THE CROSSROADS OF FILM
TEAM CROSSROADS

Ben Dalgarn, Anna Smith, Katie Schermerhorn, Pierce Cunneen

PROJECT ACCOMPLISHMENTS

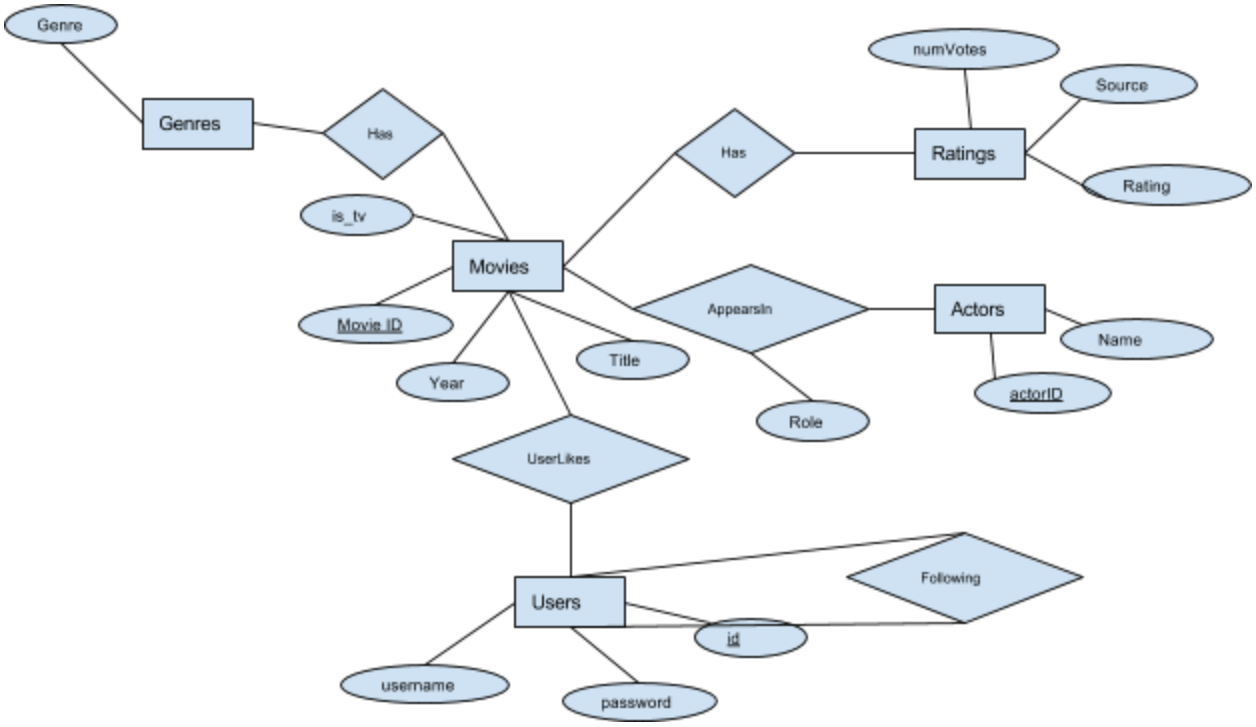
For our project, we built a movie recommendation/search service. It is a movie organizer that tracks movie data including actors, ratings, genres, etc. that gives recommendations based on different measures of similarity from multiple sources and provides users with the ability to access and filter this movie information. Users have the ability create an account and "like" different movies. Users are also able to follow other users and get recommendations for movies to watch based on the likes of the users they follow.

PROJECT USEFULNESS

Often, when thinking about or discussing movies, you'd like to be able to search for movies by various fields (actors, time-frame, genre, etc). Additionally, you'd like to get recommendations for movies based on movies that your friends enjoy. We'd like to build a service that fulfills all our movie search/recommendation needs. Rather than having to use IMBD to get movie data along with a different form of social media to get recommendations and find friends, we bundled it all into one helpful web application. We created our service to allow users to search for movies and actors in creative ways. For example, if your movie night is centered around watching an adventure movie with a certain actor, then you will be able to pull up a list of movies that fit those specifications.

DATABASE

ER DIAGRAM



SCHEMA

Genres(movieID, genre)

Movies(movieID, title, year, is_tv)

Ratings(Rating, Source, MovieID, numVotes)

Actors(actorID, name)

AppearsIn(movieID, actorID, role)

Following(Follower, Following)

UserLikes(user_id, movie_id)

Users(id, username, password)

DATA ORIGIN

We obtained our movie, rating, genre, and actor data from the IMDB database hosted on AWS. For the 'Following', 'Users', and 'UserLikes' tables, we used user data.

FUNCTIONALITIES

- Basic Search: Clicking on the ‘Search for Films’ button will allow the user to search for a film by movie title, returning the title, year, and rating. From this screen, the user can also ‘Like’ certain movies.
- Rate a Film: Clicking on the ‘Rate a Film’ button will allow the user to search from a film by movie title, and rate the movie they choose.
- Insert: Clicking on the ‘Insert’ button on the lower right corner of the dashboard will allow the user to insert a new movie into the movie database.
- Advanced Search: Click on the ‘Advanced Search’ button will allow the user to search for a film using specifications based on title, year, genre, actor, or rating.
- Follow Other Users: Clicking on the ‘Follow Other Users’ button will allow the user to find other users, view their ‘Likes’, and follow them.
- See My Likes: Clicking on the ‘See My Likes’ button will allow the user to view the movies they have ‘Liked’ and remove movies they do not like anymore.
- Get Recommendations: Clicking on the ‘Get Recommendations’ button will allow the user to view a list of recommendations based on the movies that the users they follow have liked.

SEARCH BASIC FUNCTION

The “Basic Search” function allows the user to search for a movie in the database and output a list of films that match the user’s search terms as well as the ratings for each of these movies.

SQL CALL

```
'SELECT m.movieID, m.title, m.year, r.Rating
FROM Movies m
LEFT JOIN Ratings r ON r.MovieID=m.movieID
WHERE Title LIKE "{}"
ORDER BY r.Rating DESC'
```

DATAFLOW

When a user enters the search terms into the form in the html and presses ‘Submit’, the form will send a POST request, triggering the python code to execute. The function calls the searchMovieDB() function in our code with the search terms as the parameters. This function adds ‘%’ to the beginning and end of the user’s search terms and executes the above SQL call with %searchTerms% in place of the brackets. It uses the fetchall() function to return the tuples that are returned with this query. These are sent back to the original function. The search.html page is called again with the new tuples which are output to the screen.

ADVANCED FUNCTIONS

- Collaborative Filtering Recommendations Function
  - This advanced function allows for users to make accounts and “Follow” other users. Users can then “Like” movies using the Search for Films and Advanced Search functions. Using this data, the “Get Recommendations” functions will join this information together and output a list of recommended movies to the user. This list will consist of the top 10 movies that the users you follow “Like”.
  - This function is considered advanced because it takes into account many things when it outputs a list of recommended movies. If a certain movie is liked by multiple users that you follow, that movie will ascend higher in the list. If you have already watched one of the recommended movies, it will not appear in the list.
- Advanced Search
  - This advanced function allows for users to input different parameters for movies. This include the title of a movie, a range of years that the movie was created, a name of an actor that was in the movie, a genre of the movie, and a range of Ratings that the movie could be. Once the user presses the ‘SEARCH’ button, the input parameters are given to the advMovieSearchDB() function which creates the sql query using the getAdvSearchQuery() function. The movies received from this query are output to the screen for the users to like.
  - This function is considered advanced because it creates an advanced sql query based on user input. If the user input includes an actor name or a genre, these tables are included in the query along with their conditionals. An example sql query is given below. The function is also advanced because we created many unclustered indices on the following table columns:
    - title and year in Movies
    - movieID and genre in Genres
    - Rating in Ratings
    - name in Actors
    - actorID in AppearsIn
  - Example query:
    - SELECT m.movieID, m.title, m.year, r.Rating

      FROM Movies m

      INNER JOIN Ratings r ON m.movieID=r.MovieID

      INNER JOIN Genres g ON g.movieID=m.movieID

      INNER JOIN AppearsIn ai ON ai.movieID=m.movieID

      INNER JOIN Actors a ON a.actorID=ai.actorID

      WHERE g.genre='Adventure' AND r.Rating >=5 AND m.title LIKE 'a%' AND a.name LIKE '%a%'

      GROUP BY m.movieID;

TECHNICAL CHALLENGE

One technical challenge we encountered while working on this project was the speed of the advanced search. Even after adding joins to the table, we were experiencing extremely slow performance. To remedy this situation, we looked closely at the specific tables and columns we were performing the joins and conditions on. We placed unclustered indices on the columns listed above in our description. This made our queries run much faster. Performance was remedied. Most of our advanced queries now run somewhere between 0 and 3 seconds.

ACCORDING TO PLAN

Everything should go to our plan and design specifications.

DIVISION OF LABOR

Team Crossroads split the work equally between team members. Pierce focused on parsing the IMBD data into the mySQL database. Ben and Katie focused on the Advanced Search function. Anna and Katie spent their on the basic functions and the second advanced function.