Capital University of Science and Technology

Department of Software Engineering

SE2313 – Introduction to Database Systems

ASSIGNMENT NO.03

Group Members:

Muhammad Saad Imran (BSE233117),

Abdul Waseh (BSE233),

Zohaib Fayyaz (BSE233069).

Section: 02.

Date: December **27th** ,2024.

Submitted To: Mam Hina Rashid.

Table of Contents:

➢ Include a description of the entities, attributes, and relationships:

❖ Entities and Attributes:

1. **User:** This stores user-related data like user-id, email, preferences, username, and profile picture.
2. **Movie:** This stores movie details, including movie-id, title, release year, description, and cast.
3. **Rating:** This keeps track of ratings, linking users and movies. It includes rating-id, user-id, movie-id, rating value, and timestamp.
4. **Genres:** These tracks genres and includes genre-id, genre name, description, and popularity score.
5. **Viewing History:** This table stores information on what movies users have watched, when, on what device, and for how long.
6. **Recommendations:** This table stores the movies suggested to each user, along with the recommendation score and timestamp.

❖ Relationships:

- **User and Ratings:** One user can give ratings for many movies, and each rating is linked to a single user.
- **Movie and Ratings:** Many users can rate one movie, and each rating is tied to a single movie.
- **User and Viewing History:** A user can have many records in the viewing history, and each record is connected to a single user.
- **Movie and Viewing History:** A movie can be viewed by many users, and each viewing is logged separately.
- **User and Recommendations:** A user can get many movie recommendations.
- **Movie and Recommendations:** A movie can be recommended to multiple users.
- **User and Movie:** Many users can watch the same movie, and each user can watch many movies.
- **Movie and Genre:** Movies can belong to several genres, and genres can include many movies.
- **User and Genre:** A user can have preferences for multiple genres, and genres can be liked by many users.

➢ Discuss any challenges or considerations you faced during the design process:

- **Dynamic Availability Management**
  Designing a system capable of managing live updates for parking availability required optimizing database indexing to ensure swift access and minimal latency.

- **Historical Data Retention**
  Structuring a mechanism for archiving outdated records while retaining essential historical data presented significant challenges in balancing storage efficiency and data integrity.

- **Optimized Database Structure**
  Achieving a balance between eliminating redundancy and maintaining high performance involved careful consideration of normalization techniques alongside strategic indexing.

- **Scalability for Future Growth**
  Ensuring the database and system architecture can scale to accommodate future expansions, such as additional parking lots or user traffic, required forward-thinking design strategies.

➢ Define tables, fields, and data types:

User Table:

Attribute	Data Type	Constraints
user_id	INTEGER	PRIMARY KEY
user_name	VARCHAR(100)	NOT NULL
email	VARCHAR(100)	NOT NULL
registration_date	DATE	NOT NULL
preferences	VARCHAR(100)	NOT NULL

Movie Table:

Attribute	Data Type	Constraints
movie_id	INTEGER	PRIMARY KEY
title	VARCHAR(100)	NOT NULL
release_year	INTEGER	NOT NULL
runtime	INTEGER	NOT NULL
description	VARCHAR(100)	NOT NULL
cast	VARCHAR(100)	NOT NULL

Recommendation Table:

Attribute	Data Type	Constraints
recommendation_id	INTEGER	PRIMARY KEY
recommendation_score	INTEGER	NOT NULL
timestamp	TIME	NOT NULL
movie_id	INTEGER	FOREIGN KEY
user_id	INTEGER	FOREIGN KEY

Genre Table:

Attribute	Data Type	Constraints
genre_id	INTEGER	PRIMARY KEY
genre_name	VARCHAR(100)	NOT NULL
description	VARCHAR(100)	NOT NULL
popularity_score	INTEGER	NOT NULL
parent_genre_id	INTEGER	NOT NULL

Rating Table:

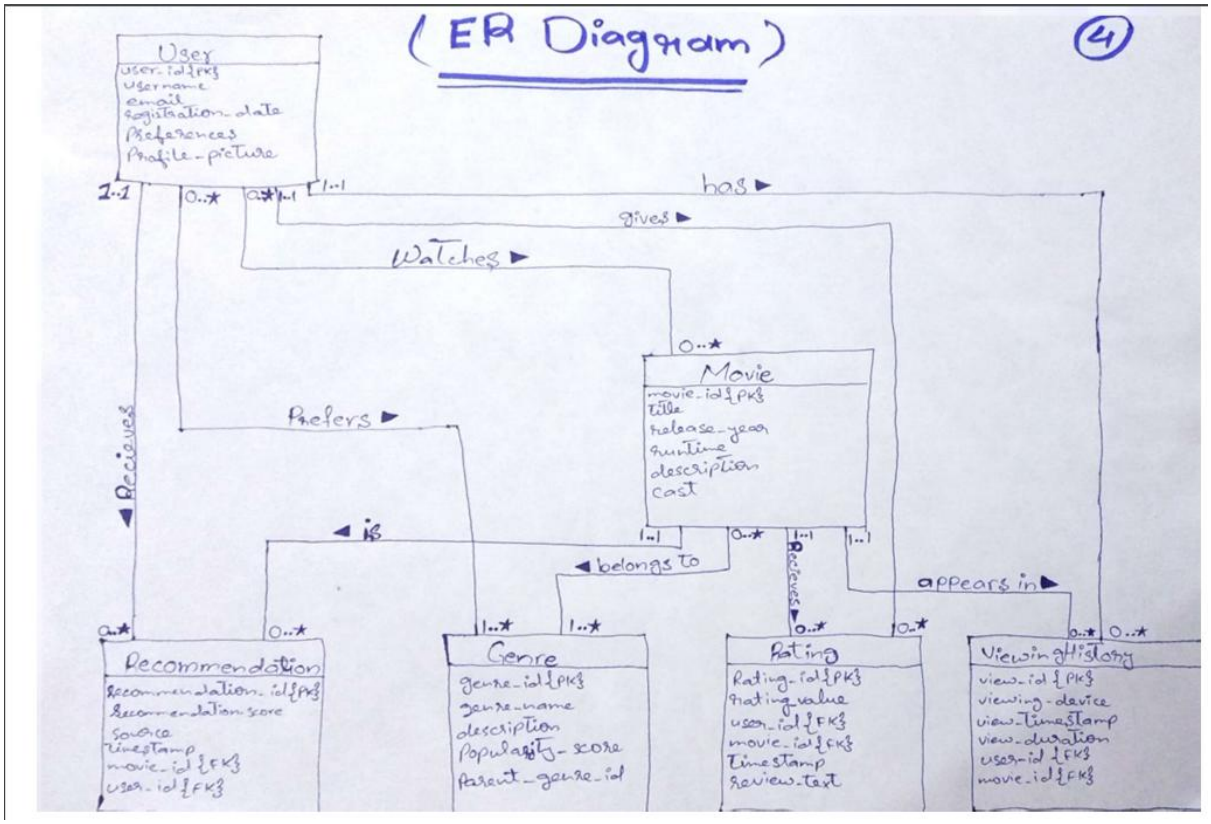
Attribute	Data Type	Constraints
rating_id	INTEGER	PRIMARY KEY
user_id	INTEGER	FOREIGN KEY
movie_id	INTEGER	FOREIGN KEY
timestamp	TIME	NOT NULL
reviews	VARCHAR(100)	NOT NULL

Viewing History Table:

Attribute	Data Type	Constraints
view_id	INTEGER	PRIMARY KEY
view_device	VARCHAR(100)	NOT NULL
view_duration	INTEGER	NOT NULL
view_timestamp	TIME	NOT NULL
user_id	INTEGER	FOREIGN KEY
movie_id	INTEGER	FOREIGN KEY

➢ Transformation of ERD into Relational Data Model (RDM):

❖ Entity Relationship Diagram:



❖ Relational Data Model (RDM):

a. User:

Attributes	Data Type	Constraints
user_id	INTEGER	PRIMARY KEY
User_name	VARCHAR(100)	NOT NULL
email	VARCHAR(100)	NOT NULL
Registration_date	DATE	NOT NULL
preferences	VARCHAR(100)	NOT NULL

b. Movie:

Attributes	Data Type	Constraints
movie_id	INTEGER	PRIMARY KEY
Title	VARCHAR(100)	NOT NULL
Release_year	INTEGER	NOT NULL
runtime	INTEGER	NOT NULL

description	VARCHAR(100)	NOT NULL
Cast	VARCHAR(100)	NOT NULL

c. Recommendation:

Attributes	Data Type	Constraints
recommendation_id	INTEGER	PRIMARY KEY
Recommendation_score	INTEGER	NOT NULL
timestamp	TIME	NOT NULL
movie_id	INTEGER	FOREIGN KEY
user_id	INTEGER	FOREIGN KEY

d. Genre:

Attributes	Data Type	Constraints
genre_id	INTEGER	PRIMARY KEY
genre_name	VARCHAR(100)	NOT NULL
description	VARCHAR(100)	NOT NULL
popularity_score	INTEGER	NOT NULL
Parent_genre_id	INTEGER	NOT NULL

e. Rating:

Attributes	Data Type	Constraints
Rating_id	INTEGER	PRIMARY KEY
user_id	INTEGER	FOREIGN KEY
movie_id	INTEGER	FOREIGN KEY
timestamp	TIME	NOT NULL
reviews	VARCHAR(100)	NOT NULL

f. Viewing history:

Attributes	Data Type	Constraints
view_id	INTEGER	PRIMARY KEY
View_device	VARCHAR	NOT NULL
View_duration	INTEGER	NOT NULL
View_timestamp	TIME	NOT NULL
user_id	INTEGER	FOREIGN KEY
movie_id	INTEGER	FOREIGN KEY

➢ Establish Relationships between the tables:

❖ Relationship between User and Ratings:

The relationship between User and Ratings is (1:N).
Each user can provide ratings for multiple movies, but each rating is associated with only one user.

- user_id is the primary key in the User table.
- user_id is a foreign key in the Ratings table.

❖ Relationship between Movie and Ratings:

The relationship between Movie and Ratings is (1:N).
Each movie can receive ratings from multiple users, but each rating corresponds to only one movie.

- movie_id is the primary key in the Movie table.
- movie_id is a foreign key in the Ratings table.

❖ Relationship between User and Viewing History:

The relationship between User and Viewing History is (1:N).
Each user can have multiple records in the viewing history, but each record belongs to only one user.

- user_id is the primary key in the User table.
- user_id is a foreign key in the Viewing History table.

❖ Relationship between Movie and Viewing History:

The relationship between Movie and Viewing History is (1:N).
Each movie can be watched by multiple users, but each viewing record pertains to only one movie.

- movie_id is the primary key in the Movie table.
- movie_id is a foreign key in the Viewing History table.

❖ Relationship between User and Recommendations:

The relationship between User and Recommendations is (1:N).
Each user can receive multiple movie recommendations, but each recommendation is tied to only one user.

- user_id is the primary key in the User table.
- user_id is a foreign key in the Recommendation table.

❖ Relationship between Movie and Recommendations:

The relationship between Movie and Recommendations is (1:N).
Each movie can be recommended to multiple users, but each recommendation corresponds to only one movie.

- movie_id is the primary key in the Movie table.
- movie_id is a foreign key in the Recommendation table.

❖ Relationship between User and Movie:

The relationship between User and Movie is (M:N).
Each user can watch multiple movies, and each movie can be watched by multiple users.

- A linking table, such as Viewing History, handles the many-to-many relationship.

❖ Relationship between Movie and Genre:

The relationship between Movie and Genre is (M:N).
Each movie can belong to multiple genres, and each genre can include multiple movies.

- A linking table, such as MovieGenre, manages this relationship.

❖ Relationship between User and Genre:

The relationship between User and Genre is (M:N).
Each user can have preferences for multiple genres, and each genre can be liked by multiple users.

- A linking table, such as UserGenrePreference, captures this association.

➢ Write a brief report explaining your design decisions:

The design of this system focuses on managing the complex connections between users, movies, ratings, genres, viewing history, and recommendations. Here's a simple explanation of the key decisions made during the design process:

❖ Choosing the Right Entities and Attributes:

The main entities in the system (User, Movie, Rating, Genre, Viewing History, and Recommendations) were chosen because they are essential for a movie recommendation system. The attributes (details) for each entity were selected to cover important information for users, movies, and their interactions. For example:

- **User** stores key details like preferences and profile information.
- **Movie** includes important movie details such as title, release year, and cast, which help with recommendations.
- **Rating** links users to movies through ratings, which is important for personalized suggestions.
- **Genre** helps categorize movies and connect them with user preferences.
- **Viewing History** keeps track of which movies users have watched, helping the system make better suggestions.
- **Recommendations** store the movies suggested to each user.

❖ Mapping Relationships:

The relationships between the entities were designed to capture how they interact:

- **One-to-many relationships**: For example, one user can give ratings for many movies, and one movie can have many ratings. Similarly, one user can have many viewing history records, and one movie can be watched by many users. This helps the system manage user interactions and movie details efficiently.
- **Many-to-many relationships**: For example, users can watch many movies, and movies can be watched by many users. Likewise, users can like many genres, and genres can be liked by many users. To manage these many-to-many relationships, linking tables like **Viewing History** and **UserGenrePreference** are used.

❖ Choosing Data Types and Constraints:

Selecting the right data types and rules for each attribute ensures the data stays organized and easy to access:

- **INTEGER** is used for IDs (like user_id and movie_id), which helps the system link different records together.
- **VARCHAR** is used for text fields (like movie titles and user names), with a limit on the number of characters to save space and keep data consistent.
- **DATE** and **TIME** are used for timestamps, helping track when users register, rate movies, and watch them.
- **NOT NULL** is used to make sure important data is always filled in and not left empty.

❖ Normalization:

Normalization is a technique used to reduce repeating data and avoid mistakes. For example, movie genres are stored separately in a different table, with a linking table managing the relationship between movies and genres. This makes it easier to update or add new genres without affecting movie data.

❖ Scalability and Flexibility:

The system is built to grow over time. By using relational tables and linking tables, we can easily add more genres, movies, and user preferences without changing the overall structure. This design allows the system to handle a growing number of movies and users in the future.

❖ Optimizing for Performance:

To make sure the system runs smoothly, we index important data (like user_id, movie_id, and rating_id). This helps speed up queries, especially when retrieving recommendations or viewing history. The design allows for easy and fast searches, like finding movies by genre or checking ratings for a specific movie.

❖ Data Integrity and Storing History:

The system focuses on keeping the data accurate and complete. For example, ratings and viewing history need to be linked properly to both users and movies. Additionally, storing recommendations and ratings with timestamps ensures that historical data is kept and can be accessed for future analysis or reporting.

❖ Conclusion:

The database design balances keeping data organized, performing well, and being flexible. The relationships between users, movies, ratings, genres, and viewing history are clearly defined, making it easy to maintain and query data. The system is designed to grow with new features or increased user traffic, ensuring it will continue to work well in the long term.