Database Systems, CSCI 4380-01
Homework # 3
Due Thursday March 16, 2023 at 11:59:59 PM

Homework Statement. This homework is worth 3% of your total grade. It has eight queries with 12.5 points for each query.

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 can be found in the database dump file at:

`https://www.cs.rpi.edu/academics/courses/spring23/csci4380/_downloads/radiodb_hw3.dmp`

You can create a database for this data, let's call it `radiodb_hw3` on `psql` shell:

```
psql> create database radiodb_hw3;
```

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

```
cat radiodb_hw3.dmp  | psql radiodb_hw3
```

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.

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 at least one song that at some point was on Bilboard for 30 weeks or longer. Order results by artist name ascending.

Query 2 Return the artist name, the name of the song, and the maximum number of streams for songs with at least 100,000 streams on Spotify and artists that have an album in the top 10 of the Rolling Stones top 500 list. Order the results first by artist name ascending, then by number of streams descending.

Query 3 Return the id and name of all songs, and the name of the artist for these songs who are on Bilboard in the fall of 2020 (assume fall is March, April, and May) and have been on Bilboard (weeksonboard) for at least 5 weeks by the end of May 2020, or have been played on radio at least 20 times in May 2020. Rename your attributes songid, songname and artistname. Order results by songname and artistname ascending.

Query 4 Find the id and name of songs listened on Spotify on weekends (Saturday and Sunday) that were not listened neither on Mondays nor on Thursdays. Return song id, name and artist name. 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 by a radio station and were streamed on spotify at least 1 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 20 for the Rolling Stones, 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 three different radio stations for at least 5 times total in May, June, or July of 2020. Sort results by artist id ascending.

Query 8 Return the name of all song genres that were genres for at least 3 songs that entered the Bilboard ranks, but never reached a rank of 3 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.

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)
);
```