Are you A Movie Junkie?

Amey Hegde
Archith Shivanagere Muralinath
Bill (Chen Ze) Dong
Tejesvi Chadag

ABSTRACT

We propose to build a Web based movie quiz application as it presents itself as an interesting project that is stimulating while at the same time socially engaging. Users can choose between different categories of quizzes and steps are taken to ensure the uniqueness of the quiz every time a user plays it. Users are tested on their knowledge of movies with some help in the form of clues. Users can also challenge their Facebook Friends who are registered users of the application, thus making the application competitive and socially engaging.

BASIC ARCHITECTURE

Server
The application backend is hosted on an instance of AWS VPC. The server uses a micro EC2 instance running nodeJS. The application relational database consists of a RDS instance using MySQL and the NoSQL databases consists of a mongoDB instance hosted using mongolab. The node server performs all database queries, to the relational and noSQL database, as well as the query to the Bing API to dynamically retrieve images.

Client
The client uses jQuery and bootstrap to perform the frontend logic. AJAX calls are used to call the server to perform database queries and receive the information as a JSON datatype. The Facebook API is also used by the client to retrieve user information such as the Facebook id and Facebook friends. This is communicated to the node server through AJAX calls.

Query Files - NodeJS modules

SQL	AreYouAMovieJunkie/nodeApp/moviesmysqldb.js
NoSQL	AreYouAMovieJunkie/nodeApp/moviesmongodb.js

DATA INSTANCE

In order to generate clues for a question, data regarding the title, genre, cast, character names, director, overview and image was required. Multiple data instances were used to acquire this data.

From TMDBMovieInfo and TMDBPersonInfo, data regarding a movie's title, genre, cast, character name, director, release date and overview are obtained. This data is stored in Tables "Movies", "Movie_genre", "Actors", "Directors" and "Overview". TMDB_ID is set as the primary key in table Movies. Relation "Roles" is used to link table "Actors" and "Movies" and relation "Directs" is used to link table "Director" and "Movies". Both the relations have TMDB_ID as the foreign key.

In order to measure the popularity, the votes for a movie was obtained from Rotten Tomatoes-MovieInfo. To link the Rotten Tomatoes and TMDB data sets, we used Links.csv to find the corresponding TMDB_ID for an IMDB_ID. once this was done, Table movies was updated with data regarding the votes for a movie.

External Dataset regarding Oscar nominees and winners is used to implement awards category of quiz. We found that for a particular award category, there are multiple nominees with no unique distinguishing feature, making it impossible to assign a key. Hence this dataset is stored in Mongolab and Nosql is used to query for clues.

DATA CLEANING AND IMPORT MECHANISM

There were two things which we considered while cleaning the data:

1) Minimum Redundancy
2) Hollywood based Movie Quiz

For minimum redundancy we have ensured that all the tables in our dataset are in First Normal Form.This guarantees that each tuple in each and every table is unique.

We found that Hollywood Movies were the most popular movies. To implement a Hollywood based Movie Quiz, we trimmed the dataset to contain only movies with votes>50, thus eliminating all international movies

Steps for Data Cleaning:

a) Access the JSON file of various datasets provided.
b) Running the python based scripts to convert from JSON to CSV file.
c) The scripts ensured that CSV files were in First Normal Form as well as encoding read the special characters.
d) Importing the CSV files into MySQL database using the MYSQL workbench.
e) The importing was done by a command (LOAD DATA LOCAL INFILE), this command loads the data from the CSV file with path to the file from the local machine used to populate the various tables in the database.

USE CASES

Selecting the right query is a key part of the project as it ensures the generation of unique questions for different categories. For the data stored in relational database, the different use cases implemented are

1. Based on Popularity:
   Higher the votes obtained by a movie, higher is its popularity. This feature is used by us to generate clues for the quiz categories Top 100(votes > 2058), Top 250(votes > 1108), Top 500(votes > 601) and All in(votes > 50). Based on the category selected by the user, a random movie is selected from all the movies having votes in that specific range.
   ```
   SELECT * FROM `MOVIES` WHERE VOTES > '+use+' ORDER BY RAND() LIMIT 0,1
   ```
   This query returns the title, image, release date and TMDB_ID of the movie. Using this

TMDB_ID(saved as bid), second query is performed to return information regarding the cast, character names, director, genre and overview of the movie.

SELECT MG.TMDB_ID, GROUP_CONCAT(DISTINCT GENRE_NAME)AS GENRE, GROUP_CONCAT(DISTINCT CHAR_NAME) AS CHAR_NAME, GROUP_CONCAT(DISTINCT ACTORS.NAME) AS ACTORS, GROUP_CONCAT(DISTINCT DIRECTORS.NAME) AS DIRECTOR, OVERVIEW FROM `MOVIES_GENRES` MG INNER JOIN `ROLES` R ON MG.TMDB_ID = R.TMDB_ID INNER JOIN `OVERVIEW` O ON MG.TMDB_ID = O.TMDB_ID, `DIRECTORS`, `ACTORS` WHERE MG.TMDB_ID = '+bid+' AND DIRECTORS.DID IN (SELECT DID FROM `DIRECTS` WHERE TMDB_ID = '+bid+') AND ACTORS.AID IN (SELECT AID FROM `ROLES` WHERE TMDB_ID = '+bid+')

This process is repeated 9 more times to generate the 10 questions for the quiz.

2. Based on Actors/Actresses:

   The user can also choose to play a quiz regarding a particular actor/actress. We have implemented quizzes for 13 actors and 13 actresses. Based on the actor/actress selected by the user, a random movie is selected from all the movies having that actor/actress in the cast.

   SELECT DISTINCT R.TMDB_ID, R.CHAR_NAME, M.TITLE, M.RELEASE_DATE, M.IMAGE FROM `ROLES` R INNER JOIN `MOVIES` M ON R.TMDB_ID = M.TMDB_ID WHERE AID = (SELECT AID FROM `ACTORS` WHERE NAME LIKE "%'+use+'%

   This query returns the title, image, release date, character name and TMDB_ID of the movie. Using this TMDB_ID(saved as bid), second query is performed to return information regarding the cast, director, genre and overview of the movie.

   SELECT MG.TMDB_ID, GROUP_CONCAT(DISTINCT GENRE_NAME)AS GENRE, GROUP_CONCAT(DISTINCT ACTORS.NAME) AS ACTORS, GROUP_CONCAT(DISTINCT DIRECTORS.NAME) AS DIRECTOR, OVERVIEW FROM `MOVIES_GENRES` MG INNER JOIN `ROLES` R ON MG.TMDB_ID = R.TMDB_ID INNER JOIN `OVERVIEW` O ON MG.TMDB_ID = O.TMDB_ID, `DIRECTORS`, `ACTORS` WHERE MG.TMDB_ID = '+bid+' AND DIRECTORS.DID IN (SELECT DID FROM `DIRECTS` WHERE TMDB_ID = '+bid+') AND ACTORS.AID IN (SELECT AID FROM `ROLES` WHERE TMDB_ID = '+bid+')

   This process is repeated 9 more times to generate the 10 questions for the quiz.

3. Based on Genre:

   The user can also choose to play a quiz based on a particular genre. we have implemented quizzes for 8 genres. Based on the genre selected by the user, a random movie is selected from all the movies in that particular genre.

   SELECT * FROM `MOVIES` WHERE TMDB_ID IN (SELECT TMDB_ID FROM `MOVIES_GENRES` WHERE GENRE_NAME LIKE "%'+use+'%") ORDER BY RAND() LIMIT 0,1

   This query returns the title, image, release date and TMDB_ID of the movie. Using this TMDB_ID in the second query of use case 1, all the remaining clue are generated. This process is repeated 9 more times to generate the 10 questions for the quiz.

4. Based on Challenge:
   Upon completion of a quiz, the user has the option to challenge a Facebook friend. if he chooses to challenge, the TMDB_ID's for each question asked in that quiz is stored along with the category and level of difficulty in table Challenge.
   INSERT INTO `CHALLENGE` (USER, FRIEND, QUIZ_ID, TMDB_ID, LEVEL) VALUES ('+user_fbid+', '+friend_fbid+', '+quiz_id+', '+tmdb_id+', '+level+')
   When the friend accepts the challenge, clues are generated using the second query of use case 1.

For the data stored as JSON file in MongoLab, the different use cases implemented are

1) Based on Roles:
   When this option is selected by the user, a random question is generated for leading and supporting roles of actors and actresses.
   db.collection('oscars').find({category: {$in: ['Actress -- Leading Role', 'Actor -- Leading Role', 'Actress -- Supporting Role', 'Actor -- Supporting Role']}}).limit(1).skip(random)
   where **random = Math.floor(Math.random()*N), N is count of total tuples returned**

2) Based on Movies:
   When this option is selected by the user, a random movie is generated for Best Picture, Directing and Animated feature film.
   db.collection('oscars').find({category: {$in: ['Best Picture', 'Directing', 'Animated Feature Film']}}).limit(1).skip(random)
   random is calculated the same way as shown above.

OPTIMIZATION TECHNIQUES

1. **Gather Statistics** - *gather_table_stats* procedure was run on all tables so that the database optimizer is familiar with tables' attributes. As this is a cost-based optimization approach, optimizer uses these statistics to calculate the selectivity of predicates and to estimate the cost of each execution plan. The optimizer uses the selectivity of a predicate to estimate the cost of a particular access method and to determine the optimal join order and join method.
   - Table statistics - number of rows, average row length
   - Column statistics - number of distinct values, number of null values, data distribution
   - Index statistics - number of leaves, number of levels

   Since all our tables, except USERS and CHALLENGE, are **static** (there is no change in volume, indexing, cardinality), we do not need to gather statistics over and over again. Optimizer chooses the best execution plan every time the query is run.

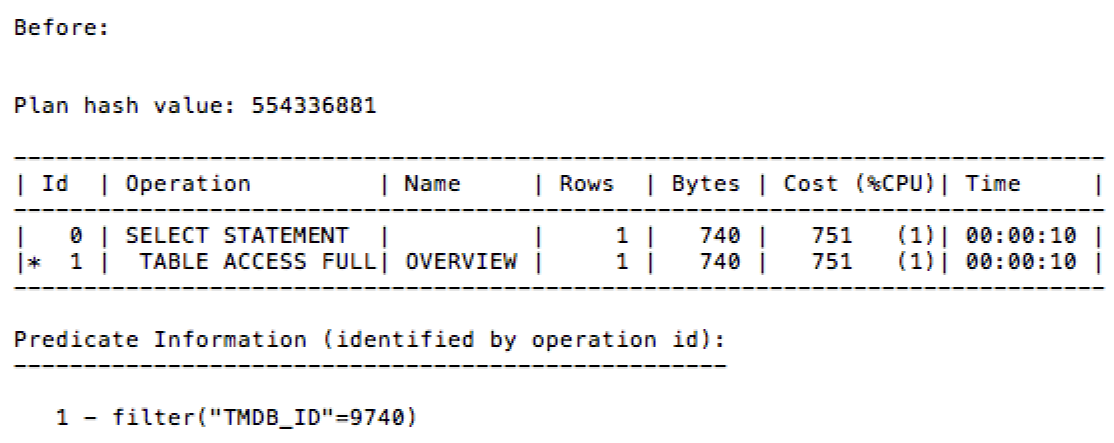
   MySQL database is integrated into Oracle SQL Developer as it offers a lot more flexibility for optimization. Below is the process followed to gather statistics on each table.

- ➢ exec dbms_stats.unlock_table_stats('Movies_Database', 'MOVIES');
  - -- this is used to unlock any existing statistics
- ➢ dbms_stats.gather_table_stats('Movie_Database', 'ACTORS',estimate_percent => 100, method_opt => 'for all columns size auto', cascade => true);
  - -- after unlocking stats, this procedure gathers latest stats on the table
- ➢ exec dbms_stats.lock_table_stats('Movies_Database', 'MOVIES');
  - -- after gathering stats, this statement locks stats for optimizer utilization

2. **Indexing** - Querying the database in our application is mostly based on equality/range searches. Hence, 5 B+ tree indexes are created on various tables to lower the cost, execution time and optimize performance. Below are the indexes created.

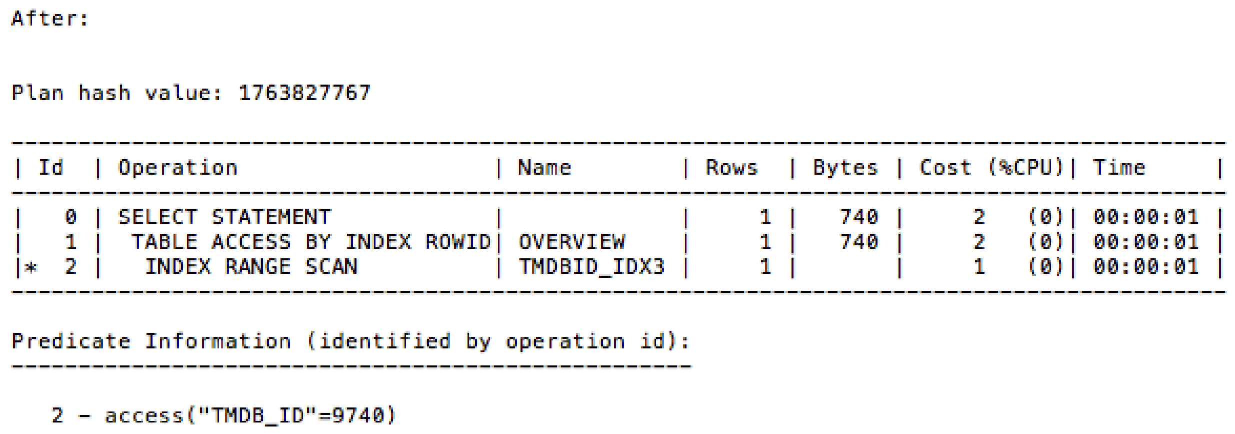
   a. create index tmdbid_idx3 on OVERVIEW(tmdb_id);
   b. create index aid_idx4 on ROLES(aid);
   c. create index tmdbid_idx on MOVIE_GENRES(tmdb_id);
   d. create index tmdbid_idx1 on DIRECTS(tmdb_id);
   e. create index tmdbid_idx2 on ROLES(tmdb_id);

   Cost and execution time reduction is shown below for one of the queries.



```
Before:


Plan hash value: 554336881

-------------------------------------------------------------------------------
| Id  | Operation          | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |          |     1 |   740 |   751   (1)| 00:00:10 |
|*  1 |  TABLE ACCESS FULL| OVERVIEW |     1 |   740 |   751   (1)| 00:00:10 |
-------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter("TMDB_ID"=9740)
```

As we can see in the above snapshot, before creation of index, cost is approximately 1500 and execution time is 10ms with a FULL TABLE ACCESS.



```
After:

Plan hash value: 1763827767

-----------------------------------------------------------------------------------------------
| Id  | Operation                    | Name          | Rows   | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |      1 |   740 |     2   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| OVERVIEW      |      1 |   740 |     2   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN           | TMDBID_IDX3 |      1 |       |     1   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("TMDB_ID"=9740)
```

After creating the index, cost is reduced to 5, execution time is reduced to 1ms with INDEX RANGE SCAN instead of a full table access.

KEY TECHNICAL CHALLENGES

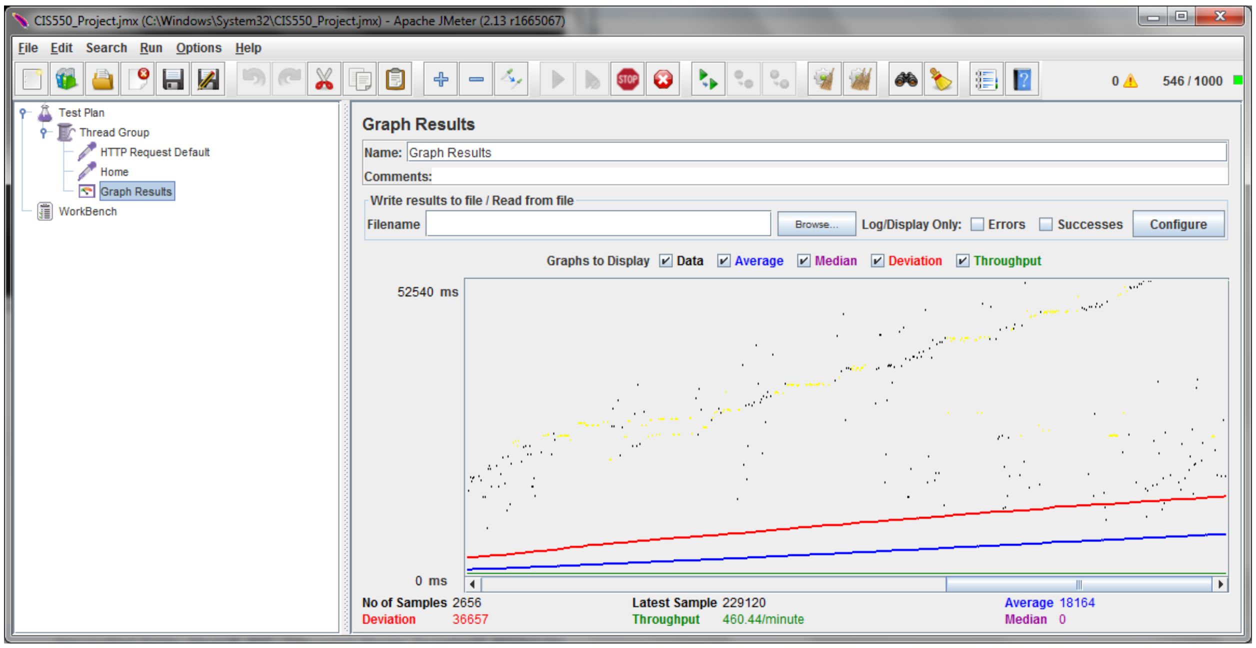
1) Database Section

The server side is written in node and database interaction is provided using existing node modules. An OracleDB was first used as the relational database but had to be switched due to incompatibilities with node. Therefore, a MySQL database was chosen.

2) Asynchronous Queries

Often times, the server needs to query the database in succession, for example when querying for the existence of a user and then retrieving the challenge information of that user. In addition, this is further complicated when adding a further Bing search query after relevant data have been retrieved from the database. Due to the event driven structure of node, an asynchronous library had to be used to ensure these queries were called in the right order with the final callback function called when all query operation have finished.

PERFORMANCE EVALUATION


CIS550_Project.jmx (C:\Windows\System32\CIS550_Project.jmx) - Apache JMeter (2.13 r1665067)
File Edit Search Run Options Help
0
546 / 1000
Test Plan
Thread Group
HTTP Request Default
Home
Graph Results
WorkBench
Browse...
Configure
52540 ms
0 ms

Graph Results

Name: Graph Results

Comments:

Write results to file / Read from file

Filename

Log/Display Only: ☐ Errors ☐ Successes

Graphs to Display ✓ Data ✓ Average ✓ Median ✓ Deviation ✓ Throughput

No of Samples 2656
Deviation 36657
Latest Sample 229120
Throughput 460.44/minute
Average 18164
Median 0

Number of threads (requests) tested for: 1000 per sec
Average time taken to execute the requests under given application is few milliseconds.
Throughput of the application is approximately 500/minute.

SPECIAL FEATURES:

1) Facebook API:
- The login API was used for users to login to our applications.
- This API also was used to get the list of friends of the user for the "Challenge a Friend" feature.

2) Bing Search API:
- This API was used to retrieve the images of the movies as one of the clues for quizzes based on Oscar awards (NoSQL component).

3) Different Level of Difficulty:
- The game is designed for three levels of difficulty: Easy, Challenging and Genius

a) Easy:
- Maximum score which can be achieved is 80 points.
- All the seven clues will be displayed at one go and you can guess the name of the movie.

b) Challenging:
- Maximum score which can be achieved is 90 points.

- Only Four out of the maximum seven clues will be displayed initially. There is an option to reveal more clues but at a cost of 10 points per clue.

c) Genius:

- Maximum score which can be achieved in 100 points.
- Only one out of the maximum seven clues will be displayed initially. There is an option to reveal more clues but at a cost of 10 points per clue.

4) Five Different Categories of Quiz:

a) Popularity Based:

- This includes Top 100, Top 250, Top 500 and All In.

b) Actors Based:

- This category includes Eight of most prominent actors in Hollywood.
- They are Brad Pitt, Leonardo DiCaprio, Christian Bale, Morgan Freeman, Johnny Depp , Tom Hanks, Bruce Willis, Liam Neeson, Russell Crowe, Matt Damon, Samuel L. Jackson, Tom Cruise and Matthew McConaughey.

c) Actress Based:

- This category includes Eight of most prominent actresses in Hollywood.
- They are Angelina Jolie, Sandra Bullock, Nicole Kidman, Cameron Diaz, Anne Hathaway, Julia Roberts, Natalie Portman,Kate Winslet, Jennifer Aniston, Halle Berry, Meryl Streep, Jennifer Lawrence and Judi Dench.

d) Genres Based:

- This category includes Eight different genres.
- They are Romance, Action, Adventure, Drama, Comedy, Thriller, Crime and Science Fiction.

e) Oscars Awards Based:

- This category includes two different variety. (Oscar Roles and Oscar Movies)

5) NoSQL Component (MongoDB):

- External dataset was used for quiz on Oscar Awards. We found that for a particular award category, there are multiple nominees with no unique distinguishing feature, making it impossible to assign a key. Hence this dataset is stored in Mongolab and Nosql is used to query for clues.

6) Global LeaderBoard:

- This leaderboard displays the top 5 scores which are registered by different users for a particular difficulty level and a particular category.
- This is unique for every level of difficulty and category.

7) Friends LeaderBoard:

- This leaderboard displays the top 5 scores which are registered only by your friends for a particular difficulty level and a particular category.

8) Timer for Score:
- The timer is used to score the user. The time remaining on the timer is the score for that particular question.
- We have implemented a timer which decrements by a point after every 2 seconds.

9) Challenge your Facebook Friends:
- There is an option to challenge your facebook friend who is an existing user of the application.
- The friend will receive the notification when he logs into the system which he can either accept or reject.

POTENTIAL FUTURE EXTENSIONS:

1) Login Authentication:
- This feature can be implemented by using text message verification using Twilio Messaging API.
- This will enable a secured login for each and every user into our applications.

2) Audio/ Visual Effect:
- If the user get the answer correct then there will be some kind of visual or audio effect which will state that “You have answered correctly”.

Division of work between Group members:

Name	Sections of Work
Tejesvi Chadag	1) Data Cleaning and Normalization 2) NoSQL Query component 3) SQL Query component
Bill (Chen Ze) Dong	1) Programming frontend logic 2) Setting up AWS infrastructure 3) Integrating query files to node server
Amey Hegde	1) Data Cleaning and Normalization 2) Populating the database 3) SQL Query component
Archith Shivanagere Muralinath	1) Optimization 2) Performance Testing 3) Indexing