Ira A. Fulton Schools of Engineering
Arizona State University

Engineering | School of Computing, Informatics, and Decision Systems Engineering

CSE 511: Data Processing at Scale

SQL Query for Movie Recommendation

Introduction:

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. To successfully complete this assignment, you will need to have correctly created the table definitions for the movie database in Assignment 1.

Requirements:

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” 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 text	average 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 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 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 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 integer	rating numeric
1	110	5

10. Write an SQL query to create a recommendation table* for a given user. Given a userID v1, you need to recommend the movies according to the movies he has rated before. In particular, you need to predict the rating P of a movie “i” that the user “Ua” didn’t rate. In the following recommendation model, P(Ua, i) is the predicted rating of movie i for User Ua. L contains all movies that have been rated by Ua. Sim(i,l) is the similarity between i and l. r is the rating that Ua gave to l.

$$P_{(u_a,i)} = \frac{\sum_{l \in \mathcal{L}} sim(i,l) * r_{u_a,l}}{\sum_{l \in \mathcal{L}} sim(i,l)}$$

11. Your SQL query should return predicted ratings from all movies that the given user hasn’t rated yet. You only return the movies whose predicted ratings are >3.9. Your query result should be saved in a table called “recommendation” which has one attribute: “title”.

	title text
1	Toy Story (1995)

12. In order to produce the recommendation table, you first need to calculate the similarities between every pair of two movies. The similarity is equal to the similarity of the average ratings of two movies (average rating over all users, the average ratings in Query 5). That means, if the average ratings of two movies are more close, the two movies are more similar. The similarity score is a fractional value E [0, 1]. 0 means not similar at all, 1 means very similar. To summarize, the similarity between two movies, i and l, is calculated using the equation below. Abs() is to take the absolute value.

$$\text{Sim}(i, l) = 1 - \frac{abs(i's\ avgrating - l's\ avgrating)}{5}$$

13. The result of similarity table should look like as follows:

	movieid1 integer	movieid2 integer	sim numeric
1	3565	2026	0.80000000000000000000

Note: You do not have to follow the exactly 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 10 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 “psql –v” option. v1 takes an integer as the input and it is the user ID v1used 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 visit the discussion forum, as many of your peers may have encountered similar problems and found a solution.

Submission:

Submit a single SQL script “solution.sql”.

Feedback:

There are ten test cases for a total of 1 point, so each test case 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 10/10 tests."

Test cases are not executed in parallel. You have to unlock all previous test cases to execute the current test case. For example, to execute test case 2, your code must be able to successfully execute test case 1. The test case number in the feedback will provide you an indication of which part of the assignment caused the error.

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.