Databases & Web Applications

Homework #3: SQL and Servers

Rutgers Course 33:136:440
Instructor: Yair Aviner

Rules

1. Upload your completed assignment to Blackboard as a single *.zip* file containing all required files to view your work.
2. You may work in groups of no larger than 3; however, **submissions must be individual**.
3. If you work in a group you must identify, in each submission, who you worked with.
4. This assignment is **optional.** If you do not turn it in it will not impact your grade. If you do turn it in, however, your lowest assignment grade for the semester will be dropped.
5. **This assignment is due December 20, 2017 at 11:00PM EST**

Part 1: Database Queries (40 Points)

For this exercise you are given *import-movies.sql*, which you must not modify in any way. You must create a new database (*schema*) on your local database server called *movies*. To import the data to your new database in MySQL Workbench:

1) Click on *Data Import/Restore* in the left column of MySQL Workbench (under Management).
2) Select *Import from Self-Contained File,* click the file-chooser and select the provided *import-movies.sql* file.
3) Click the dropdown next to *Default Target Schema* and select the 'movies' database that you created earlier.
4) Click *Start Import* in the bottom right of the screen.
5) A log should appear on your screen; the last line should indicate that the import has finished.
6) The tables may not immediately appear in the left column (under 'movies'). To fix this, click the small 'refresh' icon to the right of *Schemas* in the left column.

For the assignment you must write the following queries:

1) **Prolific Raters (10 pts):** Find the IDs of the users who have submitted the most ratings. For reference, the correct results should begin with:

userId	num_ratings
547	2391
564	1868
624	1735

2) **Highest rated popular movies (15 points):** Find the 50 highest rated movies in the dataset which have been rated *at least 100 times* by users. The returned columns should include the name of the movie, the average rating, and the number of ratings. For reference, the correct results should begin with:

rating	title	num_ratings
4.5800	Godfather, The (1972)	200
4.5691	Shawshank Redemption, The (1994)	311
4.4667	Godfather: Part II, The (1974)	135

3) **Actively changing ratings (15 points):** Find the movies that have been rated the most times since January 1, 2015. Sort the results by number of ratings, in descending order. For reference, the correct results should begin with:

title	num_ratings
Matrix, The (1999)	58
Inception (2010)	58

Star Wars: Episode IV - A New Hope	56

Part 2: Database Applications (60 Points)

For this exercise you will be writing a simple server using Node & Express.js. The server must listen on port **5000**, accept HTTP requests, and perform actions (including sending back HTTP responses) as a consequence of those requests. Additionally, the server **must connect to the MySQL database from Part 1** in order to retrieve and update data.

Specifically, it must do the following:

1) (30 pts) When the route **/rate** is reached via a GET request your server should render and send, as HTML, a simple page containing:
    a) The title 'Rate Movies!'
    b) 10 **random** (hint: use `RAND()` in your query!) movies, presented visually as a list.
        i) Each list item should include the name of the movie, its associated genres, and its average rating.
        ii) Additionally, each list item should include a numeric text input (**`<input type="number" />`**) and a button that reads 'Add Rating!'
        iii) The numeric text input should accept only integer values ranging from 0 - 5 (including 0 and 5).
            (1) For more on number inputs, see: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input/number
        iv) When the 'Add Rating!' button is clicked, a POST request should be sent via AJAX to the route '**/rate/movie**' with data that includes the movieId, the rating from the number input, and a userId that is '1000'.
            (1) It is suggested (but not required) to send this data as JSON.
            (2) You must log the response from the server in the browser console (`console.log()`)
    c) The page must be at least minimally styled, that is, it should be clear what is being presented to the user.

2) (30 pts) When the route **/rate/movie** is reached via a POST request the server must update the **ratings** table using the posted data. Specifically, it must add a row containing the given userId, movieId, rating, and timestamp.
    a) The timestamp should be calculated in the function handling this route. It should be a Unix timestamp, meaning the number of seconds that have passed since 1970. You can calculate this with the following Javascript:

    ```
    const now = new Date();
    const nowInMilliseconds = now.valueOf();
    const timestamp = Math.floor(nowInMilliseconds / 1000)
    ```

    Explanation: The above code creates an object (*now*) that represents the current date. It then gets the numeric value of that date, represented as milliseconds since 1970. We convert that into seconds by dividing the number by 1000 (since 1 millisecond === 1/1000 of a second...) and round the value down to get an integer.

    If using *moment.js* for our date handling, the above could be written simply as:

    ```
    const timestamp = moment().format('X');
    ```

    b) After the row is successfully added to the database your server must respond to the original request with a success message.