Database Design Project 4

Movie List Management System

Siwon Kim, Hyunjung Lee

1. Overview

Our group has developed a CGV movie list management system. Our text-based project was created for the purpose of managing and checking movie lists. The system has two user tiers: customers can check and search movie lists, and save movies they have watched; employees can check, add, edit, and delete movie lists. We used Python language and SQLite as our DBMS.

https://docs.google.com/presentation/d/1TPU6hfqDEsKeG6L6n-qook7GTo6T58IB4gGwC1Ghpck/edit?usp=sharing

Project Requirements:

Required Features

1. Our system is backed up using SQLite using Python.
2. Our system is a text-based interface to interact with a human user.
3. A database that consists of three relations: movies, users, and watched.
4. We have more than eight different types of queries (in terms of SQL statements) including UPDATE queries.
5. We have a text-based interface with more than three different appearances.
6. Utilization of sqlite3 to connect to the database and send in queries. It provides a convenient and easy-to-use API for working with SQLite databases from Python code.

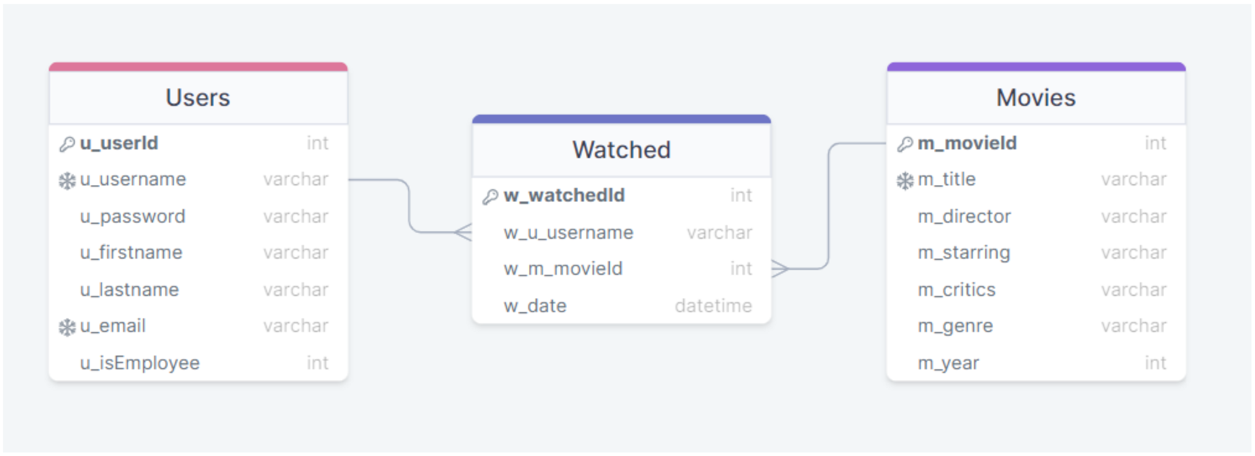
Additional Features

1. We created users for our system, assign userID/password to them
2. We created views for our database by assigning different privileges to different users (clients and employees)
3. We imported the data by using the concept of APIs.

2. Database Design

The following diagrams provide a high-level introduction to our database. They are provided as reference while learning about the system's client functionality in the following section.

Entity-Relationship Diagrams:



3. Client Functionality

This section describes how our user interface will work and how clients will view and be able to interact with our system. A more detailed description of how different pages work internally will be discussed in the following sections.

We start by going over the types of users on our system. There are three different classes:

A. Anonymous Users

All users who initially connect to our system are in this class and not able to process any interface until they log into our system. They can sign up or log in.

    a. Login
    b. SignUp - customer only
    c. Exit

B. Employees Users

    All accounts for the employees will be saved through API containing all employees information. Once they log into the system, they can view 6 different types of selections. These users will have the ability to view a movie list, add a new movie, edit critics of existing movies, delete the movies, and view their personal information.

    a. View movie list
    b. Add new movies
    c. Edit the movie list - critics only
    d. Delete the movie list
    e. View personal information
    f. Logout

C. Customers Users

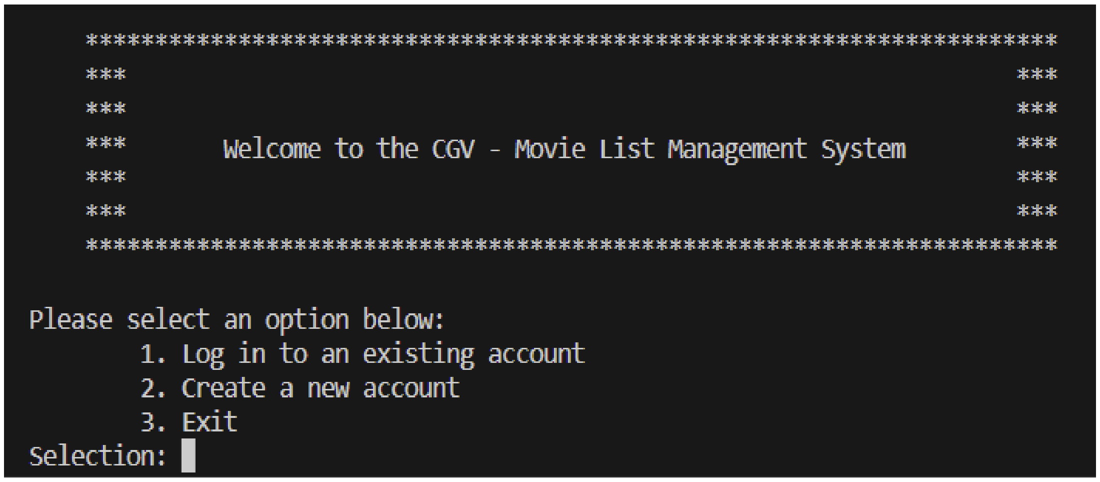
    All users should sign up for the system in order to use the system. Once they log into the system, they can view 5 different types of selections. These users will have the ability to view all movies listed, search movies, store the watched movies, view their watched movies, and view their personal information, etc.

    a. View movie list
    b. Search by Title/Director/Genre/Year
        i. Store the movie watched
    c. View watched movie List
    d. View personal Information
    e. Logout

4. Pages

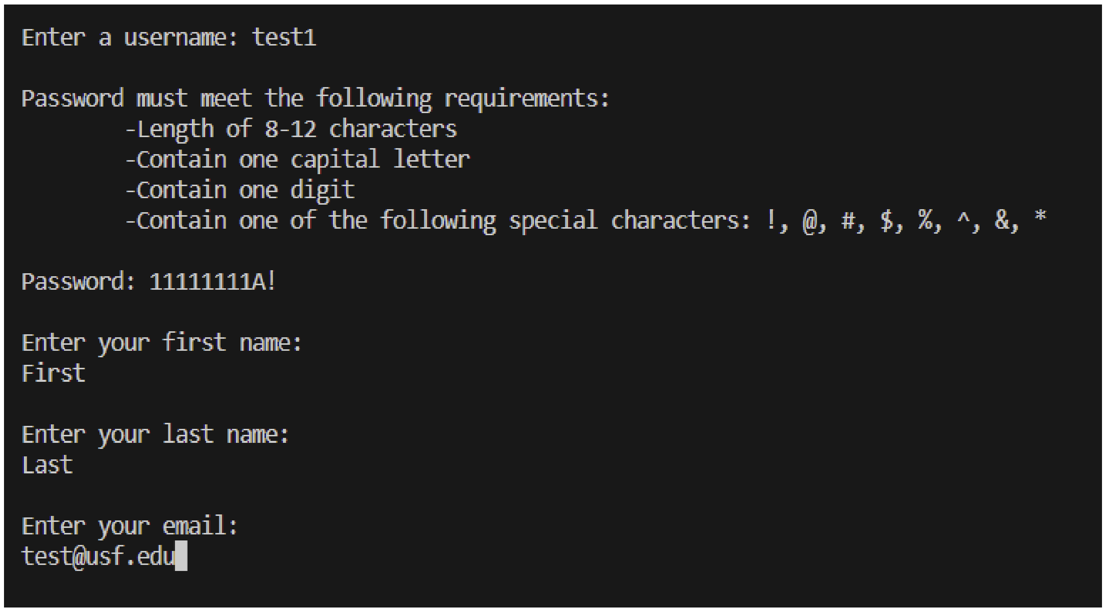
The Anonymous Users

1. Initial Page



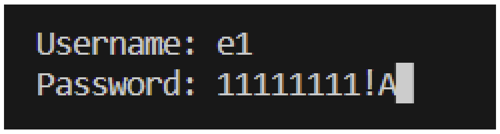
```
    **********************************************************************
    ***                                                                ***
    ***                                                                ***
    ***       Welcome to the CGV - Movie List Management System        ***
    ***                                                                ***
    ***                                                                ***
    **********************************************************************

Please select an option below:
        1. Log in to an existing account
        2. Create a new account
        3. Exit
Selection:
```

2. Sign Up Page



```
Enter a username: test1

Password must meet the following requirements:
        -Length of 8-12 characters
        -Contain one capital letter
        -Contain one digit
        -Contain one of the following special characters: !, @, #, $, %, ^, &, *

Password: 11111111A!

Enter your first name:
First

Enter your last name:
Last

Enter your email:
test@usf.edu
```

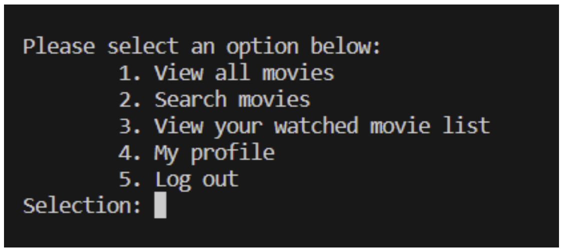
3. Sign In Page



```
Username: e1
Password: 11111111!A
```

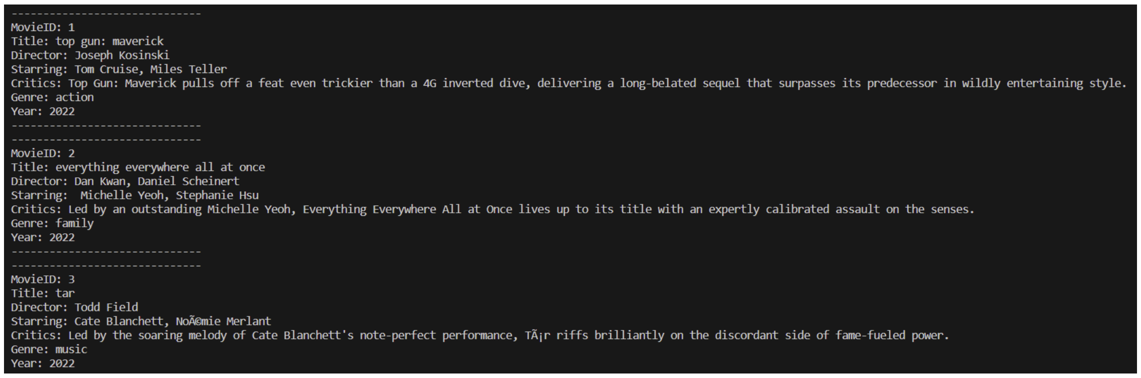
Customers Users

4. Customer Main Page



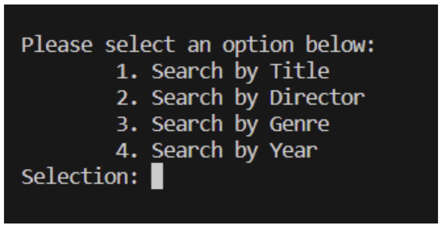
```
Please select an option below:
        1. View all movies
        2. Search movies
        3. View your watched movie list
        4. My profile
        5. Log out
Selection: 
```

5. View all movies page

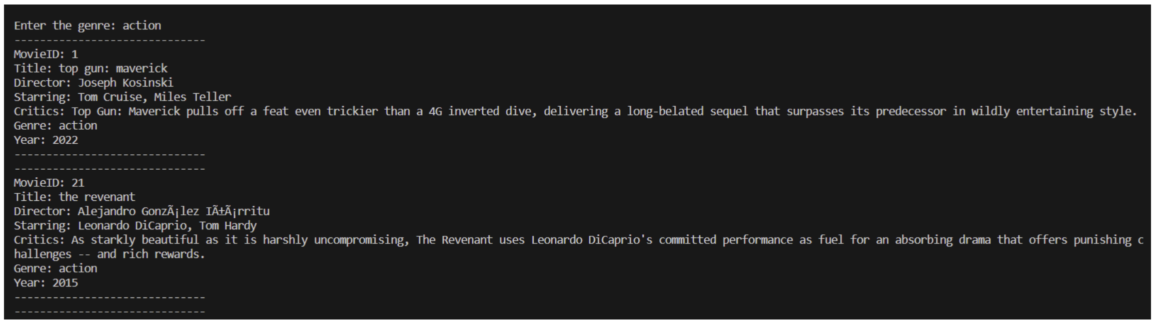


```
------------------------------
MovieID: 1
Title: top gun: maverick
Director: Joseph Kosinski
Starring: Tom Cruise, Miles Teller
Critics: Top Gun: Maverick pulls off a feat even trickier than a 4G inverted dive, delivering a long-belated sequel that surpasses its predecessor in wildly entertaining style.
Genre: action
Year: 2022
------------------------------
------------------------------
MovieID: 2
Title: everything everywhere all at once
Director: Dan Kwan, Daniel Scheinert
Starring:  Michelle Yeoh, Stephanie Hsu
Critics: Led by an outstanding Michelle Yeoh, Everything Everywhere All at Once lives up to its title with an expertly calibrated assault on the senses.
Genre: family
Year: 2022
------------------------------
------------------------------
MovieID: 3
Title: tar
Director: Todd Field
Starring: Cate Blanchett, NoÃ©mie Merlant
Critics: Led by the soaring melody of Cate Blanchett's note-perfect performance, TÃ¡r riffs brilliantly on the discordant side of fame-fueled power.
Genre: music
Year: 2022
```

6. Search Movies Page

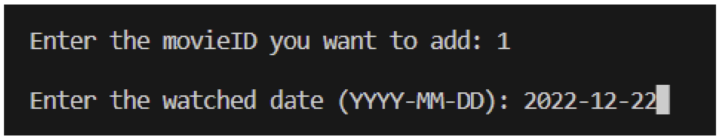


```
Please select an option below:
        1. Search by Title
        2. Search by Director
        3. Search by Genre
        4. Search by Year
Selection: 
```



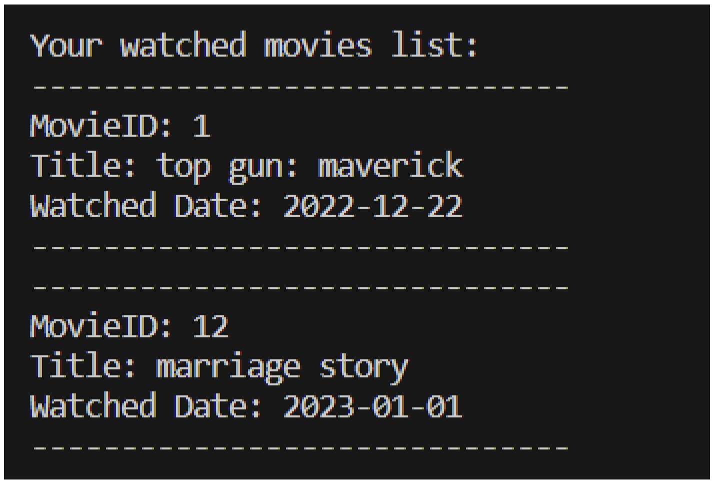
```
Enter the genre: action
------------------------------
MovieID: 1
Title: top gun: maverick
Director: Joseph Kosinski
Starring: Tom Cruise, Miles Teller
Critics: Top Gun: Maverick pulls off a feat even trickier than a 4G inverted dive, delivering a long-belated sequel that surpasses its predecessor in wildly entertaining style.
Genre: action
Year: 2022
------------------------------
------------------------------
MovieID: 21
Title: the revenant
Director: Alejandro GonzÃ¡lez IÃ±Ã¡rritu
Starring: Leonardo DiCaprio, Tom Hardy
Critics: As starkly beautiful as it is harshly uncompromising, The Revenant uses Leonardo DiCaprio's committed performance as fuel for an absorbing drama that offers punishing c
hallenges -- and rich rewards.
Genre: action
Year: 2015
------------------------------
------------------------------
```

After searching the movie, customers can add their watched movie list.



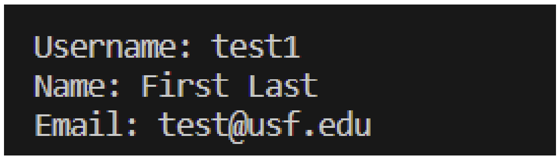
```
Enter the movieID you want to add: 1

Enter the watched date (YYYY-MM-DD): 2022-12-22
```

7. View your watched movie list Page



```
Your watched movies list:
------------------------------
MovieID: 1
Title: top gun: maverick
Watched Date: 2022-12-22
------------------------------
------------------------------
MovieID: 12
Title: marriage story
Watched Date: 2023-01-01
------------------------------
```

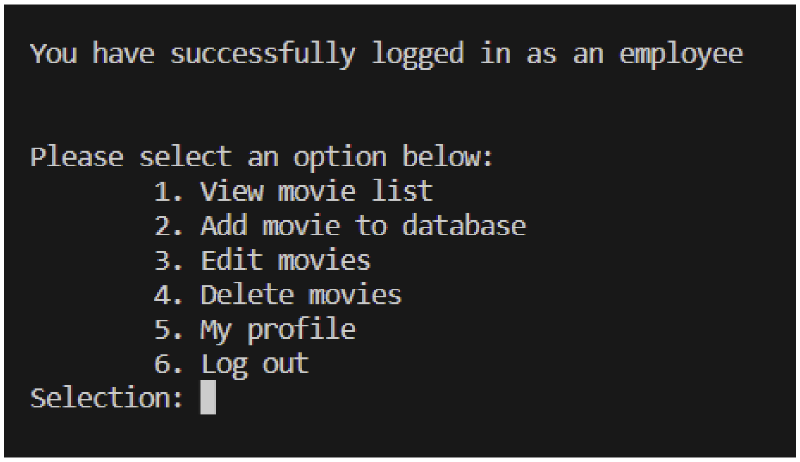
8. View personal information Page



```
Username: test1
Name: First Last
Email: test@usf.edu
```

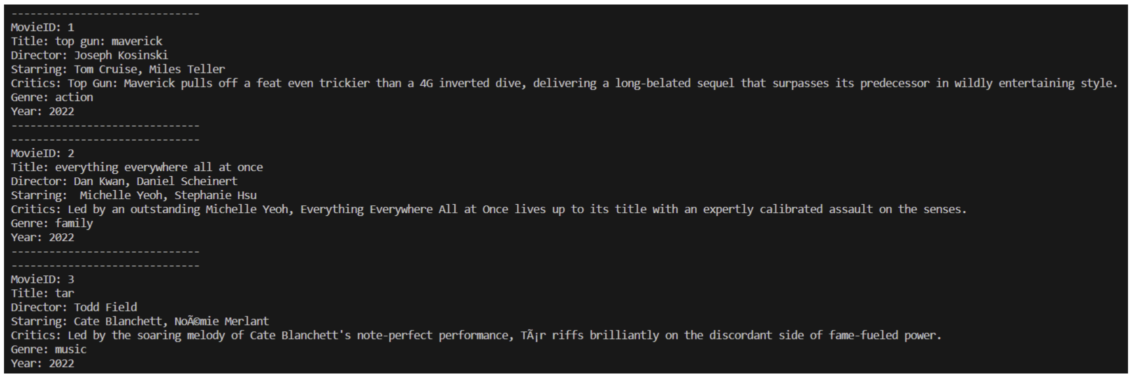
Employees Users

9. Employee Main Page



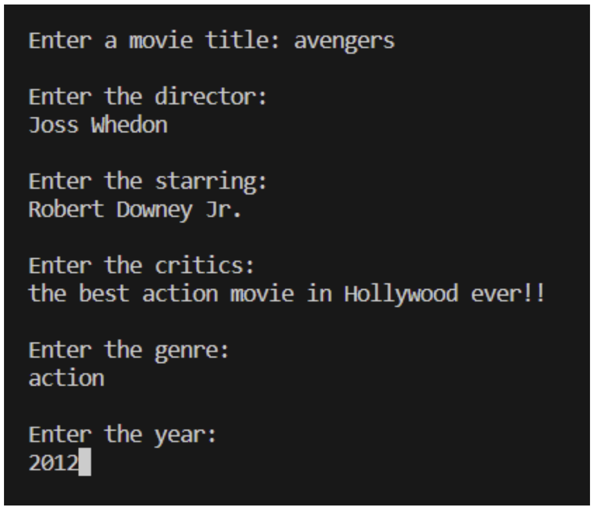
```
You have successfully logged in as an employee


Please select an option below:
        1. View movie list
        2. Add movie to database
        3. Edit movies
        4. Delete movies
        5. My profile
        6. Log out
Selection: 
```

10. View all movies Page



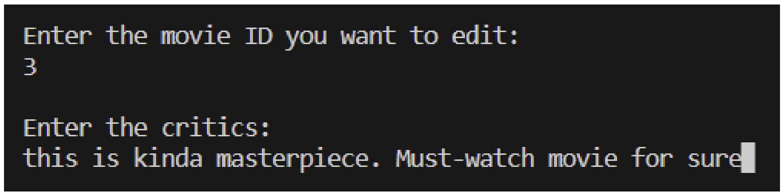
```
-----------------------------
MovieID: 1
Title: top gun: maverick
Director: Joseph Kosinski
Starring: Tom Cruise, Miles Teller
Critics: Top Gun: Maverick pulls off a feat even trickier than a 4G inverted dive, delivering a long-belated sequel that surpasses its predecessor in wildly entertaining style.
Genre: action
Year: 2022
-----------------------------
-----------------------------
MovieID: 2
Title: everything everywhere all at once
Director: Dan Kwan, Daniel Scheinert
Starring:  Michelle Yeoh, Stephanie Hsu
Critics: Led by an outstanding Michelle Yeoh, Everything Everywhere All at Once lives up to its title with an expertly calibrated assault on the senses.
Genre: family
Year: 2022
-----------------------------
-----------------------------
MovieID: 3
Title: tar
Director: Todd Field
Starring: Cate Blanchett, NoÃ©mie Merlant
Critics: Led by the soaring melody of Cate Blanchett's note-perfect performance, TÃ¡r riffs brilliantly on the discordant side of fame-fueled power.
Genre: music
Year: 2022
```

11. Add movie to database Page



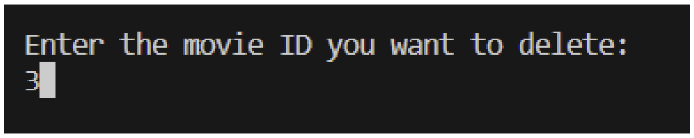
```
Enter a movie title: avengers

Enter the director:
Joss Whedon

Enter the starring:
Robert Downey Jr.

Enter the critics:
the best action movie in Hollywood ever!!

Enter the genre:
action

Enter the year:
2012
```

12. Edit movies Page



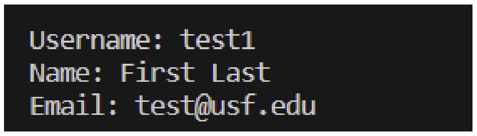
```
Enter the movie ID you want to edit:
3

Enter the critics:
this is kinda masterpiece. Must-watch movie for sure
```

13. Delete movies Page



```
Enter the movie ID you want to delete:
3
```

14. View personal information Page



```
Username: test1
Name: First Last
Email: test@usf.edu
```

5. Database Tables

This section describes what data we will store in our database as well as how our relations represent this data. The following tables will be included in our database. Please refer to the ER diagrams at the beginning of the document for a higher-level graphical description.

Users

users(u_userId: INTEGER, u_username: TEXT, u_password: TEXT, u_firstname: Text, u_lastname TEXT, u_email TEXT, u_isEmployee INTEGER)

- Primary key: u_userId
- Candidate key: u_username
- Not null: u_userId, u_username, u_password, u_firstname, u_lastname, u_email, u_isEmployee

This table stores all of the information associated with a given user. It is used to authenticate users attempting to login. A user's username is used by the "Watched" table to track a user's watched movie list.

Movies

movies(m_movieId: INTEGER, m_title: TEXT, m_director: TEXT, m_starring: TEXT, m_critics: TEXT, m_genre: TEXT, m_year: INTEGER)

- Primary key: m_movieId
- Not null: m_movieId, m_title

This table stores all of the information associated with movies. It is used to be viewed, added, edited, and deleted by users. A movie's id is used by the "Watched" table to track the user's watched movie list.

Watched

watched(w_watchedId: INTEGER, w_u_username: TEXT, w_m_movieId: INTEGER, w_date: TEXT)

- Primary key: w_watchedId
- Foreign key: w_u_username, w_m_movieId
- Not null: w_watchedId, w_u_username, w_m_movieId

This table stores all of the information associated with the watched movie list regarding all customers. It is used for customers to store and view their watched movies. A w_u_username and w_m_movieId are the foreign keys to track which user watched which movie.

6. SQL Queries

Create New Account

When a user creates a new account, the server checks if the desired username already exists in the database by running a query that counts the number of occurrences of the inputted username in the "users" table. If the count is greater than zero, it means that the username is already taken, and the user is prompted to select a new one.

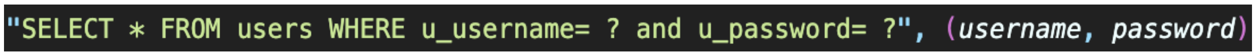
```
"SELECT COUNT(*) FROM users where u_username IS ?", (username,))
```

Once the user has chosen an available username, the server stores their account information in the "users" table using an INSERT query. This query takes the inputted username, password, first name, last name, email, and a boolean value indicating whether the user is an employee or not, and inserts this data as a new row in the "users" table.

```
INSERT INTO users (u_username, u_password, u_firstname, u_lastname, u_email, u_isEmployee) VALUES (?, ?, ?, ?, ?, ?)
, (username, password, firstname, lastname, email, isEmployee))
```

Login Page

When a user attempts to log in with a given username and password, the server searches for a matching record in the "users" table using a SELECT query. This query looks for a row in the table where the "u_username" column matches the inputted username and the "u_password" column matches the inputted password.

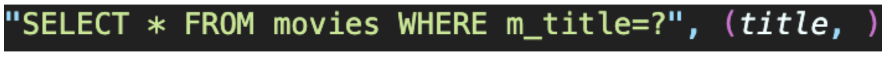


```
"SELECT * FROM users WHERE u_username= ? and u_password= ?", (username, password)
```

If the password matches, the user is considered authenticated and granted access to the requested resources. If the password does not match, the server denies access and prompts the user to try again with a correct username and password.

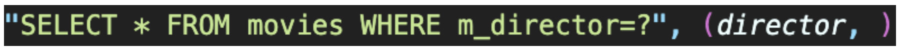
Search Movies

The customer can search movies in the “movies” table with 4 separate SELECT queries based on different criteria.
The first query searches for movies where the "m_title" column matches the inputted title. This allows users to search for a specific movie by its title.



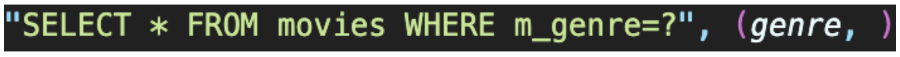
```
"SELECT * FROM movies WHERE m_title=?", (title, )
```

The second query searches for movies where the "m_director" column matches the inputted director. This allows users to search for all movies directed by a specific person.



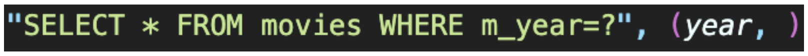
```
"SELECT * FROM movies WHERE m_director=?", (director, )
```

The third query searches for movies where the "m_genre" column matches the inputted genre. This allows users to search for all movies within a specific genre, such as action or comedy.



```
"SELECT * FROM movies WHERE m_genre=?", (genre, )
```

The fourth query searches for movies where the "m_year" column matches the inputted year. This allows users to search for all movies released in a specific year.

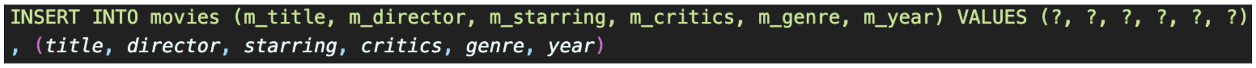


```
"SELECT * FROM movies WHERE m_year=?", (year, )
```

Manage Movies

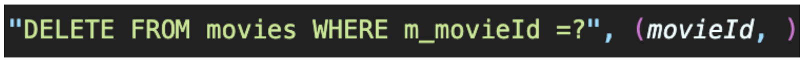
These are three separate queries for managing the "movies" table, intended for use by employees with the appropriate permissions.

The first query is an INSERT query that adds a new movie to the "movies" table. This query takes the title, director, starring, critics, genre, and release year and inserts this data as a new row in the "movies" table.



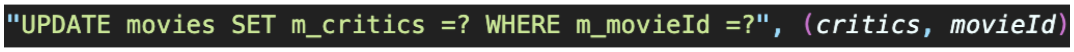
```
INSERT INTO movies (m_title, m_director, m_starring, m_critics, m_genre, m_year) VALUES (?, ?, ?, ?, ?, ?)
, (title, director, starring, critics, genre, year)
```

The second query is a DELETE query that removes a movie from the "movies" table based on its unique "m_movieId". This allows employees to remove movies that are no longer available.



```
"DELETE FROM movies WHERE m_movieId =?", (movieId, )
```

The third query is an UPDATE query that modifies the critics of a movie in the "movies" table based on its unique "m_movieId". This allows employees to update the critics of the movie.



```
"UPDATE movies SET m_critics =? WHERE m_movieId =?", (critics, movieId)
```

View Movie

This is a SELECT query that retrieves all rows from the "movies" table. When executed, this query returns a complete list of all movies in the table, along with their associated information such as title, director, starring, critics, genre, and release year.

```
"SELECT * FROM movies;"
```

This query is intended for use by both customers and employees to view the entire list of available movies in the system. Customers may use it to browse and select movies they would like to watch, while employees may use it to review the overall selection of movies in the system and make decisions about which movies to acquire or remove.

Watched Movie

These are two separate queries related to the functionality of saving and viewing watched movie lists for customers.

The first query is an INSERT query that adds a new movie to the "watched" table. This query takes the username, movieId, and watched date, and inserts this data as a new row in the "watched" table. This query is likely intended for use by customers who have watched a movie and want to record it in their watched movie list.

```
"INSERT INTO watched (w_u_username, w_m_movieId, w_date) VALUES (?, ?, ?)", (asId, movieId, date)
```

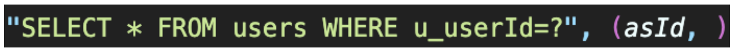
The second query is a SELECT query that retrieves a list of all movies watched by a particular customer. This query joins the "movies" and "watched" tables using the "m_movieId" and "w_m_movieId" columns, and selects the movie ID, title, and date watched for all movies that have been watched by the specified customer. This query is likely intended for use by customers who want to view their watched movie list and see when they watched each movie.

```
"""SELECT m.m_movieId, m.m_title, w.w_date
 FROM movies AS m JOIN watched AS w ON m.m_movieId = w.w_m_movieId
 WHERE w.w_u_username = ?""", (asId, )
```

When executed, this query returns a list of all movies that the specified customer has watched, along with the date that each movie was watched. This can be useful for customers who want to keep track of the movies they have seen or who want to make sure they don't accidentally re-watch a movie they've already seen.

View Profile

This is a SELECT query that retrieves a user profile from the "users" table based on a unique userId. When executed, this query returns all columns for the row in the "users" table where the "u_userId" column matches the inputted user ID. This query is likely intended for use by both customers and employees to view their own user profile information.



```
"SELECT * FROM users WHERE u_userId=?", (asId, )
```

7. Database Initialization Queries

The following 3 SQL queries are used to initialize the database tables:

Users

```
databaseCursor.execute('''CREATE TABLE IF NOT EXISTS users (
                            u_userId INTEGER PRIMARY KEY ASC,
                            u_username TEXT,
                            u_password TEXT,
                            u_firstname TEXT,
                            u_lastname TEXT,
                            u_email TEXT,
                            u_isEmployee INTEGER)''')
database.commit()
```

Movies

```
databaseCursor.execute('''Create TABLE IF NOT EXISTS movies (
                            m_movieId INTEGER PRIMARY KEY ASC,
                            m_title TEXT,
                            m_director TEXT,
                            m_starring TEXT,
                            m_critics TEXT,
                            m_genre TEXT,
                            m_year INTEGER)''')
database.commit()
```

Watched

```
databaseCursor.execute('''CREATE TABLE IF NOT EXISTS watched (
                                w_watchedId INTEGER PRIMARY KEY ACS,
                                w_u_username TEXT,
                                w_m_movieId INTEGER,
                                w_date TEXT,
                                FOREIGN KEY(w_u_username) REFERENCES
users(u_username),
                                FOREIGN KEY(w_m_movieId) REFERENCES
movies(m_movieId))''')
database.commit()
```