>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Chapter-22 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*22.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

……………………………………………….

SQL> show parameter service\_name;

………………………………………………………………………………………………\*\*\*\*\*\*\*\*\*\*\*\*\*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.

1) import cx\_Oracle

2) try:

3) con=cx\_Oracle.connect('scott/tiger@localhost')

4) cursor=con.cursor()

5) cursor.execute("insert into employees values(100,'Prasanna',1000,'Hyd')")

6) con.commit()

7) print("Record Inserted Successfully")

8) except cx\_Oracle.DatabaseError as e:

9) if con:

10) con.rollback()

11) print("There is a problem with sql",e)

12) finally:

13) if cursor:

14) cursor.close()

15) if con:

16) 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.

1) import cx\_Oracle

2) try:

3) con=cx\_Oracle.connect('scott/tiger@localhost')

4) cursor=con.cursor()

5) sql="insert into employees values(:eno,:ename,:esal,:eaddr)"

6) records=[(200,'Sunny',2000,'Mumbai'),

7) (300,'Chinny',3000,'Hyd'),

8) (400,'Bunny',4000,'Hyd')]

9) cursor.executemany(sql,records)

10) con.commit()

11) print("Records Inserted Successfully")

12) except cx\_Oracle.DatabaseError as e:

13) if con:

14) con.rollback()

15) print("There is a problem with sql",e)

16) finally:

17) if cursor:

18) cursor.close()

19) if con:

20) con.close()

……………………………………………………………………….

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

1) import cx\_Oracle

2) try:

3) con=cx\_Oracle.connect('scott/tiger@localhost')

4) cursor=con.cursor()

5) while True:

6) eno=int(input("Enter Employee Number:"))

7) ename=input("Enter Employee Name:")

8) esal=float(input("Enter Employee Salary:"))

9) eaddr=input("Enter Employee Address:")

10) sql="insert into employees values(%d,'%s',%f,'%s')"

11) cursor.execute(sql %(eno,ename,esal,eaddr))

12) print("Record Inserted Successfully")

13) option=input("Do you want to insert one more record[Yes|No] :")

14) if option=="No":

15) con.commit()

16) break

17) except cx\_Oracle.DatabaseError as e:

18) if con:

19) con.rollback()

20) print("There is a problem with sql :",e)

21) finally:

22) if cursor:

23) cursor.close()

24) if con:

25) 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

1) import cx\_Oracle

2) try:

3) con=cx\_Oracle.connect('scott/tiger@localhost')

4) cursor=con.cursor()

5) increment=float(input("Enter Increment Salary:"))

6) salrange=float(input("Enter Salary Range:"))

7) sql="update employees set esal=esal+%f where esal<%f"

8) cursor.execute(sql %(increment,salrange))

9) print("Records Updated Successfully")

10) con.commit()

11) except cx\_Oracle.DatabaseError as e:

12) if con:

13) con.rollback()

14) print("There is a problem with sql :",e)

15) finally:

16) if cursor:

17) cursor.close()

18) if con:

19) con.close()

…………………………………………………………………………………………

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

Eg: delete all employees whose salary > 5000

1) import cx\_Oracle

2) try:

3) con=cx\_Oracle.connect('scott/tiger@localhost')

4) cursor=con.cursor()

5) cutoffsalary=float(input("Enter CutOff Salary:"))

6) sql="delete from employees where esal>%f"

7) cursor.execute(sql %(cutoffsalary))

8) print("Records Deleted Successfully")

9) con.commit()

10) except cx\_Oracle.DatabaseError as e:

11) if con:

12) con.rollback()

13) print("There is a problem with sql :",e)

14) finally:

15) if cursor:

16) cursor.close()

17) if con:

18) con.close()

………………………………………………………………………..

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

1) import cx\_Oracle

2) try:

3) con=cx\_Oracle.connect('scott/tiger@localhost')

4) cursor=con.cursor()

5) cursor.execute("select \* from employees")

6) row=cursor.fetchone()

7) while row is not None:

8) print(row)

9) row=cursor.fetchone()

10) except cx\_Oracle.DatabaseError as e:

11) if con:

12) con.rollback()

13) print("There is a problem with sql :",e)

14) finally:

15) if cursor:

16) cursor.close()

17) if con:

18) con.close()

……………………………………………………………………..

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

1) import cx\_Oracle

2) try:

3) con=cx\_Oracle.connect('scott/tiger@localhost')

4) cursor=con.cursor()

5) cursor.execute("select \* from employees")

6) data=cursor.fetchall()

7) for row in data:

8) print("Employee Number:",row[0])

9) print("Employee Name:",row[1])

10) print("Employee Salary:",row[2])

11) print("Employee Address:",row[3])

12) print()

13) print()

14) except cx\_Oracle.DatabaseError as e:

15) if con:

16) con.rollback()

17) print("There is a problem with sql :",e)

18) finally:

19) if cursor:

20) cursor.close()

21) if con:

22) 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?

1) import cx\_Oracle

2) try:

3) con=cx\_Oracle.connect('scott/tiger@localhost')

4) cursor=con.cursor()

5) cursor.execute("select \* from employees")

6) n=int(input("Enter the number of required rows:"))

7) data=cursor.fetchmany(n)

8) for row in data:

9) print(row)

10) except cx\_Oracle.DatabaseError as e:

11) if con:

12) con.rollback()

13) print("There is a problem with sql :",e)

14) finally:

15) if cursor:

16) cursor.close()

17) if con:

18) con.close()

………………………………………………

Output:

D:\python\_classes>py test.py

Enter the number of required rows:3

(100, 'Prasanna', 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, 'Prasanna', 1500.0, 'Hyd')

(200, 'Sunny', 2500.0, 'Mumbai')

(300, 'Chinny', 3500.0, 'Hyd')

(400, 'Bunny', 4500.0, 'Hyd')

…………………………………………………………………………………………..

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*END\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

…………………………………………………………………………………………..