CMPE321

Database Systems

Project 3

Ali Alperen Sönmez, 2020400354
Ramazan Burak Sarıtaş, 2020400321

PART 1: SCHEMA REFINEMENT AND NORMALIZATION

We refined the schema because our previous design was containing some unnecessary tables. For instance, it is was not necessary to keep the information about a movie and the movie's director in seperate tables. Other than that, our design did not require any schema normalizations since the relations were already in Boyce-Codd Normal Form. You can see the functional dependencies of our new schema below:

1-) rating_platforms(platform_id:string, platform_name:string)

(I,N)

FDs:

I -> IN

Since "I" (platform_id) is superkey, "rating_platforms" relation is in Boyce-Codd Normal Form.

2-) directors_agreements(user_name: string, platform_id: string, director_password: string, director_name: string, director_surname: string, nationality: string)

(U,I,P,N,S,T)

FDs:

U -> UIPNST

Since "U" (user_name) is superkey, "directors_agreements" relation is in Boyce-Codd Normal Form.

3-) audience(user_name: string, audience_password: string, audience_name: string, audience_surname: string)

(U,P,N,S)

FDs:

U -> UPNS

Since "U" (user_name) is superkey, "audience" relation is in Boyce-Codd Normal Form.

4-) subscribes(user_name: string, platform_id: string)

(U,P)

FDs:

UP-> UP

Since "UP" (user_name, platform_id) is superkey, "subscribes" relation is in Boyce-Codd Normal Form.

5-) movies(movie_id: string, director_name: string, movie_name: string, duration: integer, average_rating: real)

(I,D,M,T,R)

FDs:

I -> IDMTR

Since "I" (movie_id) is superkey, "movies" relation is in Boyce-Codd Form.

Assumption: We assumed that a director can have two different movies with the same name and, two different movies can have the same name even if their directors are different.

6-) ratings(movie_id: string, user_name: string)

(I,N)

FD's:

IN -> IN

Since "IN" (movie_id, user_name) is superkey, "ratings" relation is in Boyce-Codd Normal Form.

7-) movie_predecessors(successor_id: string, predecessor_id: string)

(S,P)

FD'S:

SP -> SP

Since "SP" (successor_id, predecessor_id) is superkey, "movie_predecessors" relation is in Boyce-Codd Normal Form.

8-) genres(genre_id: integer, genre_name: string)

(I,N)

FD's:

I -> IN

N -> IN

Since "I" (genre_id) and "N" (genre_name) are both superkeys, "genres" relation is in Boyce-Codd Normal Form.

9-) movie_genres(movie_id: string, genre_id: string)

(I,G)

FD's:

IG -> IG

Since "IG" (movie_id, genre_id) is superkey, "movie_genres" relation is in Boyce-Codd Form.

10-) theatre(theatre_id: integer, theatre_name: string, capacity: integer, district: string)

(I, N, C, D)

FD's:

I -> INCD

Since "I" (theatre_id) is superkey, "theatre" relation is in Boyce-Codd Form.

11-) movie_session(session_id: string, movie_id: string)

(I, M, T, D, S)

FD's:

I -> IM

In this context, "I" (session_id) is superkey. Thus, "movie_session" relation is in Boyce-Codd Form.

12-) occupied_slots(session_id: string, theatre_id: string, session_date: string, time_slot: integer)

(I,T,D,S)

FD's:

TDS -> ITDS

Since "TDS" is superkey, this relation is in Boyce-Codd Form.

13-) tickets(ticket_id: integer, user_name: string, session_id: string)

(I, U, S)

FD's:

I -> IUS

US -> IUS

In this context, both "I" and "US" are superkeys. Thus, "tickets" relation is in Boyce-Codd Form.

14-) database_managers(user_name: string, manager_password: string)

(N, P)

FD's:

N -> NP

Since "N" (user_name) is a superkey, "database_managers" relation is in Boyce_Codd Form.

PART 2: CHECK CONSTRUCT

Below is the list of constraints that we handled using “CHECK” construct:
- We used ‘CHECK’ construct to ensure that the “rating” is between 0 and 5.
- We used ‘CHECK’ construct to ensure that the “time_slot” is between 1 and 4 because there is 4 time slots in a day.

PART 3: TRIGGER CONSTRUCT

3-) Situations handled with triggers are as follows:
- Users are unable to rate films that are not available on their subscribed platforms.
- Users can only rate films for which they have purchased tickets.
- There can be at most 4 database managers in the system.
- Users are unable to purchase tickets for sessions when the theatre capacity is already full.
- When a user rates a movie, the average rating of the movie is updated accordingly.
- The user is required to have watched all the preceding films in order to purchase a ticket for the current film.

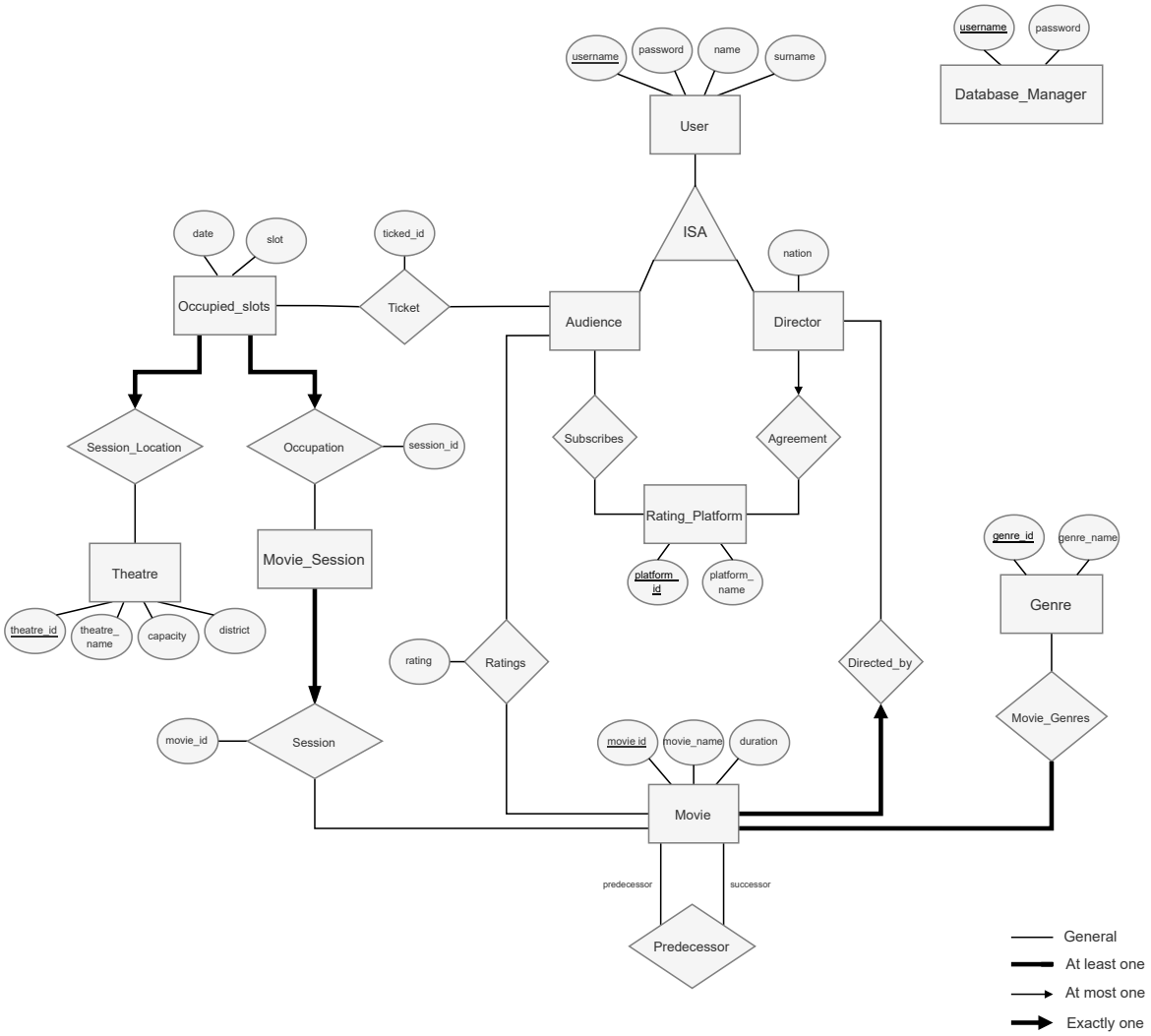
Note: The database schema file is at: app/database/database.sql

PART 4: ER DIAGRAM

In our updated ER diagram, we have made some changes to enhance the management of sessions and their associated information.

We have introduced a new table, occupied_slots. This table is responsible for storing the date, slot, and theatre information of sessions. This allows us to easily track and manage the availability of slots in different theaters.

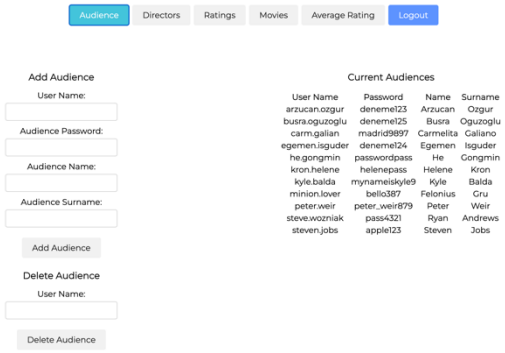
The movie_session table is now only used to establish a relationship between movies and sessions. It stores the session_id and movie_id of each session. Here is the updated ER diagram:



PART 5: APP STRUCTURE

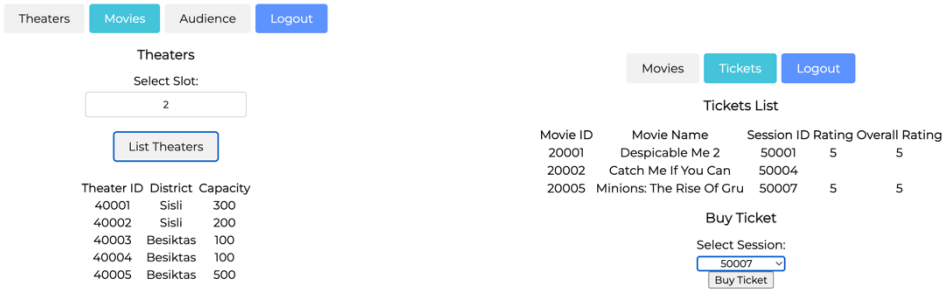
Our app is a web-based movie database management system, developed using React for the front end, Flask framework for Python for the back end, and MySQL for the database. The project follows a directory structure with separate directories for the client-side (React) code, Flask server code, and the database scripts. The back-end code, located in /app/flask-server/server.py, utilizes the mysql-connector-python library to establish a connection with the MySQL database. The app works on a local host and can be built by following the instructions in README.md.

The application features three distinct dashboards catering to different roles:
- The **Manager** dashboard consists of five tabs: Audience, Directors, Ratings, Movies, and Average Rating. Each tab provides functionality to manage audiences, directors, ratings, movies, and view average ratings respectively.



- The **Director** dashboard contains three tabs: Theaters, Movies, and Audience, allowing directors to manage theaters, movies, and view audience information.

- Lastly, the **Audience** dashboard features two tabs: Movies and Tickets, enabling audiences to browse available movies and purchase tickets.



If a user tries to access a dashboard route that their role does not allow, the app redirects the user to the login page.