PYTHON’S DATABASE CONNECTIVITY

A database represents collection of data. Just like a file, we can store data into a database. Once the data is stored in a database, we need to perform some operations on data. For example modifying the existing data or deleting the unwanted data or retrieving the data from the database, etc. operations are needed on data. To perform such operations, a database comes with a software. This is called a database management system (DBMS). Hence, we can say:

DBMS = database + software to manage the data.

Examples for DBMS are MySQL, Oracle, Sybase, SQL Server etc.

Advantages of a DBMS over files

- If we go for files, we have to write a lot of programming. For example, to store data into a file, we have to write a program, to retrieve data, we need another program and to modify data, we should write yet another program. A lot of programmers’ time is consumed in developing these programs. If we use DBMS, we can achieve all these tasks without writing any programs. We can store data, retrieve it, modify or delete it by giving simple commands. These commands are written in a language called SQL (Structured Query Language). Thus programmers’ duty is simplified.

- Generally each file contains data that is not related to another file’s data. When there is no relationship between the files, it would be difficult to fulfil the information needs of the user. The user may wish to see the information that has some relationship and that is stored in various files. Accessing such data is difficult for the programmers. In DBMS, various pieces of data can be related and retrieved easily using keys concept.

- Files are not so efficient in handling large volumes of data. It would be difficult task to enter large data into a file or retrieving it from the file. But DBMS can handle large volumes of data efficiently and easily.

- When multiple users are accessing the data concurrently from the same file, there are chances of wrong results or inconsistent changes to the existing data. DBMS will take care of such problems.

- Once changes are made to the file, we cannot roll them back. In case of DBMS, we can restore the previous state of data by rolling back the changes.

- Files offer very weak security for data whereas DBMS offers very good protection for data.

Using MySQL from Python

To work with MySQL in a Python program, we have to use *connector* sub module of *mysql* module. We can import this module as:

import pymysql

To establish connection with MySQL database, we use connect() method of *mysql.connector* module, as:

```
conn = pymysql.connect(host='localhost', database='world', user='root', password='nag123')
```

The connect() method returns MySQLConnection class object ‘conn’. This method takes the following parameters:
host= In the network, server’ ip address should be given as host. If we use individual computer, then the string ‘localhost’ can be supplied.
database= This is the database name to connect to.
user= this represents the user name. Generally this is ‘root’.
password= This indicates the root password.

The next step is to create cursor class object by calling cursor() method on ‘conn’ object as:

```
cursor = conn.cursor()
```

The ‘cursor’ object is useful to execute SQL commands on the database. For this purpose, we can use execute() method of ‘cursor’ object, as:

```
cursor.execute("select * from emptab")
```

The resultant rows retrieved from the table are stored in the ‘cursor’ object. We can get them from the ‘cursor’ object using fetchone() or fetchall() methods.

```
row = cursor.fetchone()   # get 1 row
rows = cursor.fetchall()   # get all rows
```

Finally, we can close the connection with MySQL by closing the cursor and connection objects as:

```
cursor.close()
conn.close()
```

Retrieving all rows from a table

After establishing connection with MySQL database using connect() method, we have to create ‘cursor’ object. Then we have to call the execute() method which executes our SQL command, as:

```
cursor.execute("select * from emptab")
```

Our assumption in this case is that we have ‘emptab’ table already available in the ‘world’ database in MySQL. The rows of the ‘emptab’ will be available in the ‘cursor’ object. We can retrieve them using fetchone() method as:

```
row = cursor.fetchone()   # retrieve the first row


while row is not None:   # if that row exists then
   print(row)   # display it
   row = cursor.fetchone()   # get the next row
```

PROGRAMS
71. To retrieve and display all rows from employee table.

We can also use fetchall() method that retrieves all rows from the 'emptab' table and stores them in an iterator, as:

```
rows = cursor.fetchall()
```

Now, we can retrieve the rows one by one from 'rows' iterator using a for loop as:

```
for row in rows:
   print(row)   # display each row
```

PROGRAMS
72. To retrieve and display all rows from employee table.

Inserting rows into a table

To insert a row into a table, we can use SQL insert command, as:

```
str = "insert into emptab(eno, ename, sal) values(9999, 'Srinivas', 9999.99)"
```

Here, we are inserting values for eno, ename and sal columns into 'emptab' table. The total command is taken as a string 'str'. Now we can execute this command by passing it to execute() method, as:

```
cursor.execute(str)   # SQL insert statement in the str is executed
```

After inserting a row into the table, we can save it by calling commit() method as:

```
conn.commit()
```

In case, there is an error, we can un-save the row by calling rollback() method as:

```
conn.rollback()
```

PROGRAMS
73. A Python program to insert a row into a table in MySQL.

Deleting rows from a table

We can accept the employee number from the keyboard and delete the corresponding row in the employee table. For this purpose, we can use the SQL command as:

```
delete from emptab where eno = '%d'
```

Here, '%d' represents the integer number supplied to the command as argument. This is nothing but the employee number, i.e. eno.

PROGRAMS
74. Let us delete a row from emptab by accepting the employee number.

Updating rows in a table

Suppose, we want to modify the values in a row, we can use update command. For example, to increase the salary of an employee by 1000, we can write:

```
update emptab set sal = sal+1000 where eno = '%d'
```

Here, '%d' represents the argument value, i.e. the employee number. This can be passed from the keyboard.

PROGRAMS

75. A program to increase the salary of an employee by accepting the employee number from keyboard.

Using Oracle database from Python

To work with Oracle database in a Python program, we have to use *cx_Oracle* module. This module can be imported as:

```
import cx_Oracle
```

To establish connection with Oracle database, we use connect() method of *cx_Oracle* module, as:

```
conn = cx_Oracle.connect('SYSTEM/rnr_Oracle2@localhost/orcl.Dlink')
```

The connect() method returns cx_Oracle.Connection class object 'conn'. This method takes the following parameters:

SYSTEM: this is the user name for Oracle database.
rnr_Oracle2: this is the pass word given at the time of Oracle database installation.
localhost: this indicates the IP Address of the Oracle server machine. When we are not in the network and we are running this program in an individual computer, we can mention 'localhost' to represent that the server is also found in the local system.
orcl.Dlink: this is global database name given at the time of installation.

The next step is to create cursor class object by calling cursor() method on 'conn' object as:

```
cursor = conn.cursor()
```

The 'cursor' object is useful to execute SQL commands on the database. For this purpose, we can use execute() method of 'cursor' object, as:

```
cursor.execute("select * from emptab")
```

The resultant rows retrieved from the table are stored in the 'cursor' object. We can get them from the 'cursor' object using fetchone() or fetchall() methods.

```
row = cursor.fetchone()   # get 1 row
rows = cursor.fetchall()   # get all rows
```

Finally, we can close the connection with MySQL by closing the cursor and connection objects as:

```
cursor.close()
conn.close()
```

PROGRAMS

76. A Python program to connect to Oracle database and retrieve rows from emptab table.

PROGRAM SOLUTIONS

```
# 71.displaying all rows of emptab
import pymysql

# connect to MySQL database
conn = pymysql.connect(host='localhost', database='world',
user='root', password='nag123')

# prepare a cursor object using cursor() method
cursor = conn.cursor()

# execute a SQL query using execute() method
cursor.execute("select * from emptab")

# get only one row
row = cursor.fetchone()


# if the row exists
while row is not None:
    print(row)   # display it
    row = cursor.fetchone()   # get the next row

# close connection
cursor.close()
conn.close()
```

```
# 72.displaying all rows of emptab - v2.0
import pymysql

# connect to MySQL database
conn = pymysql.connect(host='localhost', database='world',
user='root', password='nag123')


# prepare a cursor object using cursor() method
cursor = conn.cursor()

# execute a SQL query using execute() method
cursor.execute("select * from emptab")

# get all rows
rows = cursor.fetchall()

# display the number of rows
print('Total number of rows= ', cursor.rowcount)

# display the rows from rows object
for row in rows:
    print(row)   # display each row

# close connection
cursor.close()
conn.close()
```

```
# 73.inserting a row into emptab
import pymysql

# connect to MySQL database
conn = pymysql.connect(host='localhost', database='world',
user='root', password='nag123')

# prepare a cursor object using cursor() method
cursor = conn.cursor()

# prepare SQL query string to insert a row
str = "insert into emptab(eno, ename, sal) values(9999, 'Srinivas',
9999.99)"

try:
    # execute the SQL query using execute() method
    cursor.execute(str)

    # save the changes to the database
    conn.commit()
    print('1 row inserted...')

except:
    # rollback if there is any error
    conn.rollback()

# close connection
cursor.close()
conn.close()
```

```
# 74.deleting a row from emptab depending on eno
import pymysql

# function to delete row from the emptab table
def delete_rows(eno):

    # connect to MySQL database
    conn = pymysql.connect(host='localhost', database='world',
user='root', password='nag123')

    # prepare a cursor object using cursor() method
    cursor = conn.cursor()

    # prepare SQL query string to delete a row
    str = "delete from emptab where eno = '%d'"

    # define the arguments
    args = (eno)

    try:
        # execute the SQL query using execute() method
        cursor.execute(str % args)

        # save the changes to the database
        conn.commit()
        print('1 row deleted...')

    except:
        # rollback if there is any error
        conn.rollback()
    finally:
        # close connection
        cursor.close()
        conn.close()

# enter employee number whose row is to be deleted
x = int(input('Enter eno: '))

# pass eno to delete_rows() function
delete_rows(x)
```

```
# 75.updating the salary in emptab depending on eno
import pymysql

# function to update row from the emptab table
def update_rows(eno):

    # connect to MySQL database
    conn = pymysql.connect(host='localhost', database='world',
user='root', password='nag123')

    # prepare a cursor object using cursor() method
    cursor = conn.cursor()

    # prepare SQL query string to update the salary in a row
    str = "update emptab set sal = sal+1000 where eno = '%d'"

    # define the arguments
    args = (eno)

    try:
        # execute the SQL query using execute() method
        cursor.execute(str % args)

        # save the changes to the database
        conn.commit()
        print('1 row updated...')

    except:
        # rollback if there is any error
        conn.rollback()

    finally:
        # close connection
        cursor.close()
        conn.close()

# enter employee number whose row is to be updated
x = int(input('Enter eno: '))

# pass eno to update_rows() function
update_rows(x)
```

```
# 76.to retrieve all rows of emptab from Oracle database
import cx_Oracle

# connect to Oracle database
conn = cx_Oracle.connect('SYSTEM/rnr_Oracle2@localhost/orcl')

# prepare a cursor object using cursor() method
cursor = conn.cursor()
```

```
# execute a SQL query using execute() method
cursor.execute("select * from emptab")
# get only one row
row = cursor.fetchone()

# if the row exists
while row is not None:
    print(row)   # display it
    row = cursor.fetchone()   # get the next row

# close connection
cursor.close()
conn.close()
```

********* ALL THE BEST FOR YOUR CAREER ***********