Movies Database

808008228: Danish Naveed
775005575: Saadhan Pittala
792000617: Michael Ambrose
377009257: Graham Zorich
788006820: Jason Yeung

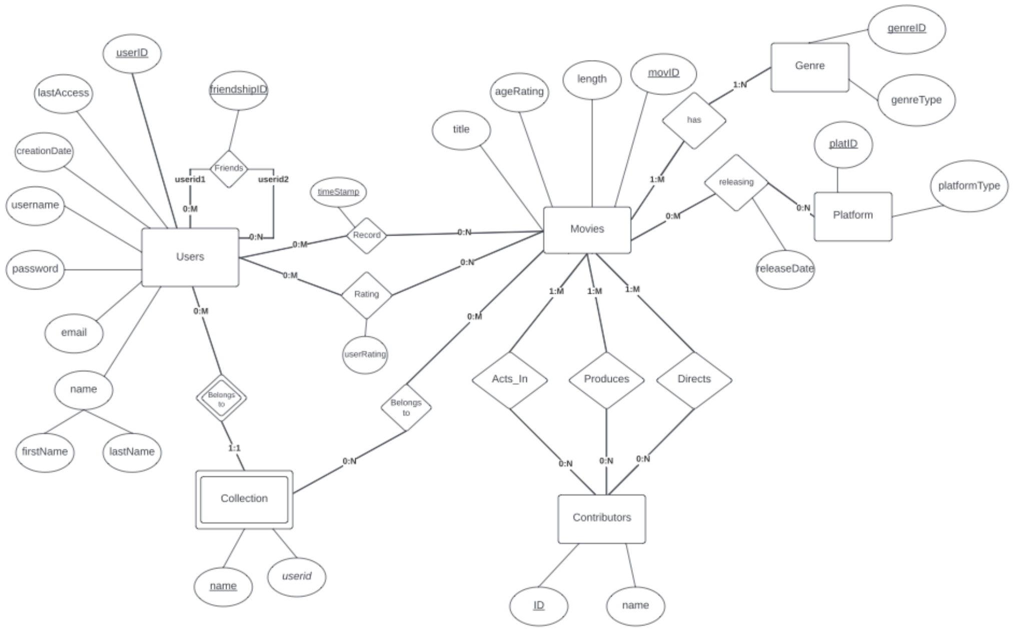
February 22, 2024

1 Introduction

Our project utilizes the Movies domain, using which we will analyze and document a variety of different movies. The planned database will be similar to a rough imitation of Netflix, on a smaller scale of course. Using the SQL language along with any potential Development Environments such as PostGres, to implement this design vision.

2 Design

2.1 Conceptual Model



User Privacy and Security
A password field suggests consideration for user security.
Flexibility and Scalability
Entities like Contributors and Movies are designed with roles and relationships (like Acts_In, Produces, and Directs) to easily accommodate new types of contributors or new relationships without major changes to the database structure. The EER is designed to handle an expanding dataset, as expected in a growing movie database.

2.2 Reduction to tables

Users(<u>userID</u>, lastAccess, creationDate, username, password, email, firstName, lastName)
Users are created using a username, password, and email
Users are found by a unique userID that is autogenerated on creation
Users can see when they last access and the creation date of the account
Users can see their own name and can friend other users by their id

Friend(<u>friendshipID</u>, *userID1*, *userID2*)

Friend has a auto generated unique friendshipID
A user can be friends with another unique user

Movies(movID, title, ageRating, length)
Movies are found by their ID which is autogenerated
Movies have a title and age rating specified from the MPAA
Movies also have a length of the movie specified

Record(*userID, movID, timestamp*)
Record are associated with a user by their userID
Record are associated with the specific timestamps or the time the user stopped the movie
Record are associated with the specific movie by the movie's id

Rating(*userID, movID*, userRating)
Rating are associated by the rater which is the user which is associated by their userID
Rating are for the specific movie specified by the movie ID
Rating are associated with a user rating of 1 - 5

Collection(name,*userID*)
Collection can be named and are associated with a user by its userID

Genre(genreID, genreType)
Genre are associated by its own unique id for each genre type

Platform(platID, platformType)
Platform are associated by its own unique id for each different platform type

Contributors(ID, name)
Contributor are associated by their IDs and name associated with the ID

ActsIn(*movID, ID*)
ActsIn are for actors and each are associated with their own unique ID
ActsIn also shows the movie the actors acts in by the movie ID

Produces(*movID, ID*)
Produces are for producers and each are associated with their own unique ID
Produces also shows the movie the producer produces by the movie ID

Directs(*movID, ID*)
Directs are for directors and each are associated with their own unique ID
Directs also shows the movie the producer produces by the movie ID

MoviesHasGenre(*movID, genreID*)
MovieHasGenre show what genres are associated by the genreID
with the movie which is by the movie ID

CollectionOfMovies(*name, movID*)
CollectionOfMovies are associated by a name
Movies in the collection are found by the unique ids

2.3 Data Requirements/Constraints

Use this section to list all the data domains and constraints that cannot be captured in your EER diagram but must be enforced by the database system.

- There are only three platforms where movies can be released on: Netflix, Theaters, and DVD.
- Length has to be in minutes, no special characters.
- Ratings can only be out of the list (G, PG, PG-13, R, X, NR).
- User Ratings must be a number from (1 - 5).
- Time Stamp will be hours:minutes:seconds
- Movie ID, User ID, platformID, genreID, friendshipID starts at 1

2.4 Sample instance data

Use this section to include a sample of entities for every entity type in your EER diagram.

Users:

- User1
  { userID: 1, username: ”moviebuff123”, password: ”encrypted_password1”, email: ”user1@example.com”, firstName: ”John”, lastName: ”Doe”, creationDate: ”2023-01-15”, lastAccess: ”2024-02-20” }
- User2
  { userID: 2, username: ”cinemalover”, password: ”encrypted_password2”, email: ”user2@example.com”, firstName: ”Jane”, lastName: ”Smith”, creationDate: ”2023-02-10”, lastAccess: ”2024-02-18” }
- User3
  { userID: 3, username: ”filmfanatic”, password: ”encrypted_password3”, email: ”user3@example.com”, firstName: ”Emily”, lastName: ”Johnson”, creationDate: ”2023-03-05”, lastAccess: ”2024-02-17” }
- User4
  { userID: 4, username: ”screenwriter”, password: ”encrypted_password4”, email: ”user4@example.com”, firstName: ”Michael”, lastName: ”Williams”, creationDate: ”2023-04-20”, lastAccess: ”2024-02-15” }
- User5
  { userID: 5, username: ”director_cut”, password: ”encrypted_password5”, email: ”user5@example.com”, firstName: ”Alice”, lastName: ”Brown”, creationDate: ”2023-05-25”, lastAccess: ”2024-02-10” }

Movies:

- Movie1
  { movID: 1, title: "Space Odyssey", ageRating: PG, length: 142 }
- Movie2
  { movID: 2, title: "The Great Adventure", ageRating: PG-13, length: 121 }
- Movie3
  { movID: 3, title: "Romance in Rome", ageRating: R, length: 113 }
- Movie4
  { movID: 4, title: "Mystery of the Abyss", ageRating: G, length: 130 }
- Movie5
  { movID: 5, title: "Comedy Nights", ageRating: R, length: 104 }

Genre:

- Genre1
  { genreID: 1, genreType: "Science Fiction" }
- Genre2
  { genreID: 2, genreType: "Adventure" }
- Genre3
  { genreID: 3, genreType: "Romance" }
- Genre4
  { genreID: 4, genreType: "Mystery" }
- Genre5
  { genreID: 5, genreType: "Comedy" }

Platform:

- Platform1
  { platID: 1, platformType: "Streaming Service", releaseDate: "2023-11-15" }
- Platform2
  { platID: 2, platformType: "Cinema", releaseDate: "2024-01-12" }
- Platform3
  { platID: 3, platformType: "Television", releaseDate: "2023-12-21" }
- Platform4
  { platID: 4, platformType: "DVD", releaseDate: "2024-02-10" }
- Platform5
  { platID: 5, platformType: "Online Rental", releaseDate: "2024-03-01" }

Contributors:

- Contributor1
  { ID: 1, name: ”Robert Phoenix” }
- Contributor2
  { ID: 2, name: ”Sophia Lorenzi” }
- Contributor3
  { ID: 3, name: ”Jonathan Creek” }
- Contributor4
  { ID: 4, name: ”Isabella Rosetti” }
- Contributor5
  { ID: 5, name: ”Markus Stahl” }

Collection:

- Collection1
  { name: ”John’s Favorites” }
- Collection2
  { name: ”Jane’s Sci-Fi Picks” }
- Collection3
  { name: ”Emily’s Romance” }
- Collection4
  { name: ”Michael’s Mystery Marathon” }
- Collection5
  { name: ”Alice’s Comedy Collection” }

Relationships:

Friends (between Users):

- User1 is friends with User2 and User3.
- User2 is friends with User4.
- User3 is friends with User5.
- User4 is friends with User1.
- User5 is friends with User2.

Acts_In (between Movies and Contributors):

- Contributor1 acts in Movie1 and Movie2.
- Contributor2 acts in Movie3.
- Contributor3 acts in Movie4.
- Contributor4 acts in Movie5.
- Contributor5 acts in Movie1.

Record (Users-Movies):

- User 1 watched Movie 1 (Space Odyssey) and left off at 00:11:15.
- User 4 watched Movie 2 (The Great Adventure) and left off at 00:50:30.
- User 7 watched Movie 3 (Love in the Time of AI) and left off at 00:45:30.
- User 1 watched Movie 4 (Cyber Detective) and left off at 00:40:33.
- User 1 watched Movie 5 (The Last Sunset) and left off at 00:10:30.

Rating (Users-Movies):

- User 1 rated Movie 1 (Space Odyssey 4).
- User 4 rated Movie 2 (The Great Adventure 4).
- User 7 rated Movie 3 (Love in the Time of AI 5).
- User 1 rated Movie 4 (Cyber Detective 3).
- User 2 rated Movie 5 (The Last Sunset 4).

3 Implementation

Use this section to describe the overall implementation of your database. Include samples of SQL statements to create the tables (DDL statements) and a description of the ETL process, including examples of the SQL insert statements used to populate each table initially.

Method in how the data was loaded into the database

For the tables users, collections, ratings, friends, and records, the data was populated just from testing alone. As in the API, we could create and delete the information in these columns. For users, we had to create accounts. For collections, users can create, modify(name or add movies), or delete. For rating users can rate movies. For friends, users can friend(follow) other users or unfriend(unfollow) them. For recording, each time the user plays a video or collection, it would be added to the table.

For all the other tables, such as movies, contributors, acts_in, and produces_in, to name a few, we took a large number of random yet unique movies, genres, and contributors, aka producers + actors + directors, and output them in a CSV file format. For each column the table has that is not auto-incremented already, such as id: the values were entirely randomized. Once we got all the variables associated with each respective table; we separated them in CSV format, aka comma-separated format, and imported the CSV file into the database. The CSV files are found in the zip under src with the application code.

Some sample SQL statements used to create the tables in the database was:

- create table users (userid integer generated by default as identity (maxvalue 100000) constraint users_pk primary key, lastaccess timestamp, creationdate timestamp, username varchar(30) constraint users_username_uk unique, password varchar(30), email varchar(50) constraint users_email_uk unique, firstname varchar(30), lastname varchar(30));

- create table friend (friendshipid integer generated by default as identity (maxvalue 100000) constraint friendship_pk primary key, userid1 integer not null constraint friend_userid1_fk references users on update cascade on delete cascade, userid2 integer not null constraint friend_userid2_fk references users on update cascade on delete cascade, constraint friend_friendships_uk unique (userid1, userid2));

Some sample SQL queries used to initially populate the database was:

- "INSERT INTO users (lastaccess, creationdate, username, password, email, firstname, lastname) VALUES (null, datetime.now(), test, test1, test@gmail.com, Danish, Naveed)"

- "INSERT INTO collection (name, userid) VALUES (horror, 1)"

- "INSERT INTO movies (movid, title, agerating, length) VALUES (63, Shrek, PG, 90)

- "INSERT INTO collectionofmovies (name, movid) VALUES (horror, 63)"

- "INSERT INTO record (userid, movid, timestamp) VALUES (1, 63, datetime.now())"

- "INSERT INTO rating (userid, movid, userrating) VALUES (1, 63, 3)"

Some sample SQL queries that were used in the API was:

- INSERT INTO users (lastaccess, creationdate, username, password, email, firstname, lastname) VALUES (null, datetime.now(), test, test1, test@gmail.com, Danish, Naveed)
- INSERT INTO users (lastaccess, creationdate, username, password, email, firstname, lastname) VALUES (null, datetime.now(), gyu2, yut2, gfuny612718@gmail.com, Goof, Funny)
- INSERT INTO friend (userid1, userid2) VALUES (1, 12)
- INSERT INTO friend (userid1, userid2) VALUES (1, 7)
- INSERT INTO collection (name, userid) VALUES (sad, 10)
- INSERT INTO collection (name, userid) VALUES (test999, 1)
- INSERT INTO genre (genretype) VALUES ("Science Fiction")
- INSERT INTO genre (genretype) VALUES ("Horror")
- INSERT INTO movies (movid, title, agerating, length) VALUES (63, Shrek, PG, 90)
- INSERT INTO movies (movid, title, agerating, length) VALUES (17, Interstellar, PG-13, 169)
- INSERT INTO movieshasgenre (movid, genreid) VALUES (1, 20)
- INSERT INTO movieshasgenre (movid, genreid) VALUES (9, 27)
- INSERT INTO collectionofmovies (name, movid) VALUES (sad, 17)
- INSERT INTO collectionofmovies (name, movid) VALUES (sad, 63)
- INSERT INTO record (userid, movid, timestamp) VALUES (1, 63, datetime.now())
- INSERT INTO record (userid, movid, timestamp) VALUES (10, 17, datetime.now())
- INSERT INTO rating (userid, movid, userrating) VALUES (1, 63, 3)
- INSERT INTO rating (userid, movid, userrating) VALUES (10, 18, 5)
- INSERT INTO acts_in (movid, id) VALUES (38, 455)
- INSERT INTO acts_in (movid, id) VALUES (96, 463)
- INSERT INTO contributors (id, name) VALUES (10, "Morgan Freeman")
- INSERT INTO contributors (id, name) VALUES (90, "Robert Downey Jr.")

- INSERT INTO produces (movid, id) VALUES (5, 262)
- INSERT INTO produces (movid, id) VALUES (59, 206)
- INSERT INTO platform (platid, platformtype) VALUES (1, ”Netflix”)
- INSERT INTO platform (platid, platformtype) VALUES (2, ”Theaters”)

Well, for data analysis, we went to DataGrip and either exported the data in each table or manually typed each value into Google Sheets before making them into graphs. For indexing, we used a total of 7 indexes, not including the auto-generated ones to use primary keys. The purpose of. all the indexes are primarily focused on reducing data redundancy by making specific variables unique.

- For the table collection, to boost performance, we added one index that prevented duplicate collection names, which would prevent data redundancy.
- For the table contributors, to boost performance, we added one index that prevented duplicate names for contributors as contributors is a table with names for actors, directors, and producers totaling over 500 different names.
- For the table friend, the table for who follows who, to prevent duplicate follows an index was added: i.e., the same user can not follow a different user twice.
- For the table genre, to prevent duplicate genre titles, an index on the genre type was added.
- For the table movies, to prevent duplicate movie titles, an index on the movie title was added.
- For the table users, to boost performance indexes were added, one prevents duplicate usernames, and the other prevents duplicate emails.

Appendix: SQL Queries

Finally, add an appendix of all the SQL statements created in your application during Phase 4.

```sql
SELECT userid1
FROM friend
WHERE userid2 = %s;
```

Listing 1: SQL query to retrieve friends of a user

```sql
SELECT userid2
FROM friend
WHERE userid1 = %s;
```

Listing 2: SQL query to who the user follows

```sql
SELECT m.title, r.userrating
FROM movies m
JOIN rating r ON m.movID = r.movID
WHERE r.userid = %s
ORDER BY r.userrating DESC
LIMIT 10;
```

Listing 3: SQL query to get top ten movies by rating

```sql
SELECT m.title, COUNT(r.movid) AS views
FROM movies m
JOIN record r ON m.movID = r.movID
WHERE r.userid = %s
GROUP BY m.title
ORDER BY views DESC
LIMIT 10;
```

Listing 4: SQL query to get top ten movies by most viewed

```sql
SELECT m.title, COUNT(r.movID) AS play_count
FROM movies m
JOIN record r ON m.movID = r.movID
WHERE r.timestamp >= NOW() - INTERVAL '90 days'
GROUP BY m.title
ORDER BY play_count DESC
LIMIT 20;
```

Listing 5: SQL query to get top 20 movies in the last 90 days

```sql
SELECT m.title, COUNT(r.movID) AS play_count
FROM movies m
JOIN record r ON m.movID = r.movID
JOIN friend f ON r.userid = f.userid2
WHERE f.userid1 = %s
GROUP BY m.title
ORDER BY play_count DESC
LIMIT 20;
```

Listing 6: SQL query to get top twenty movies among the user's followers

```sql
SELECT m.title
FROM movies m
JOIN releasing r ON m.movID = r.movID
```

```sql
WHERE EXTRACT(MONTH FROM r.releasedate) = %s
AND EXTRACT(YEAR FROM r.releasedate) = %s
ORDER BY r.releasedate DESC
LIMIT 5;
```

Listing 7: SQL query to get top five movies in this month

```sql
SELECT g.genretype, gm.genreid
FROM movies m
JOIN movieshasgenre gm ON m.movid = gm.movID
JOIN record r ON m.movID = r.movID
JOIN genre g ON gm.genreid = g.genreid
WHERE r.userid = %s
GROUP BY g.genretype, gm.genreid
ORDER BY COUNT(gm.genreid) DESC
LIMIT 1;
```

Listing 8: SQL query to get the user's favorite genre

```sql
SELECT c.name AS actor_name, a.id AS actor_id
FROM movies m
JOIN record r ON m.movID = r.movID
JOIN acts_in a ON m.movid = a.movid
JOIN contributors c ON a.id = c.id
WHERE r.userid = %s
GROUP BY actor_name, actor_id
ORDER BY COUNT(r.movID) DESC
LIMIT 1;
```

Listing 9: SQL query to get the user's favorite actor

```sql
SELECT DISTINCT m.title
FROM movies m
JOIN record r ON m.movID = r.movID
JOIN movieshasgenre gm ON m.movid = gm.movid
JOIN acts_in a ON m.movid = a.movid
WHERE r.timestamp >= NOW() - INTERVAL '90 days' AND (gm.
    genreid = %s OR a.id = %s)
GROUP BY m.title
ORDER BY m.title DESC
LIMIT 10;
```

Listing 10: SQL query to get recommendations based on favorite genre and actor of user + recently top viewed movies

4 Data Analysis

4.1 Hypothesis

Our objective for the data analysis is to better suit our users' preferences. We aim to identify the most-watched genre that also receives the highest ratings. This data would enable us to expand our database with more movies of that genre and provide users with greater choices. We also leverage data, such as the movie release dates to predict the likelihood of a being well-received or ultimately neglected.

4.2 Data Preprocessing

Data preprocessing involved several steps to ensure the data was clean and formatted correctly for analysis:

Data Cleaning: We addressed missing values by using interpolation where feasible, and removed outliers based on z-score analysis to ensure they did not skew our results. Data Formatting: We standardized the format for dates and resolved inconsistencies in genre naming (e.g., "SciFi" to "Sci-Fi") across our dataset. Data Combination: We combined data from multiple tables using SQL JOIN operations to create a comprehensive view necessary for our analysis, such as combining user ratings with movie genres. Data Enrichment: We added calculated fields such as "Month of Release" to assist in the analysis of seasonal trends in movie releases. The data extraction involved complex SQL queries and views to pull data from our relational database, focusing on aggregating user behavior and movie details.

4.3 Data Analytics & Visualization

We used R and Excel for data analytics:

R: For statistical analysis, including calculating mean, median, and standard deviation of user ratings and directorial movie counts. We utilized packages such as ggplot2 for visualization.

Excel: To generate pie charts and bar graphs that display the distribution of movies across platforms and weekdays, enhancing visual understanding. Visualizations highlighted the dominance of theatrical releases, the popularity of Fridays for movie releases, and the spread of user ratings across genres.

4.4 Conclusions

Our comprehensive data analysis yielded several critical insights that both confirmed our initial hypotheses and offered surprising discoveries:

1. **Platform Distribution**: As hypothesized, theatrical releases accounted for the largest share of movie distributions. However, the margin between theatrical and DVD releases was narrower than expected, at just around 2.9%. This finding suggests a still-robust consumer preference for physical

media, which may influence our strategy for recommending new releases or classics available on DVD.

2. **Weekday Popularity for Releases**: Our analysis confirmed Friday as the most popular day for movie releases, in line with traditional cinema release strategies aimed at maximizing weekend box office earnings. Interestingly, the low number of releases on Saturdays and the minimal activity in midweek days, particularly Wednesday, suggest a potential untapped market for midweek premieres. This could be explored further to determine if there's a niche audience that prefers less crowded viewing times.

3. **User Ratings and Genre Preferences**: The variance in user ratings across genres highlighted the diverse tastes of our user base. This variability underscores the need for a sophisticated recommendation engine that not only considers the highest-rated genres but also personalizes suggestions based on individual user preferences and viewing history.

4. **Analytical Insights for Recommendation System**:
   - The confirmation that Fridays are the most popular for movie releases could be used to time our recommendations, promoting new releases at the end of the week when users are most likely to plan their movie watching.
   - The detailed genre analysis, particularly the exploration of this year's genre ratings, could refine our recommendation algorithms to highlight genres that are trending upward in user satisfaction.

5. **Strategic Implications**: These insights have strategic implications for our movie recommendation platform. Understanding the platform distribution can help us tailor our content library and recommendations to include a balanced mix of theatrical, DVD, and streaming content. Furthermore, knowing the most popular release days can guide our promotional activities.

6. **Future Directions**: Moving forward, we recommend:
   - Further investigation into the potential for midweek movie releases to attract audiences looking for a different viewing experience.
   - Continued monitoring of genre popularity and user ratings to adapt our recommendation algorithms as trends shift over time.
   - Exploration of user segmentation by preferences to further personalize movie recommendations and marketing strategies.

Overall, our data analysis not only reinforced the importance of tailored content and strategic release timing in the entertainment industry but also highlighted areas for innovation in how we engage with and satisfy our diverse user base. These findings will guide the future enhancements of our platform, ensuring it remains responsive to the evolving preferences and behaviors of movie enthusiasts.

5 Lessons Learned

Challenges encountered and Lessons Learned:
For Phase 1, we encountered minimal challenges, primarily focused on determining the cardinalities for specific tables. We got points off the first phase due to the fact we forgot to acknowledge that a contributor can be only an actor or director and does not need to be both, so it should be a 0 to many relationships; however, we inputted in the diagram a 1 to many cardinality. Another challenge we faced was our approach to adding friends. Initially, we considered a simple recursive relationship with users but later changed it to include a unique identifier for better structurability and searchability. These challenges allowed us to understand the difficulty in implementing a schema for a database whose implementations other than the required functions were solely up to our decision.
During Phase 2, the majority of our challenges revolved around implementing the API effectively. A recurring challenge we faced was the significant time spent on bug testing to ensure all program functionalities worked. This effort was crucial in ensuring the program's correct execution under any circumstances. Another challenge we faced in API implementation was ensuring the SQL queries used to manipulate the database would work efficiently. This problem was solved relatively quickly when we noticed a console in Datagrip that would allow us to test the queries before implementing them into the API. The challenges encountered during Phase 2 highlighted the importance of thorough pre-implementation and bug testing of code and queries. Testing these components beforehand would have facilitated quicker fixes than addressing the errors and bugs after integration into the API.
For Phase 3, compared to Phase 2, the API implementation of the new functions was relatively simple and did not cause any major problems. For the data analysis portion of the phase, we had to deal with inconsistent data formats, which required extensive preprocessing that would standardize the data for proper analysis. The SQL queries used to extract the complex data also required multiple revisions and testing before use. We learned the importance of thorough data preprocessing to ensure reliable outcomes and gained proficiency in SQL and R for data manipulation and analysis, enhancing our technical skills.

6 Resources

Include in this section the resources you have used in your project beyond the normal code development such as data sets or data analytic tools (i.e. Weka, R).

- Data was primarily randomized by ChatGPT.
- Google Sheets and Excel was used to hold the data and make some of the graphs.
- R was used for exploratory analysis.