Database Systems, CSCI 4380-01
Homework # 5
Due Friday October 22, 2021 at 11:59:59 PM

Homework Statement. This homework is worth 5% 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 2.5 points). Any points that you did not complete will be added to Midterm #2. (For example, if you only solved 4 queries worth 2.5 points, your Midterm #2 will be worth 2.5% more).

This homework will concentrate on advanced SQL skills, so the queries are more complex than the previous homework. We will use `radiodb_hw4` as before to test them. You can find it at: `http://rpidbclass.info` as database: `radiodb_hw4`.

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 For each artist with a song in the database who was featured in the rollingstonetop500 list in position 20 or lower, return the name of the artist (artistname), number of songs they have in the bilboard relation (numsongs). Order results by numsongs descending and artistname ascending.

Query 2 For all songs with 0.9 or higher value of danceability that have been in bilboard rank of 10 or lower, return the song name, artist name and the number of times it has been played on radio. Rename your attributes songname artistname and numplayed. Order the results by numplayed descending, songname and artistname ascending.

Query 3 Find all songs that were in the bilboard table for at least 25 weeks (weeksonboard) but were never been played on the radio. Return the name of the song (songname) and the name of the artist (artistname) and best rank it has achieved on bilboard (minrank). Order results by minrank, songname, artistname ascending.

Solve the "NEVER" part using LEFT join! Do not use except or a set comparison (not in, not exists, etc.)

Query 4 Solve the same query as Query 3, but using NOT IN/NOT EXISTS only, no left join or except.

Query 5 Find songs with the shortest duration (in minutes, cast as an integer) in the database. Return the name of the songs (songname), its artists (artistname), the decade and the duration in minutes (duration). Order by songname and artistname.

Query 6 Find song(s) with a total stream value in this highest 50% of total stream values in the database. Return the the name of the songs (songname) and its artists (artistname) and the total streams (totalstreams). Order by songname and artistname.

In other words, if maximum total stream value for a song in the database is `X`, then you should return any song with a total stream value in the range `[X-X*0.5]`.

Format the total stream number, using the `to_char` function: For example:

```
select to_char(100000000000, '999,999,999,999');
```

Query 7 Return the name of all artists (artistname) whose songs started to climb to top 10 in bilboard rankings after being featured in the rolling stones list which appeared in 2003. Hence, the first chartdate in the top 10 for a song of these artists was after 2003. Order results by artistname.

Query 8 For each radio station whose name starts with the letter m in the playedonradio relation, return the name of the station and the number of all songs that are only played on that radio station (numsongs). Order results by station.

Note, this is a costly query, takes 3 seconds to run for me. Check your query carefully before running. If you already created this DB on your local machine, please add the following indices:

```
create index p2 on playedonradio(songid,station) ;
create index p1 on playedonradio(station,songid) ;
```

SUBMISSION INSTRUCTIONS. You will use SUBMITTY for this homework.

Please submit each query to the appropriate box in Submitty. Use an ASCII only editor for your queries.

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.
```

Please make sure your queries run in a reasonable amount of time or you may loose some small points. You can see the total run time of my queries as there will be a time in the beginning of my run and one at the end of my run! These are generated at the database server in a low load though, so your queries may be slower depending on the load on the server.

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)
);
```