CIS 450/550 Database and Information Systems Fall 2016

Homework 1 (Upload to Canvas by 23:59:59 EDT on September 21)

Please submit this homework electronically using Canvas (under hw1).

Read the SQL lecture slides as well as what was assigned on the syllabus page (textbook chapters and "SQL for Web Nerds").

The goal of this homework is to get experience with two relational databases, Oracle (running on Amazon's AWS/RDS) and mySQL (running on fling.seas.upenn.edu, or you can set up your own local instance).

The database you will use for the first part of the homework consists of a portion the Internet Movie Database (IMDB.com). It has been uploaded to Amazon's AWS/RDS and conforms to the following schema (where keys are underlined):

actors(*id*: int, *first_name*: string, *last_name*: string, *gender*: string)
directors(*id*: int, *first_name*: string, *last_name*: string)
directors_genres(*director_id*: int, *genre*: string, *prob*: float)
movies(*id*: int, *name*: string, *year*: int, *rank*: float,)
movies_directors(*director_id*: int, *movie_id*: int)
movies_genres(*movie_id*: int, *genre*: string)
roles(*actor_id*: int, *movie_id*: int, *role*: string)

1. (5 pts) Using `sqlplus` running on Eniac (or other SQL clients such as SQLDeveloper) connect to the Oracle instance IMDb that we have set up (read the `RDSHandout.doc` in the `hw1` folder on Canvas).

   Then find the names of the first five movies that occur in the table **movies** by running the query

   ```
   SELECT M.name
   FROM  movies M
   WHERE ROWNUM < 5;
   ```

   Copy these names in a file `hw1-1.txt` and submit this file using Canvas.

The next eight exercises ask you to write queries in SQL. Your answers will consist of the query itself (and *not* of the answers returned by the queries). Obviously you should run the queries to see that they work correctly. Since the dataset is rather large, you should figure out how to limit the number of tuples each query returns when you test it; a few answer tuples are just as illuminating as hundreds of them. Please put each query in a separate script file; name them `hw1-2.sql`, `hw1-3.sql`, ..., `hw1-9.sql`, and use Canvas to submit them. **Your script must compile. Your scores will be based on correctness, implementation and style. You are not allowed to hard code the answers to any of the queries.**

2. (5 pts) Find all genres that start with "A". Each genre should occur exactly once. **Hint**: the LIKE predicate allows for partial string comparisons, with the percent sign ("\%") used as a wildcard. Google if you need more details.

3. (5 pts) Print the name of all movies that appeared in 2001 and were ranked higher than 9.7.

4. (7 pts) For each rank, print the number of movies of that rank. The result should have schema (rank, num) appear ordered by rank from lowest to highest.

5. (8 pts) Print actors who have held at least 2 different roles in a single movie. Your query should work for any number of different roles by modifying a constant in the query.

6. (8 pts)Print the movie name, director last name, and director first name of all movies with genre "Comedy" that appeared in 2001 and were directed by a director whose last name starts with "L".

7. (8 pts) Print the last and first name of all directors who have never directed a top ranked movie. Print the result in alphabetical order (last name, first name). **Hint:** Your query should work regardless of what the numerical value of a top ranked movie is, i.e. it should not use a constant value for this.

8. (8 pts) For each movie, print the count of different genres it has. For movies that are not in movie_genres, the count should be 0.

9. (8 pts) Print the first name, last name and id of actors who have acted in the most movies. *Hint:* Take into account the fact that actors may have more than one role in a movie.

10. (8 pts) Place your answer to this question in a file `hw1-10.txt`.

    Consider the following MySQL session which queries the same IMDB instance as you have been using for the questions above. It consists of two queries which ask for the same information.

```
MySQL [cis550hw1]> SELECT last_name, first_name
    -> FROM directors
    -> WHERE id NOT IN
    ->  (SELECT D.id
    ->   FROM movies_directors MD, movies M, directors D
    ->   WHERE MD.movie_id=M.id AND MD.director_id=D.id
    ->   AND M.rank< (SELECT MAX(rank) FROM movies))
    -> ORDER BY last_name, first_name;
+-------------+------------+
| last_name   | first_name |
+-------------+------------+
| Cappelletti | Andrew     |
| Crandall    | Matthew    |
| Resnicoff   | Zack       |
| Tarrier     | Debs       |
| Tretbar     | Kirsten    |
+-------------+------------+
5 rows in set (0.04 sec)

MySQL [cis550hw1]> SELECT last_name, first_name
    -> FROM directors D
    -> WHERE NOT EXISTS
    ->      (SELECT *
    ->       FROM movies_directors MD, movies M
    ->       WHERE MD.movie_id=M.id AND MD.director_id=D.id
    ->                   AND M.rank< (SELECT MAX(rank) FROM movies))
    -> ORDER BY last_name, first_name;
+-------------+------------+
| last_name   | first_name |
+-------------+------------+
| Cappelletti | Andrew     |
| Crandall    | Matthew    |
| Resnicoff   | Zack       |
| Tarrier     | Debs       |
| Tretbar     | Kirsten    |
+-------------+------------+
5 rows in set (3.07 sec)
```

    (a) State in English what the queries ask for.

    (b) The first query runs in 0.04 seconds, whereas the second takes 3.07 seconds. What could account for this dramatic difference in speed?

In the next five questions, you will use MySQL. You have been given accounts on the instance hosted at fling.seas.upenn.edu – if not, please contact CETS to get one set up for you. The MySQL Answers page provides a link for resetting passwords as well as connecting to the databases, and can be found at
http://www.seas.upenn.edu/cets/answers/mysql.html
Alternatively, you can set up your own MySQL instance using a free download.

You will be creating a database with the following schema:

Person(*login*, name, sex, relationshipStatus, birthyear)
Family(*login*, *member*, role)
Friends(*login*, *friend*)

Keys are underlined; attribute login in Family is a foreign key to Person; and attributes login and friend in Friends are foreign keys to Person.

11. (10 pts) Write SQL DDL statements to define the database above and enforce the constraints below. Use varchar(255) as the type of login and name, varchar(7) for sex, and varchar(12) for relationshipsStatus and role. Put your answer in a file `hw1-11.sql`, and use this to create a mySQL database (either on fling or on your local instance).

    - Attribute sex should only take on values 'male' or 'female'.
    - Attribute relationshipStatus should only take on values 'single', 'married', 'divorced', or 'relationship'.
    - Attribute role should only take on values 'mother', 'father', 'son', 'daughter', 'aunt', 'uncle', 'cousin', 'brother', or 'sister'.

12. (5 pts) Files Person.sql, Family.sql and Friends.sql contain insert statements to populate the database. In which order(s) should these be executed and why? Put your answer in a file `hw1-12.txt`, and execute these files to populate your instance.

13. (5 pts) Suppose you wish to enforce that if X is a friend of Y then Y must also be a friend of X. Write a query returning all the logins X for which there is a tuple Friends(X,Y) but no tuple Friends(Y,X). Put your answer in a file `hw1-13.sql`.

14. (5 pts) Write a query to return all friends of friends, i.e. all (X, Z) such that Friends(X,Y) and Friends(Y, Z). Put your answer in a file `hw1-14.sql`.

15. (10 pts) **EXTRA CREDIT** This question is extra credit, so you do *not* have to submit an answer.

    Can you write a query to find the transitive closure of the friends relationship? That is, find all $Friends(X, Y_n)$ such that $Friends(X, Y_1)$, $Friends(Y_1, Y_2)$,...., $Friends(Y_{n-1}, Y_n)$. If no, explain your answer. If yes, give a query. Your query should work on any instance.

    Put your answer in a file `hw1-15.txt`.