Chapter 10 – Interface Python with MySQL

In order to connect to a database from within Python, you need a library(**mysql connector)** that provides connectivity functionality.

Steps for Creating Database Connectivity Applications

There are mainly *seven* steps that must be followed in order to create a database connectivity application.

Step 1 : Start Python.
Step 2 : Import the packages required for database programming.
Step 3 : Open a connection to database.
Step 4 : Create a cursor instance.
Step 5 : Execute a query.
Step 6 : Extract data from result set.
Step 7 : Clean up the environment.

Step 1. Start Python

- Start Python's editor where you can create your Python scripts.

Step 2. Import mysql.connector Package

- First of all you need to import **mysql.connector** package in your Python scripts. For this, write import command as shown below:

```
import mysql.connector
```

or

```
import mysql.connector as sqLtor
```

Step 3: Open a connection to database.

- The **connect( )** function of mysql.connector establishes connection to a MySQL database and requires four parameters, which are:

<connection-object> = mysql.connector.connect(**host** = <host-name> , **user** = <username> , **passwd** = <password> , **database** = <database>)

e.g.

For example :

loginid and password of your MySQL database

```
import mysql.connector as sqltor
mycon = sqltor.connect(host = "localhost", user = "root", passwd = "MyPass",
                       database = "test")
```

The connection object

a MySQL database

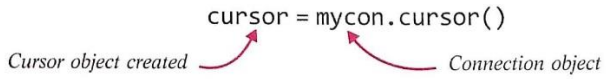
The above command will establish connection to MySQL database with user as **"root"** , password as **"MyPass"** and to the **MySQL database namely test** which exists on the MySQL.

- You can also check for successful connection using function **is_connected( )** with connected object, which returns *True* , if connection is successful.
  e.g.

Step 4 : Create a cursor instance.

- When we connect to a database from within a script/program, then the query gets sent to the server, where it gets executed, and the ***resultset*** (the set of records retrieved as per query) is sent over the connection to you, in one burst of activity, i.e. in one go. And in order to do the processing of data row by row, a special control structure is used, which is called ***Database Cursor.***
- **Syntax:**

  <cursorobject> = <connectedobject>.cursor( )
- **E.g.**



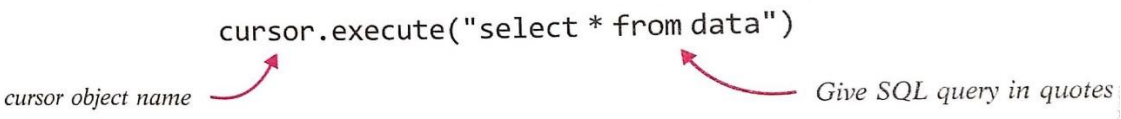
Since we established database connection through connection object **mycon** earlier, we have created a *cursor object* using the same connection object **mycon**.

Step 5 : Execute a query

- Once you have created a cursor, you can execute SQL query using **execute( )** function with cursor object as per following syntax:

  <cursorobject>.execute(<sql query string>)

- **E.g.**



The above code will execute the given SQL query and store the retrieved records(i.e. , the *resultset*) in the cursor object (namely **cursor**) which you can then use in your program/scripts as required.

Step 6 : Extract data from result set.

- Once the result of query is available in the form of a resultset stored in a cursor object, you can extract data from the resultset using any of the following ***fetch( )*** functions.
  (i) **<data>= <cursor>.fetchall( )** - It will return all the records retrieved as per query in a tuple form.
  (ii) **<data> = <cursor>.fetchone( )** - It will return one record from the *resultset* as a tuple or a list. First time it will return the first record, next time it will fetch the next record and so on.
  This method returns one record as a tuple : if there are no more records then it returns **None.**

(iii) **<data>=<cursor>.fetchmany(<n>)** - This method accepts number of records to fetch and returns a tuple where each record itself is a tuple.

(iv) **<variable>=<cursor>.rowcount** – The ***rowcount*** is a property of cursor object that returns the number of rows retrieved from the cursor so far.

For Example,

Table ***student*** of MySQL database ***test***

Rollno	Name	Marks	Grade	Section	Project
101	Ruhani	76.80	A	A	Pending
102	George	71.20	B	A	Submitted
103	Simran	81.20	A	B	Evaluated
104	Ali	61.20	B	C	Assigned
105	Kushal	51.60	C	C	Evaluated
106	Arsiya	91.60	A+	B	Submitted
107	Raunak	32.50	F	B	Submitted

Following code examples assume that the connection to the database has been established using **connect( )** method of ***mysql.connector*** as discussed in earlier steps. That is, all the following code examples of fetch functions have following code pre-executed for them :

```
import mysql.connector as sqltor
mycon = sqltor.connect(host = "localhost", user = "root", passwd = "MyPass",
                       database = "test")
if mycon.is_connected() == False:
    print('Error connecting to MySQL database')
cursor = mycon.cursor()
cursor.execute("select * from student")
```

*The SQL query retrieves all the data of table **student** of database **test***

(i) The fetchall( ) method

```
:   # database connected established and cursor object created
st = "select * from student where marks > %s" %(70,)
cursor.execute( st )
data = cursor.fetchall()
for row in data :
    print(row)
```

NOTE

Do not forget to enclose place-holder **%s** in quotes for string parameters in string template.

```
(101, 'Ruhani', Decimal('76.80'), 'A', 'A', 'Pending')
(102, 'George', Decimal('71.20'), 'B', 'A', 'Submitted')
(103, 'Simran', Decimal('81.20'), 'A', 'B', 'Evaluated')
(106, 'Arsiya', Decimal('91.60'), 'A+', 'B', 'Submitted')
```

(ii) The fetchmany( ) method

The **fetchmany(<n>)** method will return only the <n> number of rows from the resultset in the form of a tuple containing the records.

```
:    # database connected established and cursor object created
     data = cursor.fetchmany(4)      # Fetch 4 records in the resultset
     count = cursor.rowcount         # How many records returned by SQL query in the resultset
     print("Total number of rows retrieved from resultset :", count)
     for row in data :               # Now you can process the data tuple one row at a time
         print(row)
```

*The **data** variable will store the retrieved records from the resultset in the form of a tuple (a tuple of records)*

The output produced by above code is :

```
Total number of rows retrieved from resultset : 4   # Result of cursor.rowcount
(101, 'Ruhani', Decimal('76.80'), 'A', 'A', 'Pending')
(102, 'George', Decimal('71.20'), 'B', 'A', 'Submitted')
(103, 'Simran', Decimal('81.20'), 'A', 'B', 'Evaluated')
(104, 'Ali',    Decimal('61.20'), 'B', 'C', 'Assigned')
```

NOTE

The **cursor.rowcount** returns how many rows have been so far retrieved [illegible]

(iii) The fetchone( ) method

The **fetchone( )** method will return only one row from the resultset in the form of a tuple containing a record. A pointer is initialized which points to the first record of the resultset as soon as you execute a query. The **fetchone( )** returns the record pointed to by this pointer. When you fetch one record, the pointer moves to next record of the recordset. So next time , if you execute the **fetchone( )** metod, it will return only one record pointed to by the pointer and after fetching, the pointer will move to the next record of the resultset.

Also, carefully notice the behaviour of **cursor.rowcount** that always returns how many records have been **retrieved so far** using any of the *fetch..( ) methods*.

```
:    # database connected established and cursor object created
     data = cursor.fetchone()        # Fetch 1 records in the resultset (first time, only the first record is retrieved)
     count = cursor.rowcount
     print("Total number of rows retrieved from resultset :", count)
     print(data)
     print("\nAgain fetching one record")
     data = cursor.fetchone()        # Next fetchone( ) will fetch the next record from the resultset
     count = cursor.rowcount
     print("Total number of rows retrieved from resultset :", count)
     print(data)
```

```
Total number of rows retrieved in resultset : 1   # Result of cursor.rowcount
(101, 'Ruhani', Decimal('76.80'), 'A', 'A', 'Pending')

Again fetching one record
Total number of rows retrieved from resultset : 2   # Result of cursor.rowcount
(102, 'George', Decimal('71.20'), 'B', 'A', 'Submitted')
```

This time it is 1 because fetchone() method retrieved only 1 record from the cursor

This time it is 2 because fetchone() method retrieved only 1 record (next record) from the cursor but SO FAR 2 records have been retrieved.

Step 7: Clean up the Environment

- In this final step, you need to close the connection established. This you can do as follows:
  <connection object>.close( )
- **E.g.**
  mycon.close( )

Parameterised Queries

E.g.

```
inp = 70
SELECT * FROM student WHERE MARKS > inp ;
```

Two methods to form query strings based on some parameters:

(i) Old Style : String Templates with % formatting

- In this style, string formatting uses this general form : **f % v**
  Where **f** is a template string and **v** specifies the value or values to be formatted using that template.

e.g.

```
"select * from student where marks > %s"   %(70,)
            f                               v
```

Now you can store this query string in variable and then execute that variable through **cursor.execute( )** method as shown below :

```
:   # database connected established and cursor object created
st = "select * from student where marks > %s" %(70,)
cursor.execute( st )
data = cursor.fetchall()
for row in data :
    print(row)
```

NOTE

Do not forget to enclose place-holder **%s** in quotes for string parameters in string template.

```
(101, 'Ruhani', Decimal('76.80'), 'A', 'A', 'Pending')
(102, 'George', Decimal('71.20'), 'B', 'A', 'Submitted')
(103, 'Simran', Decimal('81.20'), 'A', 'B', 'Evaluated')
(106, 'Arsiya', Decimal('91.60'), 'A+', 'B', 'Submitted')
```

- In the similar manner, you can add multiple parameter values, but you must not forget to enclose placeholder **%s** in quotes for string parameters *e.g.*

```
:    # database connected established and cursor object created
st = "select * from student where marks > %s and section = '%s' " % (70,'B')
cursor.execute(st)
data = cursor.fetchall()
for row in data :
    print(row)
```

See, '%s' enclosed in quotes for string value

V tuple containing a number and a string value

```
(103, 'Simran', Decimal('81.20'), 'A', 'B', 'Evaluated')
(106, 'Arsiya', Decimal('91.60'), 'A+', 'B', 'Submitted')
```

(ii) New Style : String Templates with % formatting

- This method is based on use of **format( )** method.
- The general form for using **format( )** is :

template.format(p0 , p1 , ……., k0 = v0 , k1 = v1, ……..)

The **template** is a string containing a mixture of one or more format codes embedded in constant text. The format method uses its argument to substitute an appropriate value for each format code in the template.

e.g.1.

Consider following example. In this example, the format code "{0}" is replaced by the first positional argument (49), and "{1}" is replaced by the second positional argument, the string "okra"

These are place holders

Values tuple V. Values are substituted from here

```
"We have {0} hectares planted to {1}." .format (49, "okra")
```

The above string template will yield following string

```
'We have 49 hectares planted to okra.
```

e.g.2.

Values tuple V. Values are substituted from here

```
"{monster} has eaten {city}".format(city ='Tokyo', monster ='Tsunami')
```

Output:

```
'Tsunami has eaten Tokyo
```

e.g.3.

```
st = "select * from student where marks > {} and section = '{}' ".format(70,'B')
```

The above query string **st** stores :

```
"select * from student where marks > 70 and section = 'B' "
```

Place holder enclosed in quote, for string value

Performing INSERT and UPDATE Queries

- Insert and Update SQL commands, can also executed using SELECT queries.
- But after executing INSERT and UPDATE queries you must **commit** your query. *This makes the changes made by the INSERT and UPDATE queries permanent*. For this you must run **commit( )** method , i.e.

  <connection object>.commit( )

E.g. 1. INSERT query example

```
st = "INSERT INTO student (rollno , name , marks , marks , grade, section)
     VALUES({ } , '{ }' , { } , '{ }', '{ }' )".format( 108,'Eka' , 84.0 , 'A' , 'B')
cursor.execute(st)
mycon.commit( )
```

E.g. 2. UPDATE query example

```
st = "UPDATE student SET marks = { } WHERE marks = { }" . format(77, 76.8)
cursor.execute(st)
mycon.commit( )
```

Important Questions

Q1. What is database connectivity ?

Ans. Database connectivity refers to connection and communication between an application and a database system.

Q2. What is Connection ? What is its role?

Ans. A Connection (represented through a connection object) is the session between the application program and a database. To do anything with database, one must have a connection object.

Q3. What is a result set?

Ans. A **result set** refers to a logical set of records that are fetched from the database by executing a query and made available to the application-program.

Q4. Which package must be imported in Python to create a database connectivity application?

Ans. One such package is **mysql.connector**

****