Database Systems, CSCI 4380-01
Homework # 4
Due Friday October 16, 2019 at 11:59:59 PM

Homework Statement. This homework is worth 5% of your total grade. If you choose to skip it, Midterm #2 will be worth 5% more. This homework will concentrate on SQL skills. You will be able to test your queries against real data and expected output.

If you want to create the database on your local database server, you can find the database dump file at:

```
http://www.cs.rpi.edu/~sibel/DBS_Past_Materials/Fall2020/ministreaming.dmp
```

You can create a database for this data, let's call it `hw4` on `psql` shell:

```
psql> create database hw4;
```

and then load the data (after unzipping) using the following Unix command (on Linux and Macs):

```
cat hw4.dmp  | psql hw4
```

The data model for this homework is given in the end of this homework. We have data regarding Movies and TV Shows and which streaming platforms they are available in. This data is sampled down from a much larger database for simplicity (less than 2% of the full data) and has potentially some noise. We will use the database "as is" and not fix issues unless they are major.

Database Server Use Rules

If you want to install and create the database on your own computer, you can use the data scripts I used. You do not have to have a database server installed to do this homework. This database will be created as `ministreaming` on the shared database server at:

```
http://rpidbclass.info
```

Feel free to use it for testing your queries, but please be considerate of others when using the server. Here are a few ground rules:

- Server response to load can be unpredictable. So, be patient if you see some slow down. This is a medium sized database for a 100+ student class, so you can expect a serious slow down near the homework deadline. Please do not wait till the last minute to submit your homeworks.
- Test your queries one at a time. Best set up is using a browser and a text editor. Write queries elsewhere and test in the server with cut and paste.
- Make every effort to read your queries before submitting. A forgotten join condition may mean disaster. Check join conditions first, then run your queries.

- Remember if you have an unresponsive query, it will continue to use system resources even if you quit your browser. So, opening a new browser window will solve your problem but may slow things down for everyone. Queries should terminate after 2 minutes, so if you increased the load with a bad query, then wait for your query to end before submitting another.

  If you are experiencing problems with a query, read it carefully before running it again in a separate window. Remember: missing join conditions is the difference between: 2,094,266,610,000 tuples and 3335 tuples.

- If the server is not responsive, let us know on Submitty/Teams. I will see if some jobs need to be killed or whether server needs to be made more powerful. Please be patient.

- Please do not include a query that does not run in your homework submission. I will run all your queries in a batch job and an incomplete query will cause me a great deal of problems.

1 Problem Description

Write the following queries in SQL. In all your queries, use the simplest possible expression possible. Do not forget your JOIN conditions and pay attention to returned attributes and ordering of tuples.

Query 1 Return the title and directors of series with at least 15 seasons despite having an IMDB rating of less than 5. Order results by title and director ascending.

Query 2 Find the number of movies (nummovies) that have no imdb or rotten tomatoes rating, and either has no year value or year is after 2015.

Query 3 Return the title, director of all movies IMDB rating of 8 or higher and is available in English and in at least one other language, and are in at least one of the following genres: Action, Sci-Fi, Adventure or Drama. Order results by title and director ascending.

Query 4 Return the id, title of all movies and TVshows that are only available in Italy. Return an additional column called `streamingtype` that has the value `'movie'` or `'series'` depending on the source. Order results by the title and streamingtype in descending order.

Query 5 Return the total number of tuples in the movies relation (numtuples), total number of tuples with a date added value (numdate), earliest year and last year for a movie (minyear, maxyear), average duration for a movie and average imdb rating (avgrating). For averages, return only two digits after comma!

Note: This query requires you to do type casting. Often there are minor differences in floating point operations from different postgresql servers. So, for simplicity I have decided to ask everyone to use the same casting: cast duration as integer and averages as numeric(5,2). Format for doing this is as follows:

```
    select '120'::int, '4.1234123412'::numeric(5,2) ;
```

Query 6 Find cast members who have both been in a movie and a TV Series in the same genre for either `Mystery` or `Thriller` genres. Return the name of the cast members and the total number of movies (nummovies) and series (numseries) they have been in one of these genres. Order by the nummovies, numseries and castname ascending.

Query 7 Find series that are available in at least 3 countries. Return the title of these series, the total number countries (numcountries) and streaming platforms (numplatforms) they are available in. Order results by numcountries descending and title ascending.

Query 8 For each language, find how many movies are in that language. Return the language, min and max imdb and rotten tomatoes ratings of movies for each language that has at least 10 movies.

Rename the aggregates: minimdb, maximdb, minrotten, maxrotten respectively. Order results by minimdb and minrotten asc.

Query 9 For series that are on fuboTV, find the other platforms that they are also on. For each other platform, return the platform name and total number of series from fuboTV that they feature (numseries). Order results by numseries descending and platform name ascending.

Query 10 Return the name of all cast members who were cast in a movie added after `1/1/2019` and have also directed a movie but were not cast in any series. Order the results by castname ascending.

SUBMISSION INSTRUCTIONS. You will use SUBMITTY for this homework.

Please submit each query to the appropriate box in Submitty.

You must add a semi-colon to the end of each query to make sure it runs properly.

If you want to provide any comments, all SQL comments must be preceded with a dash:

```
-- Example comment.
```

Database Schema

This database is merged from multiple datasets, containing data for movies and TV Series, both appearing in various streaming platforms.

Note that this is real data, scraped from websites and then parsed by me, so it may be noisy. Make a habit of using simple queries to first explore the data to understand various issues.

```
create table movies(
      movieid     int  primary key
      , title     varchar(1000)
      , year      int        -- year the movie was made
      , contentrating varchar(10) -- age group the movie is intended for
      , imdbratin    float      -- imdb rating
      , rottentomatoes float -- rotten tomatoes rating
      , runtime    int
      , date_added date       -- date movie is added to Netflix
      , duration   varchar(40)
      , description text
) ;

-- Which countries the movie is available in
create table moviescountry (
     movieid int
     , country varchar(100)
     , primary key (movieid, country)
     , foreign key (movieid) references movies1(movieid)
) ;

-- Directors of the movie
create table moviesdirectors (
     movieid int
     , director varchar(100)
     , primary key (movieid, director)
     , foreign key (movieid) references movies1(movieid)
) ;

create table moviesgenres (
     movieid int
     , genre varchar(100)
     , primary key (movieid, genre)
     , foreign key (movieid) references movies1(movieid)
) ;

-- Languages the movie is available in
create table movieslanguages (
     movieid int
     , language varchar(100)
     , primary key (movieid, language)
     , foreign key (movieid) references movies1(movieid)
) ;

-- People who starred in the movie
create table moviescast (
     movieid int
     , castname varchar(100)
     , primary key (movieid, genre)
     , foreign key (movieid) references movies1(movieid)
) ;
```

```
-- Which platform the movie is in (if ispresent is True),
-- or not in (if ispresent is False)
create table moviesonplatform (
     movieid int
     , platform varchar(100)
     , ispresent boolean
     , primary key (movieid, platform)
     , foreign key (movieid) references movies1(movieid)
) ;

-- TV series on various platforms
create table series (
     seriesid int  primary key
     , title varchar(400)
     ,  yearreleased  int
     ,  contentrating varchar(40) -- age group the movie is intended for
     ,  imdbrating    float       -- imdb rating
     ,  rottentomatoes int        -- rotten tomatoes rating
     ,  description   text
     ,  seasons       int         -- how many seasons are available
     ,  date_added    date        -- date series is added to Netflix
) ;

-- Which platform the series is available in
create table seriesonplatform (
     seriesid int
     , platform varchar(100)
     , primary key (seriesid, platform)
     , foreign key (seriesid) references series(id)
) ;

-- People who starred in the series
create table seriescast (
     seriesid int
     , castname varchar(100)
     , primary key (seriesid, castname)
     , foreign key (seriesid) references series(id)
) ;

-- Categories for series
create table seriescategory (
     seriesid int
     , category varchar(100)
     , primary key (seriesid, category)
     , foreign key (seriesid) references series(id)
) ;

-- Countries the series is available in
create table seriescountry (
     seriesid int
     , country varchar(100)
     , primary key (seriesid, country)
     , foreign key (seriesid) references series(id)
) ;

-- Directors for the series
create table seriesdirectors (
     seriesid int
     , country varchar(100)
     , primary key (seriesid, country)
     , foreign key (seriesid) references series(id)
) ;

-- Genre from series
create table seriesgenres (
     seriesid int
     , genre varchar(100)
     , primary key (seriesid, genre)
     , foreign key (seriesid) references series(id)
) ;
```