Ira A. Fulton Schools of Engineering
Arizona State University

CSE 511: Data Processing at Scale

# Assignment 2: SQL Query for Movie Recommendation

## Purpose

You will be writing SQL queries for the movie recommendation database in this assignment to carry out a number of fundamental tasks.

## Objectives

Learners will be able to:

- Identify how to use the SELECT, FROM, and WHERE statements to create a basic SQL query.
- Apply the WHERE clause to filter records. It is used to extract only those records that fulfill a specified condition.
- Determine how to use aggregate functions in a SQL query, such as count, sum, average, and others, to carry out particular tasks.
- Explain to use a GROUP BY statement, which is often used with aggregate functions.

## Technology Requirements

- PostgreSQL

## Assignment Description

In Assignment 1, you learned how to design a movie recommendation database. This assignment will give you an opportunity to create such a database and build applications on top of this database. Assignment 2 uses the same background information as Assignment 1. In order to successfully complete this assignment, you will need to have correctly created the table definitions for the movie database in Assignment 1.

# Directions

In addition to the directions below, please review the "**Introduction**" and "**Submission and Feedback**" **videos** which are located in the "**Week 2: Overview**" section.

Since the data has been loaded into the database in Assignment 1, you will need to implement the following SQL queries. For each query, we provide an example of the schema of the saved query result.

1. Write a SQL query to return the total number of movies for each genre. Your query result should be saved in a table called "query1" which has two attributes: "name" attribute is a list of genres, and "moviecount" attribute is a list of movie counts for each genre.

| | name text | moviecount bigint |
|---|---|---|
| 1 | Romance | 1685 |

2. Write a SQL query to return the average rating per genre. Your query result should be saved in a table called "query2" which has two attributes: "name" attribute is a list of all genres, and "rating" attribute is a list of average rating per genre.

| | name text | rating numeric |
|---|---|---|
| 1 | Drama | 3.8204275534441805 |

3. Write a SQL query to return the movies which have at least 10 ratings. Your query result should be saved in a table called "query3" which has two attributes: "title" is a list of movie titles, and "CountOfRatings" is a list of ratings.

| | title text | countofratings bigint |
|---|---|---|
| 1 | Sleepy Hollow (1999) | 11 |

4. Write a SQL query to return all "Comedy" movies, including movieid and title. Your query result should be saved in a table called "query4" which has two attributes: "movieid" is a list of movie ids, and "title" is a list of movie titles.

| | movieid integer | title text |
|---|---|---|
| 1 | 33004 | Hitchhiker's Guide to the Galaxy, The (2005) |

5. Write a SQL query to return the average rating per movie. Your query result should be saved in a table called “query5” which has two attributes: “title” is a list of movie titles, and “average” is a list of the average rating per movie.

| | title<br>text | average<br>numeric |
|---|---|---|
| 1 | Where the Heart Is (2000) | 4.5000000000000000 |

6. Write a SQL query to return the average rating for all “Comedy” movies. Your query result should be saved in a table called “query6” which has one attribute: “average”.

| | average<br>numeric |
|---|---|
| 1 | 3.5797206165703276 |

7. Write a SQL query to return the average rating for all movies and each of these movies is both “Comedy” and “Romance”. Your query result should be saved in a table called “query7” which has one attribute: “average”.

| | average<br>numeric |
|---|---|
| 1 | 3.6989528795811518 |

8. Write a SQL query to return the average rating for all movies and each of these movies is “Romance” but not “Comedy”. Your query result should be saved in a table called “query8” which has one attribute: “average”.

| | average<br>numeric |
|---|---|
| 1 | 3.7429411764705882 |

9. Find all movies that are rated by a user such that the userId is equal to v1. The v1 will be an integer parameter passed to the SQL query. Your query result should be saved in a table called “query9” which has two attributes: “movieid” is a list of movieid’s rated by userId v1, and “rating” is a list of ratings given by userId v1 for corresponding movieid.

| | movieid<br>integer | rating<br>numeric |
|---|---|---|
| 1 | 110 | 5 |

**Note**: You do not have to follow the exact same table schema for the similarity table, as long as you can produce the correct recommendation table.

Above all, your script should be able to generate 9 tables, namely, “query1”, “query2”, ..., “query9”, and “recommendation”.

## Assignment Tips:

1. All table names and attribute names **must be in lowercase letters and match the specification**. If you have deleted your tables, you can recreate them using your Assignment 1 submission. If that is called solution1.sql, you can run it using the "psql -U postgres -f solution1.sql" command.

2. Your SQL script will be tested on PostgreSQL 9.5 using the “psql -U postgres –f solution.sql -v v1=1234567” command. Your script needs to take 1 input parameter v1 provided by the auto-grading system via the “psql –v” option. v1 takes an integer as the input and it is the user ID v1 used in your Query 9 and 10.

3. The delimiter of all files is "%".

4. You should use the following command to save your query result to a table:

   ```
   CREATE TABLE query0 AS
   YOUR SQL STATEMENT
   ```

   For instance, select the user from the users table which has userID = v1 and store it in query0 and rename the “username” column to “userfullname”.

   ```
   psql -U postgres -f solution.sql -v v1=123
   ```

   In your SQL script:

   ```
   CREATE TABLE query0 AS
   SELECT username AS userfullname
   FROM users
   WHERE users.userid = :v1
   ```

5. Do **not** put “create/select/drop database”, or “set system settings or encoding” in your SQL script. This may lead to point deductions. Do not create tables of Phase 1 and do not load any data.

6. The rows in your query result table **do not have to be sorted**.

7. You are free to create any other temp/permanent views, temp/permanent tables to help your queries.

8. Remember that you may make edits to your submission and resubmit as many times as you would like. If you have trouble submitting your assignment, we encourage you to ask questions to your peers as many of them may have encountered similar problems and found a solution.

# Submission Directions for Assignment Deliverables

Submit a **single SQL script "solution.sql"**.

## When you are ready to submit:

1. Go to "**Programming Assignment: Assignment 2: SQL Query for Movie Recommendation**".
2. Click on the "**My submissions**" tab (located at the top of the page under the deadline date).
3. Click on "**Create submission**."
4. Upload one file for the assignment and click "**Submit**."
    a. If there is an error in your assignment, you may make corrections and resubmit.

## Test Cases:

You may use the following test cases to test your code against the autograder:

- TEST CASE 1: Leave genres.name empty
- TEST CASE 2: Insert a rating larger than 5

# Evaluation

There are nine (9) test cases/queries for a total of 1 point, so each query is worth 0.1 points. **If your .sql fails, you will see the corresponding .sql error logs that indicate where the error occurred.** In the end, if the submission runs correctly, you will see feedback that states "You passed 9/9 tests."

The test cases are executed in a simultaneous manner. If you pass any 2 of the 5 test cases, you would receive 40% of the total marks.

## Common Errors:

1. Incomplete assignment (missing queries)
2. Improper join of tables leading to a syntax structure error
3. Missing constraints as per the instructions for the later part of queries
4. Bad query structure makes it difficult to execute
5. Missing a semicolon to mark an end of query
6. Avoid writing table names as public.tablename (only table name is enough)
7. Syntax error (i.e too many comments, extra lines, or white spaces)
8. Use of '=' instead of 'IN' operator while searching in nested queries

## Notes:

*The recommendation table created here uses a technique called item-based collaborative filtering to compute the predicted scores for each movie based on that user's rating history. If you are interested in learning more about models like this, the course staff recommends reading this material.

## Learner Checklist

Prior to submitting, read through the Learner Checklist to ensure you are ready to submit your best work.

- ☐ Did you title your file correctly and convert it into a single **.sql file**?
- ☐ Did you answer all of the questions to the best of your ability?
- ☐ Did you make sure your answers directly address the prompt(s) in an organized manner that is easy to follow?
- ☐ Did you proofread your work?