Homework 4

Data Mgt. for Data Science – Prof. Venugopal

Part 1: Database Schema (50 pts)

Here are the following create_table statements we used:

```
CREATE TABLE Users_Table
(
     user_name VARCHAR(210) NOT NULL,
     PRIMARY KEY(user_name)
);

CREATE TABLE Artist_Information
(
     artist VARCHAR(210) NOT NULL,
     song VARCHAR(210) NOT NULL,
     date_of_release DATETIME NOT NULL,
     PRIMARY KEY(artist, song)
);

CREATE TABLE Album_Information
(
     album VARCHAR(210) NOT NULL,
     artist VARCHAR(210) NOT NULL,
     song VARCHAR(210) NOT NULL,
     date_of_release DATETIME NOT NULL,
     PRIMARY KEY(album, artist, song),
     FOREIGN KEY(artist, song) REFERENCES
Artist_Information(artist, song)
);

CREATE TABLE Genre_Information
(
     genre VARCHAR(210) NOT NULL,
     song VARCHAR(210) NOT NULL,
     artist VARCHAR(210) NOT NULL,
     PRIMARY KEY(genre, song),
     FOREIGN KEY(artist, song) REFERENCES
Artist_Information(artist, song)
);

CREATE TABLE Playlists
(
     user_name VARCHAR(210) NOT NULL,
```

```
     playlist VARCHAR(210) NOT NULL,
     song VARCHAR(210) NOT NULL,
     artist VARCHAR(210) NOT NULL,
     date_of_creation DATETIME NOT NULL,
     PRIMARY KEY(user_name, playlist, song),
     FOREIGN KEY(user_name) REFERENCES Users_Table(user_name),
     FOREIGN KEY(artist, song) REFERENCES
Artist_Information(artist, song)
);

CREATE TABLE Song_Ratings
(
     user_name VARCHAR(210) NOT NULL,
     song VARCHAR(210) NOT NULL,
     artist VARCHAR(210) NOT NULL,
     rating INT(2) NOT NULL,
     date_of_rating DATETIME NOT NULL,
     PRIMARY KEY(user_name, song),
     FOREIGN KEY(user_name) REFERENCES Users_Table(user_name),
     FOREIGN KEY(artist, song) REFERENCES
Artist_Information(artist, song)
);

CREATE TABLE Album_Ratings
(
     user_name VARCHAR(210) NOT NULL,
     album VARCHAR(210) NOT NULL,
     rating INT(2) NOT NULL,
     date_of_rating DATETIME NOT NULL,
     PRIMARY KEY(user_name, album),
     FOREIGN KEY(user_name) REFERENCES Users_Table(user_name),
     FOREIGN KEY(album) REFERENCES Album_Information(album)
);

CREATE TABLE Playlist_Ratings
(
     user_name VARCHAR(210) NOT NULL,
     playlist VARCHAR(210) NOT NULL,
     rating int(2) NOT NULL,
     date_of_rating DATETIME NOT NULL,
     playlist_curator VARCHAR(210) NOT NULL,
     PRIMARY KEY(user_name, playlist),
     FOREIGN KEY(user_name) REFERENCES Users_Table(user_name),
     FOREIGN KEY(playlist_curator) REFERENCES
Users_Table(user_name),
     FOREIGN KEY(playlist_curator, playlist) REFERENCES
     Playlists(user_name, playlist)
);
```

Part 2: Queries (50 pts)

```
#------- Query 1 -------

SELECT g.genre AS "genre", COUNT(g.song) AS "number_of_songs"
FROM Genre_Information g
GROUP BY g.genre
ORDER BY COUNT(g.song) DESC
LIMIT 3;

#------- Query 2 -------

SELECT DISTINCT art_table.artist AS "artist_name"
FROM Artist_Information art_table
INNER JOIN Album_Information alb_table
ON art_table.artist = alb_table.artist;

#------- Query 3 -------

SELECT rat_table.album AS "album_name", AVG(rat_table.rating) AS
"average_user_rating"
FROM Album_Ratings rat_table
INNER JOIN Album_Information alb_table
ON rat_table.album = alb_table.album
WHERE alb_table.date_of_release BETWEEN "1990-01-01 00:00:00"
AND "1999-12-31 23:59:59"
GROUP BY rat_table.album
ORDER BY AVG(rat_table.rating) DESC
LIMIT 10;

#------- Query 4 -------

SELECT DISTINCT g.genre AS "genre_name", COUNT(s.rating) AS
"number_of_song_ratings"
FROM Song_Ratings s
INNER JOIN Genre_Information g
ON s.song = g.song
GROUP BY g.genre
ORDER BY COUNT(s.rating) DESC
LIMIT 3;

#------- Query 5 -------

SELECT f.user_name AS "username", f.playlist AS
"playlist_title", AVG(f.average_song_rating) AS
'average_song_rating'
FROM (
    SELECT p.playlist, p.user_name, p.song, average_song_rating
```

```
    FROM (
          Playlists p
         INNER JOIN (
                SELECT s.song, AVG(s.rating) AS
'average_song_rating'
            FROM Song_Ratings s
            GROUP BY s.song
          ) inner_table
          ON (p.song = inner_table.song)
     )
) f
GROUP BY f.playlist, f.user_name
HAVING NOT average_song_rating < 4
ORDER BY average_song_rating DESC;

#------- Query 6 -------

SELECT big_table.user_name AS "username",
COUNT(big_table.rating) AS "number_of_ratings"
FROM (
     SELECT sng_rat.user_name, sng_rat.rating
     FROM Song_Ratings sng_rat
     UNION ALL
     SELECT alb_rat.user_name, alb_rat.rating
     FROM Album_Ratings alb_rat
) big_table
GROUP BY big_table.user_name
ORDER BY COUNT(big_table.rating) DESC
LIMIT 5;

#------- Query 7 -------

SELECT art.artist AS "artist_name", COUNT(art.song) AS
"number_of_songs"
FROM Artist_Information art
WHERE art.date_of_release BETWEEN "1990-01-01 00:00:00" AND
"2010-12-31 23:59:59"
GROUP BY art.artist
ORDER BY COUNT(art.song) DESC
LIMIT 10;

#------- Query 8 -------

SELECT p.song AS "song_title", COUNT(p.playlist) AS
"number_of_playlists"
FROM Playlists p
GROUP BY p.song
ORDER BY COUNT(p.playlist) DESC
```

```
LIMIT 10;

#------- Query 9 -------

SELECT DISTINCT g2.genre AS "genre_name", g2.song AS
"song_title", number_of_ratings
FROM Genre_Information g2
INNER JOIN
(
     SELECT g1.song as "song_title", g1.genre as "genre_name",
number_of_ratings
     FROM Genre_Information g1
     INNER JOIN
     (
          SELECT s.song, count(s.rating) as "number_of_ratings"
          FROM Song_Ratings s
          GROUP BY s.song ORDER BY count(s.rating)
     ) inner_t
     ON g1.song = inner_t.song
     ORDER BY number_of_ratings
) inner_t2
ON inner_t2.song_title = g2.song
LIMIT 20;

#------- Query 10 -------

SELECT DISTINCT artist as "artist_title"
FROM Artist_Information
WHERE date_of_release < "1994-01-01 00:00:00" AND artist NOT IN
(
     SELECT artist
     FROM Artist_Information
     WHERE date_of_release >= "1994-01-01 00:00:00"
)
ORDER BY artist;
```