
mydb.commit()  
  
print(mycursor.rowcount, "record(s) affected")

**Important!:** Notice the statement: mydb.commit(). It is required to make the changes, otherwise no changes are made to the table.

**Notice the WHERE clause in the UPDATE syntax:** The WHERE clause specifies which record or records that should be updated. If you omit the WHERE clause, all records will be updated!

## Prevent SQL Injection

It is considered a good practice to escape the values of any query, also in update statements.

This is to prevent SQL injections, which is a common web hacking technique to destroy or misuse your database.

The mysql.connector module uses the placeholder %s to escape values in the update statement:

### Example

Escape values by using the placeholder %s method:

import mysql.connector  
  
mydb = mysql.connector.connect(  
  host="localhost",  
  user="root",  
  password="",  
  database="mydatabase"  
)  
  
mycursor = mydb.cursor()  
  
sql = "UPDATE customers SET address = %s WHERE address = %s"  
val = ("Thambi Nagar, Vellore", "Anna Nagar, Vellore")  
  
mycursor.execute(sql, val)  
  
mydb.commit()  
  
print(mycursor.rowcount, "record(s) affected")

# Python MySQL Limit

## Limit the Result

You can limit the number of records returned from the query, by using the "LIMIT" statement:

### Example

Select the 5 first records in the "customers" table:

import mysql.connector  
  
mydb = mysql.connector.connect(  
  host="localhost",  
  user="root",  
  password="",  
  database="mydatabase"  
)  
  
mycursor = mydb.cursor()  
  
mycursor.execute("SELECT \* FROM customers LIMIT 5")

# **LIMIT 5**: This clause restricts the number of rows returned to 5. It is used to limit the result set to a specified number of rows.  
myresult = mycursor.fetchall()  
  
for x in myresult:  
  print(x)

## Start From Another Position

If you want to return five records, starting from the third record, you can use the "OFFSET" keyword:

### Example

Start from position 3, and return 5 records:

import mysql.connector  
  
mydb = mysql.connector.connect(  
  host="localhost",  
  user="root",  
  password="",  
  database="mydatabase"  
)  
  
mycursor = mydb.cursor()  
  
mycursor.execute("SELECT \* FROM customers LIMIT 5 OFFSET 2")  
  
myresult = mycursor.fetchall()  
  
for x in myresult:  
  print(x)

# Python MySQL Drop Table

## Delete a Table

You can delete an existing table by using the "DROP TABLE" statement:

### Example

Delete the table "customers":

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

# **DROP TABLE customers**: The DROP TABLE statement is used to delete the entire table named customers from the database.  
  
mycursor.execute(sql)

## Drop Only if Exist

If the table you want to delete is already deleted, or for any other reason does not exist, you can use the IF EXISTS keyword to avoid getting an error.

### Example

Delete the table "customers" if it exists:

import mysql.connector  
  
mydb = mysql.connector.connect(  
  host="localhost",  
  user="root",  
  password="",  
  database="mydatabase"  
)  
  
mycursor = mydb.cursor()  
  
sql = "DROP TABLE IF EXISTS customers"

# **IF EXISTS**: This clause checks if the table exists before attempting to drop it. If the table does not exist, the query does nothing and does not produce an error.  
  
mycursor.execute(sql)