COMP3311 19T3 Database Systems

Prac Exercise 06
Psycopg2 - Using Python with SQL

Aims

This exercise aims to give you practice in:

- implementing Python scripts to manipulate databases

This exercise will **not** explain how to do everything in fine detail. Part of the aim of the exercise is that you explore how to use python with PostgreSQL.

Background

This document describes psycopg2, a simple programatic interface to PostgreSQL databases from Python that we'll use in COMP3311. A number of interfaces exist, but this interface is simple to learn with basic python knowledge.

Many psycopg2 tutorials can be found online, including this one.

Installation

psycopg2 is python module that can be installed with pip.

On CSE machines, psycopg2 binary is already installed

On your own unix machine, psycopg2 binary can be installed with pip:

```
pip install psycopg2-binary
```

For a full installation, or other kinds of installation, you can view the installation docs

Examples

Connecting

For any operations with the database you will need a valid connection handle. In our examples we will use the variable **conn**. The connection to the database will be omitted from all remaining examples, even though if you were to test it out you need to include it.

When connecting to a psql server on CSE's grieg, the connection string required to be passed to **psycopg2.connect()** is simply

```
dbname='myDatabaseName'
```

where myDatabaseName is the name of your database (for example "ass2")

If you are running psql on your own machine, you will likely need to provide additional credentials. An example string that would be passed to psycopg2.connect() would be:

```
dbname='myDatabaseName' user='john' host='localhost' password='shepherd'
```

This is assuming a database with credentials:

- username: john
- password: sheperd
- host: localhost
- database: comp3311

```
import psycopg2

try:
    conn = psycopg2.connect("dbname='myDatabaseName'")
except Exception as e:
    print("Unable to connect to the database")
    print(e)

# =================
# DO DATABASE STUFF
# =================

conn.close() # Close the DB connection
```

CREATE

We will use psycopg2 to create a table

```
cur = conn.cursor()
try:
    cur.execute("CREATE TABLE Person (id serial PRIMARY KEY, name varchar, age integer);")
except Exception as e:
    print("Error creating table User")
    print(e)

cur.close() # No longer need the cursor
conn.commit() # Need to commit changes to the DB
```

This will create a table in the database that looks like this (empty because no rows have been added)

id	name	age

INSERT

Note: When running an insert query, note that you may have to manually include apostrophe's such as the example below when dealing with non-numerical types

```
cur = conn.cursor()
try:
    cur.execute("INSERT INTO Person (name, age) VALUES ('{}', {})".format("John Shepherd", 38))
    cur.execute("INSERT INTO Person (name, age) VALUES ('{}', {})".format("Andrew Taylor", 20))
except Exception as e:
    print("Error inserting into Person table")
    print(e)

cur.close() # No longer need the cursor
conn.commit() # Need to commit changes to the DB
```

After this has been inserted, the current database will look like

id	name	age
1	John Shepherd	38
2	Andrew Taylor	20

UPDATE

```
cur = conn.cursor()
try:
    cur.execute("UPDATE Person SET age = {} WHERE name LIKE '%{}%'".format(50,"John"))
except Exception as e:
    print("Error updating Person table")
    print(e)

cur.close() # No longer need the cursor
conn.commit() # Need to commit changes to the DB
```

After this operation the table will now look like

id	name	age
1	John Shepherd	50
2	Andrew Taylor	20

SELECT

```
cur = conn.cursor()
try:
    cur.execute("SELECT * FROM Person WHERE age > 10")
except Exception as e:
    print("Error updating Person table")
    print(e)

rows = cur.fetchall()
for row in rows:
      print(row)
```

```
cur.close() # No longer need the cursor
```

This program will print the following:

```
(1, John Shepherd, 50)
(2, Andrew Taylor, 20)
```

Note: We don't need **conn.commit()** since nothing transactional is happening and no writing is occurring

REMOVE

```
cur = conn.cursor()
try:
    cur.execute("DELETE FROM Person WHERE age >= {}".format(30))
except Exception as e:
    print("Error deleting from Person table")
    print(e)

cur.close() # No longer need the cursor
conn.commit() # Need to commit changes to the DB
```

After this operation, the table will now look like

id	name	age
2	Andrew Taylor	20

Exercises

Database Setup You can use your database from assignment 2 for these exercises. If you need instructions on setup, please go see that.

cs3311.py

Our solutions to the prac exercises all import a common file **cs3311.py**. This file is there to abstract the common step of connecting to the DB that all files require. You could easily put the connection step explicitly in every file, if you really wanted to.

Questions

Question 1

Produce a list of names, and for each name, what their mostly highly rated title they've acted in is. If they haven't acted in a title, exclude them from the list.

Expected result

Result is too long to display, simply use the script in solutions

Question 2

Give 1 integer argument (argv), produce a histogram of all birth years of people with first names less than 4 characters and their frequency. Exclude any birth years where no names fit that criteria

Expected result

When running **python3 q2.py**

```
1953: ['Ty Henderson']
1955: ['Ed Blakely', 'Jo Gilbert', 'Al Dobalo']
1951: ['Pj Torokvei', 'Mo Claridge', 'Al Cowens']
1956: ['Zé Pedro', 'Ze Ze Ngambi']
1971: ['DJ Screw']
1967: ['Ro Gorski']
1950: ['Ko Murobushi', 'Om Puri', 'Ed Turner', 'Ad Tamboer', 'Al Moller', 'Bo Rather']
1952: ['de los Reyes', 'La Donna Mabry', 'Ed Blaylock', 'Bi Skaarup']
1966: ['Ed Forsdick', 'Bo von Der Lippe']
1969: ['Jo Dunne']
1976: ['Mo Collins', 'Yu Li']
1961: ['Os', '3 Steps Ahead']
1963: ['Al Fritsch']
```

```
1958: ['Jo Schmidt', 'El Texano', 'Ed Cray', 'Ai Saotome']
1957: ['Al Ashton', 'Ed Crabtree', 'Bo Griffin', 'Ed Dolan']
1954: ['Jo Andres', 'La Marca', 'Ed Schultz', 'El Hortelano', 'Ab Harrewijn']
1964: ['La Veneno']
1962: ['El Gaucho Bataraz']
1968: ['El Hadji Samba Sarr']
1973: ['Ai Iijima', 'DJ AM']
1972: ['Ed Vassallo', 'De Laurentis', 'Vu Anh']
1983: ['Jo Dreihann-Holenia']
1974: ['RJ Rosales', 'Jo Cox']
1978: ['DJ Megatron']
1988: ['Bo Hu']
1990: ['Jo Maycock']
1993: ['AJ Perez']
```

Question 3

Given a name, produce a filmography:

Name (born: birth_year, [died: death_year])
\t Movie1 title (year): [role(s) (i.e. actor, director,...)]
\t Movie2 title (year): [role(s) (i.e. actor, director,...)]
etc. etc.

ordered by year, then title

Expected result

When running for example **python3 q3.py "Ro Gorski"**

```
Ro Gorski (born: 1967, died: 2008)
```

Question 4

Given a movie name, produce a list of principals

Movie title (start_year)
\t Name 1, role(s) (i.e. actor, director,...)
\t Name 2, role(s) (i.e. actor, director,...)
etc. etc.

Using the ordering field from the Principals table

Expected result

When running for example **python3 q4.py "All Male, All Nude"**

```
All Male, All Nude (start_year: 2017)
    Steven Marchi: ['self']
```

Question 5

Write a function that, given part of a title, shows the full title and the total size of the cast and crew

Expected result

When running for example **python3 q4.py "All Male, All Nude"**

```
All Male, All Nude has 1 cast and crew
```

Question 6

Write python that, when given a crew member's name, the title they crewed for, and their new job, it updates the database by creating a new role to describe the relationship between the crew member and the title. If an invalid name or title is given, terminate.

Expected result

Since this is an update, play around with the solution

Solutions

You should attempt the above exercises before looking at the sample solutions.