Lab 14 – Course Final Project Report

Project Members: Kavya Moharana (kavyam3), Mahnur Khalid (mkhal2)

Part 1: *Purpose and Scope*

Our project consists of creating a IMDb-style movie reviews and ratings database: a platform for users to be able to access basic movie information such as cast and crew information, genre, and roles while also writing reviews, ratings, and interacting with other users' posts.

Our **Primary Business Objectives** would be to:

- improve marketing and industry perspectives by providing insight into what kinds of movies and media resonate the most with audiences and the general public
- streamline operations for studios and filmmakers by providing a centralized platform for movie data and analysis
- enhance the movie experience for users with up-to-date reviews, ratings, and information on trailers, streaming services, etc.
- encourage data-driven decision making in the film industry

Our **Stakeholder Analysis** consists of:

- End-Users:
  - Movie-goers and film enthusiasts are the primary consumers and users of a database containing movie features or information, reviews, and ratings
  - Film critics and journalists are also amongst the primary users of such a database
  - Studios and filmmakers would utilize the database for tracking movie performance, overall market research, and understanding audience preferences and reactions
- External Partners: Movie advertising and marketing teams may use audience trends to shape their targeted promotion plan
- Management: Database administrators who are responsible for maintaining and improving the database
- IT Staff: Database developers

In terms of **User Requirements**, the primary users of this database platform would want more detailed movie information such as plot summaries, cast and crew, release dates, and promotional content. Along with submitting their own reviews and ratings, primary users would expect to be able to search for movies based on features like genre, release year, actors, director, and ratings. Studios and their marketing teams expect to have the ability to track the performance of their film projects. Therefore, they would want access to market insights and audience trends for decision-making.

The **Functional Scope** of this project encompasses:

- Information Retrieval of Films: Users can access overall movie details, such as cast and crew information, genre details, roles, names, and ratings in an organized way
- Recommendation System: Users are recommended films based on preferences
    - Create an inventory that is useful for consumers to reference when seeking a movie recommendation
    - Users can associate a recommended movie with their designated director and other identifying information
- User Reviews and Interaction: Users have easy access to user feedback and reviews and can interact with other users' posts as well

The **Data Scope** of this project comprises:

- Movie Data: Captures identifying information about the movie such as the title, year, release date, duration, cast, director, etc.
- User Data: Identifies user information such as name, age, etc.
- Ratings Data: Organzies rating data which includes the movie id, average, ratings, total votes, and median ratings
- Genre data: Stores general information about the movie genre
- Director Mapping: Identifies the director and the movie they are associated with
- Role Mapping: Identifies the movie id, name id, and category within the role mapping

Part 2: *Entity-Relationship Diagram*

Key Entities of the database –

- Movie
- Genre
- Ratings
- Directors
- Actors
- Roles

Attributes for each Entity –

- Movie:
    - id, title, year, release_date, duration, country, worldwide_gross_income, languages, production_company
- Genre:
    - movie_id, genre_name

- Ratings:
    - movie_id, avg_rating, total_votes, median_rating
- Directors:
    - movie_id, name_id
- Actors:
    - id, name, height, date_of_birth, movies_known_for
- Roles:
    - movie_id, name_id, role_category

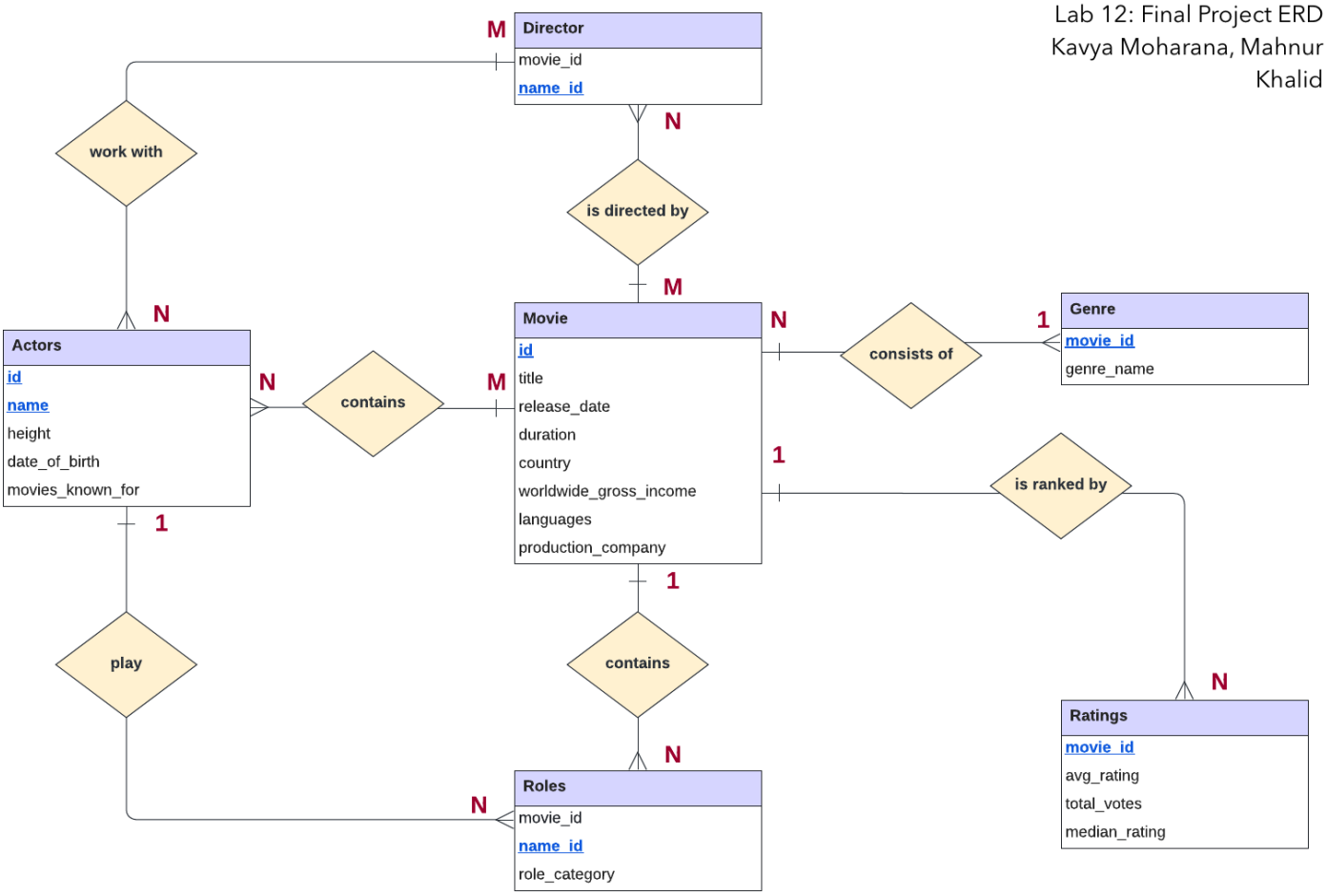
Primary Keys for each Entity –

- Movie: id
- Genre: movie_id
- Ratings: movie_id
- Directors: name_id
- Actors: name, id
- Roles: name_id

Relationships:

1. Genre consists of Movie (one-to-many)
    - One genre can consist of many movies, but one movie can only be a part of one genre
2. A Movie is ranked by Ratings (one-to-many)
    - A movie can be ranked with many ratings, but one rating can only belong to one movie
3. A Movie is directed by Directors (many-to-many)
    - A movie can be directed by many directors, and one director can direct many movies
4. Directors work with Actors (many-to-many)
    - A director can work with many actors, and one actor can work under many directors
5. Roles are played by Actors (one-to-many)
    - A role can be played by one actor, but one actor can play many roles
6. A Movie contains roles (one-to-many)
    - A movie contains many roles, but one role can only be a part of one movies
7. Movies contain many actors (many-to-many)
    - A movie contains many actors and one actor can be in many movies

Lab 12: Final Project ERD
Kavya Moharana, Mahnur Khalid



Part 3: *Database Schema*

Tables Defined in SQL:

- **Movies**:
  - id (Primary Key)
  - title
  - year
  - release_date
  - duration
  - country
  - worldwide_gross_income
  - languages
  - production_company
- **Genre**:
  - movie_id (Primary Key)
  - genre_name

- **Ratings**:
    - movie_id (Primary Key)
    - avg_rating
    - total_votes
    - median_rating
- **Directors**:
    - movie_id
    - name_id (Primary Key)
- **Actors**:
    - id (Primary Key)
    - name
    - height
    - date_of_birth
    - movies_known_for
- **Roles**:
    - movie_id
    - name_id (Primary Key)
    - Role_category
- **Directed By (Relationship between Director and Movie):**
    - name_id (Primary Key)
    - id (Foreign Key in 'Movies')
    - name_id (Foreign Key in 'Directors')
- **Works With (Relationship between Actors and Directors):**
    - name_id (Primary Key)
        - ➢ name_id (Foreign Key in 'Actors')
        - ➢ name_id (Foreign Key in 'Directors')
    - **Contains (Relationship between Movie and Actors):**
        - ➢ id (Primary Key)
        - ➢ id (Foreign Key in 'Actors')
        - ➢ id (Foreign Key in 'Movies')

Primary Keys: Used as unique identification for each movie, actor, and director in the database and their perspective ratings, genre, and roles.

Foreign Keys: Represent the established relationships (many to many) between a movie and its director(s) and actors. For example, 'id' in the 'Movies' tables corresponds to 'movie_id' in the 'Directors' table, which links a Movie to its director(s).

Revisions Made: Revisions were made to the relationships, specifically for entities that shared many to many relationships. For many to many cases, we created a relationship query that referenced the common attributes that were needed to link two distinct entities together. In the revised version, we created a separate query that made many to many relationships their own

table, which allows for the proper connection between various entities and better represents their relationship.

Movie Table

```sql
CREATE TABLE Movie(id INT PRIMARY KEY,
                   title varchar(100),
                   year INT,
                   release_date varchar(100),
                   duration INT,
                   country varchar(100),
                   worldwide_gross_income INT,
                   languages varchar(100),
                   production_company varchar(100)
                  );
```

Genre Table:

```sql
CREATE TABLE Genre(movie_id varchar(100) PRIMARY KEY,
                   genre_name varchar(100)
                  );
```

Ratings Table:

```sql
CREATE TABLE Ratings(movie_id varchar(100) PRIMARY KEY,
                     avg_rating INT,
                     total_votes INT,
                     median_rating varchar(100)
                    );
```

Roles Table:

```sql
CREATE TABLE Roles(movie_id varchar(100),
                   name_id varchar(100) PRIMARY KEY,
                   role_category varchar(100)
                  );
```

Director Table

SQLite

```sql
CREATE TABLE Directors(movie_id INT,
                       name_id varchar(100) PRIMARY KEY
                      );

```

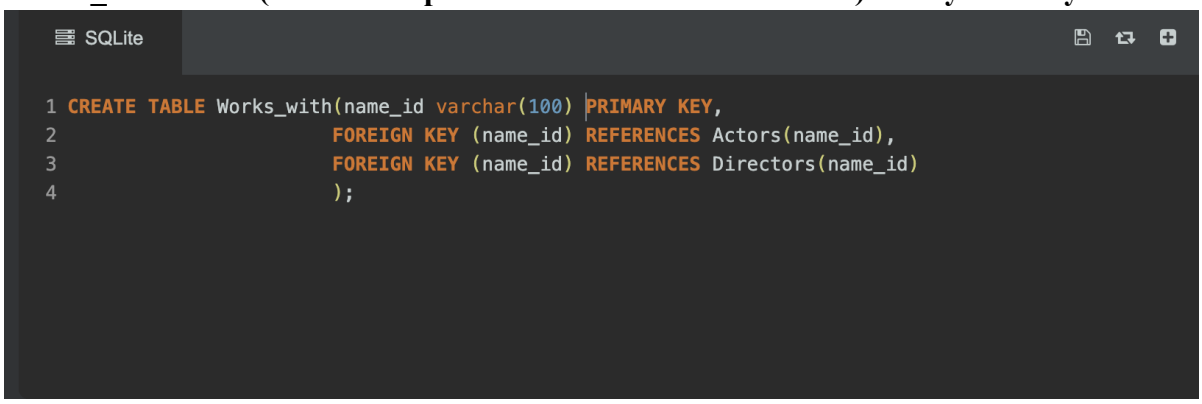
Directed Table (Relationship between Director and Movie): many to many

```sql
CREATE TABLE Directed_by(name_id PRIMARY KEY,
                         FOREIGN KEY (name_id) REFERENCES Movie(id),
                         FOREIGN KEY (name_id) REFERENCES Directors(name_id)
                        );
```

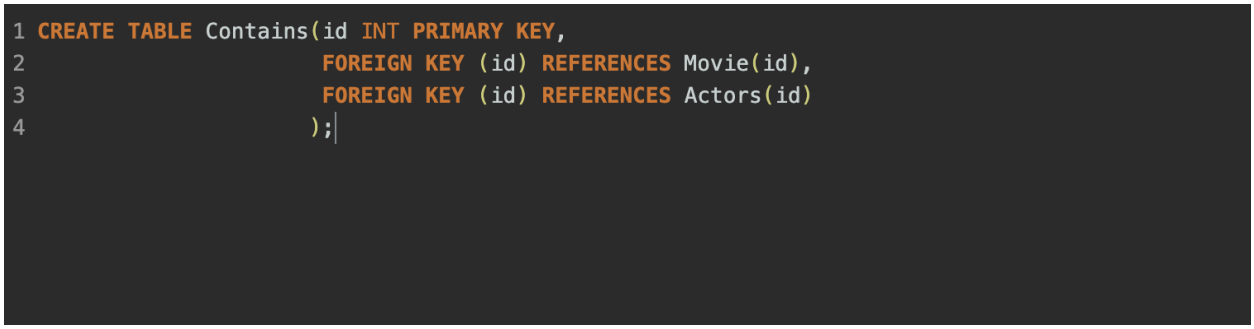
Actors Table:

```sql
CREATE TABLE Actors(id INT PRIMARY KEY,
                    name_id varchar(100),
                    height INT,
                    date_of_birth varchar(100),
                    movies_known_for varchar(100)
                   );
```

Works_with Table (Relationship between Actors and Directors): many to many



Contains Table (Relationship between Movie and Actors): many to many



Part 4: *Database Functionality and SQL Queries*

Query 1: Provide the top 5 movies that have the highest worldwide gross income.

```
SELECT title, worldwide_gross_income FROM Movie ORDER BY
worldwide_gross_income DESC LIMIT 5;
```

a. The functionality of this query demonstrates that it allows the user to understand the highest worldwide gross incomes of various movies. This gives us insight into the success of the top 5 movies and gives us an opportunity to possibly further investigate why specific movies have such high worldwide popularity, based on their financial success.

b. The results are relevant to the overall database schema because it allows us to break down the magnitude of these various movie successes, by looking at their profits and financial successes. This information can help movie production and directors make better decisions when creating movies by running their own analysis on why these 5 specific movies had such high worldwide financial success and use those attributes to make changes in their films in the future.

```
SELECT title, worldwide_gross_income FROM Movie
    ORDER BY worldwide_gross_income DESC LIMIT 5
```

title	worldwide_gross_income
Star Wars: The Force Awakens	2071310218
Barbie	1441807871
Black Panther	1349926083
Oppenheimer	951043060
La La Land	471988025

Query 2: For the movies that contain languages Korean or Mandarin, find the average movie ratings

```
SELECT Movie.languages, Movie.id, Ratings.avg_rating
FROM Movie JOIN Ratings ON Movie.id = Ratings.movie_id
WHERE Movie.languages LIKE '%Korean%' OR Movie.languages LIKE '%Mandarin%'
GROUP BY Movie.languages, Movie.id
ORDER BY avg_rating DESC;
```

a. The functionality of this query demonstrates multiple insights about the database. First, it allows us to understand the overall performance and ratings of movies produced in Korean and Mandarin. Beyond this, we are able to understand the popularity of the languages amongst other relevant languages and see the distribution of the ratings.

b. The results of running this query are relevant to the overall database schema because it shows us how a movie's language details might impact worldwide performance and audience opinions. This information can help guide movie productions by showing the popularity of movies within certain demographics. It can also help users when they are looking for a movie in either Korean or Mandarin and their understanding of average ratings.

```
SELECT Movie.languages, Movie.id, Ratings.avg_rating
    FROM Movie JOIN Ratings ON Movie.id = Ratings.movie_id
    WHERE Movie.languages LIKE '%Korean%'
        OR Movie.languages LIKE '%Mandarin%'
    GROUP BY Movie.languages, Movie.id
    ORDER BY avg_rating DESC;
```

languages	id	avg_rating
Korean, English	5	8.5
English, Mandarin	4	8
English, Mandarin, Cantonese	3	7.8
English, Swahili, Nama, Xhosa, Ko…	6	7.3

Query 3: For each genre, determine the average ratings

```
SELECT Genre.genre_name, AVG(Ratings.avg_rating)
AS average_rating
FROM Genre JOIN Movie ON Genre.movie_id = Movie.id
JOIN Ratings ON Movie.id = Ratings.movie_id
GROUP BY Genre.genre_name;
```

a. The functionality of this query overall gives insight into the average performance of various movies, aggregating by their genres. This can be helpful in predictive modeling in the future and is relevant to the database as it allows us to see the differences in the average rating, while focusing specifically on their genres.

b. The results of running this query are relevant to the overall database schema because it shows users which distinct genres have higher rated movies. It can help studios and filmmakers navigate which specific types of movies are currently performing well with audiences.

```
SELECT Genre.genre_name, AVG(Ratings.avg_rating) AS average_rating
   FROM Genre JOIN Movie ON Genre.movie_id = Movie.id
   JOIN Ratings ON Movie.id = Ratings.movie_id
   GROUP BY Genre.genre_name
```

genre_name	average_rating
Action	7.85
Biography	8.4
Fantasy	7
Horror	7.8
Romance	8
Sci-Fi	7.866666666666667
Thriller	8.5

Query 4: Provide the name and release data of the movie with the lowest total votes

```
SELECT Movie.title, Movie.release_date, MIN(Ratings.total_votes)
AS lowest_votes
FROM Movie JOIN Ratings ON Movie.id = Ratings.movie_id;
```

a. The functionality of their query allows us to view what movie had the lowest votes on IMDB and when it was released. Running this query allows us the opportunity to further investigate if the release data of these movies might be relevant to their low total votes and what can be done to improve these trends in the future.

b. The results of running this query are relevant to the overall database schema because it gives a perspective into viewers' reactions to this specific movie and allows us to investigate whether other attributes of this movie influenced its low performance and total votes.

```
SELECT Movie.title, Movie.release_date, MIN(Ratings.total_votes) AS lowest_votes
   FROM Movie JOIN Ratings ON Movie.id = Ratings.movie_id
```

title	release_date	lowest_votes
Barbie	07-21-2023	402495