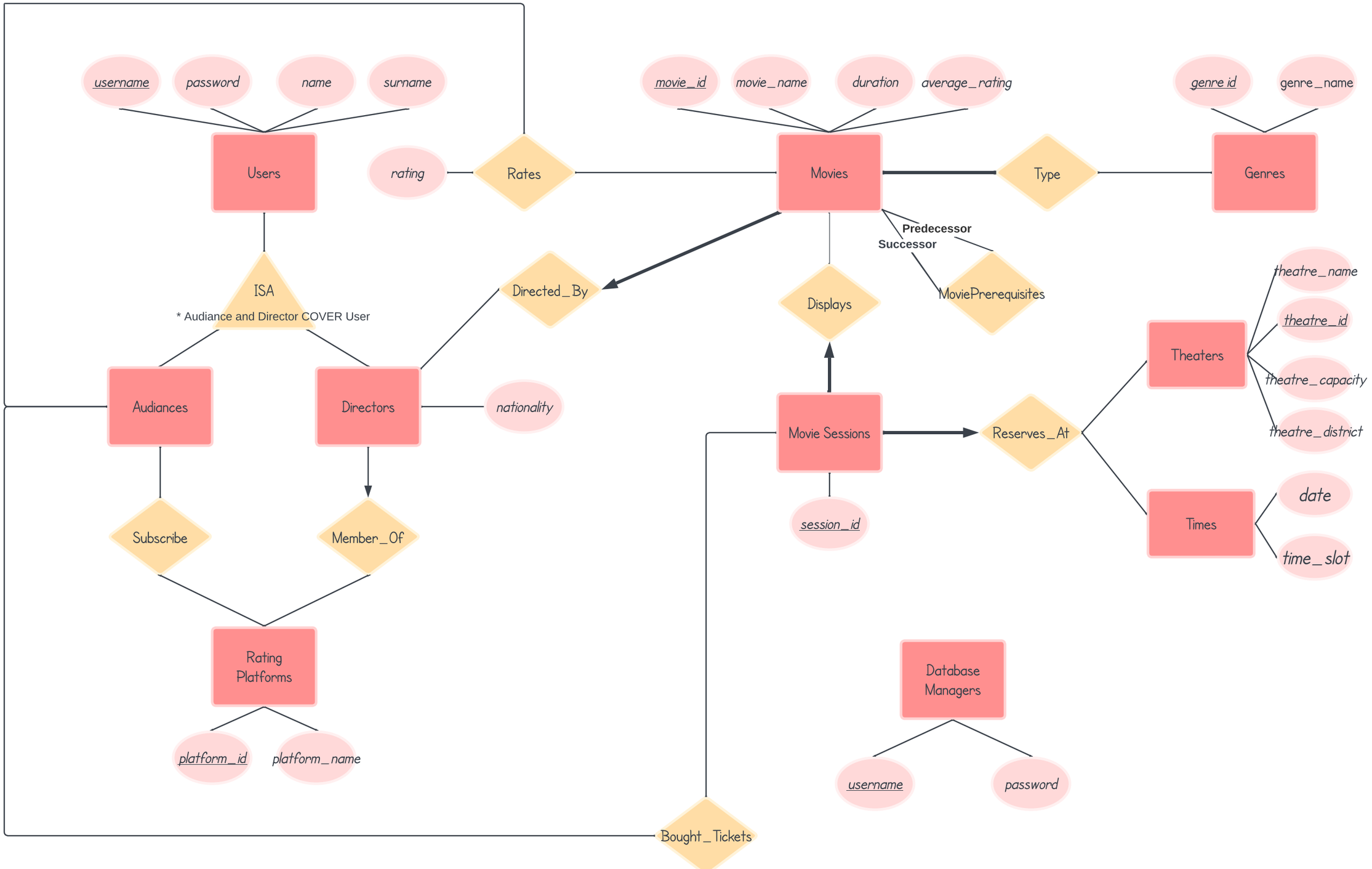
username
password
name
surname
Users
rating
Rates
movie_id
movie_name
duration
average_rating
Movies
genre id
genre_name
Genres
Type
Predecessor
Successor
MoviePrerequisites
ISA
* Audiance and Director COVER User
Directed_By
Displays
Audiances
Directors
nationality
Movie Sessions
Reserves_At
Theaters
theatre_name
theatre_id
theatre_capacity
theatre_district
session_id
Times
date
time_slot
Subscribe
Member_Of
Rating Platforms
platform_id
platform_name
Database Managers
username
password
Bought_Tickets

RELATIONS

DatabaseManagers(username: string, password_: string)

RatingPlatforms(platform_id: string, platform_name: string)

Genres(genre_id: string, genre_name: string);

Audiences(username: string, password_: string, name_: string, surname: string)

Directors(username: string, password_: string, name_: string, surname: string, nationality: string, platform_id: string)

Movies(movie_id: string, movie_name: string, duration: integer, director_username: string, average_rating: real)

MovieHasGenres(movie_id: string, genre_id: string)

Subscribes(username: string, platform_id: string)

Rates(username: string, movie_id: string, rating)

Theaters(theater_id: string, theater_name: string, theater_capacity: integer, theater_district: string)

MovieSessions(session_id: string, movie_id: string, theater_id: string, date_: date, time_slot: integer)

BoughtTickets(username: string, session_id: string)

MoviePrerequisites(movie_id_predecessor: string, movie_id_successor: string)

CMPE321 - PROJECT 1 REPORT

HASAN BAKİ KÜÇÜKÇAKIROĞLU - 2018400141

ALP TUNA – 2019400288

A) ER Diagram:

We drew the ER diagram according to the constraints we are given in the description. However, we couldn't show each restriction in the ER diagram since the expression power of ER diagrams is limited. However we tried not to miss any key and participation constraints while having an understandable ER diagram and avoiding data duplication.

Some design choices we made

- We couldn't show the following constraint although we were able to partially enforce it but decided not to implement:

  No two movie sessions can overlap in terms of theater and the time it's screened.

  When modeling the relationship between Moviesessions, theater and time, we first tried to use aggregation in order to ensure that no two movies can start at the same place at the same time. However, this approach was not natural in the ER diagram since we tied theaters to time slots which actually need to be an attribute of movie sessions and also it was not enough to catch the whole constraint. Although we don't allow the simultaneous start of two movies, their display time can still overlap. For example, movie1 may take the slots between 2-4 and movie2 may take slots between 3-4. Therefore, we decided to go with the natural ternary relationship "Reserves_At" and decided to check the constraint at other stages of the project.

- We tied Rates relationship to audience entity instead of user entity, because directors cannot rate a movie due to the constraint that users should buy a ticket to the movie to be able to rate the movie and only audience can buy the ticket.

The constraints we couldn't show

We couldn't show some constraints due to the limited expression power (at least we think so) of ER diagrams. The list of constraints we couldn't show are as follows:

- "There are four time slots for each day",
- A user can rate a movie – if they are already subscribed to the platform that the movie can be rated. AND – if they have bought a ticket to the movie

- There are four time slots for each day
- If a movie has any predecessor movies, all predecessor movies need to be watched in order to watch that movie. (See the example below: The Minions need to be watched before Minions: The Rise of Gru).
- The duration of the movie is closely related to the time slots. The time slot attribute determines the starting time of the movie and the end time is determined by the duration. (If a movie starts at time slot 2 and has a duration of 2, the theater is reserved for that movie during the following time slots: [2, 3]).

B) SQL Table Creation and Deletion:

We tried to implement all the entities and relationships together with their constraints we have designed in the ER diagram. We also implemented some extra constraints which we were not able to detect in our ER diagram.

Important Notes

1. The only constraint which we could catch in the ER diagram but not in SQL Tables is the participation constraint of Type relation in ER diagram. Movie entities have total participation, however, without a key constraint we didn't see how we could have handled that. We tried to check it using some triggers, however we were not encouraged to use triggers at this stage of our project and it was kind of an overkill for such a constraint. Therefore we omit it.

2. There are many general additional restrictions other than key/participation/nullable/uniqueness constraints that we should enforce in our project. However, without Triggers and checks we didn't see how we could have handled them. Nevertheless, we used 3 Triggers and checked the following constraints:

a. Database managers cannot have more than 4 entries.
b. A user cannot rate a movie if he/she already rated it.

3. Some of the restrictions which we couldn't enforce without Triggers and trigger constraints seemed to be a bit too complicated for those ones:

a. A user can rate a movie – if they are already subscribed to the platform that the movie can be rated. AND – if they have bought a ticket to the movie.
b. If a movie has any predecessor movies, all predecessor movies need to be watched in order to watch that movie. (See the example below: The Minions need to be watched before Minions: The Rise of Gru).

4. In the Rates relation, we thought that (audience_username and movie_id) could be valid primary keys since a user cannot rate a movie more than once. However, in this case, when we delete a user, we couldn't take any action in the Rates table, since foreign keys have to

point to the actual user and setting it to null violated the primary key constraint. Therefore, we decided that Rates relation doesn't have a primary key since its username can be null. In the end, we added a trigger to ensure that a user cannot rate the same movie twice.

5. We calculate the average rating of a movie with the help of a trigger. When an entry is given to the “Rates” table, we update the average rating of the movie. Name of the trigger is “UpdateAverageRating”

6. In the dropTables.sql file, we drop the tables without causing any dependency errors.

7. We implemented all IDs as strings although all examples given in the description were integers since most modern systems include an id generator algorithm which in turn returns a unique string ID. Also, any integer can be represented as a string, so even if all of them are sent as an integer, we can store them as strings though this would be inefficient.

8) We decided to implement a user entity using 2 tables called “Audiences” and “Directors” since they satisfy the covering constraint. Each user is either an audience or a director. The biggest downside of this approach is the slow query over all users since in such queries we need to join the two tables. However, we couldn’t see a common case where all users are queried frequently since an audience and a director has not much a lot in common other than some of their generic attributes such as “username”, “password”, and “surname”.