PYTHON EXPERIMENT 8

Name : Gini Chacko
Roll : 8942
Class : SE Comps B

Aim: Program to demonstrate CRUD (Create, Read, Update and Delete) operations on database (SQLite/MySQL) using python

CODE:

```
import mysql.connector

while True:
    print("\n\n******Operations on database******")
    print("1. Create")
    print("2. Insert")
    print("3. Read")
    print("4. Update")
    print("5. Delete")
    print("6. Exit")
    choice = int(input("\nEnter your Choice : "))
    if choice==1:
        mydb =
mysql.connector.connect(host="localhost",user="root",passwd="9869")
        mycursor = mydb.cursor()
        mycursor.execute("Show databases")

        for db in mycursor:
            print(db)
        print("\n------------------------------------------------------------")
```

```
    elif choice==2:
        mydb =
mysql.connector.connect(host="localhost",user="root",passwd="9869",database="
Gini")

        mycursor = mydb.cursor()

        sqlform = "Insert into person(firstName,lastName,age,address)
values(%s,%s,%s,%s)"

        persons =[("Gini","Chacko",20, "Mira Road")]

        mycursor.executemany(sqlform,persons)

        mydb.commit()
        print("\nSuccessfully Inserted the values!")
        print("\n------------------------------------------------------------")


    elif choice==3:
        mydb =
mysql.connector.connect(host="localhost",user="root",passwd="9869",database="
Gini")

        mycursor = mydb.cursor()
        mycursor.execute("SELECT * FROM person")

        result =mycursor.fetchall();

        for row in result:
            print("\nData in person table : ", row)
        print("\n------------------------------------------------------------")

    elif choice==4:
```

```
        mydb =
mysql.connector.connect(host="localhost",user="root",passwd="9869",database="
Gini")

        mycursor = mydb.cursor()

        query="Update person set address='Kerala' where firstName='Gini'"

        mycursor.execute(query)
        mydb.commit()
        print("\nSuccessfully Updated the values!")
        print("\n------------------------------------------------------------")

    elif choice==5:
        mydb =
mysql.connector.connect(host="localhost",user="root",passwd="9869",database="
Gini")

        mycursor = mydb.cursor()

        query="Delete from person where firstName='Gini'"

        mycursor.execute(query)
        mydb.commit()
        print("\nSuccessfully Deleted the values!")
        print("\n------------------------------------------------------------")

    elif choice==6:
        print("Exiting!!")
        print("\n------------------------------------------------------------")
        break

    else:
        print("Wrong Choice!!")
        print("\n------------------------------------------------------------")
```

OUTPUT:

```
PS C:\Users\Chacko> & python "c:/GINI/ENGG/2nd Year/Sem 4/Python/LABS/EXP 8/exp8.py"


******Operations on database******
1. Create
2. Insert
3. Read
4. Update
5. Delete
6. Exit

Enter your Choice : 1
('assign1b',)
('assign_1',)
('company',)
('exp_3',)
('exp_4',)
('exp_5a',)
('exp_6',)
('exp_7',)
('exp_8',)
('gini',)
('information_schema',)
('mysql',)
('performance_schema',)
('register',)
('sakila',)
('sales',)
('sqlconstraints',)
('sys',)
('test',)
('university',)
('university1',)
('world',)

------------------------------------------------------------
```

```
******Operations on database******
1. Create
2. Insert
3. Read
4. Update
5. Delete
6. Exit

Enter your Choice : 2

Successfully Inserted the values!

------------------------------------------------------------


******Operations on database******
1. Create
2. Insert
3. Read
4. Update
5. Delete
6. Exit

Enter your Choice : 3

Data in person table :  ('Gini', 'Chacko', '20', 'Mira Road')

------------------------------------------------------------


******Operations on database******
1. Create
2. Insert
3. Read
4. Update
5. Delete
6. Exit

Enter your Choice : 4

Successfully Updated the values!

------------------------------------------------------------


******Operations on database******
1. Create
2. Insert
3. Read
4. Update
5. Delete
6. Exit

Enter your Choice : 3

Data in person table :  ('Gini', 'Chacko', '20', 'Kerala')

------------------------------------------------------------
```

```
******Operations on database******
1. Create
2. Insert
3. Read
4. Update
5. Delete
6. Exit

Enter your Choice : 5

Successfully Deleted the values!

-----------------------------------------------------------------


******Operations on database******
1. Create
2. Insert
3. Read
4. Update
5. Delete
6. Exit

Enter your Choice : 6
Exiting!!

-----------------------------------------------------------------
```

POSTLAB QUESTIONS:

1) List and explain properties of cursor object in python.

Ans:

Cursor.__enter__()

The entry point for the cursor as a context manager. It returns itself.

Cursor.__exit__()

The exit point for the cursor as a context manager. It closes the cursor.

Cursor.arraysize

This read-write attribute can be used to tune the number of rows internally fetched and buffered by internal calls to the database when fetching rows from SELECT statements and REF CURSORS.

Cursor.bindarraysize

This read-write attribute specifies the number of rows to bind at a time and is used when creating variables via **setinputsizes()** or **var()**. It defaults to 1 meaning to bind a single row at a time.

Cursor.arrayvar(*dataType, value*[, *size*])

Create an array variable associated with the cursor of the given type and size and return a variable object.

Cursor.bindnames()

Return the list of bind variable names bound to the statement. Note that a statement must have been prepared first.

Cursor.bindvars

This read-only attribute provides the bind variables used for the last execute.

Cursor.callfunc(*name, returnType, parameters=[], keywordParameters={}*)

Call a function with the given name. The return type is specified in the same notation as is required by **setinputsizes()**.

Cursor.callproc(*name, parameters=[], keywordParameters={}*)

Call a procedure with the given name. The sequence of parameters must contain one entry for each parameter that the procedure expects.

Cursor.close()

Close the cursor now, rather than whenever __del__ is called. The cursor will be unusable from this point forward; an Error exception will be raised if any operation is attempted with the cursor.

Cursor.connection

This read-only attribute returns a reference to the connection object on which the cursor was created.

Cursor.description

This read-only attribute is a sequence of 7-item sequences. Each of these sequences contains information describing one result column: (name, type, display_size, internal_size, precision, scale, null_ok).

Cursor.execute(*statement,* **[***parameters,* **]*****keywordParameters*)

Parameters may be passed as a dictionary or sequence or as keyword parameters. If the parameters are a dictionary, the values will be bound by name and if the parameters are a sequence the values will be bound by position

Cursor.executemany(*statement, parameters, batcherrors=False, arraydmlrowcounts=False*)

The statement is managed in the same way as the **execute()** method manages it.

Cursor.executemanyprepared(*numIters*)

Execute the previously prepared and bound statement the given number of times.

Cursor.fetchall()

Fetch all (remaining) rows of a query result, returning them as a list of tuples..

Cursor.fetchmany([*numRows=cursor.arraysize*])

Fetch the next set of rows of a query result, returning a list of tuples.

Cursor.fetchone()

Fetch the next row of a query result set, returning a single tuple or None when no more data is available. An exception is raised if the previous call to **execute()** did not produce any result set or no call was issued yet.

Cursor.fetchraw([*numRows=cursor.arraysize*])

Fetch the next set of rows of a query result into the internal buffers of the defined variables for the cursor. The number of rows actually fetched is returned. \

Cursor.fetchvars

This read-only attribute specifies the list of variables created for the last query that was executed on the cursor.

Cursor.getarraydmlrowcounts()

Retrieve the DML row counts after a call to **executemany()** with arraydmlrowcounts enabled. This will return a list of integers corresponding to the number of rows affected by the DML statement for each element of the array passed to **executemany()**.

Cursor.getbatcherrors()

Retrieve the exceptions that took place after a call to **executemany()** with batcherrors enabled. This will return a list of Error objects, one error for each iteration that failed.

Cursor.getimplicitresults()

Return a list of cursors which correspond to implicit results made available from a PL/SQL block or procedure without the use of OUT ref cursor parameters

Cursor.inputtypehandler

This read-write attribute specifies a method called for each value that is bound to a statement executed on the cursor and overrides the attribute with the same name on the connection if specified.

Cursor.__iter__()

Returns the cursor itself to be used as an iterator.

.

Cursor.lastrowid

This read-only attribute returns the rowid of the last row modified by the cursor. If no row was modified by the last operation performed on the cursor, the value None is returned.

Cursor.outputtypehandler

This read-write attribute specifies a method called for each column that is to be fetched from this cursor.

Cursor.parse(*statement*)

This can be used to parse a statement without actually executing it (this step is done automatically by Oracle when a statement is executed).

Cursor.prefetchrows

This read-write attribute can be used to tune the number of rows that the Oracle Client library fetches when a SELECT statement is executed.

Cursor.prepare(*statement*[, *tag*])

This can be used before a call to **execute()** to define the statement that will be executed. When this is done, the prepare phase will not be performed when the call to **execute()** is made with None or the same string object as the statement

Cursor.rowcount

This read-only attribute specifies the number of rows that have currently been fetched from the cursor (for select statements), that have been affected by the operation (for

insert, update, delete and merge statements), or the number of successful executions of the statement (for PL/SQL statements).

Cursor.rowfactory

This read-write attribute specifies a method to call for each row that is retrieved from the database.

Cursor.scroll(*value=0, mode="relative"*)

Scroll the cursor in the result set to a new position according to the mode.

Cursor.scrollable

This read-write boolean attribute specifies whether the cursor can be scrolled or not.

Cursor.setinputsizes(**args, **keywordArgs*)

This can be used before a call to **execute()**, **callfunc()** or **callproc()** to predefine memory areas for the operation's parameters.

Cursor.setoutputsize(*size*[, *column*])

This method does nothing and is retained solely for compatibility with the DB API. The module automatically allocates as much space as needed to fetch LONG and LONG RAW columns (or CLOB as string and BLOB as bytes).

Cursor.statement

This read-only attribute provides the string object that was previously prepared with **prepare()** or executed with **execute()**.

2) Explain three different modules that can be used in python for connecting to Mysql server using example code.

Ans:

The following example shows how to connect to the MySQL server:

```
import mysql.connector

cnx = mysql.connector.connect(user='scott', password='password',
                              host='127.0.0.1',
                              database='employees')
cnx.close()
```

It is also possible to create connection objects using the connection.MySQLConnection() class:

```
from mysql.connector import (connection)

cnx = connection.MySQLConnection(user='scott', password='password',
                                 host='127.0.0.1',
                                 database='employees')
cnx.close()
```

Defining connection arguments in a dictionary and using the ** operator is another option:

```
import mysql.connector

config = {
  'user': 'scott',
  'password': 'password',
  'host': '127.0.0.1',
  'database': 'employees',
  'raise_on_warnings': True
}

cnx = mysql.connector.connect(**config)

cnx.close()
```

The following example shows how to set use_pure to False.

```
import mysql.connector

cnx = mysql.connector.connect(user='scott', password='password',
                              host='127.0.0.1',
                              database='employees',
                              use_pure=False)
cnx.close()
```