SQL Movie-Rating Query Exercises

Bookmark this page

Exercise due Aug 29, 2021 10:00 +08 Completed

You've started a new movie-rating website, and you've been collecting data on reviewers' ratings of various movies. There's not much data yet, but you can still try out some interesting queries. Here's the schema:

Movie (*mID*, title, year, director)
English: There is a movie with ID number *mID*, a title, a release *year*, and a *director*.

Reviewer (rID, name)
English: The reviewer with ID number *rID* has a certain *name*.

Rating (rID, mID, stars, ratingDate)
English: The reviewer *rID* gave the movie *mID* a number of *stars* rating (1-5) on a certain *ratingDate*.

Your queries will run over a small data set conforming to the schema. View the database. (You can also download the schema and data.)

Instructions: Each problem asks you to write a query in SQL. To run your query against our back-end sample database using SQLite, click the "Submit" button. You will see a display of your query result and the expected result. If the results match, your query will be marked "correct". You may run as many queries as you like for each question.

Important Notes:

- *Your queries are executed using SQLite, so you must conform to the SQL constructs supported by SQLite.*
- Unless a specific result ordering is asked for, you can return the result rows in any order.
- *You are to translate the English into a SQL query that computes the desired result over all possible databases.* All we actually check is that your query gets the right answer on the small sample database. Thus, even if your solution is marked as correct, it is possible that your query does not correctly reflect the problem at hand. (For example, if we ask for a complex condition that requires accessing all of the tables, but over our small data set in the end the condition is satisfied only by Star Wars, then the query "select title from Movie where title = 'Star Wars'" will be marked correct even though it doesn't reflect the actual question.) *Circumventing the system in this fashion* will get you a high score on the exercises, but it won't help you learn SQL. On the other hand, an incorrect attempt at a general solution is unlikely to produce the right answer, so you shouldn't be led astray by our checking system.

You may perform these exercises as many times as you like, so we strongly encourage you to keep working with them until you complete the exercises with full credit.

Q1
1/1 point (graded)
Find the titles of all movies directed by Steven Spielberg.

Note: Your queries are executed using SQLite, so you must conform to the SQL constructs supported by SQLite.

```
select title
from Movie
where director = 'Steven Spielberg';
```

Press ESC then TAB or click outside of the code editor to exit

Submit

Correct

Q2
1/1 point (graded)
Find all years that have a movie that received a rating of 4 or 5, and sort them in increasing order.

Note: Your queries are executed using SQLite, so you must conform to the SQL constructs supported by SQLite.

```
select distinct M.year
from Movie M, Rating R
where M.mID = R.mID and R.stars in (4,5)
order by M.year asc;
```

Press ESC then TAB or click outside of the code editor to exit

Submit

Correct

Q3
1/1 point (graded)
Find the titles of all movies that have no ratings.

Note: Your queries are executed using SQLite, so you must conform to the SQL constructs supported by SQLite.

```
select title
from Movie
where mID not in (select mID from Rating);
```

Press ESC then TAB or click outside of the code editor to exit

Submit

Correct

Q4
1/1 point (graded)
Some reviewers didn't provide a date with their rating. Find the names of all reviewers who have ratings with a NULL value for the date.

Note: Your queries are executed using SQLite, so you must conform to the SQL constructs supported by SQLite.

```
select name
from Reviewer
where rID in (select rID from Rating where ratingDate is null);
```

Press ESC then TAB or click outside of the code editor to exit

Submit

Correct

Q5
1/1 point (graded)
Write a query to return the ratings data in a more readable format: reviewer name, movie title, stars, and ratingDate. Also, sort the data, first by reviewer name, then by movie title, and lastly by number of stars.

Note: Your queries are executed using SQLite, so you must conform to the SQL constructs supported by SQLite.

```
select RV.name, M.title, R.stars, R.ratingDate
from Reviewer RV, Movie M, Rating R
where RV.rID = R.rID and M.mID = M.mID
order by RV.name, M.title, R.stars;
```

Press ESC then TAB or click outside of the code editor to exit

Submit

Correct

Q6
1/1 point (graded)
For all cases where the same reviewer rated the same movie twice and gave it a higher rating the second time, return the reviewer's name and the title of the movie.

Note: Your queries are executed using SQLite, so you must conform to the SQL constructs supported by SQLite.

```
select RV.name, M.title
from Rating R1 join Rating R2 on
     (R1.rID = R2.rID and R1.mID = R2.mID and
          (R1.rID, R1.mID) in (select rID, mID from Rating
                               group by rID, mID having count(*)=2)
      and R1.ratingDate < R2.ratingDate and R1.stars < R2.stars)
   join Movie M on (M.mID = R1.mID)
   join Reviewer RV on (RV.rID = R1.rID);
```

Press ESC then TAB or click outside of the code editor to exit

Submit

Correct

Q7
1/1 point (graded)
For each movie that has at least one rating, find the highest number of stars that movie received. Return the movie title and number of stars. Sort by movie title.

Note: Your queries are executed using SQLite, so you must conform to the SQL constructs supported by SQLite.

```
select M.title, max(R.stars)
from Movie M join Rating R on (R.mID = M.mID)
group by M.mID
order by M.title;
```

Press ESC then TAB or click outside of the code editor to exit

Submit

Correct

Q8
1/1 point (graded)
For each movie, return the title and the 'rating spread', that is, the difference between highest and lowest ratings given to that movie. Sort by rating spread from highest to lowest, then by movie title.

Note: Your queries are executed using SQLite, so you must conform to the SQL constructs supported by SQLite.

```
select M.title, (max(R.stars) - min(R.stars)) as ratingSpread
from Movie M, Rating R on (M.mID = R.mID)
group by M.mID
order by ratingSpread desc, M.title;
```

Press ESC then TAB or click outside of the code editor to exit

Submit

Correct

Q9
1/1 point (graded)
Find the difference between the average rating of movies released before 1980 and the average rating of movies released after 1980. (Make sure to calculate the average rating for each movie, then the average of those averages for movies before 1980 and movies after. Don't just calculate the overall average rating before and after 1980.)

Note: Your queries are executed using SQLite, so you must conform to the SQL constructs supported by SQLite.

```
select avg(pre1980.avgrating) - avg(post1980.avgrating)
from (select avg(stars) as avgrating
      from Rating
      join Movie on (Movie.mID = Rating.mID and Movie.year < 1980)
      group by Rating.mID) as pre1980
   join (select avg(stars) as avgrating
      from Rating
      join Movie on (Movie.mID = Rating.mID and Movie.year > 1980)
      group by Rating.mID) as post1980;
```

Press ESC then TAB or click outside of the code editor to exit

Submit

Correct

Hide Notes