Python Database Programming

Storage Areas

As the Part of our Applications, we required to store our Data like Customers Information, Billing Information, Calls Information etc..

To store this Data, we required Storage Areas. There are 2 types of Storage Areas.

1) Temporary Storage Areas
2) Permanent Storage Areas

1.Temporary Storage Areas:

These are the Memory Areas where Data will be stored temporarily.
Eg: Python objects like List, Tuple, Dictionary.

Once Python program completes its execution then these objects will be destroyed automatically and data will be lost.

2. Permanent Storage Areas:

Also known as Persistent Storage Areas. Here we can store Data permanently.
Eg: File Systems, Databases, Data warehouses, Big Data Technologies etc

File Systems:

File Systems can be provided by Local operating System. File Systems are best suitable to store very less Amount of Information.

Limitations:

1) We cannot store huge Amount of Information.
2) There is no Query Language support and hence operations will become very complex.
3) There is no Security for Data.
4) There is no Mechanism to prevent duplicate Data. Hence there may be a chance of Data Inconsistency Problems.

To overcome the above Problems of File Systems, we should go for Databases.

Databases:

1) We can store Huge Amount of Information in the Databases.

2) Query Language Support is available for every Database and hence we can perform Database Operations very easily.
3) To access Data present in the Database, compulsory username and pwd must be required. Hence Data is secured.
4) Inside Database Data will be stored in the form of Tables. While developing Database Table Schemas, Database Admin follow various Normalization Techniques and can implement various Constraints like Unique Key Constrains, Primary Key Constraints etc which prevent Data Duplication. Hence there is no chance of Data Inconsistency Problems.

Limitations of Databases:

1) Database cannot hold very Huge Amount of Information like Terabytes of Data.

2) Database can provide support only for Structured Data (Tabular Data OR Relational Data) and cannot provide support for Semi Structured Data (like XML Files) and Unstructured Data (like Video Files, Audio Files, Images etc)

To overcome these Problems we should go for more Advanced Storage Areas like Big Data Technologies, Data warehouses etc.

Python Database Programming:

Sometimes as the part of Programming requirement we have to connect to the database and we have to perform several operations like creating tables, inserting data,updating data,deleting data,selecting data etc.

We can use SQL Language to talk to the database and we can use Python to send those SQL commands to the database.

Python provides inbuilt support for several databases like Oracle, MySql, SqlServer, GadFly, sqlite, etc.
Python has seperate module for each database.
Eg: cx_Oralce module for communicating with Oracle database
pymssql module for communicating with Microsoft Sql Server

Standard Steps for Python database Programming:

1. Import database specific module
Eg: import cx_Oracle

2. Establish Connection between Python Program and database.
We can create this Connection object by using connect() function of the module.
con = cx_Oracle.connect(datbase information)

Eg: con=cx_Oracle.connect('scott/tiger@localhost')

3. To execute our sql queries and to hold results some special object is required, which is nothing but Cursor object. We can create Cursor object by using cursor() method.

```
cursor=con.cursor()
```

4. Execute SQL Queries By using Cursor object. For this we can use the following methods

i) execute(sqlquery) ➔ To execute a single sql query
ii) executescript(sqlqueries) ➔ To execute a string of sql queries seperated by semi-colon ';'
iii) executemany() ➔ To execute a Parameterized query

Eg:
```
cursor.execute("select * from employees")
```

5. commit or rollback changes based on our requirement in the case of DML Queries(insert|update|delete)

commit() ➔ Saves the changes to the database
rollback() ➔ rolls all temporary changes back

6. Fetch the result from the Cursor object in the case of select queries
fetchone() ➔ To fetch only one row
fetchall() ➔ To fetch all rows and it returns a list of rows
fecthmany(n) ➔ To fetch first n rows

Eg 1:
```
data =cursor.fetchone()
print(data)
```

Eg 2:
```
data=cursor.fetchall()
for row in data:
        print(row)
```

7. close the resources
After completing our operations it is highly recommended to close the resources in the reverse order of their opening by using close() methods.

```
cursor.close()
con.close()
```

Note: The following is the list of all important methods which can be used for python database programming.

```
connect()
cursor()
execute()
executescript()
executemany()
commit()
rollback()
fetchone()
fetchall()
fetchmany(n)
```

fetch
close()

These methods won't be changed from database to database and same for all databases.

Working with Oracle Database:

From Python Program if we want to communicate with any database,some translator must be required to translate Python calls into Database specific calls and Database specific calls into Python calls.This translator is nothing but Driver/Connector.

Diagram

For Oracle database the name of driver needed is cx_Oracle.
cx_Oracle is a Python extension module that enables access to Oracle Database.It can be used for both Python2 and Python3. It can work with any version of Oracle database like 9,10,11 and 12.

Installing cx_Oracle:

From Normal Command Prompt (But not from Python console)execute the following command

```
D:\python_classes>pip install cx_Oracle

 Collecting cx_Oracle
  Downloading cx_Oracle-6.0.2-cp36-cp36m-win32.whl (100kB)
    100% |----------| 102kB 256kB/s
Installing collected packages: cx-Oracle
Successfully installed cx-Oracle-6.0.2
```

How to Test Installation:

From python console execute the following command:

```
>>> help("modules")
```

In the output we can see cx_Oracle

```
....
_multiprocessing   crypt          ntpath         timeit
_opcode            csv            nturl2path     tkinter
_operator          csvr           numbers        token
_osx_support       csvw           opcode         tokenize
_overlapped        ctypes         operator       trace
_pickle            curses         optparse       traceback
_pydecimal         custexcept     os             tracemalloc
_pyio              cx_Oracle      parser         try
_random            data           pathlib        tty
_sha1              datetime       pdb            turtle
```


```
_sha256          dbm              pick             turtledemo
_sha3            decimal          pickle           types
_sha512          demo             pickletools      typing
_signal          difflib          pip              unicodedata
_sitebuiltins    dis              pipes            unittest
_socket          distutils        pkg_resources    unpick
_sqlite3         doctest          pkgutil          update
_sre             dummy_threading  platform         urllib
_ssl             durgamath        plistlib         uu
_stat            easy_install     polymorph        uuid
.....
```

App1: Program to connect with Oracle database and print its version.

```
1)  import cx_Oracle
2)  con=cx_Oracle.connect('scott/tiger@localhost')
3)  print(con.version)
4)  con.close()
```

Output:

```
D:\python_classes>py db1.py
11.2.0.2.0
```

App2: Write a Program to create employees table in the oracle database :

employees(eno,ename,esal,eaddr)

```
1)  import cx_Oracle
2)  try:
3)      con=cx_Oracle.connect('scott/tiger@localhost')
4)      cursor=con.cursor()
5)      cursor.execute("create table employees(eno number,ename varchar2(10),esal number(10,2),eaddr varchar2(10))")
6)      print("Table created successfully")
7)  except cx_Oracle.DatabaseError as e:
8)      if con:
9)          con.rollback()
10)         print("There is a problem with sql",e)
11) finally:
12)     if cursor:
13)         cursor.close()
14)     if con:
15)         con.close()
```

App3: Write a program to drop employees table from oracle database?

```
1)  import cx_Oracle
2)  try:
3)      con=cx_Oracle.connect('scott/tiger@localhost')
4)      cursor=con.cursor()
5)      cursor.execute("drop table employees")
6)      print("Table dropped successfully")
7)  except cx_Oracle.DatabaseError as e:
8)      if con:
9)          con.rollback()
10)         print("There is a problem with sql",e)
11) finally:
12)     if cursor:
13)         cursor.close()
14)     if con:
15)         con.close()
```

App3: Write a program to insert a single row in the employees table.

```python
import cx_Oracle
try:
    con=cx_Oracle.connect('scott/tiger@localhost')
    cursor=con.cursor()
    cursor.execute("insert into employees values(100,'Durga',1000,'Hyd')")
    con.commit()
    print("Record Inserted Successfully")
except cx_Oracle.DatabaseError as e:
    if con:
        con.rollback()
        print("There is a problem with sql",e)
finally:
    if cursor:
        cursor.close()
    if con:
        con.close()
```

Note: **While performing DML Operations (insert|update|delte), compulsory we have to use commit() method,then only the results will be reflected in the database.**

App4: Write a program to insert multiple rows in the employees table by using executemany() method.

```python
import cx_Oracle
try:
    con=cx_Oracle.connect('scott/tiger@localhost')
    cursor=con.cursor()
    sql="insert into employees values(:eno,:ename,:esal,:eaddr)"
    records=[(200,'Sunny',2000,'Mumbai'),
                    (300,'Chinny',3000,'Hyd'),
                    (400,'Bunny',4000,'Hyd')]
    cursor.executemany(sql,records)
    con.commit()
    print("Records Inserted Successfully")
except cx_Oracle.DatabaseError as e:
    if con:
        con.rollback()
        print("There is a problem with sql",e)
finally:
    if cursor:
        cursor.close()
    if con:
        con.close()
```

App5: Write a program to insert multiple rows in the employees table with dynamic input from the keyboard?

```python
import cx_Oracle
try:
    con=cx_Oracle.connect('scott/tiger@localhost')
    cursor=con.cursor()
    while True:
        eno=int(input("Enter Employee Number:"))
        ename=input("Enter Employee Name:")
        esal=float(input("Enter Employee Salary:"))
        eaddr=input("Enter Employee Address:")
        sql="insert into employees values(%d,'%s',%f,'%s')"
        cursor.execute(sql %(eno,ename,esal,eaddr))
        print("Record Inserted Successfully")
        option=input("Do you want to insert one more record[Yes|No] :")
        if option=="No":
            con.commit()
            break
except cx_Oracle.DatabaseError as e:
    if con:
        con.rollback()
        print("There is a problem with sql :",e)
finally:
    if cursor:
        cursor.close()
    if con:
        con.close()
```

App6: Write a program to update employee salaries with increment for the certain range with dynamic input.

Eg: Increment all employee salaries by 500 whose salary < 5000

```
import cx_Oracle
try:
    con=cx_Oracle.connect('scott/tiger@localhost')
    cursor=con.cursor()
    increment=float(input("Enter Increment Salary:"))
    salrange=float(input("Enter Salary Range:"))
    sql="update employees set esal=esal+%f where esal<%f"
    cursor.execute(sql %(increment,salrange))
    print("Records Updated Successfully")
    con.commit()
except cx_Oracle.DatabaseError as e:
    if con:
        con.rollback()
        print("There is a problem with sql :",e)
finally:
    if cursor:
        cursor.close()
    if con:
        con.close()
```

App7: Write a program to delete employees whose salary greater provided salary as dynamic input?

Eg: delete all employees whose salary > 5000

```
import cx_Oracle
try:
    con=cx_Oracle.connect('scott/tiger@localhost')
    cursor=con.cursor()
    cutoffsalary=float(input("Enter CutOff Salary:"))
    sql="delete from employees where esal>%f"
    cursor.execute(sql %(cutoffsalary))
    print("Records Deleted Successfully")
    con.commit()
except cx_Oracle.DatabaseError as e:
    if con:
        con.rollback()
        print("There is a problem with sql :",e)
finally:
    if cursor:
        cursor.close()
    if con:
        con.close()
```

App8: Write a program to select all employees info by using fetchone() method?

```
import cx_Oracle
try:
    con=cx_Oracle.connect('scott/tiger@localhost')
    cursor=con.cursor()
    cursor.execute("select * from employees")
    row=cursor.fetchone()
    while row is not None:
        print(row)
        row=cursor.fetchone()
except cx_Oracle.DatabaseError as e:
    if con:
        con.rollback()
        print("There is a problem with sql :",e)
finally:
    if cursor:
        cursor.close()
    if con:
        con.close()
```

App9: Write a program to select all employees info by using fetchall() method?

```
import cx_Oracle
try:
    con=cx_Oracle.connect('scott/tiger@localhost')
    cursor=con.cursor()
    cursor.execute("select * from employees")
    data=cursor.fetchall()
    for row in data:
        print("Employee Number:",row[0])
        print("Employee Name:",row[1])
        print("Employee Salary:",row[2])
        print("Employee Address:",row[3])
        print()
        print()
except cx_Oracle.DatabaseError as e:
    if con:
        con.rollback()
        print("There is a problem with sql :",e)
finally:
    if cursor:
        cursor.close()
    if con:
        con.close()
```

App10: Write a program to select employees info by using fetchmany() method and the required number of rows will be provided as dynamic input?

```
import cx_Oracle
try:
    con=cx_Oracle.connect('scott/tiger@localhost')
    cursor=con.cursor()
    cursor.execute("select * from employees")
    n=int(input("Enter the number of required rows:"))
    data=cursor.fetchmany(n)
    for row in data:
        print(row)
except cx_Oracle.DatabaseError as e:
    if con:
        con.rollback()
        print("There is a problem with sql :",e)
finally:
    if cursor:
        cursor.close()
    if con:
        con.close()
```

Output:

```
D:\python_classes>py test.py
Enter the number of required rows:3
(100, 'Durga', 1500.0, 'Hyd')
(200, 'Sunny', 2500.0, 'Mumbai')
(300, 'Chinny', 3500.0, 'Hyd')

D:\python_classes>py test.py
Enter the number of required rows:4
(100, 'Durga', 1500.0, 'Hyd')
(200, 'Sunny', 2500.0, 'Mumbai')
(300, 'Chinny', 3500.0, 'Hyd')
(400, 'Bunny', 4500.0, 'Hyd')
```

Working with Mysql database:

Current version: 5.7.19
Vendor: SUN Micro Systems/Oracle Corporation

Open Source and Freeware
Default Port: 3306
Default user: root

Note: In MySQL, everything we have to work with our own databases, which are also known as Logical Databases.

The following are 4 default databases available in mysql.
1. information_schema
2. mysql
3. performance_schema
4. test

Diagram

In the above diagram only one physical database is available and 4 logical databases are available.

Commonly used commands in MySql:

1. To know available databases:
```
mysql> show databases;
```

2. To create our own logical database
```
mysql> create database durgadb;
```

3. To drop our own database:
```
mysql> drop database durgadb;
```

4. To use a particular logical database
```
mysql> use durgadb;    OR    mysql> connect durgadb;
```

5. To create a table:
```
create table employees(eno int(5) primary key,ename varchar(10),esal double(10,2),eaddr varchar(10));
```

6. To insert data:
```
insert into employees values(100,'Durga',1000,'Hyd');
insert into employees values(200,'Ravi',2000,'Mumbai');
```

In MySQL instead of single quotes we can use double quotes also.

Driver/Connector Information:

From Python program if we want to communicates with MySql database,compulsory some translator is required to convert python specific calls into mysql database specific calls and mysql database specific calls into python specific calls. This translator is nothing but Driver or Connector.

Diagram

We have to download connector seperately from mysql database.

https://dev.mysql.com/downloads/connector/python/2.1.html

How to check installation:

From python console we have to use
help("modules")

In the list of modules,compulsory mysql should be there.

Note: In the case of Python3.4 we have to set PATH and PYTHONPATH explicitly

PATH=C:\Python34
PYTHONPATH=C:\Python34\Lib\site-packages

Q. Write a Program to create table,insert data and display data by using mysql database.

```
1)  import mysql.connector
2)  try:
3)      con=mysql.connector.connect(host='localhost',database='durgadb',user='root',password='root')
4)      cursor=con.cursor()
5)      cursor.execute("create table employees(eno int(5) primary key,ename varchar(10),esal double(10,2),eaddr varchar(
    10))")
6)      print("Table Created...")
7)
8)      sql = "insert into employees(eno, ename, esal, eaddr) VALUES(%s, %s, %s, %s)"
9)      records=[(100,'Sachin',1000,'Mumbai'),
10)                  (200,'Dhoni',2000,'Ranchi'),
11)                  (300,'Kohli',3000,'Delhi')]
12)     cursor.executemany(sql,records)
13)     con.commit()
14)     print("Records Inserted Successfully...")
15)
16)     cursor.execute("select * from employees")
17)     data=cursor.fetchall()
18)     for row in data:
19)         print("Employee Number:",row[0])
20)         print("Employee Name:",row[1])
21)         print("Employee Salary:",row[2])
22)         print("Employee Address:",row[3])
23)         print()
24)         print()
25) except mysql.connector.DatabaseError as e:
26)     if con:
27)         con.rollback()
28)         print("There is a problem with sql :",e)
29) finally:
30)     if cursor:
31)         cursor.close()
32)     if con:
33)         con.close()
```

Q. Write a Program to copy data present in employees table of mysql database into Oracle database.

```
1)  import mysql.connector
2)  import cx_Oracle
3)  try:
4)      con=mysql.connector.connect(host='localhost',database='durgadb',user='root',password='root')
```

```
5)       cursor=con.cursor()
6)       cursor.execute("select * from employees")
7)       data=cursor.fetchall()
8)       list=[]
9)       for row in data:
10)          t=(row[0],row[1],row[2],row[3])
11)          list.append(t)
12)  except mysql.connector.DatabaseError as e:
13)      if con:
14)          con.rollback()
15)          print("There is a problem with MySql :",e)
16)  finally:
17)      if cursor:
18)          cursor.close()
19)      if con:
20)          con.close()
21)
22)  try:
23)      con=cx_Oracle.connect('scott/tiger@localhost')
24)      cursor=con.cursor()
25)      sql="insert into employees values(:eno,:ename,:esal,:eaddr)"
26)      cursor.executemany(sql,list)
27)      con.commit()
28)      print("Records Copied from MySQL Database to  Oracle Database  Successfully")
29)  except cx_Oracle.DatabaseError as e:
30)      if con:
31)          con.rollback()
32)          print("There is a problem with sql",e)
33)  finally:
34)      if cursor:
35)          cursor.close()
36)      if con:
37)          con.close()
```

https://dev.mysql.com/downloads/connector/python/2.1.html

```
1)   create table employees(eno int(5) primary key,ename varchar(10),esal double(10,2),eaddr varchar(10));
2)
3)    insert into employees values(100,'Durga',1000,'Hyd');
4)    insert into employees values(200,'Ravi',2000,'Mumbai');
5)    insert into employees values(300,'Shiva',3000,'Hyd');
```