9 Interface Python with SQL

9.1 INTRODUCTION

Databases play a vital role in the efficient working of an organization. From a construction firm to a stock exchange, every organization depends on large databases. These are essentially collections of tables and are connected with each other through fields or, more precisely, columns/attributes. These database systems support SQL (Structured Query Language), which is used to access data and also to create and exploit the relationship between stored data. Additionally, these databases support database normalization rules for avoiding redundancy of data. Python programming language has powerful features for database programming. Python supports various databases like MySQL, Oracle, Sybase, PostgreSQL, etc. It also supports Data Definition Language (DDL), Data Manipulation Language (DML) and Data Query Statements. But Python standard library does not come with an RDBMS interface. For database programming, the Python **DB-API** is a widely-used module that provides a **Database Application Programming Interface**. It is a standard for database interfaces. Database Application Programming Interface is a set of tools used by an Application program to communicate with the Operating System or other programs such as Database Management System. These APIs are implemented by calling functions in the programs which provide linkage to the required program to perform a task.

Most Python database interfaces adhere to this standard.

We can choose the right database for our application. Python Database API supports a wide range of database servers such as—

- GadFly
- mSQL
- MySQL
- PostgreSQL
- Microsoft SQL Server 2000
- Informix
- Interbase
- Oracle
- Sybase

9.2 PYTHON-MySQL CONNECTIVITY

While designing real-life applications, certain situations arise pertaining to storing some important and necessary information by the user. Usually, the data inputted by the user along with the generated output is displayed but not stored because all program execution takes place inside the RAM which is a temporary memory and as soon as we close the form, its contents (form input and generated output) get erased. They can't be retrieved since they are not saved on a hard disk (or any secondary storage device). Thus, when the application is executed the next time, it requires a new set of inputs from the user. This limitation can be overcome by sending the output generated and saving the input fetched from the user in a database created at the back-end of the application. The input is fetched from the user using Python Interface. This is termed as the **Front End Interface** of the application.

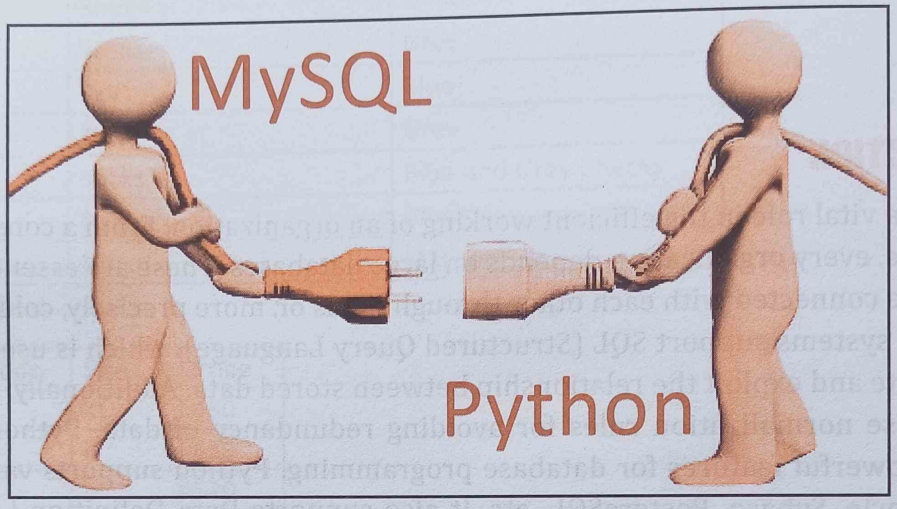


Fig. 9.1: Front-end (Python) and Back-end (MySQL)

Database (The Back-End)

While working with an application, it is required to save data permanently on some secondary storage device, which is usually the hard disk, so that stored data could be used for future reference, modification, deletion and retrieval. An application usually stores a lot of data in the form of a database which is not directly accessible to the user. This database is used by the application to give suitable response to the user. This database is called **Back-End Database**.

In Chapter 8, we learnt how to create databases, tables and how to perform query processing on these tables using SQL commands like CREATE, UPDATE, ALTER, INSERT, DELETE, SELECT and so on in various forms according to their specific syntax structures. We shall be implementing all these DDL and DML commands of SQL through Python Interface.

9.3 WHY PYTHON

Python is a flexible, portable, easy to learn and modifiable language. So, we are integrating MySQL with Python interface for executing any database applications. The various reasons to use Python for programming database applications are:

- Programming in Python is arguably more efficient and faster as compared to other languages.
- Python is known for its portability.
- It is platform-independent.

- Python supports SQL cursors.
- In many programming languages, the application developer needs to take care of the open and closed connections of the database to avoid further exceptions and errors. In Python, these connections are taken care of.
- Python supports relational database systems.
- Python database APIs are compatible with various databases, so it is very easy to migrate and port database application interfaces.

9.4 INSTALLING MySQL-CONNECTOR

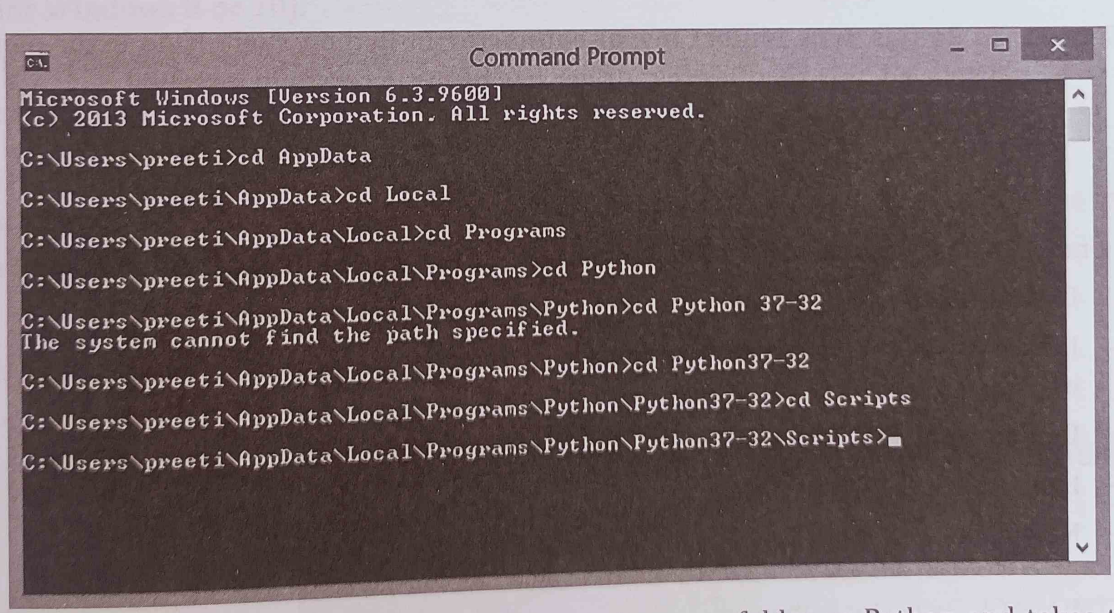
To establish connectivity between Python and MySQL, we require Python Database Interfaces and APIs. We must download a separate DB-API module for each database we need to access.

The DB-API provides a minimal standard for working with databases using Python structures and syntax wherever possible. This API includes the following:

- Importing the API module
- Acquiring a connection with the database
- Issuing SQL statements and stored procedures
- Closing the connection

Our ultimate goal is to connect Python with MySQL. To do this, we need to install a connector. Steps to install the connector and connect it to Python are explained below:

Step 1: To connect Python to MySQL, we have to install mysql-connector using 'pip' command on the command prompt (cmd).



Note: Do ensure that mysql-connector is installed in the same folder as Python and takes the path as—

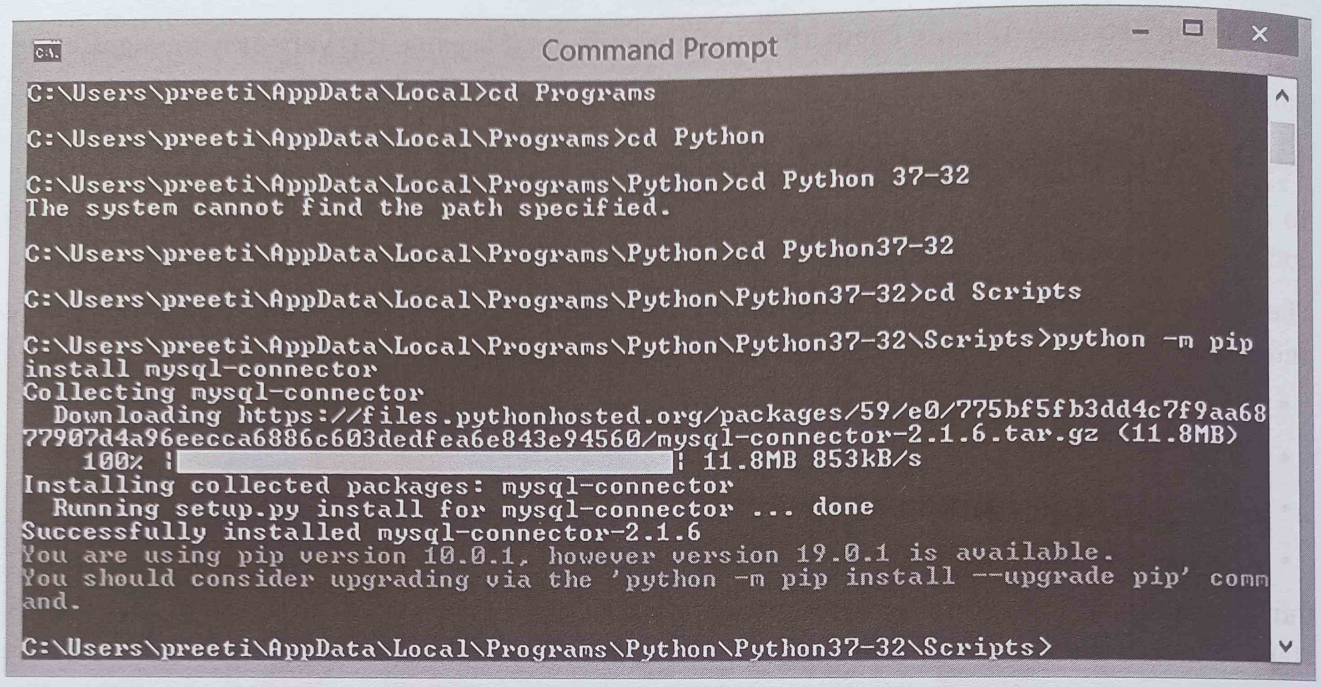
```
>cd C:\Users\your name\AppData\Local\Programs\Python\Python37-
32\Scripts>
```

Step 2: Once you have set the path, type the command as—

```
python -m pip install mysql-connector
```

Or

```
pip install mysql-connector-python
```



Mysql-connector shall download and will be installed on your system. Now we need to check whether it has been properly installed or not.

Step 3: To do this, type **import mysql-connector** in Python shell. If no error message gets displayed, this signifies that driver has been successfully installed.

```
>>> import mysql.connector
>>>
```

In a nutshell, three things are to be kept in mind for the successful installation of mysql connector:

- Ensure that Python has been already installed onto your system.
- In case there is no Python installed prior to connectivity, download Python 3.x and then install it.
- Download MySQL API, exe file will be downloaded; install it.
- Install MySQL-Python Connector
- Now connect MySQL Server using Python.

CTM: MySQL Connector Python requires Python to be in the system's PATH. Installation fails if it doesn't find Python.

Alternatively,

We can also establish the connectivity through the module "**MySQLdb**".

Let us first discuss the basic concepts related to MySQLdb.

9.4.1 MySQLdb

MySQLdb is an interface for connecting to a MySQL database server from Python. It implements the Python Database API and is built on top of the MySQL C API.

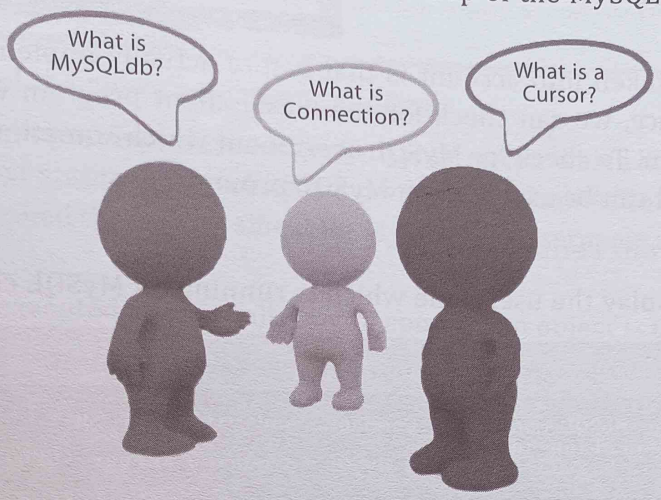


Fig. 9.2: Components of MySQLdb

Python-MySQL Database Access

MySQLdb is the Python interface to work with MySQL databases. It must be imported in Python to work with any MySQL database. To import MySQL for Python 3.x, first the module **MySQLclient** needs to be installed as follows (assuming Python is already installed with pip and its path included for Windows 8 or 10):

- Open a command window (cmd).
- Type the following at the command prompt:

```
pip install mysqlclient
```

After mysqlclient is installed on your system, MySQLdb needs to be imported as shown below:

- Open Python shell at the command prompt:

```
C:\>Python
```

- Import MySQLdb in Python shell

```
>>> import MySQLdb
```

If the above command produces the following result, it means MySQLdb module is not installed.

```
Error
Traceback (most recent call last):
File "test.py", line 3, in <module>
 import MySQLdb
ImportError: No module named MySQLdb
```

If the above error occurs and the mysql module for the Python version you are using is not installed, it can be downloaded from:

http://sourceforge.net/projects/mysqlpython

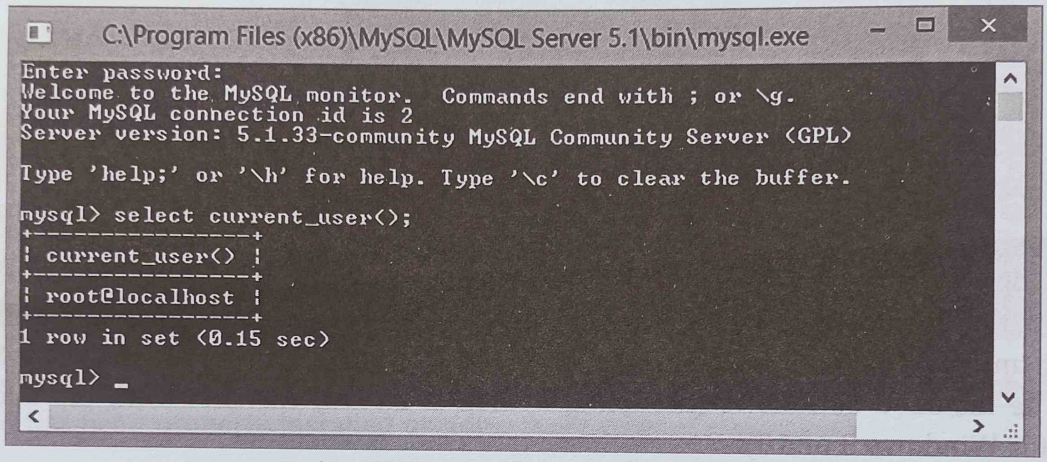
This is followed by establishing connection and creating objects for interacting with MySQL using Python. Therefore, mysql-connector installation, which we have discussed in the previous section (9.4) is recommended for use.

Learning Tip: Connector for Python—
Installing Connector/Python on Microsoft Windows and managing all your MySQL products, including MySQL-Connector/Python, with MySQL Installer is the recommended approach. It handles all requirements and prerequisites, configurations and upgrades.

One more thing to be taken into account is that before actually implementing SQL commands through Python interface, we can check for the mysql client program which is running at the back-end of the program. To check for MySQL user, about which sometimes we are not aware of, the following command can be executed on MySQL prompt.

```
mysql>SELECT current_user();
```

This command shall display the username which is running on MySQL client program.



9.5 ESTABLISHING CONNECTION

After completing the first step of installing mysql-connector, the next step to using MySQL in Python scripts is to make a connection to the database that you wish to use. All Python DB-API modules implement a function:

'module_name.connect'

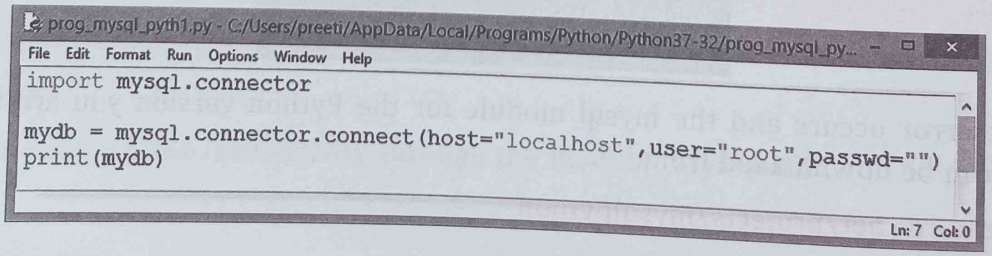
This is an essential function required to connect to the database, which is MySQL in our case.

Note: Ensure that the username and password you are entering for MySQL should be correct. We have used "root" as username and blank ("") as password for all implementations connectivity.

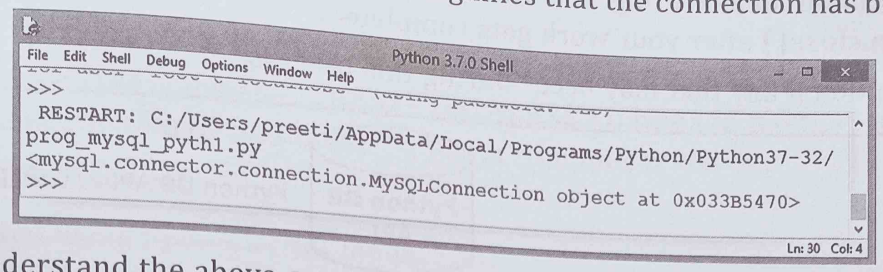
Practical Implementation-1

To establish a connection between MySQLdb and Python:

Create the first program in Python script mode for establishing this connection. Open a new script file in Python, type the following script and run it.



If the output as shown below is obtained, this signifies that the connection has been successfully established.



We will now understand the above commands in detail.

The first statement is to import mysql.connector into your script, which has already been installed on your system. The next statement involves the creation of a connection object 'mydb' once the connectivity gets established through localhost for MySQL with username as 'root' and password as " "(blank) in our case.

Once this statement is executed successfully, mydb connection object is created and the message

`<mysql.connector.connection.MySQLConnection object at 0x033B5470>`

gets displayed, else error message is displayed.

CTM: The **connect()** method creates a connection to the MySQL server and returns a MySQL Connection object.

9.6 CREATING CURSOR OBJECT

The next step for interacting with MySQL through Python is to create a cursor object. It will let us execute all the queries we need. Thus, in order to put our new connection to use, we need to create a cursor object. It is a useful control structure of database connectivity. When we fire any data manipulation query to database, it is executed and the result set (set of records) is sent over the connection in one go. Although we may want to access data one row at a time, query processing cannot happen one row at a time, so cursor helps us perform this task. Cursor stores all the data as a temporary container of returned data and allows traversal so that we can fetch data one row at a time from cursor.

The cursor object is an abstraction specified in the Python DB-API. It gives us the ability to have multiple separate working environments through the same connection to the database.

We need to create the object of a class called cursor that allows Python code to execute database command in a database session.

- Cursors are created by the connection.cursor() method: they are bound to the connection for the entire lifetime and all the commands are executed in the context of the database session wrapped by the connection.

We can create a cursor by executing the 'cursor' function of our database object.

Establishing the connection with MySQL database requires the following steps to be executed (Fig. 9.3):

1. Use **mysql.connector.connect()** method of MySQL Connector Python with required parameters such as host name, username, password and name of the database to connect to MySQL.
2. Use the connection object returned by a connect() method to create a **cursor** object to perform Database Operations.
3. Use/type the **cursor.execute()** to execute SQL queries from Python.

4. Close the Cursor object using a **cursor.close()** and MySQL database connection using **connection.close()** after your work gets completed.
5. Catch Exception if any that may occur during this process.

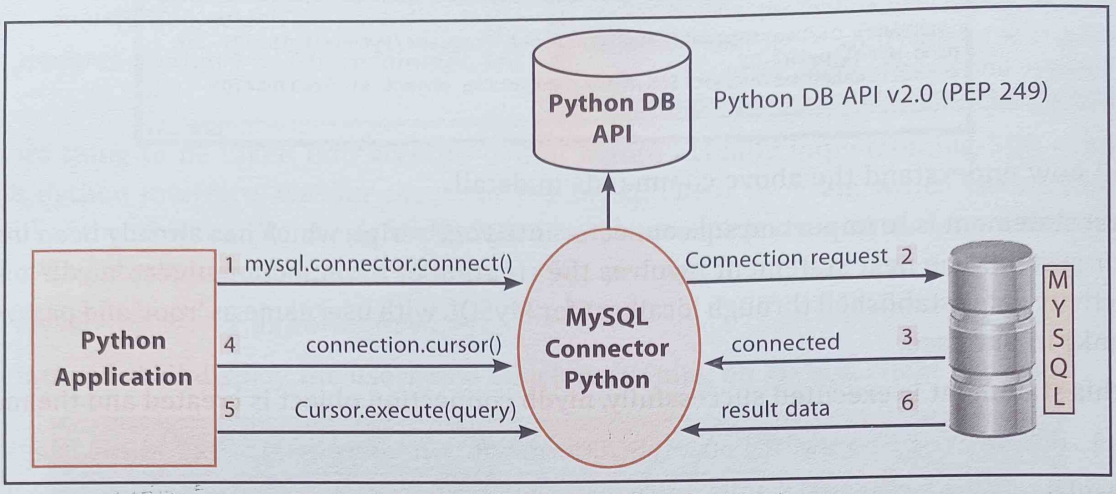


Fig. 9.3: Python-MySQL Database Connection

Arguments required for connecting MySQL database with Python:

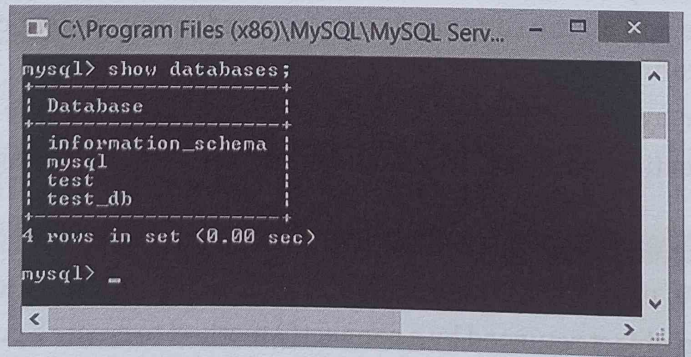
- **Username:** This is the username that you use to work with MySQL Server. The default username for the MySQL database is "**root**".
- **Password:** Password is given by the user at the time of installing the mysql database. If you are using root then you won't need the password.
- **Host Name:** This is server name or IP address on which MySQL is running. If you are running on localhost, then you can use localhost, or its IP, *i.e.*, 127.0.0.0
- **Database Name:** It is the name of the database to which connectivity is to be established. Here we are using a Database named 'school' which we will be creating in the upcoming section.

9.7 CREATING A DATABASE

Once the connection gets established, our next step is to create the new database 'school' in MySQL through Python. Before creating this database, it is advisable to check whether this database already exists or not. This is done by typing the command—

```
SHOW DATABASES;
```

in the front of MySQL prompt (mysql>).



As is evident from the above screenshot, no database named 'school' exists. So, we will now create a database named 'school'.

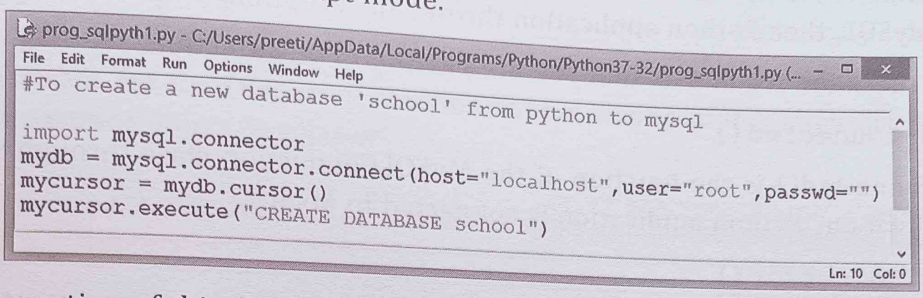
Practical Implementation–2

To create a new database 'school' in MySQL through Python:

This is done by typing the command **"CREATE DATABASE <Database_name>"** after establishing the connection and creation of cursor object. In our example, it will be:

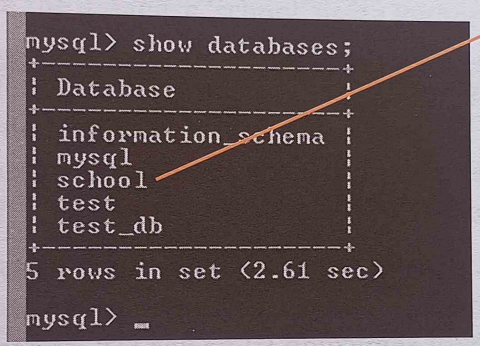
"CREATE DATABASE school"

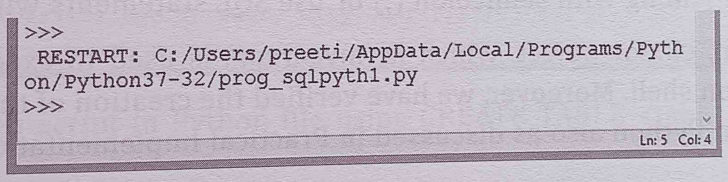
in new file created using Python script mode.



On successful creation of this database, if no error is displayed and the cursor starts blinking in front of the Python shell, this signifies that database has been successfully created.

We can further check the creation of database 'school' by typing **SHOW DATABASES**; command in MySQL prompt. The screenshot given below shows that database '**school**' exists.





Understanding the Python MySQL Database Connection Program

☞ `import mysql.connector`

- This line imports the MySQL Connector Python module in your program so you can use this modules API to connect MySQL.

In case the connection fails, import error shall be displayed—

from mysql.connector import Error

- mysql-connector Error object is used to show us an error when we fail to connect databases or if any other database error occurs while working with the database. Example: ER_ACCESS_DENIED_ERROR when username or password is wrong.

☞ `mysql.connector.connect()`

- Using this function, we can connect the MySQL Database; this function accepts four required parameters: **Host**, **Database**, **User** and **Password** that we have already discussed.

☞ `connect()`

- connect() function establishes a connection to the MySQL database from Python application and returns a MySQLConnection object. Then we can use MySQLConnection object to perform various operations on the MySQL Database.
- connect() function can throw an exception, *i.e.*, Database error, if one of the required parameters is wrong. For example, if you provide a database name that is not present in MySQL, then Python application throws an exception.

  So, check the arguments that you are passing to this function.

☞ `conn.is_connected()`

- is_connected() is the function of the MySQLConnection class through which we can verify if our Python application is connected to MySQL.

☞ `connection.cursor()`

- This method returns a cursor object. Using a cursor object, we can execute SQL queries.
- The MySQLCursor class instantiates objects that can execute operations such as SQL statements.

Cursor objects interact with the MySQL server using a MySQLConnection object.

☞ `cursor().execute()`

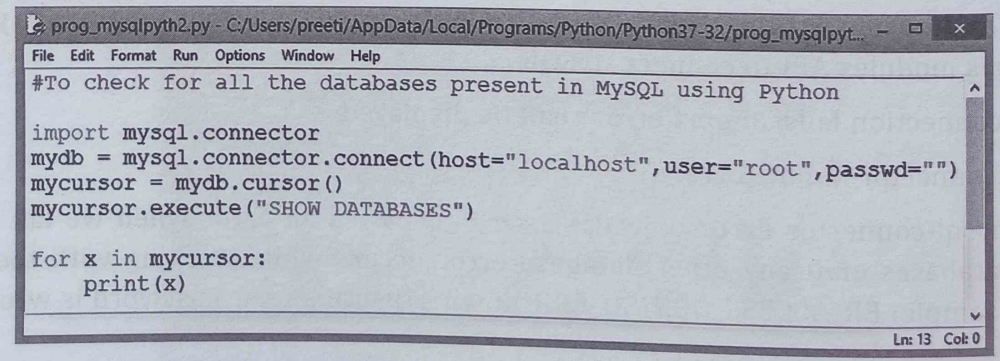
Usually, a cursor in SQL and databases is a control structure to traverse over the records in a database. So it is used for fetching the results. We get the cursor object by calling the cursor() method of connection. An arbitrary number of cursors can be created. The cursor is used to traverse the records from the result set.

Note: The execute() method accepts arguments of SQL statements in String format, *i.e.*, enclosed in double quotes. Argument can be used in any one of the forms: use SQL statements in triple quotes (''') along with semicolon (;) or use SQL statements without semicolon (;).

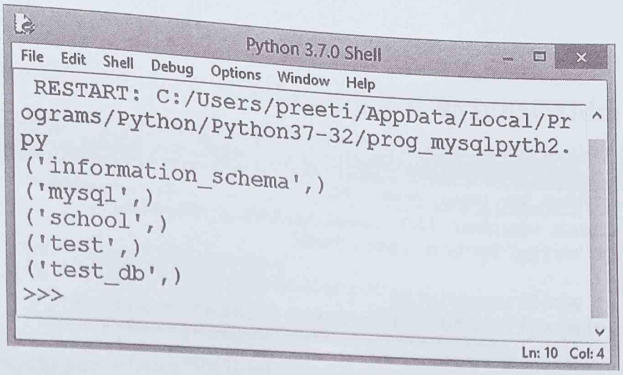
In the above program, cursor object, which is 'mycursor', is used to execute Create Database command using Python shell. Moreover, we have verified the creation of the database from SQL. This can be done from Python also as discussed in Practical Implementation–3.

Practical Implementation–3

To check whether the database has been created or not using Python interface.



Upon execution of the above code, the output window shall display all the databases present in MySQL in the front of Python shell.



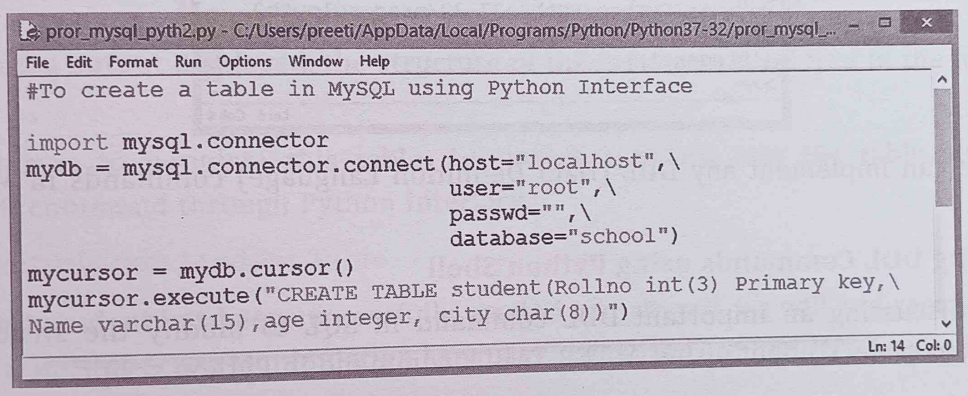
In the above program, **for** loop has been used to traverse among all the databases present inside MySQL using the cursor object 'mycursor'.

CTM: We can execute the SQL queries from Python program using execute() method associated with cursor object.

Once we are done with the creation of 'school' database in MySQL, our next step is to create a table inside this database through Python shell.

Practical Implementation–4

To create a table 'student' inside the database 'school' using Python script mode as the interface.



Type and run the given script in Python file using CREATE TABLE <table_name> command of MySQL in Python.

If no error is displayed, this means the table has been successfully created and can be verified by using **desc <tablename> command**, *i.e.*, desc student; in MySQL prompt which shall display the structure of the table student in MySQL.

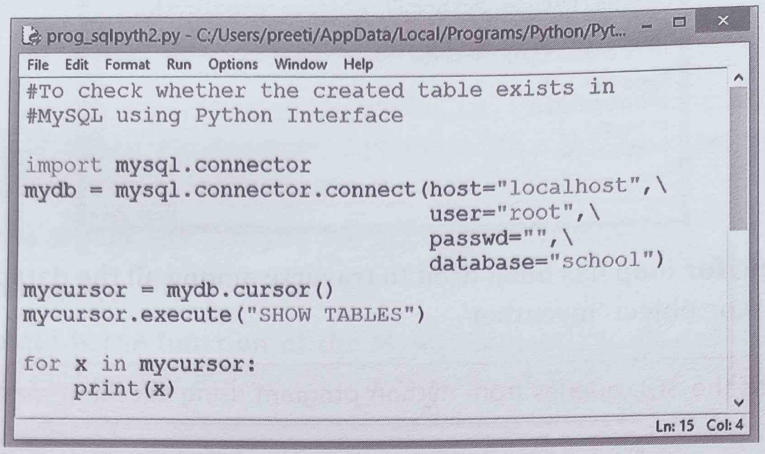
```
mysql> use school;
Database changed
mysql> desc student;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| Rollno | int(3)      | NO   | PRI | NULL    |       |
| Name   | varchar(15) | YES  |     | NULL    |       |
| age    | int(11)     | YES  |     | NULL    |       |
| city   | char(8)     | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
4 rows in set (0.13 sec)

mysql>
```

This can also be done directly from Python shell by typing the script given in Practical Implementation–5.

Practical Implementation–5

To check for the created table 'student' using Python.



In the above program, SHOW TABLES command in SQL is executed using the cursor object 'mycursor' in Python script. This shall display all the tables present inside the database school by using **for** loop for traversing all the tables present inside it and, hence, the output is obtained as shown below.

```
>>>
 RESTART: C:/Users/preeti/AppData/Local/P
rograms/Python/Python37-32/prog_sqlpyth2.
py
('student',)
>>>
Ln: 6 Col: 4
```

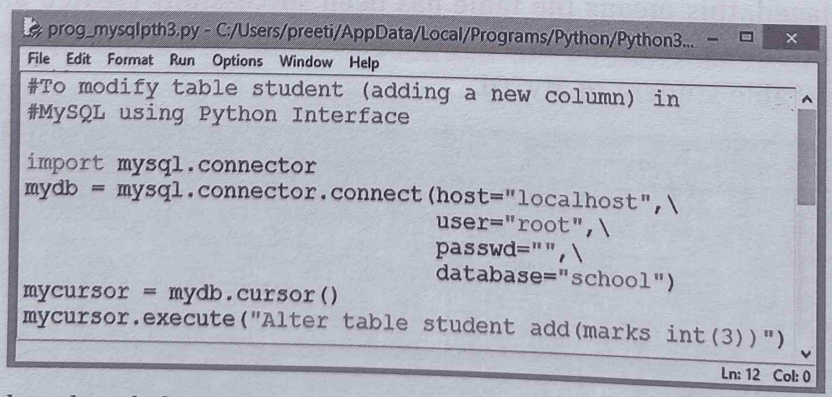
Similarly, we can implement any DDL (Data Definition Language) commands in MySQL using Python.

Implementing DDL Commands using Python Shell

We will be discussing an important DDL command in SQL to modify the structure of the already created table 'student' using ALTER TABLE command in Python script.

Practical Implementation–6

To add a new column 'marks' in the student table.



If no error is displayed and the cursor in the front of Python shell starts blinking, it indicates that the structure of the table student has been successfully modified. This can be verified in MySQL by using 'DESC STUDENT;' and also using Python shell as we will take up in Practical Implementation–7.

Practical Implementation–7

To view the modified structure of student table using Python script.

```
prog_mysqlpyth3.py - C:/Users/preeti/AppData/Local/Programs/Pytho...
File Edit Format Run Options Window Help
#To view the modified structure of table student in
#MySQL using Python Interface

import mysql.connector
mydb = mysql.connector.connect(host="localhost",\
                               user="root",\
                               passwd="",\
                               database="school")
mycursor = mydb.cursor()
mycursor.execute("Desc student")
for x in mycursor:
    print(x)
                                                    Ln: 15  Col: 4
```

Upon the execution of the above script, the following output shall be displayed as shown in the screenshot given below.

```
>>>
 RESTART: C:/Users/preeti/AppData/Local/Programs/
Python/Python37-32/prog_mysqlpyth3.py
('Rollno', 'int(3)', 'NO', 'PRI', None, '')
('Name', 'varchar(15)', 'YES', '', None, '')
('age', 'int(11)', 'YES', '', None, '')
('city', 'char(8)', 'YES', '', None, '')
('marks', 'int(3)', 'YES', '', None, '')
>>>
                                                    Ln: 10  Col: 4
```

As shown in the output displayed, the structure of the table gets displayed in the form of tuple of strings.

Our next step is to populate this table, *i.e.*, insert a record into the table student using INSERT INTO command through Python Interface.

Inserting Records into Student Table

After the table student has been successfully created and altered, we will add records to it using the popular SQL-DML (Data Manipulation Language) command INSERT INTO using Python shell.

Practical Implementation–8

To insert a record into the table student using Python Interface.

```
prog_mysqlpyth5.py - C:/Users/preeti/AppData/Local/Programs/Python/Python37-32/prog_mysqlpyth5.py...
File Edit Format Run Options Window Help
#To insert a new record into table student in
#MySQL using Python Interface

import mysql.connector
mydb = mysql.connector.connect(host="localhost",\
                               user="root",\
                               passwd="",\
                               database="school")

mycursor = mydb.cursor()
mycursor.execute("INSERT INTO student VALUES(1,'Tarun',23,'Mumbai',398)")
mydb.commit()
print(mycursor.rowcount,"Record Inserted")
                                                    Ln: 15  Col: 0
```

```
>>>
 RESTART: C:/Users/preeti/AppData/Local/
Programs/Python/Python37-32/prog_mysqlpy
th5.py
1 Record Inserted
>>>
                                                    Ln: 6  Col: 4
```

We can verify whether this record has been inserted or not using SELECT statement in MySQL.

```
mysql> select * from student;
+--------+-------+-----+--------+-------+
| Rollno | Name  | age | city   | marks |
+--------+-------+-----+--------+-------+
|      1 | Tarun |  23 | Mumbai |   398 |
+--------+-------+-----+--------+-------+
1 row in set (0.00 sec)

mysql>
```

- The **cursor.rowcount** is the property of cursor object that is used in the given program and shall return the number of records inserted which is 1 in our case since we have inserted only one record into the student table.
- For storing String data type values, all the string values should be enclosed in single quotes.

We can insert multiple records also at the same instance through Python.

Practical Implementation–9

To insert multiple records into the table student through Python interface.

prog_mysqlpyth5.py - C:\Users\preeti\AppData\Local\Programs\Python\Python37-32\prog_mysqlpyth5.py (3.7.0)

File Edit Format Run Options Window Help

```
#To insert a new record into table student in
#MySQL using Python Interface

import mysql.connector
mydb = mysql.connector.connect(host="localhost",\
                               user="root",\
                               passwd="",\
                               database="school")
mycursor = mydb.cursor()
mycursor.execute("INSERT INTO student VALUES(2,'Pooja',21,'Chail',390)")
mycursor.execute("INSERT INTO student VALUES(3,'Radhika',18,'Shimla',388)")
mycursor.execute("INSERT INTO student VALUES(4,'Sonia',24,'Goa',300)")
mycursor.execute("INSERT INTO student VALUES(5,'Vinay',25,'Pune',410)")
mycursor.execute("INSERT INTO student VALUES(10,'Shaurya',15,'Delhi',345)")
mydb.commit()
```

Ln: 17 Col: 0

Or

connect1.py - C:/Users/User2/AppData/Local/Programs/Python/Python39/connect1.py (3.9.0)

File Edit Format Run Options Window Help

```
import mysql.connector
mydb=mysql.connector.connect(host="localhost",user="root",passwd="",database="school")
mycursor=mydb.cursor()
mycursor.execute("""INSERT INTO STUDENT (Rollno, Name, age,city, marks)
                VALUES(1,'Tarun',23,'Mumbai',398),
                (2,'Pooja',21,'Chail',390),
                (3,'Radhika',18,'Shimla',388),
                (4,'Sonia',24,'Goa',300),
                (5,'Vinay',25,'Pune',410),
                (10,'Shaurya',15,'Delhi',345)""")
mydb.commit()
```

Ln: 15 Col: 0

```
mysql> select * from student;
+--------+---------+-----+--------+-------+
| Rollno | Name    | age | city   | marks |
+--------+---------+-----+--------+-------+
|      1 | Tarun   |  23 | Mumbai |   398 |
|      2 | Pooja   |  21 | Chail  |   390 |
|      3 | Radhika |  18 | Shimla |   388 |
|      4 | Sonia   |  24 | Goa    |   300 |
|      5 | Vinay   |  25 | Pune   |   410 |
|     10 | Shaurya |  15 | Delhi  |   345 |
+--------+---------+-----+--------+-------+
6 rows in set (0.00 sec)

mysql>
```

In the above code, we have used a function commit() which we will discuss now.

POINT TO REMEMBER

INSERT and **UPDATE** operations are executed in the same way we execute **SELECT** query using **execute()**. However, we must remember one thing: after executing insert or update query, we must **commit** our query using **connection object** with **commit()**.

For example , (if our connection object name is mycon)

```
mycon.commit()
```

This statement is required to make changes in database permanently, otherwise no changes are made to the table.

Methods to Manage MySQL Database Transactions in Python

Python MySQL-Connector provides the following methods to manage database transactions (Fig. 9.4).

- **commit:** MySQLConnection.commit() method sends a COMMIT statement to the MySQL server, committing the current transaction.
- **rollback:** MySQLConnection.rollback() reverts the changes made by the current transaction.
- **autocommit:** MySQLConnection.autocommit value can be assigned as True or False to enable or disable the auto-commit feature of MySQL. By default, its value is False.

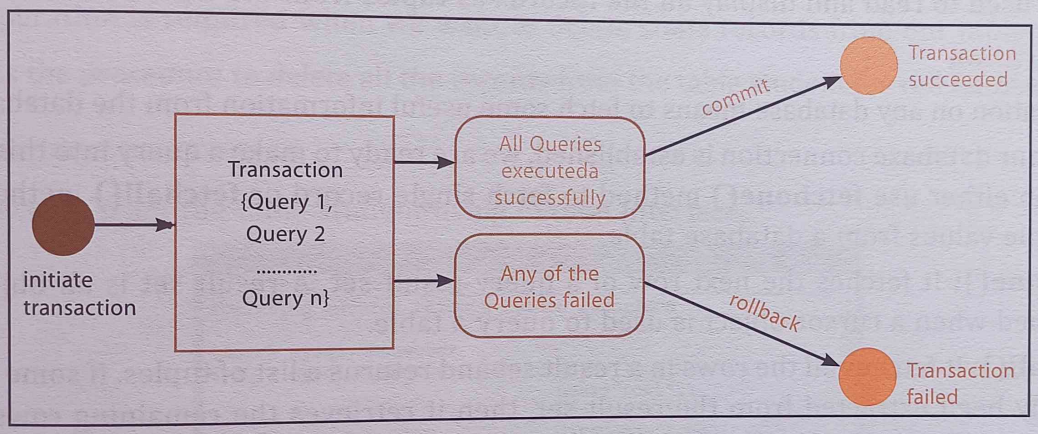


Fig. 9.4: Python-MySQL transaction management using commit

➢ Syntax of commit() method

Once a program has completed executing the query with your changes and you want to commit the changes to the database, then you need to call **commit()** method on MySQL connection object as follows:

```
connection.commit()
```

➢ Syntax of rollback() method

When one of the transactions fails to execute and you want to revert or undo all your changes, then you need to call a rollback method of MySQL connection object as follows:

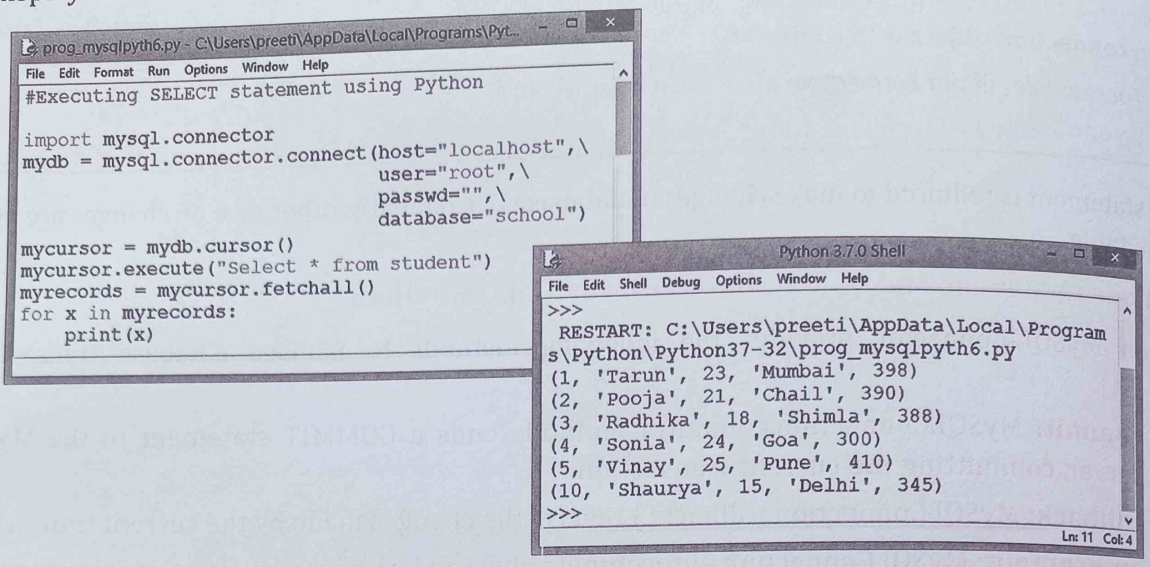
```
connection.rollback()
```

That was all about transaction handling using Python-SQL connectivity.

In Practical Implementation–9, after multiple records have been inserted into the table student, it is checked using SELECT statement in SQL. This can be accomplished in Python as well.

Practical Implementation–10

To display all the records of student table using Python shell.



In the above program, we have used SELECT statement along with fetchall() function. This function is used to read and display all the records as tuples from the table student.

READ Operation

READ Operation on any database means to fetch some useful information from the database.

- Once our database connection is established, we are ready to make a query into this database. We can either use **fetchone()** method to fetch single record or **fetchall()** method to fetch multiple values from a database table.
- **fetchone():** It fetches the next row of a query result set. A result set is an object that is returned when a cursor object is used to query a table.
- **fetchall():** It fetches all the rows in a result set and returns a list of tuples. If some rows have already been extracted from the result set, then it retrieves the remaining rows from the result set. If no more rows are available, it returns an empty list.
- **fetchmany(size):** It fetches the specified number of rows (as a list of tuples) from the result set. The default size is 1; if there are no rows in resultset, [] is returned.
- **rowcount:** This is a read-only attribute and returns the number of rows that were affected by an execute() method.

Using fetchall() method, all the records retrieved from the table student shall get stored in the object 'myrecords' which we can display one by one using **for** loop.

Selection using WHERE clause

We can retrieve selected records from the table using WHERE clause in Python shell also in the same manner as is done in MySQL.

Practical Implementation–11

To implement WHERE clause using Python interface.

This is to be implemented using MySQL Select...Where statement in Python script as shown in the given code:

```
prog_sqlpyth_where1.py - C:/Users/preeti/AppData/Local/Programs/Python/Python37-32/prog_sqlpyth_w...
File Edit Format Run Options Window Help
#Implementing SELECT statement using
#WHERE clause in Python Interface

import mysql.connector
mydb = mysql.connector.connect(host="localhost",\
                               user="root",\
                               passwd="",\
                               database="school")
mycursor = mydb.cursor()
mycursor.execute("Select name,age,marks from student where city='Delhi'")
myrecords = mycursor.fetchall()
for x in myrecords:
    print(x)
                                                                    Ln: 15  Col: 4
```

In the WHERE clause, we have used city='Delhi', *i.e.*, to fetch the records of all the students who are staying in 'Delhi' city. Since there is only one record in the table that matches with the given criteria on, it shall be displayed as shown:

```
>>>
 RESTART: C:/Users/preeti/AppData
_where1.py
('Shaurya', 15, 345)
>>>
```

Deleting Records from a Table in Python-SQL

DELETE operation is required when we want to delete some records from our table.

Following is the procedure to delete all the records from the table student for roll number 1.

Practical Implementation–12

Deleting records from the table student using Python interface.

```
prog_sqlpyth_del1.py - C:/Users/preeti/AppData/Local/Programs/Python/Pyth...
File Edit Format Run Options Window Help
#Deleting records through Python Interface

import mysql.connector
mydb = mysql.connector.connect(host="localhost",\
                               user="root",\
                               passwd="",\
                               database="school")
mycursor = mydb.cursor()
mycursor.execute("DELETE FROM student where Rollno = 1")
mydb.commit()
print(mycursor.rowcount,"Record (s) Deleted")
                                                                    Ln: 14  Col: 0
```

This can be verified using the statement–`SELECT * from student;` in MySQL.

```
mysql> select * from student;
+--------+---------+-----+--------+-------+
| Rollno | Name    | age | city   | marks |
+--------+---------+-----+--------+-------+
|      2 | Pooja   |  21 | Chail  |   390 |
|      3 | Radhika |  18 | Shimla |   388 |
|      4 | Sonia   |  24 | Goa    |   300 |
|      5 | Vinay   |  25 | Pune   |   410 |
|     10 | Shaurya |  15 | Delhi  |   345 |
+--------+---------+-----+--------+-------+
5 rows in set (0.00 sec)
```

As is evident from the above output, the record for roll number 1 has been deleted and, hence, not displayed.

```
>>>
 RESTART: C:/Users/preeti/AppData/
_del1.py
1 Record (s) Deleted
>>>
```

Updating Records in the Table using Python-SQL

UPDATE Operation on any table means to update one or more records which are already available in the table.

The following script updates all the records having Name as '**Vinay**'. Here, we increase AGE of students with name as "Vinay" to 28 years.

Practical Implementation–13

To update the student record using MySQL-Python connectivity.

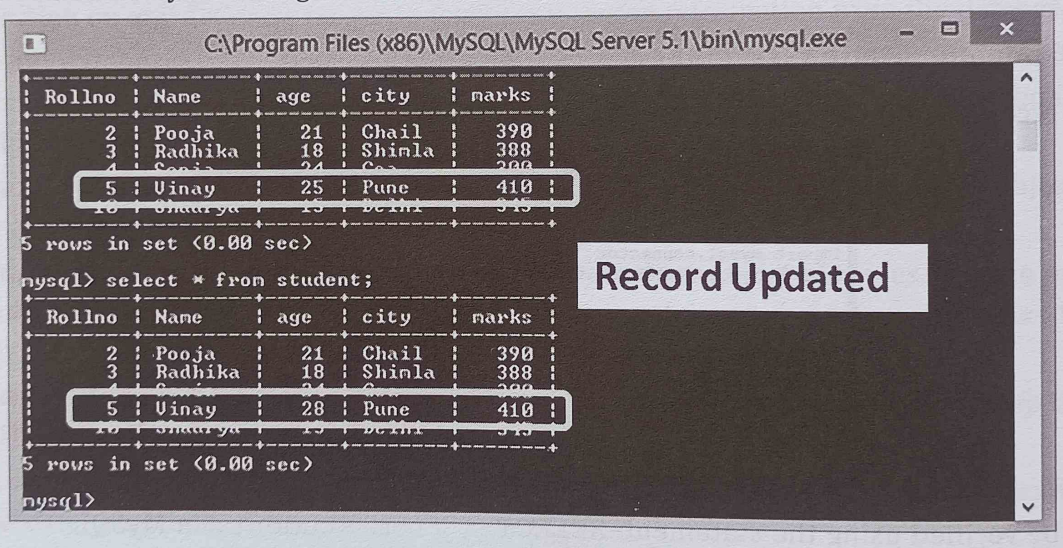
```
*prog_sqlpyth_Updt.py - C:/Users/User2/AppData/Local/Programs/Python/Python39/prog_s...
File Edit Format Run Options Window Help
#Updating records through parameterized query in Python interface

import mysql.connector
mydb = mysql.connector.connect(host="localhost",\
                               user="root",\
                               passwd="",\
                               database="school")

Age=28
Name="Vinay"
data=(Age,Name)
mycursor=mydb.cursor()
query="UPDATE student set age=%s where Name=%s"
mycursor.execute(query,data)
mydb.commit()
print(mycursor.rowcount,"Record(s) Updated")
```

```
>>>
 RESTART: C:/Users/preeti/AppData
_updt.py
1 Record (s) Updated
>>>
```

This can be verified by checking the records in MySQL.



As shown in the output window above, the age of the student 'Vinay' has been updated from 25 to 28 years.

Understanding Python MySQL Parameterized Query Program

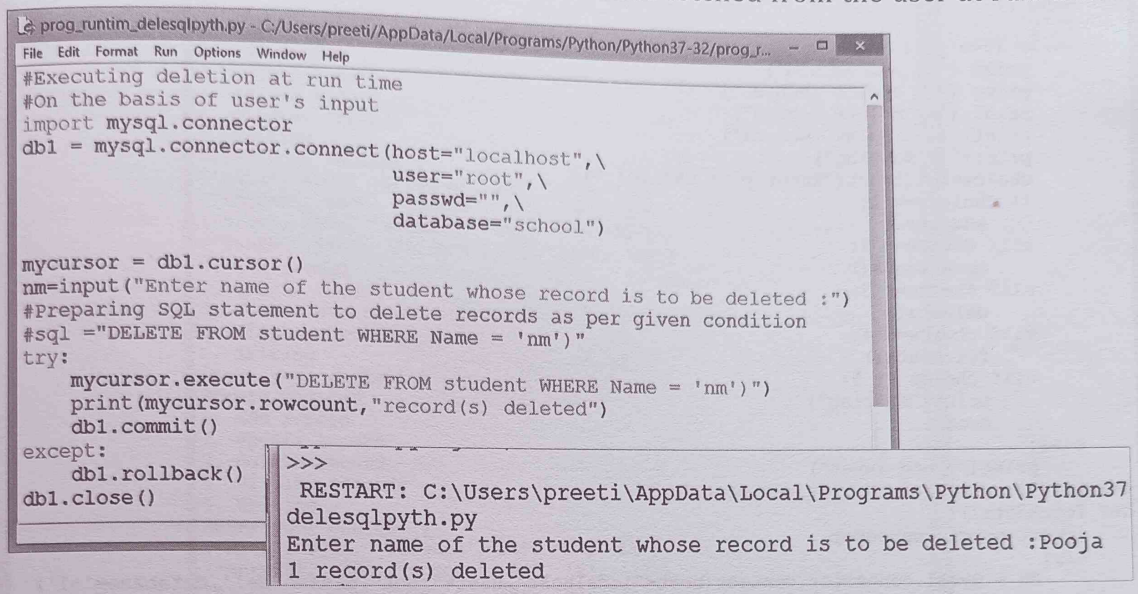
We used the function `mysql.connector.connect()` to connect the MySQL Database. This function accepts the required parameters: host, user, password and database. If a connection is successfully established, it will return the connection object.

- Firstly, we assign values to age and name variable which is to be inserted into the placeholders.
- Then we create the parameterized SQL query. In this query, we are using two **placeholders '%s'** for two columns in a table.
- Next, **we put two placeholders in update query**, one for "name" column and the other for "age" column.

- We then provide values of age and name to the tuple variable input in sequential order and pass it to **mycursor.execute()** function along with SQL update query. Remember, tuple contains user data in sequential order of placeholders.
- In the end, we commit our changes to the database using **connection.commit()**.

Practical Implementation–14

To delete the record of a student on the basis of name fetched from the user at run-time.



Explanation:

We used the function **mysql.connector.connect** to *connect the MySQL Database*. This function accepts the required parameters: host, database, user and password. If a connection is successfully established, it will return the connection object.

- Then we created the parameterized SQL query. In this query, we are using one placeholder.
- Next, we used the prepared statement to accept user input using a placeholder, *i.e.*, **we put one placeholder in delete statement** for "name" column.
- We then added this one column value in the input tuple in sequential order and passed SQL delete query and input tuple with name as 'nm' to **cursor.execute()** function.
- In the end, we commit our changes to the database using **connection.commit()**
- We placed all our code in the try-except block to catch exceptions, if any.

9.8 CLOSING CURSOR AND CONNECTION

Since the database can keep open only a limited number of connections at a time, we must close the connection using **cursorobject.close()**. For example,

`cursor.close()` or `mycursor.close()`

This method closes the cursor using cursor object which resets all results and we cannot execute any SQL statement.

We can disconnect from the database after performing all operations by explicitly calling close() method along with database object. For example,

`conn.close()`

9.9 OPERATIONS ON A TABLE IN A NUTSHELL

Following is the program for performing all the operations on a table 'student' through a menu-driven program.

prog_sqlpyth_complete.py - C:/Users/preeti/AppData/Local/Programs/Python/Python37-32/prog_sqlpyth_complete.py (3.7.0)

File Edit Format Run Options Window Help

```
#Menu-driven program to demonstrate four major operations
#performed on a table through MySQL-Python connectivity
def menu():
    c='y'
    while (c=='y'):
        print ("1. add record")
        print ("2. update record ")
        print ("3. delete record")
        print("4. display records")
        print("5. Exiting")
        choice=int(input("Enter your choice: "))
        if choice == 1:
            adddata()
        elif choice== 2:
            updatedata()
        elif choice== 3:
            deldata()
        elif choice== 4:
            fetchdata()
        elif choice == 5:
            print("Exiting")
            break
    else:
        print("wrong input")
c=input("Do you want to continue or not: ")
def fetchdata():
    import mysql.connector
    try:
        db = mysql.connector.connect(host="localhost",user="root",password='',database='s1')
        cursor = db.cursor()
        cursor.execute("SELECT * FROM student" )
        results = cursor.fetchall()
        for x in results:
            print(x)
    except:
        print ("Error: unable to fetch data")

def adddata():
    import mysql.connector
    db = mysql.connector.connect(host='localhost',user='root',password='',database='s1')
    cursor = db.cursor()
    cursor.execute("INSERT INTO student VALUES('Ritu',4000,'Science',345,'B','11')")
    cursor.execute("INSERT INTO student VALUES('Ankush',6000,'Commce',445,'A','12')")
    cursor.execute("INSERT INTO student VALUES('Pihu',3566,'Humanis',446,'A','11')")
    cursor.execute("INSERT INTO student VALUES('Tinku',8900,'Science',545,'A+','12')")
    db.commit()
    print("Records added")

def updatedata():
    import mysql.connector
    try:
        db = mysql.connector.connect(host="localhost",user="root",password='',database='s1')
        cursor = db.cursor()
        sql = ("Update student set stipend=5000 where name='Ritu'")
        cursor.execute(sql)
        print("Record Updated")
        db.commit()
    except Exception as e:
        print (e)
def deldata():
    import mysql.connector
    db = mysql.connector.connect(host="localhost",user="root",password='',database='s1')
    cursor = db.cursor()
    sql = "delete from student where name='Ritu'"
    cursor.execute(sql)
    print("Record Deleted")
    db.commit()

menu()
```

Ln: 44 Col: 26

```
File  Edit  Shell  Debug  Options  Window  Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.19
14 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:/Users/preeti/AppData/Local/Programs/Python/Python37-
32/prog_sqlpyth_complete.py
Do you want to continue or not: y
1. add record
2. update record
3. delete record
4. display records
5. Exiting
Enter your choice: 1
Records added
1. add record
2. update record
3. delete record
4. display records
5. Exiting
Enter your choice: 4
('Ritu', 4000, 'Science', 345, 'B', '11')
('Ankush', 6000, 'Commce', 445, 'A', '12')
('Pihu', 3566, 'Humanis', 446, 'A', '11')
('Tinku', 8900, 'Science', 545, 'A+', '12')
1. add record
2. update record
3. delete record
4. display records
5. Exiting
Enter your choice: 2
Record Updated
1. add record
2. update record
3. delete record
4. display records
5. Exiting
Enter your choice: 4
('Ritu', 5000, 'Science', 345, 'B', '11')
('Ankush', 6000, 'Commce', 445, 'A', '12')
('Pihu', 3566, 'Humanis', 446, 'A', '11')
('Tinku', 8900, 'Science', 545, 'A+', '12')
1. add record
2. update record
3. delete record
4. display records
5. Exiting
Enter your choice: 3
Record Deleted
1. add record
2. update record
3. delete record
4. display records
5. Exiting
Enter your choice: 4
('Ankush', 6000, 'Commce', 445, 'A', '12')
('Pihu', 3566, 'Humanis', 446, 'A', '11')
('Tinku', 8900, 'Science', 545, 'A+', '12')
1. add record
2. update record
3. delete record
4. display records
5. Exiting
Enter your choice: 5
Exiting
>>>                                                    Ln: 63  Col: 4
```

MEMORY BYTES

- We use MySQL-Connector Python to connect MySQL.
- **mysql.connector.connect()** method of MySQL-Connector Python is used with the required parameters to connect MySQL.
- MySQL-Connector Python requires Python to be in the system's PATH. Installation fails if it doesn't find Python.
- Install MySQL-Connector Python using pip command.
- MySQLdb is an interface for connecting to a MySQL database server from Python. It implements the Python Database API and is built on top of the MySQL C API.

- 'module_name.connect' is used to make a connection to the database that you wish to use.
- In order to put our new connection to good use, we need to create a cursor object. It gives us the ability to have multiple separate working environments through the same connection to the database.
- Once a Database Connection is established, we are ready to create tables using execute() method of the created cursor.
- Read operation on any table means to fetch some useful information from the table.
- Use fetchall() method to fetch multiple values from a database table.
- fetchone() fetches the next row of a query result set. A result set is an object that is returned when a cursor object is used to query a table.
- rowcount is a read-only attribute and returns the number of rows that were affected by an execute() method.
- UPDATE Operation on any database means to update one or more records, which are already available in the table.
- To disconnect Database Connection, use close() method.

OBJECTIVE TYPE QUESTIONS

1. Fill in the blanks.

(a) ..... methods will return only one row from the result set in the form of a tuple containing a record.

(b) ..... method returns the result set in the form of tuples containing the records or rows returned by the sql table.

(c) A ..... refers to a logical set of records that is fetched from the database.

(d) A session between the application program and the database is called ..... .

(e) A ..... is a special control structure that facilitates the row-by-row processing of records in the result set.

(f) A ..... query is used to check if data has been added to the table or not.

(g) To connect MySQL database ..... script is used.

(h) The ..... is a property of cursor object that returns the number of rows allowed from the cursor so far.

(i) ..... package can be imported in place of mysql-connector.

(j) A ..... is a collection of organized information that can be easily used, managed, updated, and they are classified according to their organizational approach.

2. State whether the following statements are True or False.

(a) Read operation on any table means to fetch some useful information from the table.

(b) Use fetchall() method to retrieve only one value from a database table.

(c) rowcount is a read-only attribute.

(d) To disconnect database connection, use connect() method.

(e) Update statement is used to insert data into the table.

(f) A resultset is an object that is returned when a cursor object is used to query a table.

(g) After establishing connection, execute() method is used.

(h) fetchone() fetches the next row of a query result set.

(i) DB-API is the standard for Python's database interface.

(j) connect() method does not require any parameters.

3. Multiple Choice Questions (MCQs)

(a) Which connector is used for linking the database with Python code?

(i) MySQL-connector (ii) YesSQL: connector

(iii) PostSQL: connector (iv) None of these

(b) To open a connector to Mysql database, which statement is used to connect with mysql?

(i) connector (ii) connect (iii) password (iv) username

(c) Which software is used for coding the Python language?

(i) Idea (ii) IDLE (iii) Python Script (iv) Connector

(d) Which command is used for counting the number of rows in a database?
(i) row (ii) rowcount (iii) count() (iv) row_count

(e) Name the method which is used for displaying only one resultset.
(i) fetchmany (ii) fetchno (iii) fetchall (iv) fetchone

(f) To execute all the rows from the result set, which method is used?
(i) fetchall() (ii) fetchone() (iii) fetchmany() (iv) none of these

(g) Pick the correct username used for logging in database.
(i) root (ii) local (iii) directory (iv) host

(h) Name the host name used for signing in the database.
(i) localhost (ii) localpost (iii) localcost (iv) none of these

(i) The command used for modifying the records is:
(i) update() (ii) add() (iii) updateall() (iv) none of these

(j) Which command is used for cleaning up the environment?
(i) my.closed() (ii) is.end() (iii) con.quit() (iv) mycon.close()

(k) Which function of connection is used to check whether connection to mysql is successfully done or not?

```
import mysql.connector as mycon
con = mycon.connect                #ConnectionString
if ..........................:
    print("Connected!")
else:
    print("Error! Not Connected")
```

(i) con.connected() (ii) con.isconnected()
(iii) con.is_connected() (iv) con.is_connect()

(l) Identify the correct statement to create cursor:

```
import mysql.connector as msq
con = msq.connect()                #ConnectionString
mycursor = ............
```

(i) con.cursor() (ii) con.create_cursor() (iii) con.open_cursor() (iv) con.get_cursor()

SOLVED QUESTIONS

1. Which file do we import in Python script to establish MySQL-Python connectivity?

Ans. mysql-connector or mysqldb.

2. What is the significance of using connect()?

Ans. connect() function is used to connect or establish a connection with MySQL database using Python script.

3. What is the role of execute()?

Ans. The role of execute() is execution of MySQL queries such as Create, Insert, Delete, Select, etc., along with Python interface.

4. What are the features of MySQL?

Ans. MySQL provides cross-platform support with a wide range of interfaces for application programming, and has many stored procedures like triggers and cursors that help in managing the database.

5. What are the basic steps to connect with MySQL using table Members? Also retrieve the total number of members whose ID is 1.

Ans. The basic steps to connect Python script with MySQL (table Members) are:
(i) Open the Python script and import any one package for database connectivity: MySQLdb or mysql-connector.
(ii) Create a connection to database using connect() method and provide parameters such as host, user, password and database name.
(iii) Create a cursor object.
(iv) Execute MySQL query (here retrieving total number of members whose ID is 1).
(v) Retrieve data from resultset variable and display using print() command.
(vi) Close the connection environment.

For example:

```
import MySQLdb
conn = MySQLdb.connect(host='host', user='user', passwd='passwd', db='db')
cursor = conn.cursor()
cursor.execute('SELECT COUNT(MemberID) as count FROM Members WHERE id = 1')
row = cursor.fetchone()
print(row)
conn.close()
```

6. What is Python's database interface known as?

Ans. DB-API is the standard for Python's database interface. Database Application Programming Interface is a set of tools used by an Application program to communicate with the Operating System or other programs such as Database Management System.

7. What does database DB-API include?

Ans. Using Python structure, DB-API provides standard and support for working with databases. The API working consists of the following steps:

(i) Bring in the API module
(ii) Obtain database connection
(iii) Issue SQL statements and then store procedures
(iv) Close the connection

8. In the following connection string, identify the elements:
connect(<<1>> = 127.0.0.1, <<2>> = "root", <<3>> = "admin")

Ans. <<1>> host, <<2>> user, <<3> passwd
connect(host= 127.0.0.1, user= 'root', passwd= 'admin')

9. Explain the benefits of Python Database Programming.

Ans. (i) Programming in Python is considerably simple and efficient as compared to other languages. Likewise, Database programming is much easier using Python-MySQL connectivity.
(ii) Python database is portable and the program is also portable, which is an advantage in terms of portability.
(iii) Python supports SQL cursors.
(iv) It also supports Relational Database Systems.
(v) The API of Python for the database is also compatible with other databases.
(vi) It is platform-independent.

10. How can we implement MySQL Database?

Ans. To use MySQL database using Python, we need to first install it on our machine and then type the script given below to implement MySQL in the program:

```
import MySQLdb
```

11. Explain the various database operations one can perform using MySQL-Python connectivity.

Ans. There are various operations that can be performed within a Python program. To deal with these statements, one must have a good knowledge of Database programming and SQL.

Database Operations Environment Variables	Description
INSERT	It is an SQL statement used to create a record into a table.
READ	Fetches useful information from the database.
UPDATE	It is used to update already existing or available record(s).
DELETE	It is used to delete records from the database.
ROLLBACK	It works like "undo" which reverts all the changes that you have made.

12. Consider the information stored in the table 'EMP':

EMPNO	ENAME	DEPT	SALARY
1	ALEX	MUSIC	60000
2	PETER	ART	67000
3	JOHNY	WE	55000
4	RAMBO	P&HE	48000

The following Python code is written to access the records of table 'EMP'. What will be the output of the following code:

```
# Assume all basic setup related to connection and cursor creation is already done.
query="select * from EMP"
mycursor.execute(query)
results = mycursor.fetchone()
results = mycursor.fetchone()
results = mycursor.fetchone()
d = int(results[3])
print(d*3)
```

Ans. 165000

13. Consider the given code and give the output:

```
import mysql.connector as mys
mycon = mys.connect(host='localhost',user='root',passwd='admin',database='company')
mycursor = mycon.cursor()
mycursor.execute("select * from emp")
mydata = mycursor.fetchone()
nrec = mycursor.rowcount
print("Total records fetched so far are",nrec)
mydata = mycursor.fetchone()
nrec = mycursor.rowcount
print("Total records fetched so far are",nrec)
mydata = mycursor.fetchmany(2)
nrec = mycursor.rowcount
print("Total records fetched so far are",nrec)
```

Ans. Total records fetched so far are 1
Total records fetched so far are 2
Total records fetched so far are 4

14. Consider a database 'company' that has a table 'Emp' that stores IDs of employees. Write a MySQL-Python connectivity to retrieve data, one record at a time, for employees with IDs less than 10.

Ans.
```
import MySQLdb as mycon
try:
  db = mycon.connect(host="localhost", user="root", passwd="",
                     database="company")
  cursor = db.cursor()
  sql = "select * from Emp where id < 10"
  number_of_rows = cursor.execute(sql)
  print(cursor.fetchone()) # fetch the first row only
  db.close()
except mycon.DataError as e:
  print("DataError")
  print(e)
```

15. Write code to connect to a MySQL database namely School and then fetch all those records from table Student where grade is 'A'.

Ans.
```
import mysql.connector as a
mydb = a.connect(host="localhost",user="root",passwd=" ", database = "School")
cur = mydb.cursor()
run = "select * from Student where grade = 'A' "
cur.execute(run)
data = cur.fetchall()
for i in data :
   print(i)
mydb.close()
```

16. Explain what will the following query do?

```
import mysql.connector
db = mysql.connector.connect(….) #connection string
cursor = db.cursor()
person_id = input("Enter required person id")
lastname = input("Enter required lastname")
```

```
cursor.execute("INSERT INTO staff (person_id, lastname) VALUES ({}, '{}') ".
format (person_id, lastname))
db.commit()
db.close()
```

Ans. The above program shall insert a new record in table staff with input being fetched from the user for the fields person_id and last name on execution.

17. What are the basic steps to connect Python with MySQL using table Members present in the database 'Society'?

Ans.
```
import MySQLdb
conn = MySQLdb.connect(host="localhost",user='root',passwd=' ',\
                       database="Society")
cursor= conn.cursor()
cursor.execute('SELECT COUNT(MemberID) as count FROM Members
                WHERE id = 1')
row = cursor.fetchone()
print(row)
conn.close()
```

18. SS Public School is managing student data in 'Student' table in 'school' database. Write a Python code that connects to database school and retrieves all records and displays total number of students.

Ans.
```
import mysql.connector
conn = mysql.connector.connect(host="localhost",user='root',
                               passwd =''database="school")
cursor= conn.cursor()
cursor.execute("Select * from Student")
records = cursor.fetchall()
count=0
for x in records:
    count+=1
    print(x)
print("Total number of records are:",count)
conn.close()
```

UNSOLVED QUESTIONS

1. Explain the following 'results' retrieval methods with examples.
   A. `fetchone()`
   B. `rowcount()`
   C. `fetchall()`
2. What is the significance of connecting Python with MySQL?
3. Why do we use connect() function?
4. Explain the steps for establishing MySQL connection with Python.
5. Explain the transaction keywords used with MySQL-Python connectivity.
6. Give the significance of using execute() function.
7. Differentiate between commit() and rollback() statements.
8. Which function is used to connect to database?
9. Which function is used to run the SQL query?
10. Which function is used to read one record from the database?
11. Write a program to display all records in ascending order of their salary from table employee.
12. Write a program to increase salary of the employee, whose name is "MANOJ KUMAR", by ₹ 3000.
13. Write a program to delete the employee record whose name is read from keyboard at execution time.
14. Create a database TESTDB.
    - Create a table EMPLOYEE with Fields FIRST_NAME, LAST_NAME, AGE, SEX and INCOME.

CASE-BASED/SOURCE-BASED INTEGRATED QUESTIONS

1. ABC Infotech Pvt. Ltd. needs to store, retrieve and delete the records of its employees. Develop an interface that provides front-end interaction through Python, and stores and updates records using MySQL.

   The operations on MySQL table "emp" involve reading, searching, updating and deleting the records of employees.

   (a) Program to read and fetch all the records from EMP table having salary more than ₹ 70000.

Ans.

File Edit Format Run Options Window Help

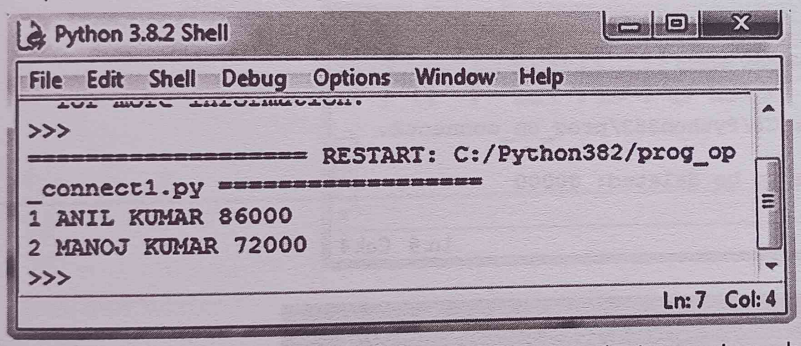
```
#Program to read and search record of employees having salary more than 70000
import mysql.connector
db1 = mysql.connector.connect(host="localhost", user="root", password="",database="company")
cursor = db1.cursor()

sql = "SELECT * FROM EMP WHERE SALARY > 70000;"

try:
    cursor.execute(sql)
    #using fetchall() function to fetch all records from the table EMP and store in resultset
    resultset = cursor.fetchall()
    for row in resultset:
        empno = row[0]
        ename = row[1]
        salary = row[2]

    print(("empno=%d, ename=%s, salary=%f") % (empno, ename, salary))
except:
    print("Error: unable to fetch data")
db1.close()
```

Note: To access data in individual columns, use the index. Also, notice that data in column is converted to its equivalent Python type.



   (b) Program to update the records of employees by increasing salary by ₹ 1000 of all those employees who are getting less than ₹ 80000.

Ans.

File Edit Format Run Options Window Help

```
#Program to update the salaries of all employees
#who have salary less than 80000
import mysql.connector
db1 = mysql.connector.connect(host="localhost",user="root",password="",database="company")
cursor = db1.cursor()
#Preparing SQL statement to increase salary of all employees whose salary is less than 80000
sql = "UPDATE EMP SET salary = salary +1000 WHERE salary<80000;"

try:
    cursor.execute(sql)
    db1.commit()
except:
    db1.rollback()
db1.close()
```

```
mysql> select * from emp;
+-------+--------------+--------+
| empno | ename        | salary |
+-------+--------------+--------+
|     1 | ANIL KUMAR   |  86000 |
|     2 | MANOJ KUMAR  |  72000 |
+-------+--------------+--------+
2 rows in set (0.03 sec)

mysql> select * from emp;
+-------+--------------+--------+
| empno | ename        | salary |
+-------+--------------+--------+
|     1 | ANIL KUMAR   |  86000 |
|     2 | MANOJ KUMAR  |  73000 |
+-------+--------------+--------+
2 rows in set (0.00 sec)
```

(c) Program to delete the record on the basis of inputted salary.

Ans.

```
File Edit Format Run Options Window Help
import mysql.connector
db1 = mysql.connector.connect(host="localhost", user="root", password="",database="company")
cursor = db1.cursor()
sal = int(input("Enter salary whose record to be deleted: "))
#Preparing SQL Statement to delete records as per given condition
sql = "DELETE FROM EMP WHERE salary < '%d';" %(sal)
try:
    cursor.execute(sql)
    print(cursor.rowcount, end=" record(s) deleted")
    db1.commit()
except:
    db1.rollback()
db1.close()
```

```
Python 3.8.2 Shell
File Edit Shell Debug Options Window Help
>>>
========== RESTART: C:/Python382/prog_op_connect2.
py ==========
Enter salary whose record is to be deleted: 80000
>>>
                                                Ln: 6 Col: 4
```

```
mysql> select * from emp;
+-------+--------------+--------+
| empno | ename        | salary |
+-------+--------------+--------+
|     1 | ANIL KUMAR   |  86000 |
|     2 | MANOJ KUMAR  |  72000 |
+-------+--------------+--------+
2 rows in set (0.03 sec)

mysql> select * from emp;
+-------+--------------+--------+
| empno | ename        | salary |
+-------+--------------+--------+
|     1 | ANIL KUMAR   |  86000 |
+-------+--------------+--------+
1 row in set (0.00 sec)
```