



MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

**Mini Project Report
of
Database Systems Lab (CSE 2262)**

Movie Booking Database System

**SUBMITTED
BY**

**Tejeswar Pokuri 220905236, CSE-B, 33
Devansh Desai, 220905368, CSE-B, 43**

**Department of Computer Science and Engineering
Manipal Institute of Technology, Manipal.
April 2024**



MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Manipal
15/04/2024

CERTIFICATE

This is to certify that the project titled **MiniProject Title** is a record of the bonafide work done by **Student(s) (Reg. Nos. 220905236, 220905368)** submitted in partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology (B.Tech.) in COMPUTER SCIENCE & ENGINEERING of Manipal Institute of Technology, Manipal, Karnataka, (A Constituent Institute of Manipal Academy of Higher Education), during the academic year 2022-2023.

Name and Signature of Examiners:

1. **Dr. Dinesh Acharya U, Professor, Department of Computer Science & Engineering**

TABLE OF CONTENTS

Abstract

Problem Statement

ER Diagram and Relational Tables

DDL Commands

List of SQL Queries

UI Design

Procedures/ Triggers

References

Abstract

Created a movie booking system that allows users to view details of any movie such as name, run time, genre and then book tickets by seeing available theaters, showtimes and seats. On the admin side, admins should be able to insert, update or delete movies and showtimes and manage and view the bookings made by the users.

Problem Statement

Data requirements

- To store movies and all their details such as name, run time, genre.
- To store showtimes available for movies.
- To store the available theatres.
- To store the seats available identified by the row and the theatre they belong to.
- To store users along with their login credentials, contact details, name.
- To store bookings along with all the necessary details such as movie, showtime, theatre, seat, and user.

Functional requirements

- To allow users and admins to login using their credentials.
- To allow admins to view and edit the available movies.
- To allow users to view available movies and all their details.
- To allow users to make bookings for movies by seeing the available showtimes, theatres.
- To allow admins to make bookings for movies by seeing the available showtimes, theatres.
- To prevent duplicate booking of seats for specific showtimes of movies.
- To allow users to view the bookings made by them.
- To allow admins to view and edit all the bookings.
- To allow admins to manually execute SQL queries to carry out specific data manipulation tasks.

Relational Tables and ER Diagram

Relational Tables

Our relational database design is in BCNF form.

Movies (movie-id, name, run-time, genre)

Bookings (booking-id, username, movie-id, date, time, row, seat, price)

Showtimes (movie-id, time)

Theatres (t-id, t-name, location)

Seats (t-name, row, seat)

Users (user-id, username, password, name, phone-no)

Admin (username, password)

ER Diagram

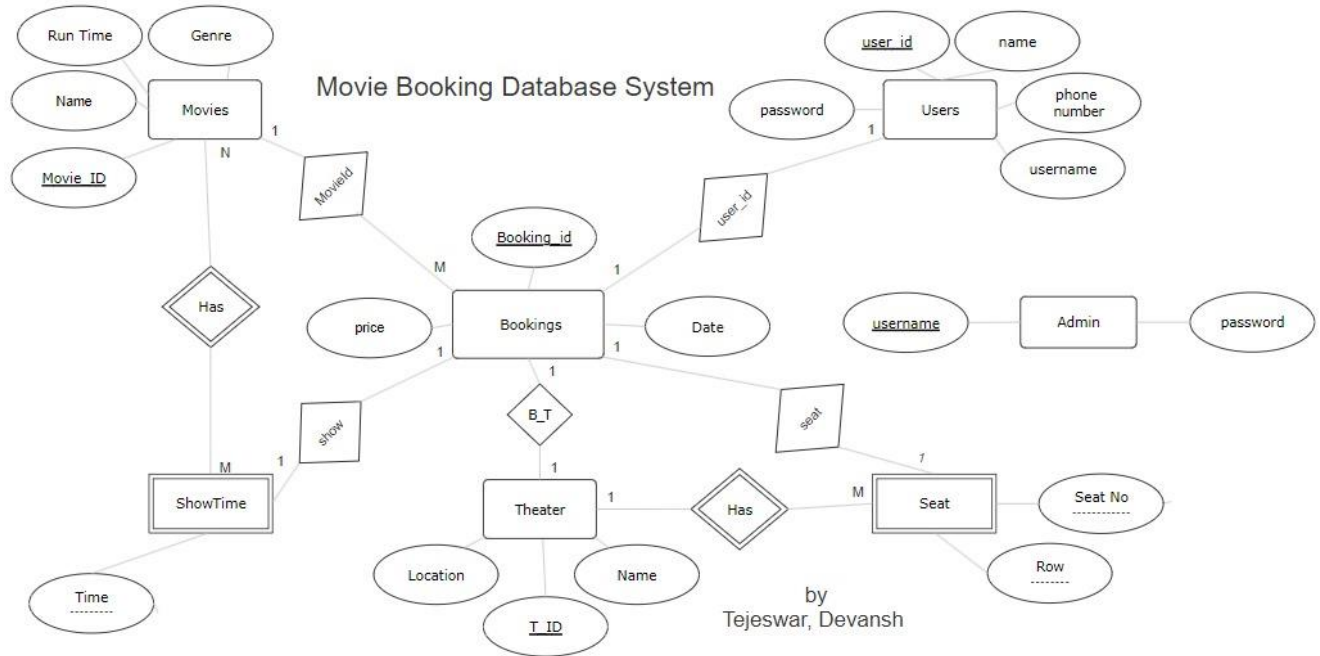


Fig 1 : ER Diagram

DDL Commands

```
CREATE TABLE IF NOT EXISTS movies (
    movie_id integer PRIMARY KEY AUTOINCREMENT,
    name text NOT NULL,
    run_time int NOT NULL,
    genre text NOT NULL
);
```

```
CREATE TABLE IF NOT EXISTS bookings (
    booking_id integer PRIMARY KEY AUTOINCREMENT,
    username text NOT NULL,
    movie_id integer NOT NULL,
    movie_name text NOT NULL,
    date text NOT NULL,
    time text NOT NULL,
    row text NOT NULL,
    seat int NOT NULL,
    price real NOT NULL,
    FOREIGN KEY (movie_id)
        REFERENCES movies (id)
        ON UPDATE SET NULL
        ON DELETE SET NULL
);
```

```

);

CREATE TABLE IF NOT EXISTS showtimes (
    movie_id integer NOT NULL,
    time text NOT NULL,
    FOREIGN KEY (movie_id)
        REFERENCES movies (id)
        ON UPDATE SET NULL
        ON DELETE SET NULL
);

CREATE TABLE IF NOT EXISTS theaters (
    t_name text PRIMARY KEY,
    location text
);

CREATE TABLE IF NOT EXISTS seats (
    t_name text,
    row text,
    seat text,
    FOREIGN KEY (t_name)
        REFERENCES theaters (t_name)
        ON UPDATE SET NULL
        ON DELETE SET NULL
);

CREATE TABLE IF NOT EXISTS users (
    user_id integer PRIMARY KEY AUTOINCREMENT,
    username text NOT NULL,
    password text NOT NULL,
    name text NOT NULL,
    ph text NOT NULL
);

```

List of Queries

Our project consists of multiple implicit SQL queries which are run by interaction with the UI elements. For eg: When a particular movie is selected, an SQL query is run to get the showtimes corresponding to that movie from the showtimes table. Then, only these showtimes are displayed in the dropdown box for the selection of show timing.

Our project also contains a run query text field that allows admins to run SQL queries by manually typing them into the text box. For the purpose of demonstration, we also came up with 6 SQL queries of varying complexity to be run using this method.

Implicit Queries

To Insert into movies

```
con.execute("INSERT INTO movies (name, run_time, genre) VALUES (?, ?, ?)",  
            (name, run_time, genre))
```

To return the movies ordered by one of the attributes.

```
cur.execute("SELECT * FROM movies ORDER BY {}".format(order))
```

To return the names of all the theatres

```
cur.execute("SELECT t_name FROM theaters ")
```

To return the showtimes corresponding to a selected movie name

```
cur.execute("SELECT time FROM showtimes WHERE movie_id = (SELECT movie_id FROM  
movies WHERE name = ?)", (movie,))
```

To return the rows available in a particular theatre

```
cur.execute("SELECT distinct row FROM seats WHERE t_name = ?", (theater,))
```

To return the seats available in a particular row of a particular theatre

```
cur.execute("SELECT seat FROM seats WHERE t_name = ? AND row = ?", (theater, row))
```

To delete a movie

```
con.execute("DELETE FROM movies WHERE movie_id = :id", {'id': movie_id})
```

To add a booking

```
cur.execute("""INSERT INTO bookings (username, movie_id, movie_name, date, time, row,  
seat, price, tname) VALUES (?, ?, ?, ?, ?, ?, ?, ?)""", (username, movie_id, movie, date, time,  
row, seat, price, tname))
```

To show all bookings

```
cur.execute("SELECT * FROM bookings ORDER BY {}".format(order))
```

To delete a particular booking

```
con.execute("DELETE FROM bookings WHERE booking_id=:id", {'id': booking_id})
```

To insert a new user

```
con.execute("INSERT INTO users (username, name, ph, password) VALUES (?, ?, ?, ?)",  
(username, name, ph, password))
```

To get a particular users password

```
cur.execute("""SELECT password FROM users WHERE username LIKE :username""",  
{'username': username})
```

To check if a particular username already exists

```
cur.execute("""SELECT username FROM users WHERE username LIKE :username""",  
{'username': username})
```

To update a user's password

```
cur.execute("""UPDATE logins SET password = :password WHERE username = :username""",
{'password': enc_password, 'username': username})
cur.execute("""SELECT password FROM admin
WHERE username LIKE :username""", {'username': username})
```

To display a particular user's bookings

```
cur.execute("SELECT * FROM bookings WHERE username = :id", {'id': username})
```

Note: The question marks inside insertion snippets such as **VALUES(?, ?)**, **(name, run_time, genre)** followed by variable names are the syntax specified by sqlite3 for the purpose of passing values of python variables to SQL insert statements.

Explicit Queries

Show users who have booked all the movies

```
SELECT u.username
FROM logins u
WHERE NOT EXISTS (
    SELECT DISTINCT m.movie_id
    FROM Movies m
    EXCEPT
    SELECT DISTINCT b.movie_id
    FROM Bookings b
    WHERE b.username = u.username
);
```

Show the movies in descending order of average booking price per seat

```
SELECT m.name AS movie_name, AVG(b.price) AS avg_price_per_seat
FROM Movies m
JOIN Bookings b ON m.movie_id = b.movie_id
GROUP BY m.name
ORDER BY avg_price_per_seat DESC;
```

Top 3 movies with highest total earnings

```
SELECT m.name AS movie_name, SUM(b.price) AS total_revenue
FROM Movies m
LEFT JOIN Bookings b ON m.movie_id = b.movie_id
GROUP BY m.name
ORDER BY total_revenue DESC
LIMIT 3;
```

Display all the movies having a particular genre

```
SELECT name FROM movies WHERE genre='Comedy';
```

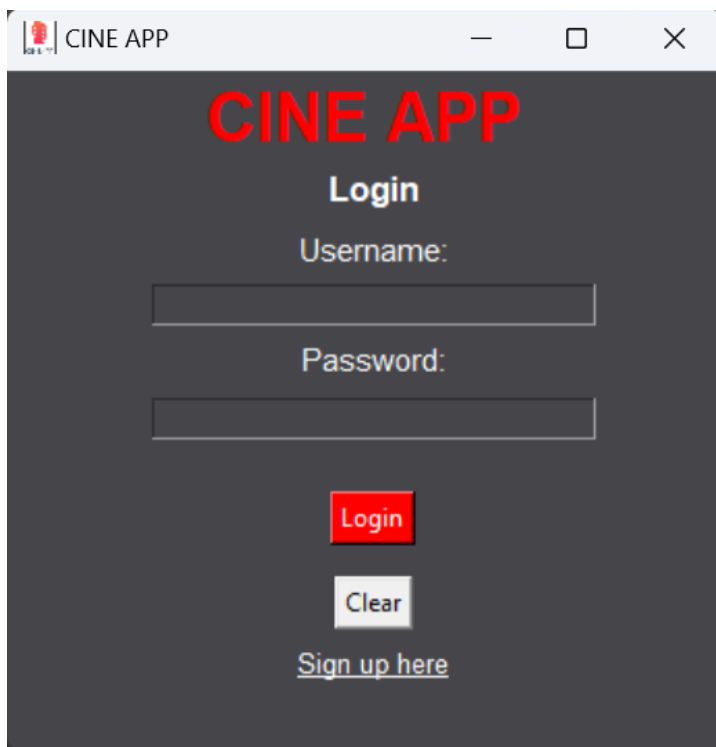
Show the total number of bookings for each movie

```
SELECT movie_name, count(*)
FROM bookings
GROUP BY movie_name;
```


To find the average runtime of each genre

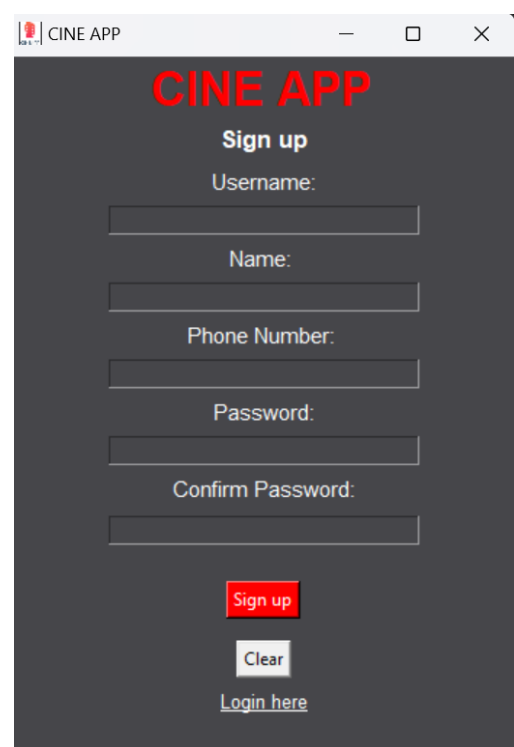
```
WITH AvgRuntimePerGenre AS (  
    SELECT genre, AVG(run_time) AS avg_runtime  
    FROM Movies  
    GROUP BY genre  
)  
SELECT genre, avg_runtime  
FROM AvgRuntimePerGenre  
ORDER BY avg_runtime DESC;
```

UI Design



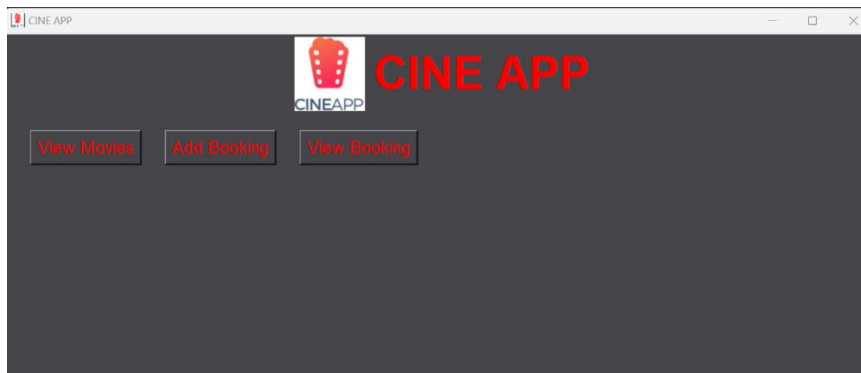
The login page features a dark gray background with the 'CINE APP' logo in red at the top. Below the logo, the word 'Login' is centered in white. There are two input fields: 'Username:' and 'Password:'. Below the password field is a red 'Login' button, a white 'Clear' button, and a link 'Sign up here'.

Fig 2: Login In page



The sign up page features a dark gray background with the 'CINE APP' logo in red at the top. Below the logo, the word 'Sign up' is centered in white. There are four input fields: 'Username:', 'Name:', 'Phone Number:', and 'Password:'. Below the password field is a 'Confirm Password:' field. At the bottom, there is a red 'Sign up' button, a white 'Clear' button, and a link 'Login here'.

Fig 3: Sign Up page



The user interface features a dark gray background with the 'CINE APP' logo in red at the top. Below the logo, there are three buttons: 'View Movies', 'Add Booking', and 'View Booking'.

Fig 4: User Interface



Fig 5: View Movies Interface



Fig 6: View Bookings Interface

CINE APP

View Movies Add Booking View Booking

Movie Name:

Choose Date:

April 2024

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
14	1	2	3	4	5	6	7
15	8	9	10	11	12	13	14
16	15	16	17	18	19	20	21
17	22	23	24	25	26	27	28
18	29	30	1	2	3	4	5
19	6	7	8	9	10	11	12

Select Date

Choose Time: Choose Theater:

Choose Row: Choose Seat:

☐ Adult ₹ 300
 ☐ Child ₹ 150
 ☐ Student ₹ 200

Submit Clear

Fig 7: Add Booking Interface

CINE APP

View Movies Add Movie View Bookings Add Booking Run Query

Fig 8 : Admin Interface

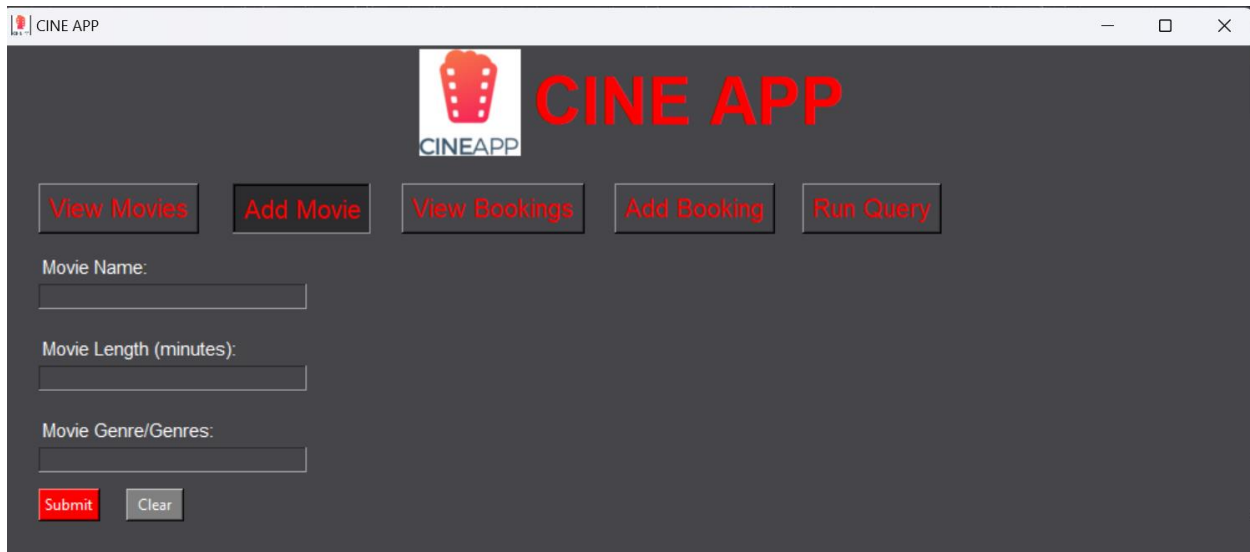


Fig 9 : Add movie Interface

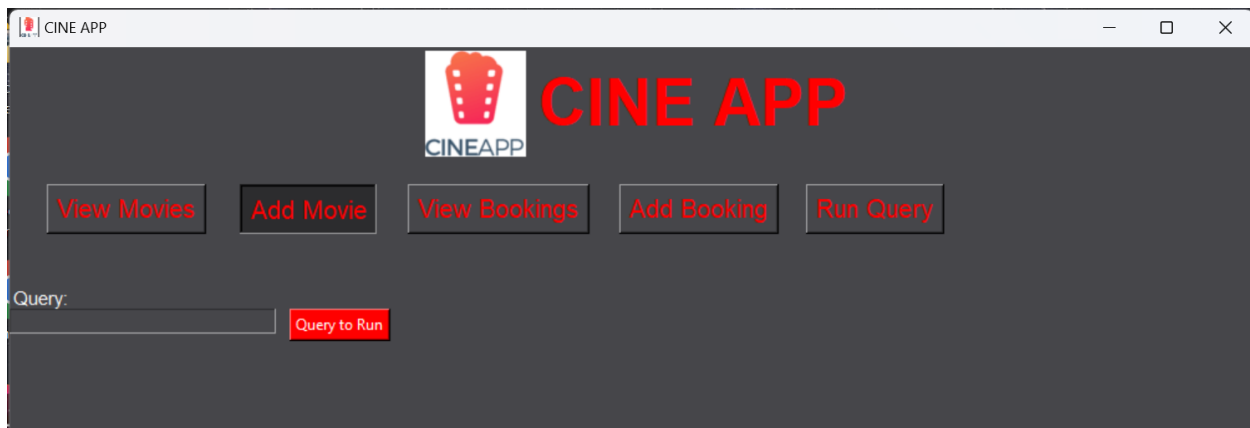


Fig 10: Query Interface

Note: The view movies, view bookings and add bookings Interface is same for both user and admin

Procedures / Triggers

To ensure that duplicate bookings of a seat for a particular showtime of a movie do not take place, we implemented a trigger. The trigger was defined as follows:

```
CREATE TRIGGER IF NOT EXISTS prevent_duplicate_booking
BEFORE INSERT ON bookings
FOR EACH ROW
BEGIN
    SELECT RAISE(ABORT, 'Seat already booked for this movie and time')
    FROM bookings
    WHERE movie_id = NEW.movie_id
    AND date = NEW.date
    AND time = NEW.time
```

```
AND row = NEW.row  
AND seat = NEW.seat;  
END;
```

We implemented several SQLite3 functions and procedures that get called by the UI elements. For eg: When a particular movie is selected, only the showtimes corresponding to that movie will be available in the showtimes dropdown. This is achieved by querying the showtimes table of the database in the background to return only the showtimes which correspond to that movie.

References

<https://docs.python.org/3/library/sqlite3.html>

<https://docs.python.org/3/library/tkinter.html>

<https://www.youtube.com/watch?v=YXPyB4XeYLA>

<https://www.youtube.com/watch?v=byHcYRpMgl4>

<https://www.youtube.com/watch?v=TY6RDEG9bhw>

<https://pypi.org/project/simple-crypt/>

<https://www.smartdraw.com/entity-relationship-diagram/er-diagram-tool.htm>

<https://www.youtube.com/watch?v=uQrJ0TkZlc>