Database Systems, CSCI 4380-01
Homework # 4
Due Friday October 15, 2021 at 11:59:59 PM

Homework Statement. This homework is worth 3% of your total grade. It has 8 queries with 12.5 points for each query. You are required to complete at least 4 queries (equivalent of 1.5 points). Any points that you did not complete will be added to Midterm #2. (For example, if you only solved 4 queries worth 1.5 points, your Midterm #2 will be worth 1.5% more).

This homework will concentrate on SQL skills. You will be able to test your queries against real data and expected output.

The testing data for this database is already created in the class server at: `http://rpidbclass.info`. When logging in, choose: `radiodb_hw4`.

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/Fall2021/radiodb_hw4.dmp`

You can create a database for this data, let's call it `radiodb_hw4` on `psql` shell:

```
psql> create database radiodb_hw4;
```

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

```
cat radiodb_hw4.dmp  | psql radiodb_hw4
```

The data model for this homework is given in the end of this homework. We have data regarding songs, artists, their popularity in various platforms. This data is sampled down from a much larger database for simplicity (less than 5% 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 `radiodb_hw4` 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. Even though this is a very small slice of the actual data, it is not so small. There are tables with thousands of tuples.
- 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 name of all artists in our database who have a song with at least 10 million streams on Spotify. Order results by artist name ascending.

Query 2 Return the name of all artists in our database who have a song on the bilboard with rank 1 and have an album in the top 50 of the Rolling Stones top 500 list. Order the results by artist name ascending.

Query 3 Return the id and name of all songs, and the name of the artist for these songs who are on Bilboard on August 2002 and have been on Bilboard (weeksonboard) for at least 25 weeks by August 2020, or have been streamed at least 5 million times in August 2020. Order the results by song id and name ascending. Rename your attributes songid, songname and artistname. Order results by songname and artistname ascending.

Query 4 Find the id and name of songs played on the radio between 08AM (time '08:00:00', inclusive) and 10AM (time '10:00:00', inclusive) that are not played at any other time of the day. Order the results of id and name ascending. Rename your attributes songid, songname and artistname. Order results by songname and artistname ascending.

Query 5 Return id, name and artist name of all songs that were played in a radio station and were streamed on spotify at least 4 million streams, but were not in Bilboard charts. Rename your attributes songid, songname and artistname. Order results by songname and artistname ascending.

Query 6 For each artist in the top 25 for the Rolling Stones list who was on bilboard charts, return their id, name, the total number of songs they have had on the bilboard charts (named songsonbilboard) and their highest bilboard rank (minbilboardrank). Order by name of artist ascending.

Query 7 Find the id and name of all artists who were in the top 500 Rolling Stones list and were played by at least two different radio stations for at least 20 times total in March of 2020. Sort results by artist id ascending.

Query 8 Return the name of all song genres that where for genres for at least 4 songs that entered the Bilboard ranks, but never reached a rank of 5 or higher. Order the results by genre name 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 songs (tracks) from different artists. We have data regarding the songs performance on spotify, bilboard and on various radio stations. As with Homework #3, I dedicate this database to WRPI!

Note that this is real data, scraped from websites, parsed and matched. It is likely very noisy. Make a habit of using simple queries to first explore the data to understand various issues.

```
-- All artists in the database
CREATE TABLE artists (
    id       bigint NOT NULL
    , name   text
    , PRIMARY KEY (id)
);

-- Song (or track) information: name, its artist as well as
-- features based on the audio analysis of the song from 1 million song db
-- decade is which decade the song is from

CREATE TABLE songs (
    id              bigint NOT NULL
    , name          text
    , artistid      bigint
    , uri           text
    , danceability double precision
    , energy        double precision
    , key           double precision
    , loudness      double precision
    , mode          double precision
    , speechiness   double precision
    , acousticness double precision
    , instrumentalness double precision
    , liveness      double precision
    , valence       double precision
    , tempo         double precision
    , duration\_ms  double precision
    , time_signature integer
    , chorus_hit    double precision
    , sections      integer
    , popularity    integer
    , decade        text
    , PRIMARY KEY (id)
    , FOREIGN KEY (artistid) REFERENCES artists(id)
);

-- Genre of the songs in the database, from spotify.
CREATE TABLE song\_genre (
    songid     bigint NOT NULL
    , genre    varchar(100) NOT NULL
    , PRIMARY KEY(songid, genre)
    , FOREIGN KEY (songid) REFERENCES songs(id)
);


-- Bilboard rank (between 1-100) of the songs in the database
-- Each row is for a specific song in a specific date of the Bilboard chart
-- Multiple songs may share a rank in a given chart date.
-- Lastweek, peakrank and weeksonboard are the statistics for a specific
```

```
-- song at the given chartdate.

CREATE TABLE bilboard (
    rank         integer
    , songid     bigint NOT NULL
    , lastweek   integer
    , peakrank   integer
    , weeksonboard integer
    , chartdate  date NOT NULL
    , PRIMARY KEY(songid, chartdate)
    , FOREIGN KEY (songid) REFERENCES songs(id)
);

-- For each song, we store when they were played on radio (based on
-- 8 different radio channels: mai,george,sound,rock,breeze,edge,magic,more
-- All radio stations are based in New Zealand (where I could find data from!)
-- Each time the song is played, we store the timestamp.
-- Note that this is a random sample of tuples from the original 855K tuples.

CREATE TABLE playedonradio (
    id           integer NOT NULL
    , songid     bigint
    , station    varchar(40)
    , playedtime timestamp without time zone
    , PRIMARY KEY (id)
    , FOREIGN KEY (songid) REFERENCES songs(id)
);

-- Based on Rolling Stone Magazine's list of the 500 Greatest Albums of
-- All Time, originally published in 2003, slightly updated in 2012.
-- For each album, there is a description of the reason why it was chosen
-- in the critic attribute, as well as the year the album came out
-- and its label.

CREATE TABLE rollingstonetop500 (
    position     integer NOT NULL
    , artistid   bigint
    , album      varchar(255)
    , label      varchar(255)
    , year       integer
    , critic     text
    , PRIMARY KEY (position)
    , FOREIGN KEY (artistid) REFERENCES artists(id)
);

-- Daily top 200 tracks listened to by users of the Spotify platform.
-- Position of the song in the top 200 for a given date (streamdate)
-- streams is the number of listens for this song.

CREATE TABLE spotify (
    position       integer NOT NULL
    , songid       bigint
    , streams      integer
    , streamdate   date NOT NULL
    , PRIMARY KEY (position, streamdate)
    , FOREIGN KEY (songid) REFERENCES songs(id)
);
```