CSL332 - Project Report

I Siddharth 2009CE10304, K Sai Teja Reddy 2009PH10723

April 26, 2012

Motivation:

In this age of multimedia and movie fanatics, there are millions of movies to choose from. But which movie to watch next? We need a comprehensive method to analyse user tastes and suggest him the movies hed love to watch but doesnt know about.

Description:

We all watch movies. Many times, we wish we could find movies which were similar to the movies we liked the most. We hope theres something which read our minds and suggested us on movies. This application has a database which notes down a users rating on different genres of movies and suggests movies based on those and can further be sorted by custom preference of the user. The application tracks a users ratings on different movies and gives suggestions on movies using an algorithm.

Data Gathering and Cleaning:

- Data Download - The Rotten Tomatoes API allows for a search query of a string of minimum 3 characters. We used a script to generate all possible permutations of alphabets and numbers, which gives 36*36 (1296) permutations. For each of these permutations, we made requests to the api server using cURL in PHP and querying for movies which have one or more word which start with the given permutation which gave us the movie id, year and title in JSON format. We fed this data directly into a table called 'extra'. Before inserting into extra table, we checked if the id already exists in extra table, if it does we took next movie. This gave us a total of 101434 of unique movies.

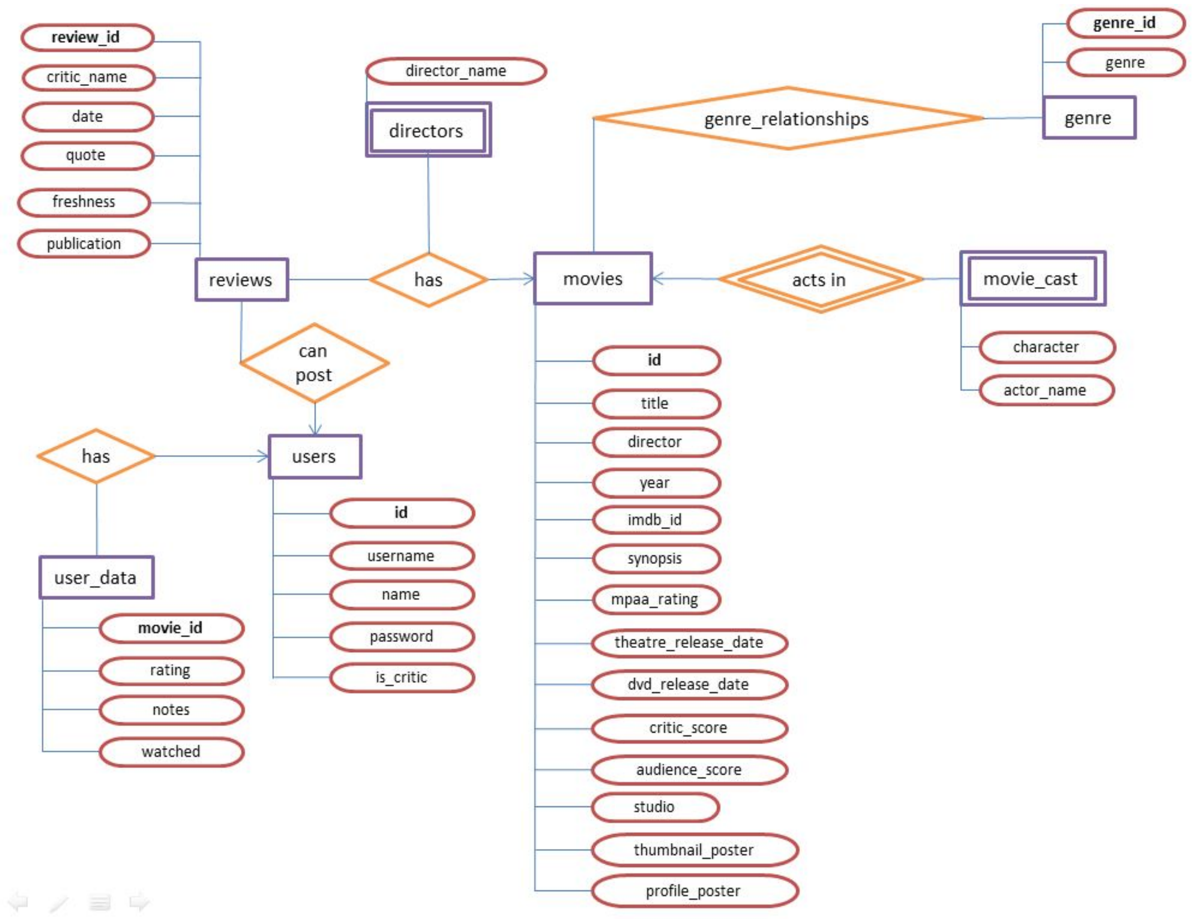
  Now that we got all movie id's, we queried Rotten Tomatoes for movie info and reviews for each of these movies. To be on the safe side, we saved the JSON returned into our tables json_dumps and json_dumps_reviews for movie info and reviews, respectively, from where we could later take and process the entries. We chose to do this instead of directly processing the data and inserting into the final database since the processing and insertion overheads would have driven up the time taken for the data extraction, which was already taken As soon as we got info for a movie, we marked the 'done' column in extra as true for that movie and as soon as we got all reviews for a movie, we marked 'done_review' as true for that movie. To make the process fast, we used 4 different files. One read all ids less than average ids from top, the other from bottom in the table sorted by id's. Similarly, remaining two read all ids greater than average ids from top and bottom respectively. This was done since the number of queries was limited to 10 per second by Rotten Tomatoes and the actual speed was around 1 to 2 results per second. So we split the queries into the above explained categories and used multiple API keys.

- Clean Up Steps - After downloading the JSONs for info and review, we eliminated duplicates using the PostgreSQL's hidden id 'ctid' for each table.

  Sample SQL: DELETE FROM json_dumps where ctid not in (SELECT MAX(ctid) FROM json_dumps GROUP BY id,json);

- Scripts - As the API we used doesn't provide the whole information at once, we've used different scripts to download different parts at different times. We've used feed-extra.php to feed extra table, movie_reviews.php to get JSON for movie reviews and movie_info.php to get JSON for movie info.

  After we got JSONs, we've used info_decoder.php and review_decoder.php to feed the data into final tables.



Interface and Back-end Sketch:

User's views of the system

1. Home screen - This is a minimalistic view with just a search box and has a text box element to enter the keyword for search and a dropdown which contains options for categories of search. The possible options are movie names, directors, genres, actor names. We are still trying to implement a holistic search using php to build the queries. Detailed Movie View After a user selects a movie for detailed view, the movie id is captured and the following details of the movie are extracted from the database: synopsis, poster url, director, cast, rating, imdb id, critic/user score, personal notes, watched status. The user can mark a movie as Watched (for his personal reference) or Wanna Watch (user can mark movies that he is interested in and view them later). Also, he can rate the movies (in which case the movie will automatically bemarked as "Watched").

   - WannaWatch List View - While browsing for movies, a user can mark movies he'd like to watch as "WannaWatch" for later reference. The "WannaWatch" view lists the movies earlier marked as "WannaWatch" by the logged in user, which can be marked as "Watched" from here. This changes (or adds) the watched column of the corresponding tuple in user_data.
   - Watched List View - While browsing for movies, a user can mark and rate movies he watched. This viewlists the movies marked "Watched" by the logged in user. The data is read from the user_data for the current logged in user.
   - Recommended Movies - Movies are recommended by picking actors in all the movies rated by the current user so far, sorting them by the number of times an actor (or director) appears in the result, picking the top actors, and fetching top movies (by audience_score) in which they have acted (or directed).
   - Advanced Search Form View - This view contains a form where a user can search a movie by multiple attributes such as director, actor, genre, date of release, rating (greater or less than) etc.

After submitting the form, the query will be built using PHP and the results will be shown on Search Results view as explained below.

- Search Results View - This is a common view for every search a user performs. This has movie listings with limited attribute values like title, year, cast(3 or 4), director, genre etc, and in case it shows advanced search results, it will show the search attributes. These are by default sorted by audience_score but can also be sorted by any attribute.

2. System view -

   - Foreign keys - We've used for foreign keys for table user data which checks for user id in user's table and movie id in movies table.
   - Constraints - We've used a constraint to check if the watched status is one of the following: watched, wannawatch, not watched.
   - Admin(for adding data) - An option for admin to add a movie, genre, cast, etc. This view is visible only for admin (after admin login).This presents all the fields to add a new movie to the database.

Queries:

The sql files are attached with this file in queries.sql.

- These queries are fired from php and include php variable and hence most of these will not run unless the variables are replaced with actual values.
- Also, since most of the queries fired are search queries and the response time for search queries depends on the string length (and regex pattern length), the variations in response times varies from a few milliseconds to a few seconds.
- The queries for recommended movies are formed and processed on the fly through php and so, determining the total time for all the sub-queries to be processed for the recommended movies view will not yield the overall response time in the front-end.