

Python Programming

Unit-V

Database Programming

- **Introduction:**

To build the real world applications, connecting with the databases is the necessity for the programming languages. However, python allows us to connect our application to the databases like MySQL, SQLite, MongoDB, and many others.

Python also supports Data Definition Language (DDL), Data Manipulation Language (DML) and Data Query Statements. For database programming, the Python DB-API is a widely used module that provides a database application programming interface.

The Python programming language has powerful features for database programming, those are

- Database Programming in Python is arguably more efficient and faster compared to other languages.
- Python is famous 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.

Environment Setup:

In this topic we will discuss Python-MySQL database connectivity, and we will perform the database operations in python. . The Python DB API implementation for MySQL is possible by MySQLdb or mysql.connector.

Note: The Python DB-API implementation for MySQL is possible by **MySQLdb** in python2.x but it deprecated in python3.x. In Python3.x, DB-API implementation for MySQL is possible by **mysql.connector**.

- ***You should have MySQL installed on your computer***

Windows:

You can download a free MySQL database at <https://www.mysql.com/downloads/>.

Linux(Ubuntu):

sudo apt-get install mysql-server

- ***You need to install mysql.connector***

To connect the python application with the MySQL database, we must import the mysql.connector module in the program.

The mysql.connector is not a built-in module that comes with the python installation. We need to install it to get it working.

Execute the following command to install it using pip installer.

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

(OR)

- ***You need to install MySQLdb:***

MySQLdb is an interface for connecting to a MySQL database server from Python. The **MySQLdb** is not a built-in module that comes with the python installation. We need to install it to get it working.

Execute the following command to install it using pip installer.

- For Ubuntu, use the following command -

```
$sudo apt-get install python2.7-mysqldb
```

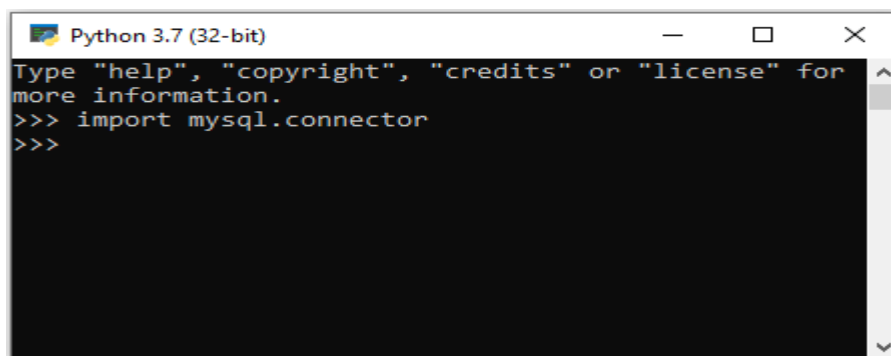
- For Windows command prompt, use the following command -

```
pip install MySQL-python
```

- To test if the installation was successful, or if you already have "MySQL Connector" installed, execute following python statement at terminal or CMD.

```
import mysql.connector
```

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

A screenshot of a terminal window titled "Python 3.7 (32-bit)". The window has a black background with white text. At the top, it says "Type 'help', 'copyright', 'credits' or 'license' for more information." Below that, the user has entered the command ">>> import mysql.connector" and the prompt ">>>" appears again on the next line, indicating the command was executed successfully without any error messages.

```
Python 3.7 (32-bit)
Type "help", "copyright", "credits" or "license" for
more information.
>>> import mysql.connector
>>>
```

Hence, we have successfully installed mysql-connector for python on our system.

- In the same way to test if the installation was successful, or if you already have "MySQLdb" installed, execute following python statement at terminal or CMD.

```
import MySQLdb
```

If the above statement was executed with no errors, "MySQLdb" is installed and ready to be used.

- **Python Database Application Programmer's Interface (DB-API):**

Python DB-API is independent of any database engine, which enables you to write Python scripts to access any database engine. The Python DB API implementation for MySQL is possible by MySQLdb or mysql.connector.

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

- Import module(**mysql.connector** or **MySQLdb**)
- Create the connection object.
- Create the cursor object
- Execute the query
- Close the connection

- **Import module(mysql.connector or MySQLdb):**

To interact with MySQL database using Python, you need first to import **mysql.connector** or **MySQLdb** module by using following statement.

- **MySQLdb(in python2.x)**

import MySQLdb

- **mysql.connector(in python3.x)**

import mysql.connector

- **Create the connection object:**

After importing **mysql.connector** or **MySQLdb** module, we need to create connection object, for that python DB-API supports one method i.e. **connect ()** method.

It creates connection between MySQL database and Python Application.

- If you import **MySQLdb**(in python2.x) then we need to use following code to create connection.

Syntax:

*conn-name= MySQLdb.**connect** (<host-name>,<username>,<password>,<database>)*

Example:

*Myconn =MySQLdb.**connect** ("localhost","root","root","emp")*

(Or)

- If you import **mysql.connector**(in python3.x) then we need to use following code to create connection.

Syntax:

*conn-name= mysql.connector.**connect** (**host**=<host-name>,
user=<username>,**passwd**=<pwd>,**database**=<dbname>)*

Example:

*myconn=mysql.connector.**connect**(**host**="localhost",**user**="root",**passwd**="root",**database**="emp")*

- **Create the cursor object:**

After creation of connection object we need to create cursor object to execute SQL queries in MySQL database. The cursor object facilitates us to have multiple separate working environments through the same connection to the database.

The Cursor object can be created by using **cursor ()** method.

Syntax:

```
cur_came = conn-name.cursor()
```

Example:

```
my_cur=myconn.cursor()
```

- **Execute the query:**

After creation of cursor object we need to execute required queries by using cursor object. To execute SQL queries, python DB-API supports following method i.e. **execute ()**.

Syntax:

```
cur-name.execute(query)
```

Example:

```
my_cur.execute ("select * from Employee")
```

- **Close the connection:**

After completion of all required queries we need to close the connection.

Syntax:

```
conn-name.close()
```

Example:

```
conn-name.close()
```

➤ **MySQLdb(in python2.x):**

MySQLdb is an interface for connecting to a MySQL database server from Python. The following are example programs demonstrate interactions with MySQL database using **MySQLdb** module.

Note – Make sure you have root privileges of MySQL database to interact with database.i.e. User and password.

We are going to perform the following operations on MySQL database.

- Show databases
- Create database
- Create table
- To insert data into table
- Read/Select data from table
- Update data in table
- Delete data from table

Example Programs

- **To display databases :**

We can get the list of all the databases by using the following MySQL query.

>show databases;

Example: – showdb.py

```
import MySQLdb

#Create the connection object
myconn = MySQLdb.connect("localhost","root","root")

#creating the cursor object
cur = myconn.cursor()

#executing the query
dbs = cur.execute("show databases")

#display the result
for x in cur:
    print(x)

#close the connection
myconn.close()
```

Output:

```
>>>python showdb.py
('information_schema',)
('mysql',)
('performance_schema',)
('phpmyadmin',)
('test',)
('Sampledb',)
```

- **To Create database :**

The new database can be created by using the following SQL query.

> create database <database-name>

Example: – createdb.py

```
import MySQLdb

#Create the connection object
myconn = MySQLdb.connect("localhost","root","root")

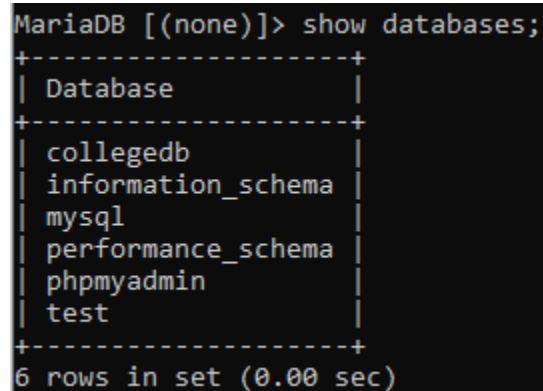
#creating the cursor object
cur = myconn.cursor()

#executing the query
cur.execute("create database Collegedb")
print("Database created successfully")

#close the connection
myconn.close()
```

Output:

```
>>>python createdb.py
Database created successfully
```



```
MariaDB [(none)]> show databases;
+-----+
| Database |
+-----+
| collegedb |
| information_schema |
| mysql |
| performance_schema |
| phpmyadmin |
| test |
+-----+
6 rows in set (0.00 sec)
```

- **To Create table :**

The new table can be created by using the following SQL query.

> create table <table-name> (column-name1 datatype, column-name2 datatype,...)

Example: – createtable.py

```
import MySQLdb
```

```
#Create the connection object
```

```
myconn = MySQLdb.connect("localhost","root","root","Collegedb")
```

```
#creating the cursor object
```

```
cur = myconn.cursor()
```

```
#executing the query
```

```
cur.execute("create table students(sid varchar(20) primary key,sname varchar(25),age int(10))")
```

```
print("Table created successfully")
```

```
#close the connection
```

```
myconn.close()
```

Output:

```
>>>python createtable.py
```

```
Table created successfully
```

```
MariaDB [(none)]> use Collegedb
Database changed
MariaDB [Collegedb]> show tables;
+-----+
| Tables_in_collegedb |
+-----+
| students             |
+-----+
1 row in set (0.00 sec)

MariaDB [Collegedb]>
```

- **To Insert data into table :**

The data can be inserted into table by using the following SQL query.

> *insert into <table-name> values (value1, value2,...)*

Example: – insertdata.py

```
import MySQLdb
```

```
#Create the connection object
```

```
myconn = MySQLdb.connect("localhost","root","root","Collegedb")
```

```
#creating the cursor object
```

```
cur = myconn.cursor()
```

```
#executing the queries
```

```
cur.execute("INSERT INTO students VALUES ('501', 'ABC', 23)")
```

```
cur.execute("INSERT INTO students VALUES ('502', 'XYZ', 22)")
```

```
#commit the transaction
```

```
myconn.commit()
```

```
print("Data inserted successfully")
```

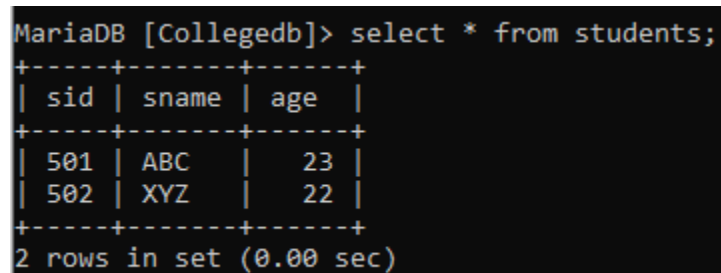
```
#close the connection
```

```
myconn.close()
```

Output:

```
>>>python insertdata.py
```

```
Data inserted successfully
```



```
MariaDB [Collegedb]> select * from students;
+-----+-----+-----+
| sid | sname | age |
+-----+-----+-----+
| 501 | ABC   | 23  |
| 502 | XYZ   | 22  |
+-----+-----+-----+
2 rows in set (0.00 sec)
```


- **To Read/Select data from table :**

The data can be read/select data from table by using the following SQL query.

>select column-names from <table-name>

Python provides the fetchall() method and fetchone() method.

- **fetchall()** method returns all rows in the table. We can iterate the result to get the individual rows.
- **fetchone()** method returns one row from table.

Example: – selectdata.py

```
import MySQLdb

#Create the connection object
myconn = MySQLdb.connect("localhost","root","root","Collegedb")

#creating the cursor object
cur = myconn.cursor()

#executing the query
cur.execute("select * from students")

#fetching all the rows from the cursor object
result = cur.fetchall()
print("Student Details are :")

#printing the result
for x in result:
    print(x);

#close the connection
myconn.close()
```

Output:

```
>>>python selectdata.py
```

```
Student Details are:
('501', 'ABC', 23)
('502', 'XYZ', 22)
```

Example: – selectone.py

```
import MySQLdb

#Create the connection object
myconn = MySQLdb.connect("localhost","root","root","Collegedb")

#creating the cursor object
cur = myconn.cursor()

#executing the query
cur.execute("select * from students")

#fetching all the rows from the cursor object
result = cur.fetchone()
print("One student Details are :")

#printing the result
print(result)

#close the connection
myconn.close()
```

Output:

```
>>>python selectone.py
```

```
One student Details are:
('501', 'ABC', 23)
```

- **To Update data in table :**

The data can be updated in table by using the following SQL query.

> update <table-name> set column-name=value where condition

Example: – updatedata.py

```
import MySQLdb

#Create the connection object
myconn = MySQLdb.connect("localhost","root","root","Collegedb")

#creating the cursor object
cur = myconn.cursor()

#executing the queries
cur.execute("update students set sname='Kumar' where sid='502'")

#commit the transaction
myconn.commit()
print("Data updated successfully")

#close the connection
myconn.close()
```

Output:

```
>>>python updatedata.py  
Data updated successfully
```

```
MariaDB [Collegedb]> select * from students;  
+-----+-----+-----+  
|  sid  |  sname  |   age   |  
+-----+-----+-----+  
|  501  |   ABC   |    23   |  
|  502  |  Kumar  |    22   |  
+-----+-----+-----+  
2 rows in set (0.00 sec)
```

- **To Delete data from table :**

The data can be deleted from table by using the following SQL query.

> *delete from <table-name> where condition*

Example: – deletedata.py

```
import MySQLdb
```

```
#Create the connection object
```

```
myconn = MySQLdb.connect("localhost","root","root","Collegedb")
```

```
#creating the cursor object
```

```
cur = myconn.cursor()
```

```
#executing the queries
```

```
cur.execute("delete from students where sid='502'")
```

```
#commit the transaction
```

```
myconn.commit()
```

```
print("Data deleted successfully")
```

```
#close the connection
```

```
myconn.close()
```

Output:

```
>>>python deletedata.py
```

```
Data deleted successfully
```

```
MariaDB [Collegedb]> select * from students;  
+-----+-----+-----+  
|  sid  |  sname  |   age   |  
+-----+-----+-----+  
|  501  |   ABC   |    23   |  
+-----+-----+-----+  
1 row in set (0.00 sec)
```

➤ **mysql.connector(in python3.x):**

MySQL Connector enables Python programs to access MySQL databases. The following are example programs demonstrate interactions with MySQL database using **mysql.connector** module.

Note – Make sure you have root privileges of MySQL database to interact with database.i.e. User and password.

We are going to perform the following operations on MySQL database.

- Show databases
- Create database
- Create table
- To insert data into table
- Read/Select data from table
- Update data in table
- Delete data from table

Example Programs

• **To display databases :**

We can get the list of all the databases by using the following MySQL query.

>show databases;

Example: – showdb.py

```
import mysql.connector

#Create the connection object
myconn = mysql.connector.connect(host = "localhost", user = "root",passwd = "root")

#creating the cursor object
cur = myconn.cursor()

#executing the query
dbs = cur.execute("show databases")

#display the result
for x in cur:
    print(x)

#close the connection
myconn.close()
```

Output:

```
>>>python showdb.py
('information_schema',)
('mysql',)
('performance_schema',)
('phpmyadmin',)
('test',)
```

- **To Create database :**

The new database can be created by using the following SQL query.

> create database <database-name>

Example: – createdb.py

```
import mysql.connector

#Create the connection object
myconn = mysql.connector.connect(host = "localhost",
                                user = "root",
                                passwd = "root")

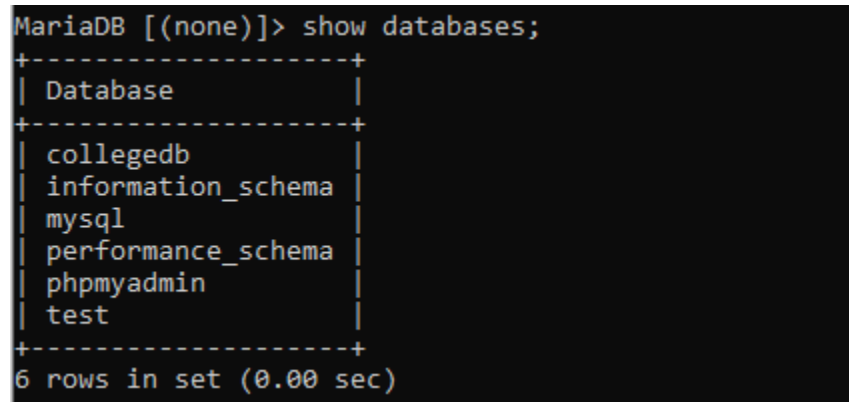
#creating the cursor object
cur = myconn.cursor()

#executing the query
cur.execute("create database Collegedb")
print("Database created successfully")

#close the connection
myconn.close()
```

Output:

```
>>>python createdb.py
Database created successfully
```



```
MariaDB [(none)]> show databases;
+-----+
| Database |
+-----+
| collegedb |
| information_schema |
| mysql |
| performance_schema |
| phpmyadmin |
| test |
+-----+
6 rows in set (0.00 sec)
```

- **To Create table :**

The new table can be created by using the following SQL query.

> *create table <table-name> (column-name1 datatype, column-name2 datatype,...)*

Example: – createtable.py

```
import mysql.connector
```

```
#Create the connection object
```

```
myconn=mysql.connector.connect(host="localhost",  
                               user="root",  
                               passwd="root",  
                               database="Collegedb")
```

```
#creating the cursor object
```

```
cur = myconn.cursor()
```

```
#executing the query
```

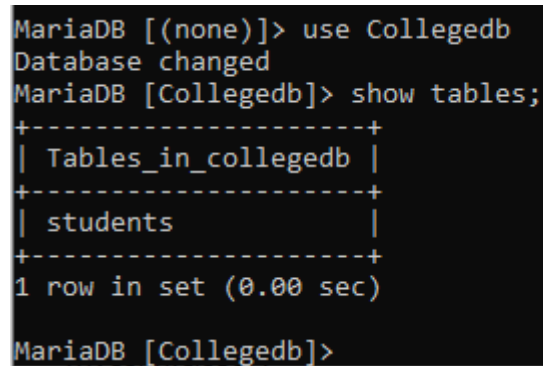
```
cur.execute("create table students(sid varchar(20) primary key,sname varchar(25),age int(10))")  
print("Table created successfully")
```

```
#close the connection
```

```
myconn.close()
```

Output:

```
>>>python createtable.py  
Table created successfully
```



```
MariaDB [(none)]> use Collegedb  
Database changed  
MariaDB [Collegedb]> show tables;  
+-----+  
| Tables_in_collegedb |  
+-----+  
| students              |  
+-----+  
1 row in set (0.00 sec)  
  
MariaDB [Collegedb]>
```

- **To Insert data into table :**

The data can be inserted into table by using the following SQL query.

> *insert into <table-name> values (value1, value2,...)*

Example: – insertdata.py

```
import mysql.connector

#Create the connection object
myconn=mysql.connector.connect(host="localhost",
                               user="root",
                               passwd="root",
                               database="Collegedb")

#creating the cursor object
cur = myconn.cursor()

#executing the queries
cur.execute("INSERT INTO students VALUES ('501', 'ABC', 23)")

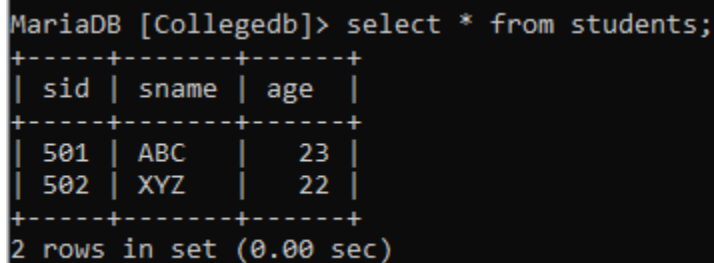
cur.execute("INSERT INTO students VALUES ('502', 'XYZ', 22)")

#commit the transaction
myconn.commit()
print("Data inserted successfully")

#close the connection
myconn.close()
```

Output:

```
>>>python insertdata.py
Data inserted successfully
```



The screenshot shows a terminal window with a MySQL prompt. The user has entered the command 'select * from students;'. The output is a table with three columns: 'sid', 'sname', and 'age'. There are two rows of data: one with '501', 'ABC', and '23', and another with '502', 'XYZ', and '22'. Below the table, it says '2 rows in set (0.00 sec)'.

```
MariaDB [Collegedb]> select * from students;
+-----+-----+-----+
| sid | sname | age |
+-----+-----+-----+
| 501 | ABC   | 23  |
| 502 | XYZ   | 22  |
+-----+-----+-----+
2 rows in set (0.00 sec)
```

- **To Read/Select data from table :**

The data can be read/select data from table by using the following SQL query.

>select column-names from <table-name>

Python provides the fetchall() method and fetchone() method.

- **fetchall()** method returns all rows in the table. We can iterate the result to get the individual rows.
- **fetchone()** method returns one row from table.

Example: – selectdata.py

```
import mysql.connector

#Create the connection object
myconn=mysql.connector.connect(host="localhost",
                               user="root",
                               passwd="root",
                               database="Collegedb")

#creating the cursor object
cur = myconn.cursor()

#executing the query
cur.execute("select * from students")

#fetching all the rows from the cursor object
result = cur.fetchall()
print("Student Details are :")

#printing the result
for x in result:
    print(x);

#close the connection
myconn.close()
```

Output:

```
>>>python selectdata.py
```

```
Student Details are:
('501', 'ABC', 23)
('502', 'XYZ', 22)
```


Example: – selectone.py

```
import mysql.connector

#Create the connection object
myconn=mysql.connector.connect(host="localhost",user="root",passwd="root",
                                database="Collegedb")

#creating the cursor object
cur = myconn.cursor()

#executing the query
cur.execute("select * from students")

#fetching all the rows from the cursor object
result = cur.fetchone()
print("One student Details are :")

#printing the result
print(result)

#close the connection
myconn.close()
```

Output:

```
>>>python selectone.py
```

```
One student Details are:
('501', 'ABC', 23)
```

- **To Update data in table :**

The data can be updated in table by using the following SQL query.

> update <table-name> set column-name=value where condition

Example: – updatedata.py

```
import mysql.connector

#Create the connection object
myconn=mysql.connector.connect(host="localhost",user="root",passwd="root",
                                database="Collegedb")

#creating the cursor object
cur = myconn.cursor()

#executing the queries
cur.execute("update students set sname='Kumar' where sid='502'")

#commit the transaction
myconn.commit()
print("Data updated successfully")

#close the connection
myconn.close()
```

Output:

```
>>>python updatedata.py  
Data updated successfully
```

```
MariaDB [Collegedb]> select * from students;  
+-----+-----+-----+  
| sid | sname | age |  
+-----+-----+-----+  
| 501 | ABC   | 23  |  
| 502 | Kumar | 22  |  
+-----+-----+-----+  
2 rows in set (0.00 sec)
```

- **To Delete data from table :**

The data can be deleted from table by using the following SQL query.

> *delete from <table-name> where condition*

Example: – deletedata.py

```
import mysql.connector
```

```
#Create the connection object
```

```
myconn=mysql.connector.connect(host="localhost",user="root",passwd="root",  
                                database="Collegedb")
```

```
#creating the cursor object
```

```
cur = myconn.cursor()
```

```
#executing the queries
```

```
cur.execute("delete from students where sid='502'")
```

```
#commit the transaction
```

```
myconn.commit()
```

```
print("Data deleted successfully")
```

```
#close the connection
```

```
myconn.close()
```

Output:

```
>>>python deletedata.py  
Data deleted successfully
```

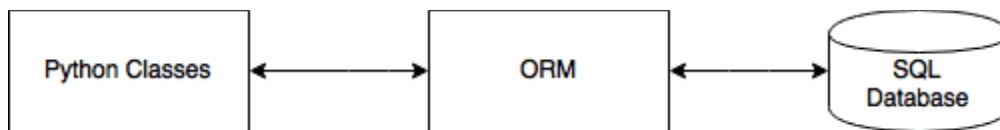
```
MariaDB [Collegedb]> select * from students;  
+-----+-----+-----+  
| sid | sname | age |  
+-----+-----+-----+  
| 501 | ABC   | 23  |  
+-----+-----+-----+  
1 row in set (0.00 sec)
```

Object Relational Mapping (ORM) in Python

- **Introduction:**

Object Relational Mapping is a system of mapping objects to a database. That means it automates the transfer of data stored in relational databases tables into objects that are commonly used in application code.

An object relational mapper maps a relational database system to objects. The ORM is independent of which relational database system is used. From within Python, you can talk to objects and the ORM will map it to the database.



ORMs provide a high-level abstraction upon a relational database that allows a developer to write Python code instead of SQL to interact (create, read, update and delete data and schemas) with database.

Relational database (such as PostgreSQL or MySQL)

ID	FIRST_NAME	LAST_NAME	PHONE
1	John	Connor	+16105551234
2	Matt	Makai	+12025555689
3	Sarah	Smith	+19735554512
...

Python objects

```
class Person:
    first_name = "John"
    last_name = "Connor"
    phone_number = "+16105551234"
```

```
class Person:
    first_name = "Matt"
    last_name = "Makai"
    phone_number = "+12025555689"
```

```
class Person:
    first_name = "Sarah"
    last_name = "Smith"
    phone_number = "+19735554512"
```

ORMs provide a bridge between
**relational database tables, relationships
and fields** and **Python objects**

The mapping like this...

- Python Class == SQL Table
- Instance of the Class == Row in the Table

Developers can use the programming language they are comfortable with to work with a database instead of writing SQL statements or stored procedures.

ORMs also make it possible to switch an application between various relational databases. For example, a developer could use **SQLite** for local development and **MySQL** in production. A production application could be switched from **SQLite** to **MySQL** with minimal code modifications.

There are many ORM implementations written in Python, including

- SQLAlchemy
- Peewee
- The Django ORM
- PonyORM
- SQLAlchemy
- Tortoise ORM

- **SQLAlchemy:**

SQLAlchemy provides a standard interface that allows developers to create database-code to communicate with a wide variety of database engines.

SQLAlchemy is a library used to interact with a wide variety of databases. It enables you to create data models and queries in a manner that feels like normal Python classes and statements. It can be used to connect to most common databases such as Postgres, MySQL, SQLite, Oracle, and many others.

SQLAlchemy is a popular SQL toolkit and Object Relational Mapper. It is written in Python and gives full power and flexibility of SQL to an application developer.

It is necessary to install SQLAlchemy. To install we have to use following code at Terminal or CMD.

pip install sqlalchemy

SQLAlchemy is designed to operate with a DBAPI implementation built for a particular database. It uses system to communicate with various types of DBAPI implementations and databases.

To check if SQLAlchemy is properly installed or not, enter the following command in the Python prompt

```
– >>>import sqlalchemy
```

If the above statement was executed with no errors, "sqlalchemy " is installed and ready to be used.

- **Connecting to Database:**

To connect with database using SQLAlchemy, we have to create engine for this purpose SQLAlchemy supports one function is **create_engine()**.

The **create_engine()** function is used to create engine; it takes overall responsibilities of database connectivity.

Syntax:

Database-server[+driver]://user:password@host/dbname

Example:

mysql+mysqldb://root:root@localhost/collegedb

The main objective of the Object Relational Mapper API of SQLAlchemy is to facilitate associating user-defined Python classes with database tables, and objects of those classes with rows in their corresponding tables.

SQLAlchemy enables expressing database queries in terms of user defined classes and their defined relationships.

- **Declare Mapping:**

First of all, `create_engine()` function is called to set up an engine object which is subsequently used to perform SQL operations.

The Engine establishes a real DBAPI connection to the database, In case of ORM, the configurational process starts by describing the database tables and then by defining classes which will be mapped to those tables.

To create engine in case of MySQL:

Example:

```
from sqlalchemy import create_engine
engine = create_engine('mysql+mysqlconnector://root:@localhost/Collegedb', echo = True)
```

In SQLAlchemy, these two tasks are performed together. This is done by using Declarative system; the classes created include directives to describe the actual database table they are mapped to.

The SQLAlchemy Object Relational Mapper maps user-defined Python classes to database tables, table rows to instance objects, and columns to instance attributes. The SQLAlchemy ORM is built on the SQLAlchemy Expression Language.

When using ORM, we first configure database tables that we will be using. Then we define classes that will be mapped to them. Modern SQLAlchemy uses *Declarative* system to do these tasks.

A *declarative base class* is created, which maintains a catalog of classes and tables. A declarative base class is created with the `declarative_base()` function.

*The `declarative_base()` function is used to create base class. This function is defined in **sqlalchemy.ext.declarative** module.*

To create declarative base class:

Example:

```
from sqlalchemy.ext.declarative import declarative_base
Base = declarative_base()
```

Example: tabledef.py

```
from sqlalchemy import Column, Integer, String
from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
# create a engine
engine = create_engine('mysql+mysqlconnector://root:@localhost/Sampledbs', echo = True)
# create a declarative base class
Base = declarative_base()

class Students(Base):
    __tablename__ = 'students'
    id = Column(Integer, primary_key=True)
    name = Column(String(10))
    address = Column(String(10))
    email = Column(String(10))
Base.metadata.create_all(engine)
```

Output:

```
E:\pythonprgms>python tabledef.py
2019-11-04 10:30:02,283 INFO sqlalchemy.engine.base.Engine
CREATE TABLE students (
  id INTEGER NOT NULL AUTO_INCREMENT,
  name VARCHAR(10),
  address VARCHAR(10),
  email VARCHAR(10),
  PRIMARY KEY (id)
)
2019-11-04 10:30:02,283 INFO sqlalchemy.engine.base.Engine {}
2019-11-04 10:30:02,523 INFO sqlalchemy.engine.base.Engine COMMIT
```

```
MariaDB [sampledb]> use sampledb;
Database changed
MariaDB [sampledb]> show tables;
+-----+
| Tables_in_sampledb |
+-----+
| students            |
+-----+
1 row in set (0.00 sec)

MariaDB [sampledb]>
```

- **Inserting data into the database table:**

The database table is still empty. We can insert data into the database using Python objects. Because we use the SQLAlchemy ORM we do not have to write a single SQL query.

To interact with database tables python supports one inbuilt object i.e. **Session object**. A Session object is created with the help of **sessionmaker()** Module. This module creates a Session class.

To create object for Session Class:

Example:

```
from sqlalchemy.orm import sessionmaker
Session = sessionmaker(bind = engine)
session = Session()
```

The Session class supports following methods:

- **begin():**begins a transaction on this session
- **add():**places an object in the session.
- **add_all():**adds a collection of objects to the session
- **commit():**flushes all items and any transaction in progress
- **delete():**marks a transaction as deleted.
- **query():**To retrieve/select data from database table.
- **close():**Closes current session by clearing all items and ending any transaction in progress.

Example: insertdata.py

```
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from tabledef import *
engine = create_engine('mysql+mysqlconnector://root:@localhost/Sampledbs', echo = True)

# create a Session
Session = sessionmaker(bind=engine)
session = Session()

# Create objects
user = Students(name = 'ABC', address = 'Hyderabad', email = 'abc@abc.com')
# To add record, we can use add() method
session.add(user)

#To add multiple records, we can use add_all() method
session.add_all([
    Students(name = 'BCD', address = 'Hyderabad', email = 'bcd@gmail.com'),
    Students(name = 'CDE', address = 'Gurgaon', email = 'cde@gmail.com'),
    Students(name = 'DEF', address = 'Pune', email = 'def@gmail.com')])
)
# commit the record the database
session.commit()
print("Inserted Successfully..")
```

Output:

Inserted Successfully..

```
MariaDB [sampledb]> select * from students;
+----+-----+-----+-----+
| id | name | address | email |
+----+-----+-----+-----+
| 2  | ABC  | Hyderabad | abc@abc.co |
| 3  | BCD  | Hyderabad | bcd@gmail. |
| 4  | CDE  | Gurgaon  | cde@gmail. |
| 5  | DEF  | Pune     | def@gmail. |
+----+-----+-----+-----+
4 rows in set (0.00 sec)
```

- **Retrieve/Select data from the database table:**

To retrieve/select data from table, the Session class supports one method i.e. query (). By using this we need to create query object. The query object has **all()** method which returns a result set in the form of list of objects

Example: **selectdata.py**

```
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from tabledef import *
engine = create_engine('mysql+mysqlconnector://root:@localhost/Sampledbs', echo = False)
# create a Session
Session = sessionmaker(bind=engine)
session = Session()
# create a query object
result = session.query(Students).all()
# Display records
for row in result:
    print ("Id: ",row.id,"Name: ",row.name, "Address:",row.address, "Email:",row.email)

print("Display record of student ABC")
# Display records with condition
result = session.query(Students).filter(Students.name=="ABC")
for row in result:
    print ("Id: ",row.id,"Name: ",row.name, "Address:",row.address, "Email:",row.email)
```

Output:

Id: 2 Name: ABC Address: Hyderabad Email: abc@abc.co
Id: 3 Name: BCD Address: Hyderabad Email: bcd@gmail.
Id: 4 Name: CDE Address: Gurgaon Email: cde@gmail.
Id: 5 Name: DEF Address: Pune Email: def@gmail.

Display record of student ABC

Id: 2 Name: ABC Address: Hyderabad Email: abc@abc.co