Project 3: MovieDB System

CMPE 321, Introduction to Database Systems, Spring 2023

Due: 22 May 2023, 23:59 o'clock

1 Introduction

As part of today's youth, you should have been to a movie event at least once. You may have bought a movie ticket, either in a movie theater or through an online interface. Now it's time to explore what's behind the online interface, and also combine it with the rating experience. A well-designed database system is of utmost importance, since otherwise, we face the risk of having erroneous data, unsatisfied customers, and loss of revenue. This project aims to develop a database application for a movie ticket booking and rating system.

2 Project Description

In project 1, you performed conceptual database design, drew ER diagrams, and converted these ER diagrams into relational tables for MovieDB. In this project, you will go one step further and implement the MovieDB database with a user interface (UI) using the structure you have designed in Project 1 or its improved version. Refining your schema using what you learned in the lectures is strongly advised. MovieDB should contain the following information:

1. **User** includes the following attributes; *username, password, name, surname.* Each user has a unique username and each user is either an audience or a director.
   (a) **Audience** additionally have a list of bought *tickets for movie sessions* and a list of *rating platforms* to which they are subscribed.

   An audience can buy tickets for different movies (or movie sessions) and can subscribe to different rating platforms such as IMDB and Letterboxd.

   (b) **Directors** additionally have *nation* information and *platform id.*
   - Each director **must** have **only one** *nation.*
   - Each director **can** have **at most one** *platform id.*

   That is, dual citizenship is not allowed and agreement with more than one platform is not possible for directors.

2. **Rating Platform** includes the following attributes: *platform id, platform name*. Both name and id must be unique.

3. **Movie Sessions** includes the following attributes: *session id, movie id, movie name, duration, genre list, average rating, director username, platform id, predecessors, theatre id, theatre name, theatre capacity, theatre district, time slot, date*. The session id must be unique.

   - Each movie's platform is the same as the platform of its director.
   - No two movie sessions can overlap in terms of theatre and the time it's screened.
   - There are four time slots for each day.
   - 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 theatre is reserved for that movie during the following time slots: [2, 3]).
   - 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).
   - Each theatre id corresponds to a physical location. Hence, theatre capacity and theatre district depend solely on the theatre id.
   - Same as above; movie id determines the movie name, duration, genre list, overall rating, director username, and platform id.
   - Every movie needs to have at least one genre.
   - Every movie needs to have exactly one director.

4. **Ratings** have the following attributes: *username, movie id, rating*. A pair of username and movie id uniquely identifies a rating. Ratings will be given as floats between 0 - 5 (e.g., 4.5).

   - A user can rate the same movie only once.
   - 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.

5. **Genre** have following attributes: *genre id, genre name*. Both name and id must be unique.

6. **Database Managers** consists of the following attributes: username and password. There exists only one database manager with a certain username. There can be at most 4 database managers registered to the system.

3 Schema refinement and normalization

You should analyze your design from Project 1 in terms of functional dependencies (FDs) and normal forms, then, refine your design if needed. You should explicitly list all of the non-trivial FDs. Then, for each relation you should determine if it is in Boyce-Codd Normal Form (BCNF) and you should explain how the requirements of BCNF are met (or not met) in terms of FDs. If a relation is not in BCNF, you should check whether it is 3NF and explain how the requirements for 3NF are met (or not met). If a relation is not in BCNF, you should either decompose it into BCNF relations or provide a justification if you decide not to decompose it. If you decompose a relation, you should explain whether the decomposition is lossless-join and dependency preserving. It is possible that your initial schema is already in BCNF. If this is the case, you still need to explain how the requirements of BCNF are met in terms of FDs for each one of your relations.

In addition, you are expected to refine your design by capturing the constrains that you were not able to handle in Project 1 by using the CHECK construct. Please describe which additional constraints you were able to handle by using CHECK.

4 Requirements

Sample database manager, genre, and rating platform data are provided in the tables below. You can use these to test your system. Note that we will use the same data for testing your application in the demo, so please do not manipulate these provided data. That is, you **shouldn't create/delete or change** that data that we provide below for Managers, Genres, and Rating Platforms.

Table 1: Sample Database Managers Data.

username	**password**
manager1	managerpass1
manager2	managerpass2
manager35	managerpass35

Table 2: Sample Genre Data

genre_id	**genre_name**
80001	Animation
80002	Comedy
80003	Adventure
80004	Real Story
80005	Thriller
80006	Drama

Table 3: Sample Rating Platforms Data.

platform_id	**platform_name**
10130	IMDB
10131	Letterboxd
10132	FilmIzle
10133	Filmora
10134	BollywoodMDB

Your UI must support the following operations:

1. Database managers shall be able to log in to the system with their credentials (username and password).

2. Database managers shall be able to add new Users (Audiences or Directors) to the system.

3. Database managers shall be able to delete audience by providing username. When an audience is deleted, all personal data regarding that audience must be deleted including the *list_of_bought_sessions* for movie sessions and the *list_of_rating_platforms* to which they are subscribed.

4. Database managers shall be able to update *platform id* of the directors by providing director *username* and *platform_id*.

5. Database managers shall be able to view all directors. The list must include the following attributes: *username, name, surname, nation, platform_id*.

6. Database managers shall be able to view all ratings of a specific audience by providing *username*. The list must include the following information: *movie_id, movie name, rating*.

7. Database managers shall be able to view all movies of a specific director by providing the director's *username*. The list must include the following attributes: *movie_id, movie name, theatre_id, district, time_slot*.

8. Database managers shall be able to view the average rating of a movie by providing *movie_id*. The list must include the following attributes: *movie_id, movie_name, overall_rating*.

9. Directors shall be able to log in to the system with their credentials (username and password).

10. Directors shall be able to list all of the theatres available for a given slot. The list must include the following attributes: theatre_id, district, theatre_capacity.

11. Directors shall be able to add movies by providing *movie_id, movie_name, theatre_id, time_slot*. The *platform_id* of the movie should be the same as the director's platform.

12. Directors shall be able to add predecessor(s) to a movie by providing its *movie_id* and the *movie_id of the predecessors*.

13. Directors shall be able to view all movies that they directed in ascending order of *movie_id*. The list must include the following attributes: *movie_id, movie name, theatre_id, time_slot, predecessors_list*. Predecessors list must be a string in the form "movie1_id, movie2_id, ..."

14. Directors shall be able to view all audiences who bought a ticket for a specific movie directed by themselves. To view this information, the director should provide a *movie_id*. The list must include the following attributes: *username, name and surname*.

15. Directors shall be able to update the name of a movie directed by themselves by providing a *movie_id* and *movie name.*

16. Audiences shall be able to list all the movies. The list must include the following attributes: *movie_id, movie name, director's surname, platform, theatre_id, time_slot, predecessors_list. predecessors_list* must be a string in the form "*movie1_id, movie2_id, ...*"

17. Audiences shall be able to buy a movie ticket by providing a *session_id.* There are several constraints:

    (a) Audiences can buy tickets for different movies or movie sessions (i.e., different sessions of the same movie)

    (b) To buy the movie tickets, all predecessor movies need to be watched in order to watch that movie. (See the example below: Deli Yürek to be watched before the Deli Yürek: Bumerang Cehennemi). We will not test the case of a director adding a predecessor for a movie after an audience has bought the movie tickets.

    (c) Audience cannot buy a ticket if the *theatre_capacity* is full. At the beginning, bought *list_of_bought_sessions* of audiences will be provided. The capacity for a movie session is the maximum number of audiences who can buy ticket for that movie session.

18. Audiences shall be able to view the tickets that they're currently buying and have bought previously. The list must include the following attributes: *movie_id, movie name, session_id, rating, overall_rating.* If the audience is buying the ticket and the movie isn't screened yet, the rating field will be null.

19. You must handle the following cases with triggers:

    (a) When audience rates a session, the average rating of the movie should be updated automatically in the ratings table.

    You can write more triggers if it is needed.

5 Notes

- In the scope of this project, we are more interested in the functionality of the web interface, rather than the styling of UI. In other words, it is totally fine to create a system that satisfies all requirements and has zero line of CSS and Javascript.
- The allowed languages are PHP, Java, JavaScript, and Python. You can use a framework, however, you **must** write the SQL queries and boot the database server yourself. Note that you should set up the database and create the tables on your own. You are **not** allowed to use any tool that helps with these parts such as ORM(Object Relational Mapping).
- Also, do **not** use database connector as your query supplier. Use it only as a query executor by running it with SQL strings written by you.
- You are not expected to deploy your system. So, it is **fine** if it runs in your local.
- You are free to use **relational database** server of your choice. However, we will **not** accept submissions with non-relational databases or no database servers. Also, **SQLite is not applicable** for this project.

6 Submission

This project can be implemented either individually or as a team of two people. You are free to change teams in the upcoming projects. The submission must include your **code**, your **description of the schema refinement step, your updated database design and ER diagrams**, and a **READ.ME** file that describes how to **run** your code. **Please note that if we are unable to run your code or if there are not any README instructions, there will be 50% penalty.** Place all the required files into a folder named with the student IDs of the team members separated by an underscore (e.g., 2017400200_2018700120). Zip the folder for submission and name the .zip file with the same name. Submit the .zip file through Moodle until the deadline. **Any other submission method is not allowed**. Each group should submit **one** .zip file. The system will be evaluated during a demo session that will be arranged. Demo day(s) will be announced. Before the demo date, we will share the data that will be used during the demo in Excel format. You **must** add the entries **before** the demo.

7 Late Submission Policy

You are allowed a total of 7 late days on the projects with no late penalties applied. You can use these 7 days as you wish. For example, you can submit the first project on time, the second project 2 days late and then the third project 3 days late. In that case, you will have to submit the fourth project on time. No late submissions for any of the project will be accepted after you use these 7 extra days. If you change your team, the team's late days used so far will be the maximum number of late days used by any of the team members.