Project 3 : MovieDB - Report

CMPE 321 Introduction to Database Systems

Necdet Can Uzduran

Student ID : 2019400195

22/05/2023

Description of the Schema Refinement Step

Non-trivial Functional Dependencies

In this section I will be analyzing each table one by one and explicitly list non-trivial functional dependencies that exist on that particular table.

Audience

username → username, password, name, surname

Director

username → username, password, name, surname, nation, platform_id

Rating Platform

platform_id → platform_id, platform_name

platform_name → plaform_id

Audience_RPlatform

There is not any non-trivial functional dependency in Audience_RPlatform table

username, platform_id → username, platform_id (trivial)

Rating

username, movie_id → username, movie_id, rating

username, movie_id → rating (decomposing the statement above)

Movie Session

session_id → session_id, time_slot, date, movie_id, theatre_id

Ticket

There is not any non-trivial functional dependency in Ticket table

username, session_id → username, session_id (trivial)

Theatre

theatre_id → theatre_id, theatre_name, theatre_cap, district

Genre

genre_id → genre_id, genre_name

genre_name → genre_id

Movie

movie_id → movie_id, movie_name, duration, director_username

Movie Genres

There is not any non-trivial functional dependency in Movie Genres table

movie_id, genre_id → movie_id, genre_id (trivial)

Predecessor

There is not any non-trivial functional dependency in Predecessor table

movie_id, pred_id → movie_id, pred_id

DB_Manager

username → username, password

username → password (decomposing the statement above)

Reserves*

Since there cannot exist more than one movie session that starts at the same date and same starting slot, "theatre_id,date,starting_slot" becomes the primary key (composite) of the table. Hence this composite key basically enough to determine a session's ending_slot from the table.

theatre_id, date, starting_slot □ theatre_id, date, starting_slot, ending_slot

Average_Rating*

movie_id → movie_id, avg_rating

***** Newly added tables

Evaluation of Tables w.r.t Normal Forms

By the definition of Boyce-Codd Normal Form, one can observe that all tables that are created in BCNF since for all functional dependencies, left side of the FDs consist of candidate keys (superkey) only. Hence there isn't any need for redesigning my schemas.

Additional Constraints

In this section I will be mentioning the additional constraints that I couldn't manage to cover in previous project and I will be providing how I handled these constraints in this project.

- **No two movie sessions can overlap in terms of theatre and the time it's screened**
  I couldn't manage to cover this constraint in previous project since my knowledge about concepts such as triggers and checks were limited. However, in this project I've managed to handle this constraint via creating a trigger called **"overlapingTimeSlots"**

Trigger is activated before insert on movie session table. Basically let a director create a movie session if there is no overlap on screen times of another movie session by checking reservations of all other theatres in "Reserves" table.

- **If a movie has any predecessor movies, all predecessor movies need to be watched in order to watch that movie**
  I've managed to cover this constraint via creating a trigger called **"mustWatchPreds"** This trigger is activated before insert on Ticket table. Trigger basically loops through all predecessor movies of the movie that is being screened on the movie session that is trying to be bought from the audience, and if there doesn't exist a ticket bought from the audience for any of the predecessor movie **before the date of the new ticket that the audience trying to buy**, then trigger force system to reject the buying process of that ticket.

- **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**
  I've managed to cover this constraint via creating a trigger called **"ratingConditions"** This trigger basically make sure the audience that is trying to rate a movie, already subscribed to the movie's rating platform (rating platform of the director of the movie) and also bought a ticket for that movie (movie session that is screening that movie).

- **There can be at most 4 database managers registered to the system**
  I've managed to cover this constraint via creating a trigger called **"atMostFourDBM"** This trigger is more trivial than the other ones. Trigger basically being activated before insert on DB_Manager table, and reject the insertion if there are already 4 database manager registered to the system.

- **Audience cannot buy a ticket if the theatre capacity is full**
  I've managed to cover this constraint via creating a trigger called "theatreCapChecker". This trigger is activated before insert on ticket, and reject the buying process of that ticket, if the theatre is full for the corresponding movie session.

- **Average rating updates**
  There are two update constraint that need to be covered on average ratings.
  1 - Update on average rating of a movie if there are a new rating inserted
  2 - Update on average rating of a movie if an audience is removed
  Both of these update constraints are covered via triggers called **"updateAvgRating"** and "**avg_rat_after_audience_del"** respectively.

- Most of the constraints are handled via using triggers. However, check is also used to ensure that given rating is between values of 0 and 5 (inclusive)

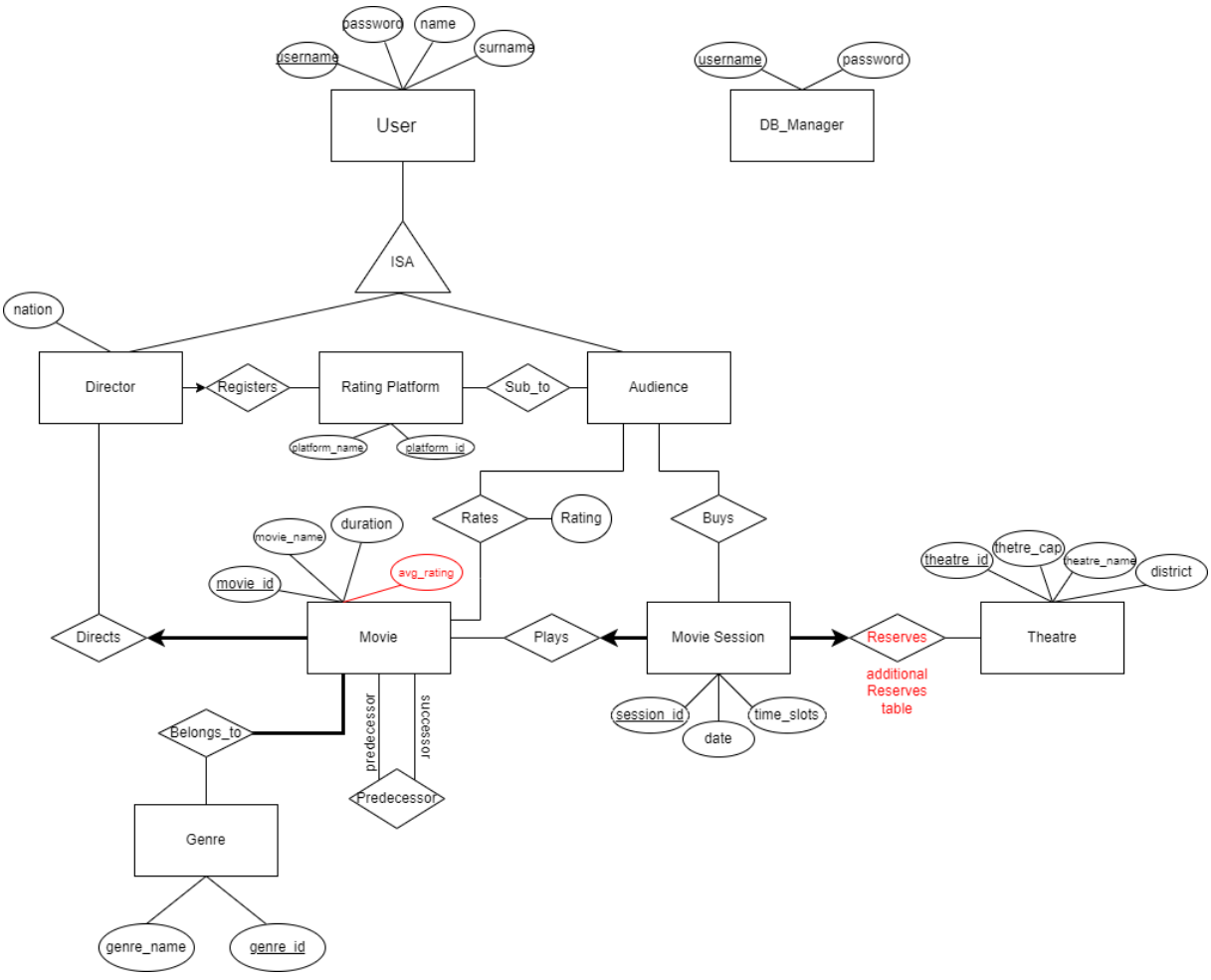
Updated Database Design

There is no change on schemas (tables) since all of them are in BCNF. However, in order to manage additional information and constraints that are stated in the project description, I've added two tables additional to the previous ones that are "Reserves" and "Average Rating" tables. "Reserves" table is updated once a movie session is created. Logic behind this table is basically making a reservation to the theatre corresponding to the movie session that is inserted to "Movie Session" table, and with the help of this table we can ensure that there is no overlapping between movie sessions in terms of thatre, date and screen time.

"Average Rating" table, basically holds the average rating information of the movies in the system. Creating additional table for average rating of the movies rather than calculating the average rating of a movie from "Rating" table each time we want to access to average rating of a movie, ease our way into updating the average rating of a movie and make this updating and retrieving process more trackable.

All other table designs are kept same, apart from additional check constraints mentioned in the section above.

Updated ER Diagram



*Updates are shown in red

Application

Project is implemented using **Python** language via using **Flask** back-end framework. PyMySQL is used as Python-MySQL client library. One need to install Flask, and pymysql in order to run the project.

- **Install Flask via command below:**

  >> pip install flask

- **Install pymysql via command below:**

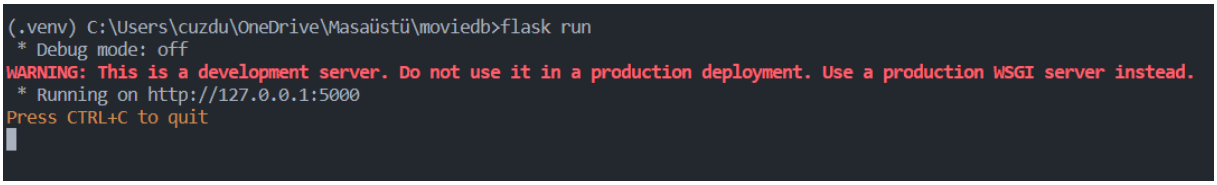
  >> pip install pymysql (Windows) -- I've still got module not found error if I install pymysql with this command

  >> sudo apt-get install python3-pymysql (bash) -- this installation seemed unproblematic on my side

- **After installation run the program:**

  >> flask run

While application is running (on port 5000), one can Ctrl + Click to the URL below, or simply write the URL of the application into any browser's URL section.



- Database configuration:
- host = "127.0.0.1"
- user = "root"
- password = "admin123"
- db = "MovieDB"

- User interface of the application is pretty simple and easy to follow. One can run the application and make use of the application features pretty easily via any browser.