Building MySQL python applications

Topic 4 Lesson 8

Python database objects

- 2 main classes for processing database queries
  - **Connection object**
    - Connection to the database
    - Object created via the connection (.connection) method
  - **Cursor object**
    - Query statement execution
      - Method to execute a statement (.execute)
    - Result to the results
      - Method to retrieve row of data from the results (variations of fetch)
    - Cursor object created by the cursor method (.cursor) of the connection object.
    - Method to run a MySQL procedure (.callproc)

Process for accessing database

1. Import the MySQL API module
2. Acquire a connection to a specific database
3. Issue SQL statements and stored procedures.
4. Close the connection

Database (API)s

Add a library with database calls (API)

- Special standardized interface: procedures/objects
- Pass SQL strings from host language, presents result sets in a host language-friendly way

A "driver" traps the calls and translates them into DBMS specific code (Oracle, MySQL, SQL Server etc.)

- database can be across a network

GOAL: applications are independent of database systems and operating systems

Python connection library

Mysqlclient: a wrapper around the mysql-connector-c C library. You should have a development C environment set up to compile C code to use this library.

Pymysql: pure python implementation. It tends to be available quicker for the newer versions of python.

mysql-connection-python: Developed from the MySQL group at Oracle. Another pure python implementation.

mysql-connector: Original connector from MySQL

Python mysql.connector example

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# Simple MySQL database connection

import flask
import mysql.connector


def main(config):
    output = []
    cnx = mysql.connector.connect(**config)

    cur = cnx.cursor()
    cur2 = cnx.cursor()
    reb = 'rebels'
    movie_id = 1
    stmt_select = "select * from characters order by character_name"

    cur.execute(stmt_select)
    for row in cur.fetchall():
        output.append('{0:20s}  {1:15s} {2:15s}
            {3:15s}'.format(row[0], row[1], row[2], row[3]))
    cur.close()
```

Python mysql.connector (cont.)

```
s2 = 'SELECT * FROM movies WHERE movie_id = {}'.format(movie_id)
        cur2.execute(s2)
        for row in cur.fetchall():
            print(row)

        cur2.callproc('track_planet', args=['Endor'])

        for result in cur2.stored_results():
            print(result.fetchall())

        cur2.close()

        return output


    if __name__ == '__main__':
        config = {
            'host': 'localhost',
            'port': 3306,
            'database': 'starwarsfinal',
            'user': 'root',
            'password': 'root',
            'charset': 'utf8',
            'use_unicode': True,
            'get_warnings': True,
        }

        out = main(config)
        print('\n'.join(out))
```

Example pymysql (connect & retrieve data)

```
import pymysql


    cnx = pymysql.connect(host='localhost', user='root', password='root',
                          db='lotrfinal', charset='utf8mb4',
    cursorclass=pymysql.cursors.DictCursor)

    cur = cnx.cursor()
    stmt_select = "select * from lotr_character order by
    character_name"

    cur.execute(stmt_select)

    rows = cur.fetchall()
```

Pymysql provides different cursors

Pymysql.cursors.SSCursor : an unbuffered cursor, useful for queries that returns many rows or for connections on remote servers. Instead of copying every row of data to the buffer, this will fetch rows as needed

Pymyql.cursors.DictCursor: returns the result as a dictionary, where the key is the field name and the value is the field value

Pymysql.cursors.SSDictCursor: an unbuffered cursor, which returns the results as a dictionary {field_name: field_value}

Example pymysql (process cursor)

```
for row in rows:
    print(row) # prints each field as a key value pair
    print(row["character_name"], row['species'])
    #reference field by name
    c_name_var = row["character_name"]
    # get specific values
  cur.close()
```

Prepared statements

Any SQL statement can be made into a prepared statement by using the character string %s to specify a value that will be provided at execution time:

Example:

```
species = 'elf'
cursor = cnx.cursor()
query = "SELECT character_name FROM lotr_character WHERE species=%s"
cursor.execute(query, species)
# ... retrieve data ...
```

Tuples affected by the query

The cursor method, rowcount, returns the number of tuples affected or returned by the SQL statement. For example, if cur is the cursor result of a SELECT statement

```
print("The query returned {} rows".format(cur.rowcount))
```

Prints the number of rows returned.

```
The query returned 2 rows
```

Starting Points

For pymysql:
https://pypi.org/project/PyMySQL/
https://www.tutorialspoint.com/python3/python_database_access.htm
https://pymysql.readthedocs.io/en/latest/modules/index.html

For mysqlclient-python:
https://pypi.org/project/mysqlclient/

For mysql-connection
https://dev.mysql.com/doc/connector-python/en/connector-python-versions.html

For a comparison of the approaches
https://wiki.openstack.org/wiki/PyMySQL_evaluation

Python Summary

There are many different libraries for connecting a python application to a MySQL database. Pymysql is written entirely in python and does not require a C development environment. It also provides 3 different types of cursor objects.

Handling OUT and INOUT parameters to python from MySQL requires the use of wrapper parameter that runs on the DB server. It extracts the values from the session variables into a cursor.