**MySQL-Connector-Python module in Python**

* Last Updated : 09 Mar, 2020

MySQL is a Relational Database Management System (RDBMS) whereas the structured Query Language (SQL) is the language used for handling the RDBMS using commands i.e Creating, Inserting, Updating and Deleting the data from the databases. SQL commands are case insensitive i.e CREATE and create signify the same command.

In this article, we will be discussing the MySQL Connector module of Python, how to install this module and a piece of code on how to connect this with the MySQL database. For any application, it is very important to store the database on a server for easy data access.

**What is MYSQL Connector/Python?**

MySQL Connector/Python enables Python programs to access MySQL databases, using an API that is compliant with the Python Database API Specification v2.0 (PEP 249). It is written in pure Python and does not have any dependencies except for the Python Standard Library.

**How to Connect to MySQL Database in Python**

1. **Install MySQL connector module**

Use the pip command to [install MySQL connector Python](https://pynative.com/install-mysql-connector-python/).  
pip install mysql-connector-python

1. **Import MySQL connector module**

Import using a import mysql.connector statement so you can use this module’s methods to communicate with the MySQL database.

1. **Use the connect() method**

Use the connect() method of the MySQL Connector class with the required arguments to connect MySQL. It would return a MySQLConnection object if the connection established successfully

1. **Use the cursor() method**

Use the cursor() method of a MySQLConnection object to create a cursor object to perform various SQL operations.

1. **Use the execute() method**

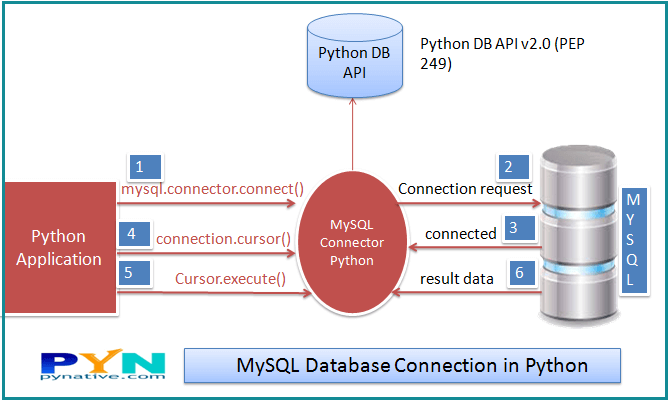
The execute() methods run the SQL query and return the result.

1. **Extract result using [fetchall()](https://pynative.com/python-cursor-fetchall-fetchmany-fetchone-to-read-rows-from-table/)**

Use cursor.fetchall() or fetchone() or fetchmany() to read query result.

1. **Close cursor and connection objects**

use cursor.clsoe() and connection.clsoe() method to close open connections after your work completes

MySQL database connection in Python

Run the below query on the MySQL console if you have not created any database in MySQL. Otherwise, you can skip the below query.

**Python MySQL CRUD Operation**

Also, learn how to execute various MySQL operations from Python by referring to the following articles.

**Click on each tutorial to study operations in detail**.

* [Insert rows into MySQL table from Python](https://pynative.com/python-mysql-insert-data-into-database-table/): Insert a single and multiple rows into the MySQL table. Also, learn how to use Python variables in the parameterized query to insert dynamic data into a table.
* [Select rows from MySQL table using Python](https://pynative.com/python-mysql-select-query-to-fetch-data/): Execute a SQL SELECT query from a Python application to fetch rows from MySQL table. Also, learn how to process SELECT query results, Fetch all rows or single rows from the table, and count total rows of a table.
* [Update rows of MySQL table from Python](https://pynative.com/python-mysql-update-data/): Update a single row, multiple rows, a single column, and various columns. Additionally, learn how to use python variables in the parameterized query to update table data.
* [Delete table rows from Python](https://pynative.com/python-mysql-delete-data/): Delete a single row, multiple rows, a single column, and various columns. Also, learn to Delete all Rows, Delete tables, and an entire database from MySQL using python.
* [Execute MySQL stored procedures from Python](https://pynative.com/python-mysql-execute-stored-procedure/) and learn how to pass IN and OUT parameters to the MySQL stored procedure.
* [Python MySQL Parameterized Query and Prepared Statement](https://pynative.com/python-mysql-execute-parameterized-query-using-prepared-statement/): Learn to use Parameterized Queries or Prepared Statement to use Python variables in the SQL query to pass dynamic data to MySQL table.
* [Python MySQL Commit and Rollback to Manage Transactions](https://pynative.com/python-mysql-transaction-management-using-commit-rollback/): Manage MySQL database transactions from Python to maintain the ACID property of MySQL transactions using the commit() and rollback() methods.
* [Python Database Connection Pooling With MySQL](https://pynative.com/python-database-connection-pooling-with-mysql/): Learn to create and use a connection pool to increase the performance of your Python MySQL applications.
* [Python MySQL BLOB Insert and Retrieve digital data](https://pynative.com/python-mysql-blob-insert-retrieve-file-image-as-a-blob-in-mysql/): Learn to insert or fetch any digital information such as a file, image, video, or song as blob data into MySQL table from Python

**Python MySQL Connection arguments list**

port: The TCP/IP port of the MySQL server. This value must be an integer. We can specify the different port number if the MySQL server is listening to a different port. The default value for this port argument is 3306.

* use\_unicode: Specify whether to use Unicode or not. The default value is True.
* charset: MySQL character set to use, character set variables relate to a client’s interaction with the server. There are almost 30 to 40 charset MySQL server supports. The default value of the charset argument is “utf8″.
* auto-commit: Set it to true if you want to auto-commit transactions. If you wish to manage transactions in MySQL from Python, you need to set this value true or false. The default value is False, i.e., the changes are not committed to the database. You need to explicitly call a commit method to persist your changes in the database.
* get\_warnings: To fetch warning, this is helpful to know the connection is established but with warnings. The default value is False.
* raise\_on\_warnings: Set it when you want to raise an exception on warnings. The Default value is False.
* connection\_timeout (connect\_timeout\*) : Timeout for the TCP and Unix socket connections. The connection terminates after this timeout expired.
* buffered: If true, the cursor objects fetch the results immediately after executing queries. The default value is False.
* raw: If true, MySQL results are returned as-is rather than converting into Python types. The default value is False. You can set it to true if you want a query result in MySQL type.
* force\_ipv6: When setting to True, uses IPv6 when an address resolves to both IPv4 and IPv6. By default, IPv4 is used in such cases. The default value for this argument is false.
* pool\_name: It is the Connection pool name that you are creating or using.
* pool\_size: Connection pool size that you want to create. The default value is 5.
* pool\_reset\_session: Reset session variables when the connection is returned to the pool. The default is True.
* use\_pure: Specify whether to use pure Python or C Extension. If use\_pure=False, then a pure Python module is used; otherwise, it connects MySQL using C extension. Moreover, if C Extension is not available, MySQL Connector Python automatically falls back to the pure Python implementation.
* unix\_socket: The location of the Unix socket file. These enable communication between two processes.
* auth\_plugin: Authentication plugin to use, Added in 1.2.1.
* collation: MySQL collation to use. You can use the collation that you set while installing MySQL Server. The default value is utf8\_generalW\_chiich.
* sql\_mode: Set the sql\_mode session variable at connection time.