CMU - SCS
15-415/15-615 Database Applications
Spring 2013, C. Faloutsos
Homework 2: SQL queries
Released: Tuesday, 02/05/2013
Deadline: Tuesday, 02/12/2013

Reminders - IMPORTANT:

- Like all homework, it has to be done **individually**.
- Please submit a **printout of both your SQL commands and the output of the queries** in a **hard copy, in class**, on Tuesday, 02/12/2013, 1:30pm.
- Further, submit your **code electronically**, as described in details below, by 1:30pm on Tuesday, 02/12/2013.
- For ease of grading, please print each of the eight questions on a **separate** page, i.e., eight pages in total for this homework. If you need more pages for one problem, please staple them together. Type your **name and andrew ID on each** of the eight pages. FYI, we will have eight labeled piles at the front of the classroom, one for each problem.

Reminders - FYI:

- Weight: 15% of homework grade.
- The points of this homework add up to 100.
- Rough time estimates: 3 - 6 hours.

Electronic submission

1. Submit one SQL file for each of the eight questions - name them `query1.sql`, `query2.sql` and so on. In your home directory on the server (details below), there will be a directory named `your-andrew-id415_hw2` (e.g. for andrew-id etaralova, the directory is `etaralova415_hw2`).

2. Copy **all of the requested .sql** files into that directory with the command:

   ```
   cp query[1-8].sql your-andrew-id415_hw2/
   ```

   For example, for the andrew-id etaralova, the command will be:

   ```
   cp query[1-8].sql etaralova415_hw2/
   ```

Notes

1. Feel free to use views (give the SQL definitions before using them). Please **do not** create additional tables.

2. Some questions can be answered with more than one correct SQL query - we will accept them all.

3. **Please make sure your queries take no more than several seconds to execute!** In our solutions, they all take less than 5 seconds.

4. We will automatically run your scripts and diff the output against correct answers. For your convenience, we are providing a Makefile in your `andrew-d415_hw2` directory: e.g. for the andrew-id etaralova, run the following command to make the output of problem1:

   ```
   cd etaralova415_hw2
   ```

   ```
   make problem1
   ```

   To compare your answer with (very fake) solutions (provided only so you have an idea how we will grade your problems), run the following command (from the dir `etaralova415_hw2`):

   ```
   make grade1
   ```

   If the output is: *Files sol/problem1_clean.txt and sol/sol1_clean.txt are identical*, then you got it right. Since we are providing fake answer files, you will get an error with a lot of output, ending in *make: *** [grade1] Error 1*.

Database setup

We will work with PostgreSQL - a popular open-source database. We have put up a readme file on the course homepage with detailed instructions here:
`http://www.cs.cmu.edu/~christos/courses/dbms.S13/hws/HW2/PostgreSQLReadme.html`
Here is the quick summary to get you started:

1. Log into newcastle.db.cs.cmu.edu using `your-andrew-id415` as your user-id, e.g., for andrew-id etaralova, the user-id is etaralova415. The initial password of your account is same as the user-id, e.g., for andrew-id etaralova, password is etaralova415 (ignore the prompt for *CS.CMU.EDU's Password:*, hit enter, and also ignore the following error *kinit: krb5_get_init_creds: Client NNN unknown*). You will be prompted to change the password immediately after the first login, by first entering the initial password, and then giving your new password twice. Please pick a strong new password which meets the password safety requirement of the server machine (like, YW9c10bK etc.).

2. On first login, run `../etaralova415/setup_db.sh`, press"y" to continue when prompted.

3. Run `pg_ctl start -o -i` and when script is done, press "Enter" again.

4. Run `psql`.

5. Run `SELECT COUNT(*) FROM movies;`. The count should equal $2,680$.

6. Run `SELECT COUNT(*) FROM play_in;`. The count should equal $74,772$.

7. Run `q` to quit PostgreSQL.

8. **IMPORTANT:** Please **stop the server** using `pg_ctl stop` before logging out.

SQL queries on the MovieLens dataset (100 points) [Kate]

In this homework we will use the MovieLens dataset released in May 2011. For more details, please refer to:

`http://www.cs.cmu.edu/~christos/courses/dbms.S13/hws/HW2/movielens-readme.txt`

We preprocessed the original dataset and loaded the following two tables to PostgreSQL:

```
movies ( mid, title, year, num_ratings, rating)
play_in (mid, name, cast_position)
```

In the table `movies`, `mid` is the unique identifier for each movie, `title` is the movie's title, and `year` is the movie's year-of-release. Each movie receives a total number of `num_ratings` ratings from users, and the average `rating` is a rating on a scale of $0.0 - 10.0$.

The table `play_in` contains the main cast of movies. name is the actor's name (assume each actor has an unique name). `cast_position` is the order of the actor where he/she appears on the movie cast list (for example, in the movie *Titanic*, the `cast_positions` of *Leonardo DiCaprio* and *Kate Winslet* are 1 and 2, respectively).

Queries

Write SQL queries for the following:

Q 1. Print all movie titles starring *Daniel Craig*, sorted in ascending alpha order. SUBMIT ON SEPARATE PAGE [5 points]

Q 2. Print names of the cast of the movie *"The Dark Knight"* in ascending alpha order. SUBMIT ON SEPARATE PAGE [5 points]

Q 3. One would expect that the movie with the highest number of user ratings is either the highest rated movie or perhaps the lowest rated movie. Let's find out if this is the case here. SUBMIT ON SEPARATE PAGE [10 points]

- **3.1.** Print all information (mid, title, year, num_ratings, rating) for the movie with the most number of ratings (return only the top one result: there is only one in first place). [2 points]
- **3.2.** Print all information (mid, title, year, num_ratings, rating) for the movie(s) with the **highest** rating (include all that tie for first place). Order by ascending `mid`. [2 points]
- **3.3.** Is the movie with the most number of user ratings among these **highest** rated movies? Print the output of the query that will check our conjecture, i.e. it will print the movies that have both: a) the **highest** number of ratings; and b) the **highest** average rating. [2 points]

3.4. Print all information (mid, title, year, num_ratings, rating) for the movie(s) with the **lowest** rating, ordered by ascending `mid`. [2 points]

3.5. Is the movie with the most number of user ratings among these **lowest** rated movies? Print the output of the query that will check our conjecture, i.e. it will print the movies that have both: a) the **highest** number of ratings; and b) the **lowest** average rating. [2 points]

Q 4. Let's find out the movies with extreme ratings for the last 8 years. We split this question into the following steps. SUBMIT ON SEPARATE PAGE [13 points]

4.1. Print the movie year, title and rating of the **lowest** rated movie for each years in the period 2005-present, inclusive, in ascending year order. In case of a tie, print them all in ascending alpha order on title. [5 points]

Hints:

1. *You might want to create a view.*
2. *For example, the list of the worst rated movies for the years 2003-2005 is:*

```
 year |       title        | rating
------+--------------------+--------
 2003 | House of the Dead  |    3.8
 2004 | Catwoman           |    4.4
 2005 | Alone in the Dark  |    4.2
```

4.2. Print the movie year, title and rating of the **highest** rated movie for each years in the period 2005-present, inclusive, in ascending year order. In case of a tie, print all, sorted in ascending alpha order on the title. [5 points]

Hints:

1. *You might want to create a view.*
2. *For example, the list of the highest rated movies for the years 2003-2005 is:*

```
year |           title             | rating
-----+-----------------------------+--------
2003 | Oldeuboi                    |    8.6
2004 | Bin-jip                     |    8.6
2005 | Chinjeolhan geumjassi       |      8
2005 | Diary of a Mad Black Woman  |      8
```

4.3. Print the combined output (movie year, title and rating), for both the lowest and highest rated movies per year from 2005 to present, in ascending year order. As before, break ties by printing them in ascending alpha order on the title. [3 points]

Q 5. Let's find out who are the "no flop" actors: we will define a "no flop" actor as one who has played **only** in movies which have a rating greater than or equal to 8. We split this problem into the following steps. SUBMIT ON SEPARATE PAGE [12 points]

5.1. Create a view called `high_ratings` which contains the distinct names of all actors who have played in movies with a rating greater than or equal to 8. Similarly, create a view called `low_ratings` which contains the distinct names of all actors who have played in movies with a rating less than 8. Print a) the number of rows in the view `high_ratings` and b) the number of rows in the view `low_ratings`. [3 points]

5.2. Use the above views to print the number of "no flop" actors in the database. [2 points]

5.3. Finally, use the above view to print the names of these "no flop" actors, along with the number M of movies they have played in, sorted by descending M and then by ascending name, and print only the top 10. [7 points]

Q 6. Let's find out who is the actor with the highest "longevity." Print the name of the actor/actress who has been playing in movies for the longest period of time (i.e. the time interval between their first movie and their last movie is the greatest). SUBMIT ON SEPARATE PAGE [15 points]

Q 7. Let's find the close buddies of *Annette Nicole*: print the names of all actors who have starred in **all** movies in which *Annette Nicole* has starred in (it's ok to report the name of *Annette Nicole* in the result; also, it is ok if these actors have starred in more movies than *Annette Nicole* has played in). PostgreSQL does not have a relational division operator, so we will guide you through the following steps (you might find it useful to consult the slides or the textbook for the alternative "double negation" method of performing relational division). SUBMIT ON SEPARATE PAGE [15 points]

7.1. First, create a view `co_actors` which contains the distinct names of actors who have played in at least one movie with *Annette Nicole*. Print the number of rows in this view. [3 points]

7.2. Second, create a view `all_combinations` which contains all possible combinations of `co_actors` and the movie ids in which *Annette Nicole* has played in. This view should have two columns: co-actor names and movie ids (*note: this view contains `co_actor,mid` combinations which are fake, because they never happened!*). Print the number of rows in this view. [4 points]

7.3. Third, create a view `non_existent` from the view `all_combinations` by removing all (`co_actor,mid`) pairs that did in fact happen (i.e. these pairs exist in the table `play_in`). Print the number of rows in this view. [4 points]

7.4. Finally, from all of *Annette Nicole*'s co-actors (view `co_actors`) remove the distinct names of the actors from the `non_existent` view. Print the names of the actors who

have actually played in (at least) all movies that *Annette Nicole* has played in (*note: it is ok to report Annette Nicole in the results*).[4 points]

Q 8. Let's find out who is the most "networked" actor - the actor who has the highest number of **distinct** co-actors.

To help you with partial credit, we split this question into the following steps. SUBMIT ON SEPARATE PAGE [25 points]

8.1. First, for a warm-up, print all of *Jim Parsons*' distinct co-actors, ordered by ascending co-actor name. We will call *Jim Parsons* the "source actor." [1] [10 points]

Hint: For example, the distinct co-actors of Rupert Murray are:

```
     src_name   |      co_actor_name
----------------+-----------------
 Rupert Murray | Daniel Schacter
 Rupert Murray | Doug Bruce
 Rupert Murray | Douglas Bruce
```

8.2. Now, print only the name of the source actor (*Jim Parsons*) and the count of all of his distinct co-actors. [5 points]

Hint: in our previous example for Rupert Murray, this would be:

```
     src_name   | coactors_n
----------------+------------
 Rupert Murray |          3
```

8.3. Finally, use the above query to get the distinct count of everyone's co-actors and print the (source) actor with the highest such count. The output should have two columns: the (source) actor name and the count of his distinct co-actors. In case of a tie, print all of them, sorted in alpha order on the source name. [10 points]

[1] In IMDB, he is an emerging star of 2012 and a two-time Emmy-winning star of "The Big Bang Theory."