Assignment 2

Python, PostgreSQL, IMDB

Last updated: **Wednesday 1st March 3:47pm**

Most recent changes are shown in red ... older changes are shown in brown.

[Specification] [Database] [Examples] [Testing] [Submitting]

Aims

This assignment aims to give you practice in

- implementing Python scripts to extract and display data from a database
- [optionally] implementing SQL views and PLpgSQL functions to support the scripts
- [optionally] implementing a collection of Python functions to support the scripts

The goal is to build some useful data access operations on the Movie database.

Summary

Marks:	This assignment contributes **20 marks** toward your total mark for this course.
Submission:	via Webcms3 or `give`, submit the files `dirby`, `releases`, `genres`, `roles`, `movie`, `helpers.py`, `helpers.sql`
Deadline:	Friday 11 November 2022 @ 21:00
Late Penalty:	0.05 *marks* off the maximum achievable mark for each hour late (i.e., approx 1 mark per day) for the first 5 days, then 100% penalty after that

How to do this assignment:

- read this specification carefully and completely
- familiarise yourself with the database schema
- create a database `ass2` on the `d2` host
- explore the database
- make a private directory for this assignment (**not** under `/localstorage`)
- put a copy of the NOT YET AVAIALBLE template files there
- edit the files in this directory on a host other than `d2`
- on `d2`, test that your Python scripts produce the expected output
- submit the assignment via WebCMS3 or `give` (as described on the What to Submit page)

And, of course, if you have PostgreSQL installed on your home machine, you can do all of the `d2` work there. BUT don't forget to test it on `d2` before submitting.

The "template files" aim to save you some time in writing Python code. E.g. they do handle the command-line arguments and let you focus on the database interaction. The `helpers.*` files are provided in case you want to defined Python functions or PLpgSQL functions that might be useful in several of your scripts. You are not required to use them (i.e. you can leave them unchanged), but they must still be submitted.

The template files are available in a single ZIP file ass2.zip, which contains the following

- `helpers.sql` ... any views or PLpgSQL functions to assist your Python
- `helpers.py` ... any Python function to share between scripts
- `dirby` ... Python script to give a list of films for a given director
- `releases` ... Python script to show countries where a film was released
- `genres` ... Python script to show list of genres for movies released in a given year
- `roles` ... Python script to display roles played by an actor/actress
- `movie` ... Python script to display information about a movie

There is even a function already in `helpers.py`. Freebie!

Setting Up

To install the Movie database under your PostgreSQL server on `d2`, simply run the following two commands (after ensuring that your server is running):

```
$ createdb ass2
$ psql ass2 -f /home/cs3311/web/22T3/assignments/ass2/files/ass2.dump
```

If everything proceeds correctly, the load output should look something like:

```
SET
SET
...
SET
CREATE DOMAIN
... a few of these
CREATE TABLE
... a few of these
COPY n
... a few of these
ALTER TABLE
... a whole bunch of these
ALTER TABLE
```

You should get no `ERROR` messages. The database loading should take less than 5 seconds on `d2`, assuming that `d2` is not under heavy load. (If you leave your assignment until the last minute, loading the database on `d2` may be considerably slower, thus delaying your work even more. The solution: at least load the database *Right Now*, even if you don't start using it for a while.) (Note that the `ass2.dump` file is 19MB in size; copying it under your home directory on the CSE machines is not a good idea.)

Note that the database is called `ass2` (when you want to access it via `psql` or Python script).

A useful thing to do initially is to get a feeling for what data is actually there. This may help you understand the schema better, and will make the descriptions of the exercises easier to understand. Look at the schema. Ask some queries. Do it now.

Examples ...

```
$ psql ass2
... PostgreSQL welcome stuff ...
ass2=# \d
... look at the schema ...
ass2=# select * from Movies;
... look at the Movies table ...
ass2=# select * from People;
... look at the People table ...
ass2=# select * from dbpop();
... how many records in all tables ...
ass2=# ... etc. etc. etc.
ass2=# \q
```

Summary on Getting Started

To set up your database for this assignment, run the following commands:

```
$ ssh zID@d2.cse.unsw.edu.au   or   ssh nw-syd-vxdb2
... and then on d2 ...
$ source /localstorage/$USER/env
$ p1
... you shut down the server after your last session, didn't you?
$ createdb  ass2
$ psql  ass2  -f  /home/cs3311/web/22T3/assignments/ass2/files/ass2.dump
$ psql  ass2
... run some checks to make sure the database is ok
$ mkdir  Assignment2Directory
... make a working directory for Assignment 2
$ cd Assignment2Directory
$ unzip /home/cs3311/web/22T3/assignments/ass2/files/ass2.zip
... puts the template files in your working directory
```

The only messages produced by these commands should be those noted above. If you omit any of the steps, then things will not work as planned. If you subsequently ask questions on the Forums, where it's clear that you have *not* done and checked the above steps, the questions will not be answered.

Exercises

Q1: The `dirby` (Directed By) script(3 marks)

Complete the Python3 `dirby` script that takes one command-line argument, the name of a person, and prints:

- a list of films that this person directed, or
- a message that the person didn't direct any films, or
- a message that the person does not exist in the database

Note that there are some names that occur multiple times. Choose only people who have worked as directors, and then choose the first one from that list ordered by `People.id`. If none of the matching people are directors, exit with the message "None of the people called *Name* has directed any films".

Examples of using the script can be found on the Examples page.

Q2: The `releases` script(4 marks)

Complete the Python3 `releases` script, that takes two command-line arguments, the complete title of a movie and the year it was made, and lists all of the countries where it was released. The movie title must be an exact match to the title in the database (the words, the punctutation and the capitalzation). If the title contains multiple words, you will need to put them in quotes on the command line (e.g. "`Bad Influence`"). The year must be a four-digit string (e.g 2012).

You can assume that the combination (title,year) is unique, even though there are some counter-examples in the database. We will not test any cases where (title,year) is not unique.

First check that the year is valid (i.e. four digits). If not, exit with the message "Invalid year".

If the movie does not exist in the database, print the message "No such movie", and exit the program. If the movie does exist, but has no releases mentioned in the database, print the message "No releases" and exit the program. Otherwise, print a list of country names, one per line, in alphabetical order, for all of the countries where the movie was released.

Examples of using the script can be found on the Examples page.

Q3: The `genres` script(3 marks)

Complete the Python3 `genres` script, that takes one command-line argument, a year, and prints a list of the top-10 genres of movies released during that year. Order the list in descending order based on the number of movies in each genre. For genres that have equal numbers of movies, order by the genre names.

First check that the year is valid (i.e. four digits). If not, exit with the message "Invalid year". If there are no movies in the given valid year, exit with the message "No movies".

To extract the top 10, use the following PostgreSQL construct:

```
FETCH FIRST 10 ROWS WITH TIES
```

For details on this, see the entry for SELECT in the PostgreSQL documentation.

Examples of using the script can be found on the Examples page.

Q4: The `roles` script(4 marks)

Complete the Python3 `roles` script, that takes a single command-line argument, the name of an actor/actress, and prints a list of the roles they played. (Note: treat a principal whose `job` is `self` the same as an actor/actress.) Each entry in the list gives the name of the character and the title and year of the movie it's in. The list should be ordered by year, then by movie title, then by role name.

If the name does not exist in the database, then print the message "No such person". If the person had no acting roles, then print the message "No acting roles".

There are instances where several people have the same name. In these cases, it should print a list for each person, introduced by their name and a number. If only one person has the name, we simply print the role list without name or number. Order the list of people by their `People.id`.

Examples of using the script can be found on the Examples page.

Q5: The `movie` script(6 marks)

Complete the Python3 `movie` script, that takes a single command-line argument, the (partial) name of a movie, and prints a list of all the principals in the movie. Order this list based on the ordering given in `Principals.ord`.

Actors and actresses should have the role they played in the movie next to their name; if there is no role information for that person, display "???" as the role. Treat principals with the job `self` the same as actors/actresses. Other principals should display their name and the job they performed (e.g. director, cinematographer).

If more than one movie matches the partial title, display a numbered list, ordered by title, then year, of matching movies, followed by a prompt to select one movie (by its number). Then display information about that particular movie as described above.

Examples of using the script can be found on the Examples page.