**CAPITAL UNIVERSITY OF SCIENCE AND TECHNOLOGY**

**DATABASE PROJECT**

**MOVIE RECOMMENDATION SYSTEM**

**GROUP MEMBERS**

|  |  |
| --- | --- |
| **REGNO** | **NAME** |
| BSE233073 | ABDUL WASEH |
| BSE233118 | SAAD IMRAN |
| BSE233069 | ZOHAIB FAYYAZ |

**DATABASE PROJECT**

**MOVIE RECOMMENDATION SYSTEM**

* **Description of the entities, attributes, and relationships**

In a **Movie Recommendation System**, there are several main entities, each with specific information (attributes) associated with them. The relationships between these entities help the system function correctly by organizing how they connect to each other.

#### ****Entities & Attributes:****

**1.User** (The person using the system)

* 1. user\_id: A unique identifier for each user (Primary Key).
  2. username: The user's chosen name.
  3. email: The user’s email address.
  4. created\_at: The date the user registered.

**2.Movie** (A film in the system)

* 1. movie\_id: A unique identifier for each movie (Primary Key).
  2. title: The name of the movie.
  3. release\_year: The year the movie was released.
  4. description: A brief summary of the movie.
  5. duration: How long the movie is (in minutes).
  6. genre\_id: The category of the movie (e.g., Action, Comedy).

**3.Genre** (The type or category of the movie, such as Comedy, Drama, etc.)

* 1. genre\_id: A unique identifier for each genre (Primary Key).
  2. genre\_name: The name of the genre (e.g., Comedy, Action).

**4.Rating** (The rating given by a user for a movie)

* 1. rating\_id: A unique identifier for each rating (Primary Key).
  2. user\_id: The ID of the user who gave the rating (Foreign Key to User).
  3. movie\_id: The ID of the movie being rated (Foreign Key to Movie).
  4. rating\_value: The rating value (usually 1 to 5 stars).
  5. rating\_date: The date when the rating was given.

**5.ViewingHistory** (Records of movies watched by users)

* 1. history\_id: A unique identifier for each viewing record (Primary Key).
  2. user\_id: The ID of the user who watched the movie (Foreign Key to User).
  3. movie\_id: The ID of the movie being watched (Foreign Key to Movie).
  4. view\_date: The date when the movie was watched.

**6.Recommendation** (Movies recommended to a user based on their preferences)

* 1. recommendation\_id: A unique identifier for each recommendation (Primary Key).
  2. user\_id: The ID of the user who is being recommended the movie (Foreign Key to User).
  3. movie\_id: The ID of the recommended movie (Foreign Key to Movie).
  4. recommendation\_score: A score (e.g., 1 to 10) showing how strongly the movie is recommended to the user.
  5. created\_at: The date when the recommendation was made.

#### ****Relationships:****

* **User ↔ Rating:**

A user can rate many movies, and each movie can be rated by many users (Many-to-Many). This is managed using the **Rating** table.

* **User ↔ ViewingHistory:**

A user can watch many movies, and each movie can be watched by many users (Many-to-Many). This is managed using the **ViewingHistory** table.

* **User ↔ Recommendation:**

A user can receive many movie recommendations (One-to-Many).

* **Movie ↔ Genre:**

A movie can have one genre, but each genre can have many movies (One-to-Many).

* **Movie ↔ Rating:**

A movie can have many ratings (One-to-Many).

* **Movie ↔ ViewingHistory:**

A movie can be watched by many users (One-to-Many)

### ****Challenges or Considerations During the Design Process****

* **Data Redundancy:**

A big concern was avoiding repeating information across different tables. For example, movies should not have their genre information repeated every time, so genres are stored in a separate table.

* **Recommendation Accuracy:**

The core purpose of this system is to provide personalized recommendations. Making sure that recommendations are based on a user's past behavior (ratings and viewing history) requires complex algorithms. We store all the user interactions, but the actual recommendation system will work through algorithms that are applied later, outside of the database.

* **Scalability:**

The system needs to handle a growing number of users, movies, and ratings without slowing down. Efficient indexing (like indexing on user\_id, movie\_id, etc.) will help speed up queries.

* **User Privacy:**

We need to ensure that user data (such as emails and viewing history) is kept private and secure. This means implementing encryption and secure data storage.

* **Many-to-Many Relationships:**

Handling relationships where users can rate many movies, and each movie can be rated by many users, was complex but necessary for storing all the ratings data.

* **Define tables, fields, and data types**

### ****1. User Table****



### ****2. Movie Table****

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Data Type** | **Description** |
| movie\_id | INT (Primary Key) | Unique identifier for each movie. |
| title | VARCHAR(100) | The title of the movie. |
| release\_year | YEAR | The year the movie was released. |
| description | TEXT | A brief description of the movie. |
| duration | INT | Duration of the movie in minutes. |
| genre\_id | INT (Foreign Key) | The genre of the movie, linking to the Genre table. |

### ****3. Genre Table****



### ****4. Rating Table****



### ****5. ViewingHistory Table****



### ****6.Recommendation Table****

### 

### ****Relationships Between Tables****

**User ↔ Rating**: A **User** can rate many **Movies**. Each **Movie** can be rated by many **Users**. The **Rating** table links them through **user\_id** and **movie\_id**.

**User ↔ ViewingHistory**: A **User** can watch many **Movies**. Each **Movie** can be watched by many **Users**. The **ViewingHistory** table links them through **user\_id** and **movie\_id**.

**Movie ↔ Genre**: Each **Movie** is categorized by a **Genre**. A **Genre** can have many **Movies**. The **Genre** table is linked to the **Movie** table through **genre\_id**.

**User ↔ Recommendation**: A **User** can receive many **Recommendations**, but each **Recommendation** belongs to only one **User**. Each **Movie** can be recommended to many **Users**. The **Recommendation** table links them through **user\_id** and **movie\_id**.

### ****Establish Relationships Between Tables****

The relationships between tables are established by using **Foreign Keys**:

* **Users ↔ Ratings:**

A user can rate many movies. This relationship is managed through the Ratings table.

* **Movies ↔ Ratings:**

A movie can have many ratings. This relationship is managed through the Ratings table.

* **Users ↔ ViewingHistory:**

A user can watch many movies. This relationship is managed through the ViewingHistory table.

* **Movies ↔ ViewingHistory:**

A movie can be watched by many users. This relationship is managed through the ViewingHistory table.

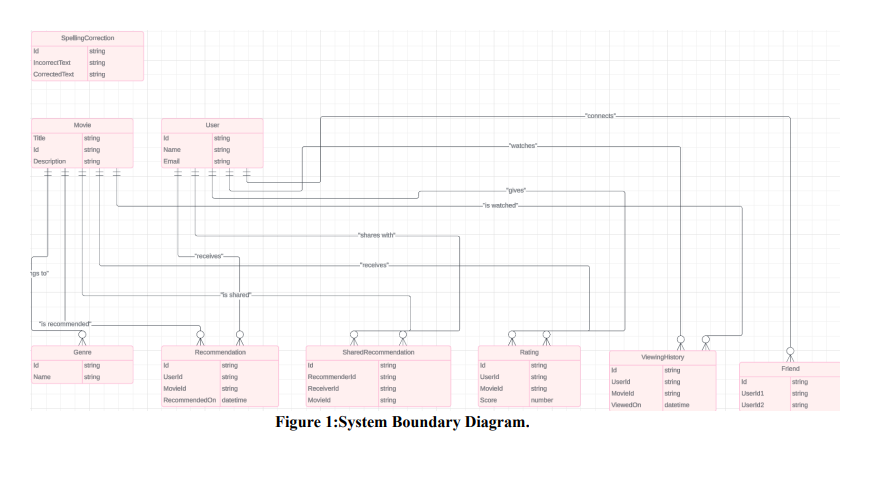
* **Users ↔ Recommendations:**

A user can receive many recommendations. This relationship is managed through the Recommendations table.

* **Movies ↔ Recommendations:**

A movie can be recommended to many users. This relationship is managed through the Recommendations table.

* **Transform ERD into relational data model (RDM).**



**Relational model**

**USER**



**MOVIE**



**RECOMMENDATION**



**SHARED RECOMMENDTION**



**VIEWING**



**RATING TABLE**



**SPELLING CORRECTION**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraint** |
| id | INT | PRIMARY KEY |
| incorrectText | TEXT | NOT NULL |
| correctedText | TEXT | NOT NULL |

### ****Movie Recommendation System: Report****

#### ****Introduction****

The **Movie Recommendation System** is designed to help users discover movies tailored to their preferences. With the large volume of content available on streaming platforms, users often find it difficult to choose what to watch. The system uses user data (such as past viewing history and ratings) to recommend movies that align with their tastes, improving the overall user experience.

This report describes the core elements of the database used in the **Movie Recommendation System**, focusing on the entities, their attributes, and the relationships that ensure the system functions efficiently.

### ****System Overview****

The **Movie Recommendation System** is built around several key functionalities:

* **User Profiles:** The system stores user information, such as personal preferences and interaction history.
* **Movie Ratings:** Users can rate movies, which directly influences future recommendations.
* **Personalized Recommendations:** Based on user behavior, the system generates movie suggestions.
* **Genre Classification:** Movies are categorized by genre (e.g., action, comedy, drama) to facilitate genre-based searches.
* **Dynamic Suggestions:** As users interact more with the system (watching movies, giving ratings), recommendations are continuously updated.

The core data is organized into tables that store information about users, movies, ratings, genres, viewing history, and recommendations.

### ****Entities and Their Descriptions****

#### ****1. User Table****

The **User** table holds information about the people using the system. Each user has a unique profile in the system, which includes:

* **user\_id**: A unique identifier for each user (primary key).
* **username**: The chosen username for the user.
* **email**: The user's email address (used for login and communication).
* **created\_at**: The timestamp when the user created their account.

This table is essential for storing and managing user-specific data. It supports personalization by linking users to their ratings, viewing history, and movie recommendations.

#### ****2. Movie Table****

The **Movie** table stores details about the movies available in the recommendation system. It includes:

* **movie\_id**: A unique identifier for each movie (primary key).
* **title**: The title of the movie.
* **release\_year**: The year the movie was released.
* **description**: A brief summary of the movie.
* **duration**: The length of the movie in minutes.
* **genre\_id**: A foreign key linking each movie to a specific genre from the **Genre** table.

This table is crucial for storing movie-specific details that are needed to generate recommendations. It allows users to search for movies based on their title, genre, and other attributes.

#### ****3. Genre Table****

The **Genre** table categorizes movies into various genres. Each genre helps to classify movies so users can filter recommendations based on their preferences (e.g., comedy, action, romance). It includes:

* **genre\_id**: A unique identifier for each genre (primary key).
* **genre\_name**: The name of the genre (e.g., Action, Comedy, Drama).

The genre table is important for organizing movies into categories, enabling users to find movies by genre or to recommend new movies within a specific genre.

#### ****4. Rating Table****

The **Rating** table stores the ratings that users give to movies. It enables the system to understand user preferences and adjust future recommendations accordingly. The table includes:

* **rating\_id**: A unique identifier for each rating (primary key).
* **user\_id**: A foreign key linking the rating to a specific user (from the **User** table).
* **movie\_id**: A foreign key linking the rating to a specific movie (from the **Movie** table).
* **rating\_value**: The numerical rating given to the movie (e.g., a score from 1 to 5 stars).
* **rating\_date**: The date and time the rating was given.

Ratings help shape the personalized recommendations the system provides to each user. By analyzing ratings, the system learns the user's tastes and preferences, improving the quality of future suggestions.

#### ****5. ViewingHistory Table****

The **ViewingHistory** table records which movies each user has watched. This information is essential for generating accurate recommendations, as it provides context to the system about the user’s movie-watching habits. It includes:

* **history\_id**: A unique identifier for each record in the history (primary key).
* **user\_id**: A foreign key linking the viewing history to a specific user (from the **User** table).
* **movie\_id**: A foreign key linking the viewing history to a specific movie (from the **Movie** table).
* **view\_date**: The date and time the movie was watched.

By tracking what movies users have watched, the system can recommend similar or related movies based on viewing history. It also helps the system understand the user's changing preferences over time.

#### ****6. Recommendation Table****

The **Recommendation** table stores personalized movie suggestions for each user. The system generates these recommendations based on user interactions, ratings, and viewing history. It includes:

* **recommendation\_id**: A unique identifier for each recommendation (primary key).
* **user\_id**: A foreign key linking the recommendation to a specific user (from the **User** table).
* **movie\_id**: A foreign key linking the recommendation to a specific movie (from the **Movie** table).
* **recommendation\_score**: A score that indicates how relevant the recommendation is to the user (e.g., on a scale from 1 to 10).
* **created\_at**: The date and time when the recommendation was generated.

The **Recommendation** table is central to the functionality of the system. It stores the actual movie suggestions provided to users and helps the system keep track of which movies have been recommended.

### ****Relationships Between the Tables****

The relationships between the tables are designed to ensure the integrity of the data and facilitate complex queries required for generating personalized recommendations.

**User ↔ Rating**:

A **User** can rate many **Movies**, and each **Movie** can be rated by many **Users**. This creates a **many-to-many** relationship between users and movies, with the **Rating** table serving as the intermediary.

**User ↔ ViewingHistory**:

A **User** can watch many **Movies**, and each **Movie** can be watched by many **Users**. This forms a **many-to-many** relationship, which is managed by the **ViewingHistory** table.

**Movie ↔ Genre**:

Each **Movie** is assigned a specific **Genre**. A **Genre** can have many **Movies**, but each **Movie** belongs to only one **Genre**. This creates a **one-to-many** relationship between genres and movies.

**User ↔ Recommendation**:

A **User** can receive many **Recommendations**, but each **Recommendation** is unique to a **User**. A **Movie** can be recommended to multiple **Users**. This forms a **one-to-many** relationship between **User** and **Recommendation**.

**Movie ↔ Rating**:

Each **Movie** can have many **Ratings** from different users, but each rating corresponds to a specific movie. This creates a **one-to-many** relationship between **Movie** and **Rating**.

### ****Challenges and Considerations****

**Data Redundancy**: One of the key challenges was avoiding redundancy in the database design. By normalizing the data (e.g., storing genres in a separate

**Genre** table instead of repeating genre information in the **Movie** table), the system ensures consistency and reduces data duplication.

**Recommendation Accuracy**: Building a recommendation engine based on ratings, viewing history, and other user behaviors is complex. The system’s effectiveness depends on how well it can track and analyze user preferences. While the database stores necessary data, the actual recommendation process involves more sophisticated algorithms outside the database.

**Performance**: As the database grows with more users and movies, ensuring optimal performance becomes crucial. Proper indexing and query optimization are important to ensure fast and efficient data retrieval, especially for generating real-time movie recommendations.

**Scalability**: The system must be scalable to handle a growing number of users, movies, ratings, and recommendations. It should be able to support the addition of new features, such as integrating machine learning algorithms for more accurate recommendations.

### ****Future Enhancements****

**Real-Time Recommendations**: Implementing real-time recommendation systems that update suggestions as users interact with the platform (e.g., after watching a movie or rating a movie) can improve the accuracy and relevance of suggestions.

**Social Features**: Allowing users to share movie recommendations with friends and see what movies their friends are watching could create a more interactive and social experience.

**Advanced Recommendation Algorithms**: Exploring machine learning models, such as collaborative filtering or hybrid recommendation systems, could improve the quality of recommendations based on user data.

**Multi-Language Support**: Expanding the system to support multiple languages would increase its usability across diverse regions.