

MySQL

It is an open source database management system application which will help us to manage the database like store and retrieve data.

To work with MySQL in Python program we have to import **connector** sub module of **mysql** module.

```
import mysql.connector
```

Creating Connection

`connect()` – This method is used to open or establish a new connection. It returns an object representing the connection.

Syntax: -

```
connection_object = connect(user='username', password='pass' host='localhost',  
port=3306);
```

eg: -

```
import mysql.connector
```

```
conn = mysql.connector.connect(user='root', password='geek', host='localhost',  
port=3306)
```

Creating Connection

```
import mysql.connector  
config = {  
    'user': 'root',  
    'password': 'geek',  
    'host': 'localhost',  
    'port': 3306  
}  
conn = mysql.connector.connect(**config)
```

Check Connection

`is_connected()` – This method is used to check if the connection to MySQL is established or not. It returns True if the connection is established successfully.

Syntax:- `connection_object.is_connected()`

eg:-

```
import mysql.connector
```

```
conn = mysql.connector.connect(user='root', password='geek',  
host='localhost')
```

```
print(conn.is_connected())
```

Close Connection

close() – This method is used to close the connection.

Syntax:- connection_object.close()

eg:-

```
import mysql.connector
```

```
conn = mysql.connector.connect(user='root', password='geek',  
host='localhost')
```

```
# Do your work
```

```
conn.close()
```

Operations

- Create Database
- Show Database

cursor() Method

This method is used to create cursor class object.

We need cursor object so we can call execute() method.

Syntax:- `cursor_object = connection_object.cursor()`

Arguments may be passed to the cursor() method to control what type of cursor to create:

- If *buffered* is *True*, the cursor fetches all rows from the server after an operation is executed. This is useful when queries return small result sets. *buffered* can be used alone, or in combination with the *dictionary* or *named_tuple* argument.
- If *dictionary* is *True*, the cursor returns rows as dictionaries.
- If *named_tuple* is *True*, the cursor returns rows as named tuples.
- If *prepared* is *True*, the cursor is used for executing prepared statements.

cursor() Method

- if *raw* is *True*, the cursor skips the conversion from MySQL data types to Python types when fetching rows. A raw cursor is usually used to get better performance or when you want to do the conversion yourself.
- The *cursor_class* argument can be used to pass a class to use for instantiating a new cursor. It must be a subclass of `cursor.CursorBase`.

The returned object depends on the combination of the arguments. Examples:


- If not buffered and not raw: `MySQLCursor`
- If buffered and not raw: `MySQLCursorBuffered`
- If not buffered and raw: `MySQLCursorRaw`
- If buffered and raw: `MySQLCursorBufferedRaw`

cursor() Method

The returned object depends on the combination of the arguments. Examples:

- If not buffered and not raw: MySQLCursor
- If buffered and not raw: MySQLCursorBuffered
- If not buffered and raw: MySQLCursorRaw
- If buffered and raw: MySQLCursorBufferedRaw

eg:- myc = conn.cursor() 

eg:- myc = conn.cursor(buffered=True) 

eg:- myc = conn.cursor(prepared=True) 

execute() Method

This method is used to execute given SQL queries.

We need cursor object so we can call execute() method.

Syntax:- cursor_object.execute(sql, param=None, multi=False)

Sql – It is sql query.

Param – The parameters found in the tuple or dictionary params are bound to the variables in the operation.

Multi – execute() returns an iterator if multi is True.

eg:-

```
myc = conn.cursor()
```

```
myc.execute('SELECT * FROM student')
```

```
sql = 'SELECT * FROM student'
```

```
myc = conn.cursor()
```

```
myc.execute(sql)
```

Close Cursor

close () method closes the cursor, resets all results, and ensures that the cursor object has no reference to its original connection object.

Syntax:- `cursor_object.close()`

eg:- `myc.close()`

Connecting to Database

`connect()` – This method is used to open or establish a new connection. It returns an object representing the connection.

Syntax: -

```
connection_object = connect(user='username', password='pass', database='dbname',  
host='localhost', port=3306);
```

eg: -

```
import mysql.connector
```

```
conn = mysql.connector.connect(user='root', password='geek', host='localhost',  
database='pdb', port=3306)
```

Connecting to Database

```
import mysql.connector
config = {
    'user': 'root',
    'password': 'geek',
    'host': 'localhost',
    'database': 'pdb',
    'port': 3307
}
conn = mysql.connector.connect(**config)
```

executemany() Method

This method is used to execute given SQL query against all parameter sequences or mappings found in the sequence `seq_of_params`.

With the `executemany()` method, it is not possible to specify multiple statements to execute in the operation argument.

Syntax:- `cursor_object.executemany(sql, seq_of_param)`

eg:-

```
myc = conn.cursor()
```

```
myc.executemany(sql, seq_of_params)
```

Operations

- Create Table
- Show Table
- Insert Data
- Delete Data
- Update Data

commit() Method

This method is used to save inserted row in the table. It is required to make the changes, otherwise no changes are made to the table.

This method sends a COMMIT statement to the MySQL server, committing the current transaction. Since by default Connector/Python does not autocommit, it is important to call this method after every transaction that modifies data for tables that use transactional storage engines.

Syntax:- `connection_object.commit()`

eg:- `conn.commit()`

rollback() Method

This method is used to un-save row, if there is an error.

This method sends a ROLLBACK statement to the MySQL server, undoing all data changes from the current transaction. By default, Connector/Python does not autocommit, so it is possible to cancel transactions when using transactional storage engines such as InnoDB.

Syntax:- `connection_object.rollback()`

eg:- `conn.rollback()`

try:

```
myc.execute(sql)
```

```
conn.commit()
```

except:

```
conn.rollback()
```

rowcount Property

This read-only property returns the number of rows returned for SELECT statements, or the number of rows affected by DML statements such as INSERT or UPDATE.

Syntax:- `cursor_object.rowcount`

eg:- `myc.rowcount`

lastrowid Property

This read-only property returns the value generated for an `AUTO_INCREMENT` column by the previous `INSERT` or `UPDATE` statement or `None` when there is no such value available.

If you perform an `INSERT` into a table that contains an `AUTO_INCREMENT` column, `lastrowid` returns the `AUTO_INCREMENT` value for the new row.

If you insert multiple rows into a table using a single `INSERT` statement, the `lastrowid` property contains the last insert id of the first row.

Syntax:- `cursor_object.lastrowid`

eg:- `myc.lastrowid`

fetchone() Method

This method retrieves the next row of a query result set and returns a single sequence, or None if no more rows are available. By default, the returned tuple consists of data returned by the MySQL server, converted to Python objects. If the cursor is a raw cursor, no such conversion occurs.

You must fetch all rows for the current query before executing new statements using the same connection.

Syntax:- `row = cursor_object.fetchone()`

eg:- `row = myc.fetchone()`

fetchall() Method

This method fetches all (or all remaining) rows of a query result set and returns a list of tuples. If no more rows are available, it returns an empty list.

You must fetch all rows for the current query before executing new statements using the same connection.

Syntax:- `rows = cursor_object.fetchall()`

eg:- `rows = myc.fetchall()`

fetchmany() Method

This method fetches the next set of rows of a query result and returns a list of tuples. If no more rows are available, it returns an empty list.

The number of rows returned can be specified using the size argument, which defaults to one. Fewer rows are returned if fewer rows are available than specified.

You must fetch all rows for the current query before executing new statements using the same connection.

Syntax:- `rows = cursor_object.fetchmany(size=1)`

eg:- `rows = myc.fetchmany(3)`

Parameterized Query

A parameterized query is a query which can use the format or pyformat parameterization style for parameters and the parameter values supplied at execution.

These executed with MySQLCursor can use the %s and %(key)s format style. %s is used as format style in the sql queries, while using tuple parameters. %(key)s is used as format style in the sql queries, while using dictionary parameters.

```
myc = conn.cursor()
```

Tuple Parameters

```
sql = 'INSERT INTO student(name, roll, fees) VALUES(%s, %s, %s)'  
myc = conn.cursor()  
myc.execute(sql, ("Rohan", 111, 60000.50))
```

```
sql = 'INSERT INTO student(name, roll, fees) VALUES(%s, %s, %s)'  
myc = conn.cursor()  
params = ("Rohan", 111, 60000.50)  
myc.execute(sql, params)
```


Dictionary Parameters

```
sql = 'INSERT INTO student(name, roll, fees) VALUES(%(name)s, %(roll)s, %  
(fees)s)'
```

```
myc = conn.cursor()
```

```
myc.execute(sql, {'name':'Kajal', 'roll':777, 'fees': 54100})
```

```
sql = 'INSERT INTO student(name, roll, fees) VALUES(%(name)s, %(roll)s, %  
(fees)s)'
```

```
myc = conn.cursor()
```

```
params = {'name':'Kajal', 'roll':777, 'fees': 54100}
```

```
myc.execute(sql, params)
```

executemany() Method

This method is used to prepare given SQL query and executes it against all parameter sequences or mappings found in the sequence seq_of_params.

With the executemany() method, it is not possible to specify multiple statements to execute in the sql argument.

Syntax:- cursor_object.executemany(sql, seq_of_params)

sql – It is sql query

seq_of_params – It is a list of tuples, containing the data to insert.

Prepared Statement

A prepared statement is used to execute the same statement repeatedly with high efficiency. The prepared statement execution consists of two stages: prepare and execute.

At the prepare stage a statement template is sent to the database server. The server performs a syntax check and initializes server internal resources for later use.

At the Execute Stage the parameter values are sent to the server. The server creates a statement from the statement template and these values to execute it.

Prepared statements executed with `MySQLCursorPrepared` can use the format `%s` or `qmark ?` parameterization style.

`%s` and `?` are called as parameter marker.

This differs from nonprepared statements executed with `MySQLCursor`, which can use the `format` or `pyformat` parameterization style.

Advantage

- Prepared statements are very useful against SQL injections.
- Prepared statements reduce parsing time as the preparation on the query is done only once (although the statement is executed multiple times)

Creating a Cursor

Using *prepared=True* argument to the `cursor()` method, creates a cursor that enables execution of prepared statements using the binary protocol.

In this case, the `cursor()` method of the connection object returns a `MySQLCursorPrepared` object.

e.g:-

```
myc = conn.cursor(prepared=True)
```

```
sql = 'INSERT INTO student(name, roll, fees) VALUES(%s, %s, %s)'  
myc = conn.cursor(prepared=True)  
myc.execute(sql, ("Rohan", 111, 60000.50))
```

```
sql = 'INSERT INTO student(name, roll, fees) VALUES(%s, %s, %s)'  
myc = conn.cursor(prepared=True)  
params = ("Rohan", 111, 60000.50)  
myc.execute(sql, params)
```

```
sql = 'INSERT INTO student(name, roll, fees) VALUES(?, ?, ?)'  
myc = conn.cursor(prepared=True)  
myc.execute(sql, ("Rohan", 111, 60000.50))
```

```
sql = 'INSERT INTO student(name, roll, fees) VALUES(?, ?, ?)'  
myc = conn.cursor(prepared=True)  
params = ("Rohan", 111, 60000.50)  
myc.execute(sql, params)
```

How it works

- For the first call to the `execute()` method, the cursor prepares the statement. If data is given in the same call, it also executes the statement and you should fetch the data.
- For subsequent `execute()` calls that pass the same SQL statement, the cursor skips the preparation phase.