Persistent

Database Programming

Objectives

At the end of this second session, you will be able to understand:

- Database Programming in Python

Database connectivity modules - sqlite3, MySQLdb, mysql.connector

Introduction to database programming in Python

Database access is a large part of many applications, including dynamic web applications. By using external modules, Python can access most popular databases, and in general the interface for each follows the DB-API 2.0 standard database specification detailed in PEP (Python Enhancement Proposal) 249.

The specification calls for the use of a connection object to manage the connection to the database and for the use of cursor objects to manage the interaction with the database, for fetching data from the database and updating its contents.

The fact that Python database libraries conform to the DB-API 2.0 spec has a couple of obvious advantages. For one thing, writing code for different databases is easier, because the general rules are the same.

The other advantage is that it's fairly easy to prototype an application using a lightweight database and then switch the application over to a production database after the basic design of the application has been finalized.

Using the sqlite3 database

- Python provides many modules for many databases. This presentation will cover the database that comes included with Python: sqlite3.
- SQLite3 is not suited for large, high-traffic applications.
- Sqlite3 however provides two advantages. The first is, because it’s part of the standard library it can be used anywhere you need a database, without worrying about adding dependencies.
- The second benefit is that sqlite3 stores all its records in a local file, so it doesn’t need both a client and server, like MySQL or other common databases.
- These features make sqlite3 a handy option for both smaller applications and quick prototypes.

Accessing a database from Python using sqlite3

- To use a sqlite3 database, the first thing is to import the sqlite3 module as shown below:

```
>>> import sqlite3
```

- Then get a connection object also known as database handle by calling the connect function with the name of file that will be used to store the data as shown below:

```
>>> conn = sqlite3.connect(‘<name of database file>')
```

- The next step is to create a cursor object from the connection as shown below:

```
>>> cursor = conn.cursor()
>>> cursor
<sqlite3.Cursor object at 0xb7a12980>
```

- At this point, its possible to make queries against the database.

Accessing a database from Python using sqlite3 (contd.)

- The cursor object can be used not only to query database but also to create tables and insert records into database tables.
- The example below creates a table and inserts a couple of records as shown below:

```
>>> cursor.execute("create table test (name text, count integer)")

>>> cursor.execute("insert into test (name, count) values ('Bob', 1)")

>>> cursor.execute("insert into test (name, count) values (?, ?)",

... ("Jill", 15))
```

- The last insert query illustrates the preferred way to make a query with variables. Rather than constructing the query string, it's more secure to use a? for each variable and then pass the variables as a tuple parameter to the execute method.
- The advantage is that a developer doesn't need to worry about incorrectly escaping a value; sqlite3 takes care of it.

Using the cursor object

- With the table populated as shown in the previous slide the following code shows how the table can be queried using SQL commands using ? for variable binding as shown below.

```
>>> result = cursor.execute("select * from test")

>>> print(result.fetchall())

[('Bob', 1), ('Jill', 15), ('Joe', 10)]

>>> result = cursor.execute("select * from test where name like :name",

... {"name": "bob"})

>>> print(result.fetchall())

[('Bob', 1)]
```

Using the cursor object (contd.)

- The cursor object can also be used to update records in a database table as shown below:

```
>>> cursor.execute("update test set count=? where name=?", (20, "Jill"))
>>> result = cursor.execute("select * from test")
>>> print(result.fetchall())
[('Bob', 1), ('Jill', 20), ('Joe', 10)]
```

- In addition to the fetchall method, the fetchone method gets one row of the result as shown below:

```
>>> row = result.fetchone()
```

- The fetchmany returns an arbitrary number of rows.

Using a for loop to display query results

- Python allows to use a for loop to process the results of a query one record at a time as shown below:
- **Syntax:** for record in cursor:

```
        # code to process the record goes here
```

- **Example:**

```
>>> result = cursor.execute("select * from test")
>>> for row in result:
... print(row)
```

The above code gives the following output/

```
('Bob', 1)
('Jill', 20)
('Joe', 10)
```

Python MySQL database access

- The Python standard for database interfaces is the Python DB-API. Most Python database interfaces adhere to this standard.
- You can choose the right database for your application. Python Database API supports a wide range of database servers such as:
  - GadFly
  - mSQL
  - MySQL
  - PostgreSQL
  - Microsoft SQL Server 2000
  - Informix
  - Interbase
  - Oracle
  - Sybase

MySQLdb module

You must download a separate DB API module for each database you need to access. For example, if you need to access an Oracle database as well as a MySQL database, you must download both the Oracle and the MySQL database modules. The DB API provides a minimal standard for working with databases using Python structures and syntax wherever possible. This API includes the following:

- Importing the API module.
- Acquiring a connection with the database.
- Issuing SQL statements and stored procedures.
- Closing the connection.
- We would learn all the concepts using MySQL, so let us talk about MySQLdb module for Python2 and mysql.connector for Python3 version.

MySQLdb module (contd.)

What is MySQLdb?

- MySQLdb is an interface for connecting to a MySQL database server from Python. It implements the Python Database API v2.0 and is built on top of the MySQL C API.

How do I Install MySQLdb?

- Before proceeding, you make sure you have MySQLdb installed on your machine. Just type the following in your Python script and execute it:

```
#!/usr/bin/python

import MySQLdb
```

MySQLdb module (contd.)

If it produces the following result, then it means MySQLdb module is not installed:

Traceback (most recent call last): File "test.py", line 3, in <module> import MySQLdb

ImportError: No module named MySQLdb

To install MySQLdb module, use the following commands:

For Ubuntu, use the following commands:

$ sudo apt-get install python-pip python-dev libmysqlclient-dev

For Fedora, use the following commands:

$ sudo dnf install python python-devel mysql-devel redhat-rpm-config gcc

For Window Python command prompt, use the following command –

pip install MySQLdb

MySQLdb module (contd.)

Database Connection

Before connecting to a MySQL database, make sure of the following:

- You have created a database TEST.
- You have created a table EMPLOYEE in TEST.
- This table has fields FIRST_NAME, LAST_NAME, AGE, SEX and INCOME.
- User ID “root" and password “root" are set to access TEST.
- Python module MySQLdb is installed properly on your machine.
- You have gone through MySQL tutorial to understand MySQL Basics.

MySQLdb module (contd.)

Creating Database Table

import MySQLdb #this is not a part of standard Python library set

# Open database connection
db = MySQLdb.connect("localhost","root","root","test")

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

# Drop table if it already exist using execute() method.
cursor.execute("DROP TABLE IF EXISTS EMPLOYEE")

# Create table as per requirement

```
sql = """CREATE TABLE EMPLOYEE (
         FIRST_NAME  CHAR(20) NOT NULL,
         LAST_NAME  CHAR(20),
         AGE INT,
         SEX CHAR(1),
         INCOME FLOAT )"""
```

cursor.execute(sql)

# disconnect from server db.close()

Persistent

MySQLdb module (contd.)

INSERT Operation

It is required when you want to create your records into a database table.

```
import MySQLdb

# Open database connection
db = MySQLdb.connect("localhost","root","root","test" )

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

# Prepare SQL query to INSERT a record into the database.
sql = """INSERT INTO EMPLOYEE(FIRST_NAME,
         LAST_NAME, AGE, SEX, INCOME)
         VALUES ('Mac', 'Mohan', 20, 'M', 2000)"""

try:
    # Execute the SQL command
    cursor.execute(sql)

    # Commit your changes in the database
    db.commit()

except:
    # Rollback in case there is any error
    db.rollback()

    # disconnect from server
    db.close()
```

MySQLdb module (contd.)

INSERT Operation

```
#Previous example can be written as follows to create
SQL queries dynamically: import MySQLdb

# Open database connection
db = MySQLdb.connect("localhost","root","root","test" )

# prepare a cursor object using cursor() method
cursor = db.cursor()
# Prepare SQL query to INSERT a record into the
database.
sql = "INSERT INTO EMPLOYEE(FIRST_NAME, \
        LAST_NAME, AGE, SEX, INCOME) \
        VALUES ('%s', '%s', '%d', '%c', '%d' )" % \
        ('Mac', 'Mohan', 20, 'M', 2000)
try:
        # Execute the SQL command
        cursor.execute(sql)

        # Commit your changes in the database
        db.commit()

except:
        # Rollback in case there is any error
        db.rollback()

        # disconnect from server
        db.close()
```

MySQLdb module (contd.)

INSERT Operation

Example

- Following code segment is another form of execution where you can pass parameters directly:

```
..................................
user_id = "test123"
password = "password"
con.execute('insert into Login values("%s", "%s")' % \
(user_id, password))
..................................
```

Persistent

MySQLdb module (contd.)

READ Operation

- READ Operation on any database means to fetch some useful information from the database.
- Once our database connection is established, you are ready to make a query into this database. You can use either **fetchone()** method to fetch single record or **fetchall()** method to fetch multiple values from a database table.
- **fetchone():** It fetches the next row of a query result set. A result set is an object that is returned when a cursor object is used to query a table.
- **fetchall():** It fetches all the rows in a result set. If some rows have already been extracted from the result set, then it retrieves the remaining rows from the result set.
- **rowcount:** This is a read-only attribute and returns the number of rows that were affected by an execute() method.

MySQLdb module (contd.)

READ Operation Example

```
import MySQLdb
# Open database connection
db = MySQLdb.connect("localhost","root","root","test" )
# prepare a cursor object using cursor() method
cursor = db.cursor()
# Prepare SQL query to INSERT a record into the
database.
sql = "SELECT * FROM EMPLOYEE \
        WHERE INCOME > '%d'" % (1000)
try:
        # Execute the SQL command cursor.execute(sql)
        # Fetch all the rows in a list of lists.
        results = cursor.fetchall() for row in results:
                fname = row[0]
                lname = row[1]
                age = row[2]
                sex = row[3]
                income = row[4]
                # Now print fetched result
                Print "fname=%s,lname=%s,age=%d,sex=
                        %s,income=%d" % \ (fname, lname,
                        age, sex, income)
except:
        print "Error: unable to fecth data"
        # disconnect from server
        db.close()  #o/p: fname=Mac, lname=Mohan,
                          age=20, sex=M, income=2000
```

Persistent

MySQLdb module (contd.)

UPDATE Operation

```
import MySQLdb

# Open database connection
db = MySQLdb.connect("localhost","root","root","test" )

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

# Prepare SQL query to UPDATE required records
sql = "UPDATE EMPLOYEE SET AGE = AGE + 1
        WHERE SEX = '%c'" % ('M')

try:
    # Execute the SQL command
    cursor.execute(sql)

    # Commit your changes in the database
    db.commit()

except:
    # Rollback in case there is any error
    db.rollback()

    # disconnect from server
    db.close()
```

MySQLdb module (contd.)

DELETE Operation

```
import MySQLdb
# Open database connection
db = MySQLdb.connect("localhost","root","root","test" )
# prepare a cursor object using cursor() method
cursor = db.cursor()
# Prepare SQL query to DELETE required records
sql = "DELETE FROM EMPLOYEE WHERE AGE >
        '%d'" % (20)
try:
        # Execute the SQL command
        cursor.execute(sql)
        # Commit your changes in the database
        db.commit()
except:
        # Rollback in case there is any error
        db.rollback()
        # disconnect from server
        db.close()
```

MySQLdb module (contd.)

DELETE Operation

```
import MySQLdb

# Open database connection
db = MySQLdb.connect("localhost","root","root","test" )

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

# Prepare SQL query to DELETE required records
sql = "DELETE FROM EMPLOYEE WHERE AGE >
      '%d'" % (20)

try:
    # Execute the SQL command
    cursor.execute(sql)
    # Commit your changes in the database
    db.commit()
except:
    # Rollback in case there is any error
    db.rollback()
    # disconnect from server
    db.close()
```

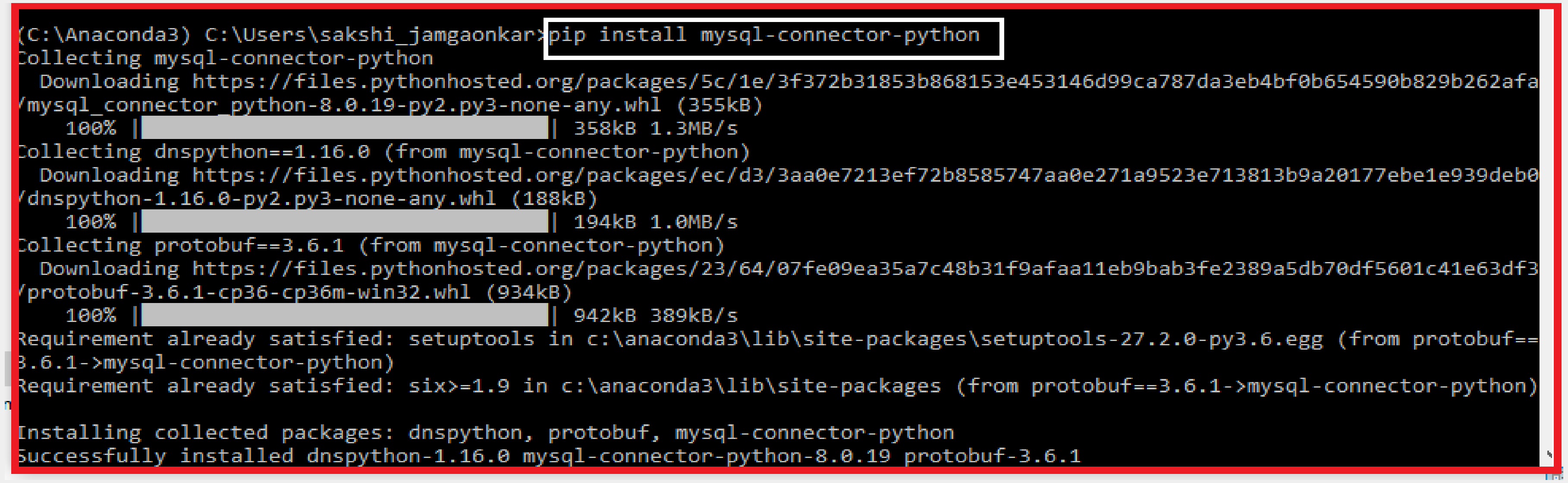
MySQL Connector Python - mysql.connector module

- MySQL Connector Python has the following advantages: –
  - MySQL Connector Python is written in pure Python, and it is self-sufficient to execute database queries through python.
  - It is an official Oracle-supported driver to work with MySQL and python.
  - It is Python 3 compatible, actively maintained.

- This Python MySQL session mainly focuses on: –
  - How to install MySQL Connector Python and use its functions to access the MySQL database.
  - Perform MySQL CRUD operations such as data insertion, data retrieval, data update, and data deletion using Python.

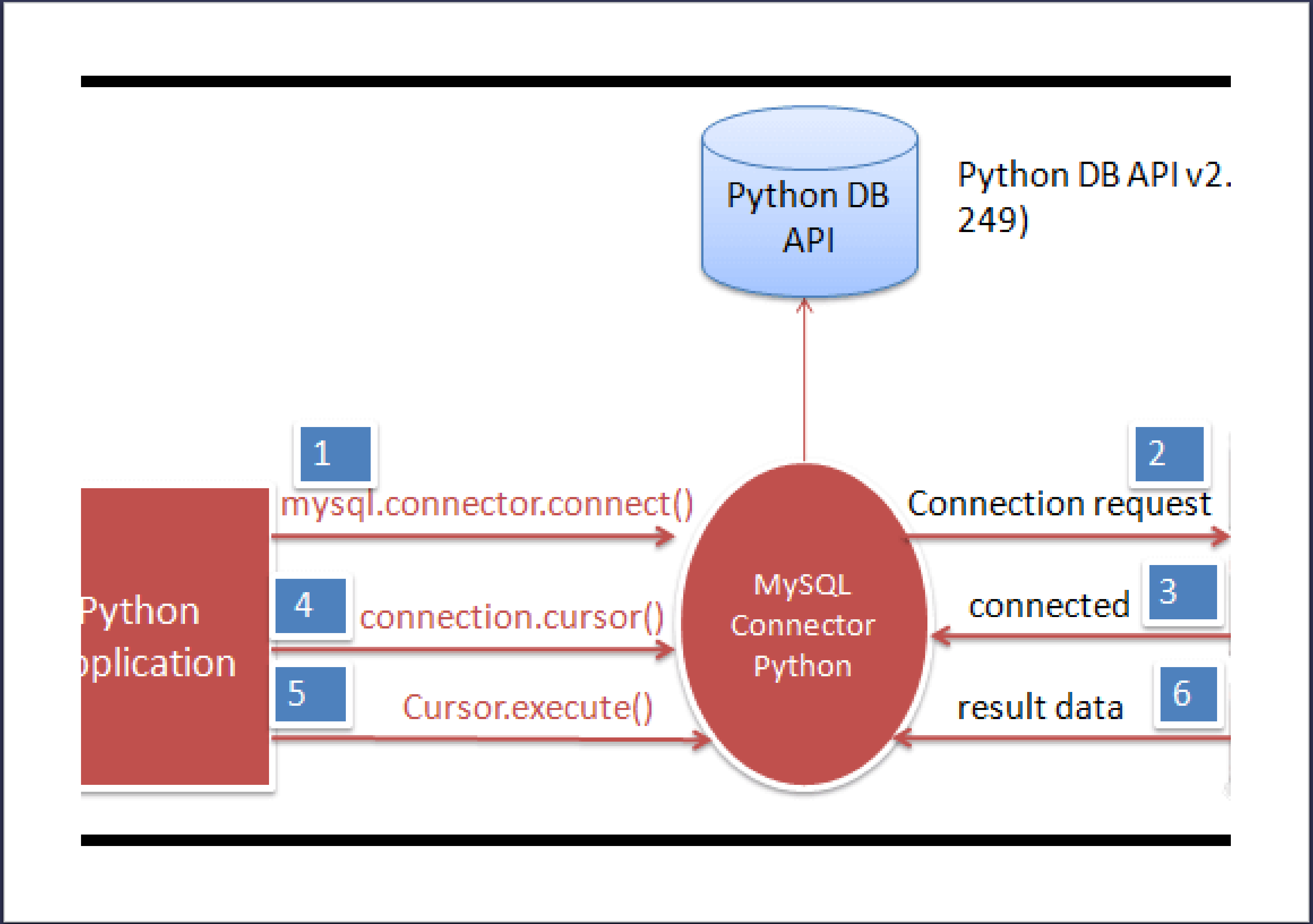
mysql.connector module Installation

- **mysql-connector-python module Installation:**
- **Using *pip* command :**
  - Open Anaconda Command prompt as administrator :
  - Go to start menu ->Anaconda 3->Anaconda prompt
  - Use **cd** to come out of set directory or path.
  - Run **pip install** command as:
  - **>pip install mysql-connector-python**



```
(C:\Anaconda3) C:\Users\sakshi_jamgaonkar>pip install mysql-connector-python
Collecting mysql-connector-python
  Downloading https://files.pythonhosted.org/packages/5c/1e/3f372b31853b868153e453146d99ca787da3eb4bf0b654590b829b262afa
/mysql_connector_python-8.0.19-py2.py3-none-any.whl (355kB)
    100% |████████████████████████████████| 358kB 1.3MB/s
Collecting dnspython==1.16.0 (from mysql-connector-python)
  Downloading https://files.pythonhosted.org/packages/ec/d3/3aa0e7213ef72b8585747aa0e271a9523e713813b9a20177ebe1e939deb0
/dnspython-1.16.0-py2.py3-none-any.whl (188kB)
    100% |████████████████████████████████| 194kB 1.0MB/s
Collecting protobuf==3.6.1 (from mysql-connector-python)
  Downloading https://files.pythonhosted.org/packages/23/64/07fe09ea35a7c48b31f9afaa11eb9bab3fe2389a5db70df5601c41e63df3
/protobuf-3.6.1-cp36-cp36m-win32.whl (934kB)
    100% |████████████████████████████████| 942kB 389kB/s
Requirement already satisfied: setuptools in c:\anaconda3\lib\site-packages\setuptools-27.2.0-py3.6.egg (from protobuf==
3.6.1->mysql-connector-python)
Requirement already satisfied: six>=1.9 in c:\anaconda3\lib\site-packages (from protobuf==3.6.1->mysql-connector-python)

Installing collected packages: dnspython, protobuf, mysql-connector-python
Successfully installed dnspython-1.16.0 mysql-connector-python-8.0.19 protobuf-3.6.1
```



Steps to connect MySQL database in Python using MySQL Connector Python

Python Example to connect MySQL Database

```
import mysql.connector

from mysql.connector import Error

try:

    connection =
mysql.connector.connect(host='localhost',

                                         database='test',

                                         user='root',

                                         password='root')

    if connection.is_connected():

        db_Info = connection.get_server_info()

        print("Connected to MySQL Server version ",
db_Info)
```

```
cursor = connection.cursor()

        cursor.execute("select database();")

 record = cursor.fetchone()

        print("You're connected to database: ", record)


except Error as e:

    print("Error while connecting to MySQL", e)

finally:

    if (connection.is_connected()):

        cursor.close()

        connection.close()

        print("MySQL connection is closed")
```

Persistent

Understand the Python MySQL Database connection program

- **import mysql.connector**

This line imports the MySQL Connector Python module in your program so you can use this module's API to connect MySQL.

- **from mysql.connector import Error**

mysql connector Error object is used to show us an error when we failed to connect Databases or if any other database error occurred while working with the database. Example ACCESS DENIED ERROR when username or password is wrong.

- **mysql.connector.connect()**

Using this method we can connect the MySQL Database, this method accepts four required parameters: Host, Database, User and Password that we already discussed.

connect() method established a connection to the MySQL database from Python application and returned a MySQLConnection object. Then we can use MySQLConnection object to perform various operations on the MySQL Database.

The Connect() method can throw an exception, i.e. Database error if one of the required parameters is wrong. For example, if you provide a database name that is not present in MySQL, then Python application throws an exception. So check the arguments that you are passing to this method.

Understand the Python MySQL Database connection program

- **connection.is_connected()**

is_connected() is the method of the MySQLConnection class through which we can verify is our python application connected to MySQL.

- **connection.cursor()**

This method returns a cursor object. Using a cursor object, we can execute SQL queries.

The MySQLCursor class instantiates objects that can execute operations such as SQL statements.

Cursor objects interact with the MySQL server using a MySQLConnection object.

- **cursor.close()**

Using the cursor's close method we can close the cursor object. Once we close the cursor object, we can not execute any SQL statement.

- **connection.close()**

At last, we are closing the MySQL database connection using a close() method of MySQLConnection class.

x

Python MySQL Create Table

```
import mysql.connector
from mysql.connector import Error
try:
    connection =
mysql.connector.connect(host='localhost',
                                         database='test',
                                         user='root',
                                         password='root')
    mySql_Create_Table_Query = """CREATE TABLE
Laptop (
                             Id int(11) NOT NULL,
                             Name varchar(250) NOT NULL,
                             Price float NOT NULL,
                             Purchase_date Date NOT NULL,
                             PRIMARY KEY (Id)) """

    cursor = connection.cursor()
    result =
cursor.execute(mySql_Create_Table_Query)
    print("Laptop Table created successfully ")
except Error as e:
    print("Error while connecting to MySQL", e)
finally:
    if (connection.is_connected()):
        cursor.close()
        connection.close()
        print("MySQL connection is closed")
```

Python Insert Into MySQL Table

```
import mysql.connector
from mysql.connector import Error
try:
    connection = mysql.connector.connect(host='localhost',
                                         database='test',
                                         user='root',
                                         password='root')
    mySql_insert_query = """INSERT INTO Laptop (Id, Name, Price, Purchase_date)
                           VALUES
                           (10, 'Lenovo ThinkPad P71', 6459, '2019-08-14') """
    cursor = connection.cursor()
    cursor.execute(mySql_insert_query)
    connection.commit()
    print(cursor.rowcount, "Record inserted successfully into Laptop table")
    cursor.close()

except Error as e:
    print("Error while connecting to MySQL", e)
finally:
    if (connection.is_connected()):
        cursor.close()
        connection.close()
        print("MySQL connection is closed")
```

Python Select from MySQL Table

```
import mysql.connector
from mysql.connector import Error
try:
    connection = 
mysql.connector.connect(host='localhost',
                                         database='test',
                                         user='root',
                                         password='root')
    sql_select_Query = "select * from Laptop"
    cursor = connection.cursor()
    cursor.execute(sql_select_Query)
    records = cursor.fetchall()
    print("Total number of rows in Laptop is: ", 
cursor.rowcount)
 print ("All fetched records = ",records)#list of tuple 
records
    print("\nPrinting each laptop record")
    for row in records:
        print("Id = ", row[0], )
        print("Name = ", row[1])
        print("Price  = ", row[2])
        print("Purchase date  = ", row[3], "\n")
except Error as e:
    print("Error reading data from MySQL table", e)
finally:
    if (connection.is_connected()):
        connection.close()
        cursor.close()
        print("MySQL connection is closed")
```

Python access Oracle database: cx_Oracle

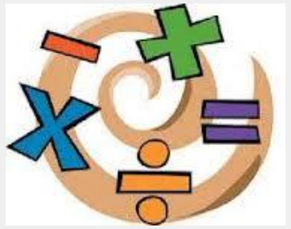
- **cx_Oracle** is the module used to connect to Oracle Database from python. cx_Oracle.connect() is used to establish the connection and execute() method is used to run any valid query over the database.
- **cur.fetchall()** method can be used to retrieve all the records from the cursor object.
- cx_Oracle is not a part of standard installation of python. It can be downloaded using command prompt and executing command: **pip install cx_Oracle** at the Scripts folder location of your Python installation.

```
>>> import cx_Oracle
>>> con = cx_Oracle.connect("username/password@127.0.0.1/oracleSID")
>>> cur = con.cursor()
>>> cur.execute('select * from departments order by department_id')
>>> for fetchedRows in cur:
                print (fetchedRows)
>>> cur.close()
>>> con.close()
```

Persistent

Assignments

1. Write Python script for database connectivity using sqlite3 or MySQLdb module. Create a table from the script, please drop it if it already exists.



Persistent

Assignments (contd.)

CREATE TABLE users(id INTEGER PRIMARY KEY, name TEXT, phone TEXT, email TEXT unique, password TEXT)

Accept some values for variables –

name1, phone1, email1, password1 , eg.

name1 = 'Ravi Verma'

phone1 = '9923849335'

email1 = 'ravi_verma@gmail.com’

password1 = 'tough@password!!’

- Insert these values in table users.
- Read the data back and display.
- Accept name from user, display the details for that user if it exists in table users.

Summary

With this we have come to the end of our session, where we discussed about:

- Database Programming

Reference material

- http://www.tutorialspoint.com/python
- http://www.learnpython.org/
- http://www.sqlite.org
- http://docs.python.org/2/tutorial/
- https://docs.python.org/2/tutorial/stdlib.html
- https://docs.python.org/2/tutorial/stdlib2.html
- https://packaging.python.org/installing/
- https://docs.python.org/2/library/socket.html

Questions

Key contacts

Sakshi Jamgaonkar

sakshi_jamgaonkar@persistent.com

Asif Immanad

asif_immanad@persistent.co.in

Persistent

Persistent

Thank you!