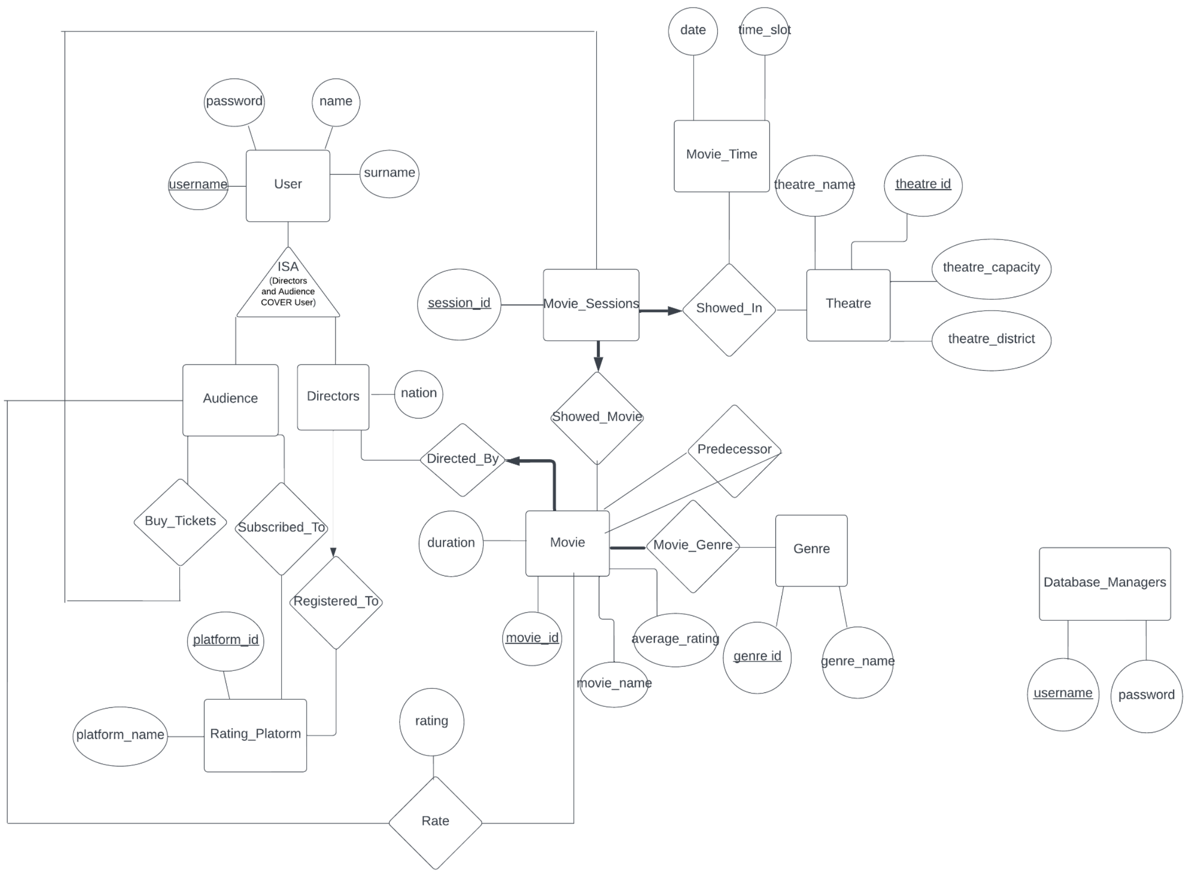
Project 3: MovieDB System
CMPE 321 Introduction to Database Systems, Spring 2023

Ali Üçer 2020400348, Aras Güngöre 2018401117

Introduction

This database schema organizes a movie rating platform where users can buy tickets, rate movies, and subscribe to movie platforms. The schema comprises entities such as users, directors, movie platforms, movie sessions, genres, and theaters, among others. It incorporates checks and constraints to ensure data consistency and reliability, such as limits on rating scores and the number of database managers. Triggers are also implemented to maintain data integrity.
Code and readme file are given in the file.

ER Diagram



Logical Database Design

Audience(username: string, _password: string, _name: string, surname: string)
Directors(username: string, _password: string, _name: string, surname: string, nation: string, platform_id: integer)
Subscribed_To(username: string, platform_id: integer)
Buy_Ticket(username: string, session_id: integer)
Rate(username: string, movie_id: integer, rating: real)
Rating_Platform(platform_id: integer, platform_name: string)
Movie_Sessions(session_id: integer, movie_id: integer, theater_id: integer)
Movie_Time(session_id: integer, _date: date, time_slot: integer)
Movie(movie_id: integer, movie_name: string, duration: integer, director_username: string, average_rating: real)
Movie_Genre(movie_id: integer, genre_id: integer)
Predecessor_Movies(predecessor_id: integer, successor_id: integer)
Theater(theater_id: integer, theater_name: string, Theater_capacity: integer, theater_district: string)
Genre(genre_id: integer, genre_name: string)
Database_Managers(username: string, _password: string)

Schema Refinement Steps

Table: Audience(username, password, name, surname)
Functional Dependencies:
    username -> password
    username -> name
    username -> surname
Using union property: {username} -> {password, name, surname}

The Audience table is in Boyce-Codd Normal Form (BCNF), following the rule that all non-key attributes (password, name, surname) are functionally dependent on the primary key (username). This implies that each username uniquely determines an audience member's password, name, and surname.

Table: Directors(<u>username</u>, password, name, surname, nation, platform_id)
Functional Dependencies:

```
        username -> password
        username -> name
        username -> surname
        username -> nation
        username -> platform_id
```

Using union property: {username} -> {password, name, surname, nation, platform_id}

The Directors table is in Boyce-Codd Normal Form (BCNF), following the principle that every non-key attribute (password, name, surname, nation, platform_id) is functionally dependent on the primary key (username). This indicates that each username uniquely determines a director's password, name, surname, nation, and platform_id.

Table: Subscribed_To(<u>username</u>, <u>platform_id</u>)
Functional Dependencies:

```
        {username, platform_id} -> {}           // There is no non-trivial dependency
```

The Subscribed_To table is in Boyce-Codd Normal Form (BCNF). This table essentially acts as a junction table to establish a relationship between the Audience and Rating_Platform tables. Its primary key is a composite key of both username and platform_id.

Table: Buy_Ticket(<u>username</u>, <u>session_id</u>)
Functional Dependencies:

```
        {username, session_id} -> {}            // There is no non-trivial dependency
```

The Buy_Ticket table is in Boyce-Codd Normal Form (BCNF). Similar to the Subscribed_To table, Buy_Ticket acts as a junction table connecting the Audience and Movie_Sessions tables. The primary key is a composite of username and session_id.

Table: Rate(username, movie_id, rating)
Functional Dependencies:
{username, movie_id} -> {rating}

The Rate table adheres to Boyce-Codd Normal Form (BCNF). Each non-key attribute (rating) is functionally dependent on the primary key (username, movie_id). This means that each pair of username and movie_id uniquely identifies a rating.

Table: Rating_Platform(platform_id, platform_name)
Functional Dependencies:
{platform_id} -> {platform_name}

The Rating_Platform table adheres to Boyce-Codd Normal Form (BCNF). This is because the non-key attribute (platform_name) is functionally dependent on the primary key (platform_id). Thus, each platform_id uniquely determines a platform_name.

Table: Movie_Sessions(session_id, movie_id, theater_id)
Functional Dependencies:
session_id -> movie_id
session_id -> theater_id
Using union property: {session_id} -> {movie_id, theater_id}

The Movie_Sessions table is in Boyce-Codd Normal Form (BCNF), as all non-key attributes (movie_id, theater_id) are functionally dependent on the primary key (session_id). This means that each session_id uniquely determines a movie_id and a theater_id.

Table: Movie_Time(session_id, _date, time_slot)
Functional Dependencies:
session_id -> _date
session_id -> time_slot
Using union property: {session_id} -> {_date, time_slot}

The Movie_Time table follows the Boyce-Codd Normal Form (BCNF). Each non-key attribute (_date, time_slot) is functionally dependent on the primary key (session_id), indicating that a session_id uniquely determines a specific date and a time_slot.

Table: Movie(movie_id, movie_name, duration, director_username, average_rating)
Functional Dependencies:
movie_id -> movie_name
movie_id -> duration
movie_id -> director_username
movie_id -> average_rating
Using union property: {movie_id} -> {movie_name, duration, director_username, average_rating}

The Movie table complies with the Boyce-Codd Normal Form (BCNF). This is because all non-key attributes (movie_name, duration, director_username, average_rating) are functionally dependent on the primary key (movie_id). Therefore, each movie_id uniquely identifies a movie_name, duration, director_username, and average_rating.

Table: Movie_Genre(movie_id, genre_id)
Functional Dependencies:
{movie_id, genre_id} -> {}   // There is no non-trivial dependency

The Movie_Genre table adheres to the Boyce-Codd Normal Form (BCNF) as there are no non-key attributes. The primary key is a composite key consisting of movie_id and genre_id, and there are no additional attributes.

Table: Predecessor_Movies(predecessor_id, successor_id)
Functional Dependencies:
{predecessor_id, successor_id} -> {}   // There is no non-trivial dependency

The Predecessor_Movies table complies with the Boyce-Codd Normal Form (BCNF), as there are no non-key attributes. The primary key is a composite key made up of predecessor_id and successor_id, and there are no additional attributes.

Table: Theater(theater_id, theater_name, theater_capacity, theater_district)
Functional Dependencies:
theater_id -> theater_name
theater_id -> theater_capacity
theater_id -> theater_district
Using union property: {theater_id} -> {theater_name, theater_capacity, theater_district}

The Theater table adheres to the Boyce-Codd Normal Form (BCNF). Every non-key attribute (theater_name, theater_capacity, theater_district) is functionally dependent on the primary key (theater_id), which means that each theater_id uniquely identifies a theater_name, theater_capacity, and theater_district.

Table: Genre(genre_id, genre_name)
Functional Dependencies:
{genre_id} -> {genre_name}

The Genre table conforms with the Boyce-Codd Normal Form (BCNF). This is because the non-key attribute, genre_name, is functionally dependent on the primary key, genre_id. Therefore, each genre_id uniquely identifies a genre_name.

Table: Database_Managers(username, _password)
Functional Dependencies:
{username} -> {_password}

The Database_Managers table adheres to the Boyce-Codd Normal Form (BCNF). The non-key attribute _password is functionally dependent on the primary key username. Thus, each username uniquely determines a _password.

These constraints and triggers are used for maintaining the consistency and reliability of the database.

1. **check_rating Constraint:** This is a check constraint added to the Rate table to ensure that the value for the rating field always falls between 0 and 5.
2. **check_timeslot Constraint:** This is a check constraint added to the Movie_Time table to ensure that the time_slot is always between 1 and 4.
3. **MaxDatabaseManagers Trigger:** This trigger is set up to limit the total number of database managers. It checks whether there are already 4 entries in the Database_Managers table before inserting a new row. If there are already 4 managers, it raises an error and rejects the insertion.
4. **UpdateAverageRating Trigger:** This trigger operates on the Rate table after an insertion operation. It updates the average_rating field in the Movie table with the new average rating for the movie that just received the new rating.

5. **CheckUserRating Trigger:** This trigger operates before an insertion into the Rate table. It ensures that a user must be subscribed to the movie platform and must have bought a ticket before they can rate a movie. If these conditions are not met, the trigger raises an error.
6. **CheckMovieDuration Trigger:** This trigger operates before an insertion into the Movie_Time table. It ensures that the duration of a movie doesn't exceed the available time slots. If the movie's duration is too long for the allocated time slot, it raises an error.
7. **CheckTheatreCapacity Trigger:** This trigger operates before an insertion into the Buy_Ticket table. It checks that the number of tickets sold for a session doesn't exceed the capacity of the theater. If the theater is already at capacity, it raises an error.
8. **CheckTimeSlot Trigger:** This trigger checks for time conflicts in the Movie_Time table before a new session's insertion. If the new movie session overlaps with an existing one at the same theater and date, it raises an error and deletes the new session from the Movie_Sessions table to prevent double-booking.
9. **CheckPredecessorMovies Trigger:** This trigger operates before an insertion into the Buy_Ticket table. It ensures that a user has watched all predecessor movies before they can buy a ticket for a new movie. If they haven't watched the predecessors, it raises an error.