Movie Theater Database

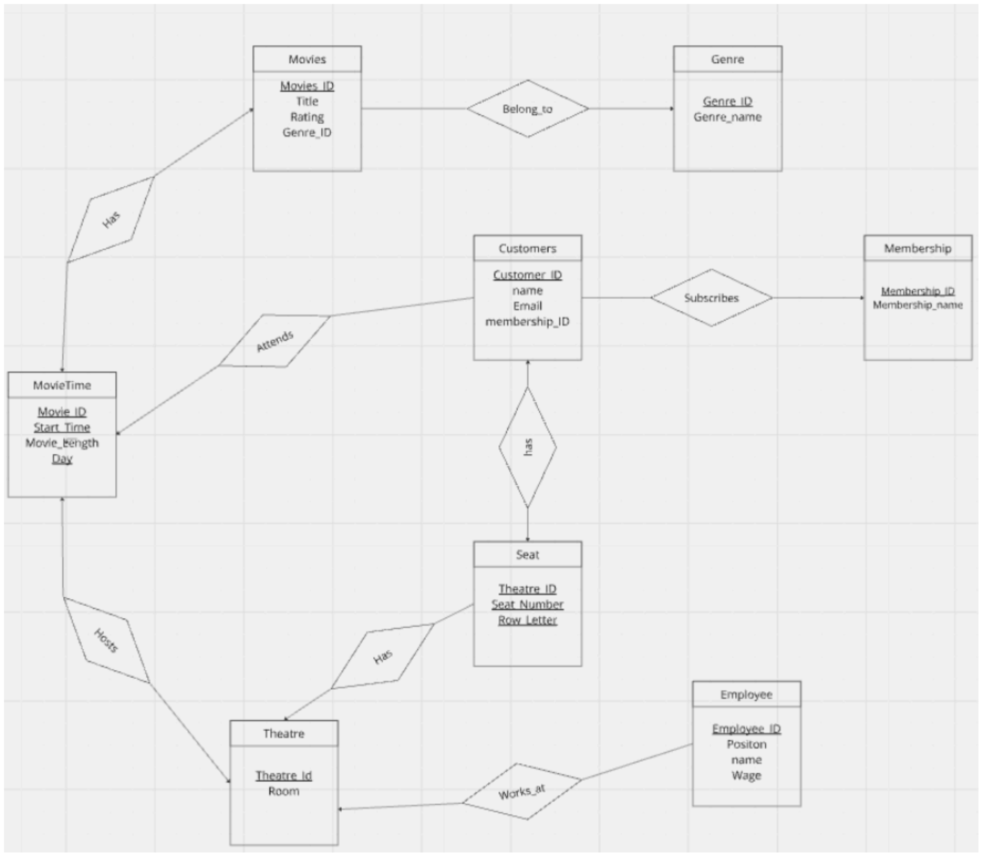
Serene Qasem, Adam Azmi, Chloe Gray, Skyler Dowling, Joseph Aycock, Brett Bradley

Enterprise

The enterprise selected for our project is a movie theater committed to delivering a fully functional cinematic experience. It is meant to be business and user friendly by screening a wide range of movies across many genres while managing scheduling, seating arrangements, customer memberships, and employee roles, and movie ratings. The theater provides customers with membership plans that give exclusive benefits, such as Basic, Premium, and Super Premium. Functionality is important to make sure things run smoothly, so we needed a strong database to manage everything. The idea of this is to store and manage data efficiently by using a relational database. We wanted a flexible query for the business such as when movies are available on a specific day, customer membership details, and employees earning above a specific wage.

Customer management is important for maintaining their details such as their membership plans, emails, and names which can be determined from their unique ID. It is important to have movie scheduling with flexible start times and durations, avoiding duplicates occurring at the same time, so we managed movie time and day data accordingly. Theater and seating arrangements are important for multiple theaters, each with unique room numbers and seating configurations with letters and numbers. Employee management is essential for tracking each person's positions, whether they be a cashier, manager, or usher. Their wages are important for payroll purposes so it is important to maintain that data.

E-R Diagram



Relational Schema

- Movies (movies_ID, title, rating, genre_ID)

movies_ID: Unique identifier for each movie; title: The title of the movie; rating: The movie's rating; genre_ID: Links to the genre of the movie from the **Genre** table.

- Genre (genre_ID, genre_name)

genre_ID: Unique identifier for each genre; genre_name: The name of the genre.

- Employee (employee_ID, position, name, wage)

employee_ID: Unique identifier for each employee; position: The employee’s job title or role; name: The employee's name; wage: The employee's hourly wage or salary.

- Theater (theater_ID, room)

theater_ID: Unique identifier for each theater; room: The specific room or hall within the theater.

- Membership (membership_ID, membership_name)

membership_ID: Unique identifier for each membership type; membership_name: The name of the membership.

- Seat (theater_ID, seat_num, row_letter)

theater_ID: Identifies the theater room; seat_num: The seat number in the room; row_letter: The letter corresponding to the row.

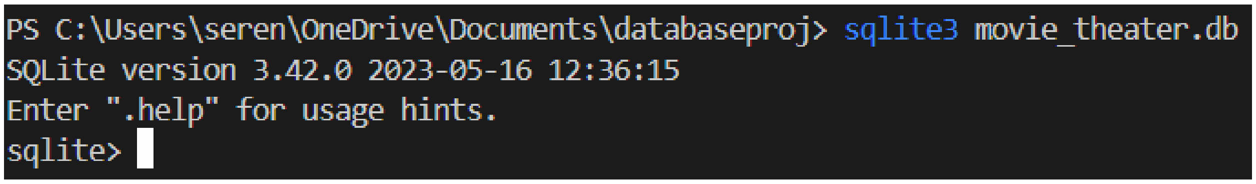
- Customer (customer_ID, name, email, membership_ID)

customer_ID: Unique identifier for each customer; name: The customer’s name; email: The customer’s email address; membership_ID: Links to the **Membership** table, indicating the customer's membership type.

- MovieTime (movie_ID, start_time, movie_length, day)

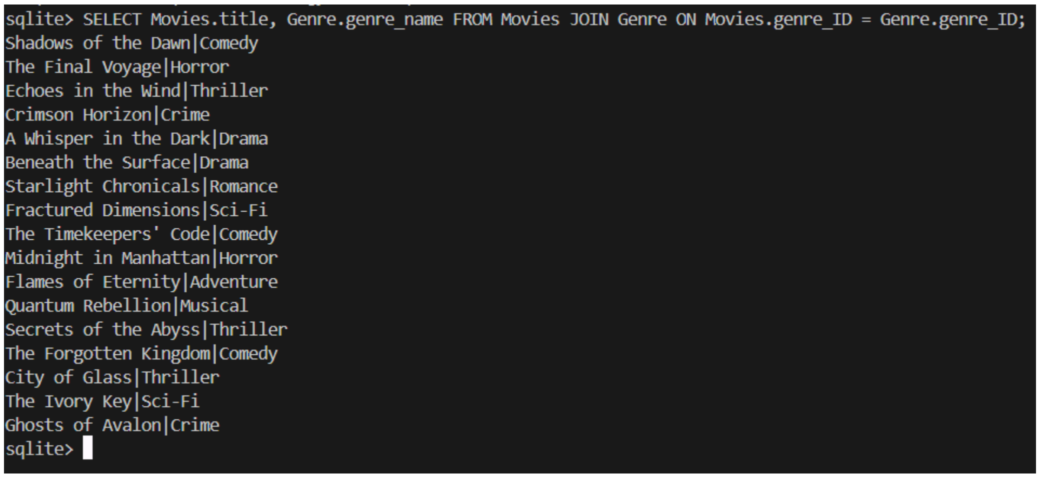
movie_ID: Links to the **Movies** table, indicating which movie is being shown; start_time: The time the movie starts; movie_length: The duration of the movie; day: The day of the week when the movie is shown.

Test Queries



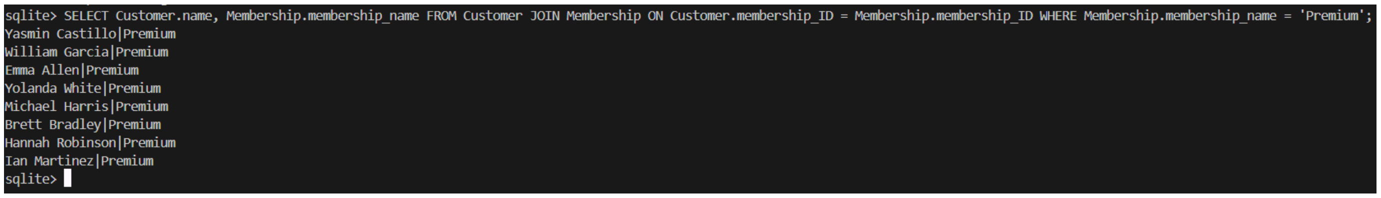
```
PS C:\Users\seren\OneDrive\Documents\databaseproj> sqlite3 movie_theater.db
SQLite version 3.42.0 2023-05-16 12:36:15
Enter ".help" for usage hints.
sqlite>
```

This is the first step for triggering the prompt to test queries. Make sure you put a semicolon after every query you want to test/run!



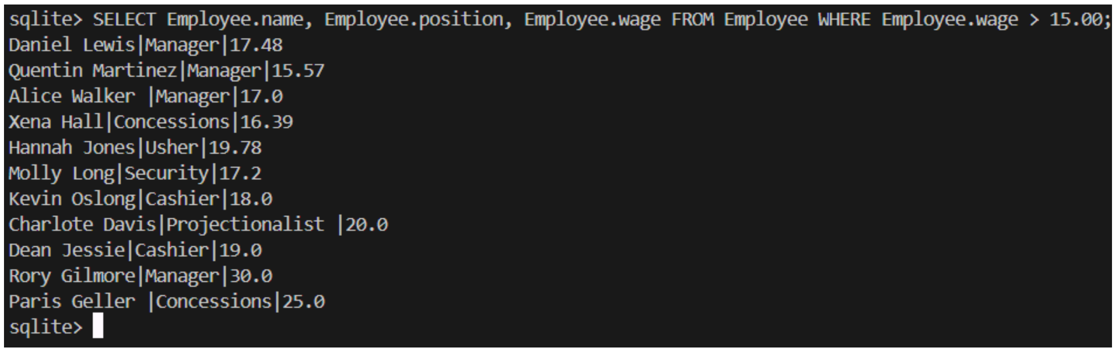
```
sqlite> SELECT Movies.title, Genre.genre_name FROM Movies JOIN Genre ON Movies.genre_ID = Genre.genre_ID;
Shadows of the Dawn|Comedy
The Final Voyage|Horror
Echoes in the Wind|Thriller
Crimson Horizon|Crime
A Whisper in the Dark|Drama
Beneath the Surface|Drama
Starlight Chronicals|Romance
Fractured Dimensions|Sci-Fi
The Timekeepers' Code|Comedy
Midnight in Manhattan|Horror
Flames of Eternity|Adventure
Quantum Rebellion|Musical
Secrets of the Abyss|Thriller
The Forgotten Kingdom|Comedy
City of Glass|Thriller
The Ivory Key|Sci-Fi
Ghosts of Avalon|Crime
sqlite>
```

Query #1: Lists all the movies and their corresponding genre



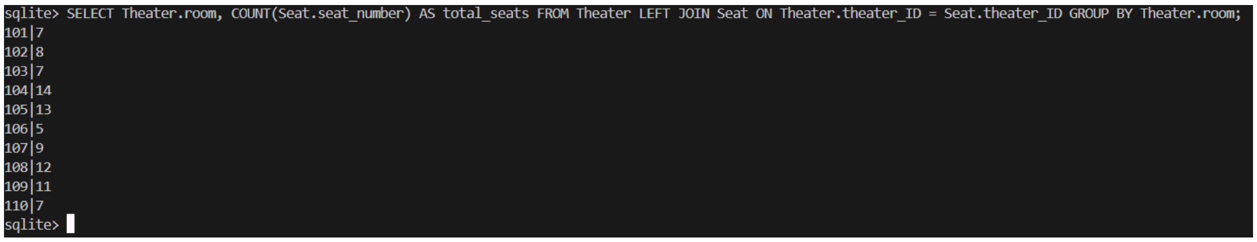
```
sqlite> SELECT Customer.name, Membership.membership_name FROM Customer JOIN Membership ON Customer.membership_ID = Membership.membership_ID WHERE Membership.membership_name = 'Premium';
Yasmin Castillo|Premium
William Garcia|Premium
Emma Allen|Premium
Yolanda White|Premium
Michael Harris|Premium
Brett Bradley|Premium
Hannah Robinson|Premium
Ian Martinez|Premium
sqlite>
```

Query #2: Lists all customers with a Premium membership



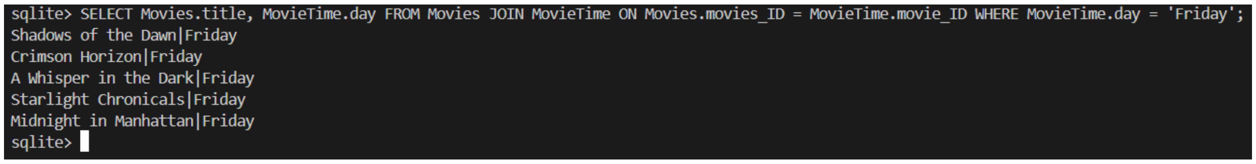
```
sqlite> SELECT Employee.name, Employee.position, Employee.wage FROM Employee WHERE Employee.wage > 15.00;
Daniel Lewis|Manager|17.48
Quentin Martinez|Manager|15.57
Alice Walker |Manager|17.0
Xena Hall|Concessions|16.39
Hannah Jones|Usher|19.78
Molly Long|Security|17.2
Kevin Oslong|Cashier|18.0
Charlote Davis|Projectionalist |20.0
Dean Jessie|Cashier|19.0
Rory Gilmore|Manager|30.0
Paris Geller |Concessions|25.0
sqlite>
```

Query #3: Lists all employees earning a wage of above $15



```
sqlite> SELECT Theater.room, COUNT(Seat.seat_number) AS total_seats FROM Theater LEFT JOIN Seat ON Theater.theater_ID = Seat.theater_ID GROUP BY Theater.room;
101|7
102|8
103|7
104|14
105|13
106|5
107|9
108|12
109|11
110|7
sqlite>
```

Query #4: Gets all the theaters and the number of seats in each theater

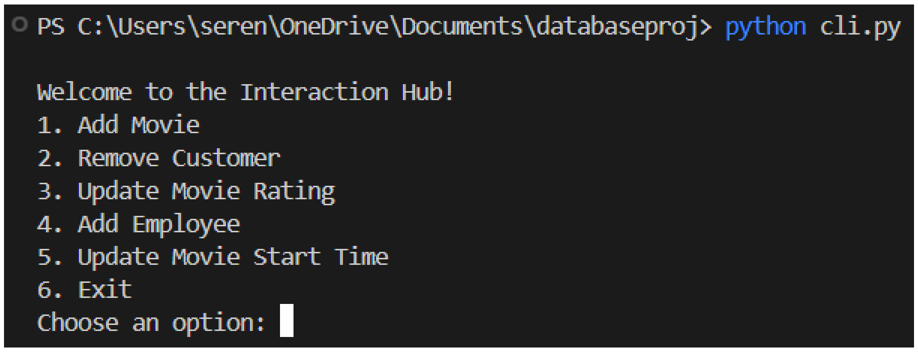


```
sqlite> SELECT Movies.title, MovieTime.day FROM Movies JOIN MovieTime ON Movies.movies_ID = MovieTime.movie_ID WHERE MovieTime.day = 'Friday';
Shadows of the Dawn|Friday
Crimson Horizon|Friday
A Whisper in the Dark|Friday
Starlight Chronicals|Friday
Midnight in Manhattan|Friday
sqlite>
```

Query #5: Lists all the movies being shown on a Friday

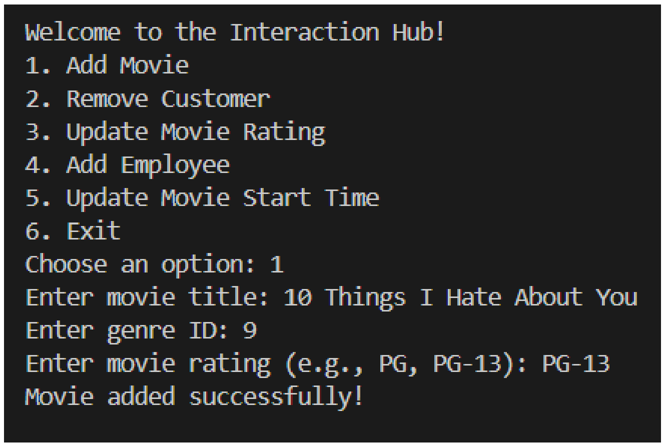
Command Line Interface(CLI)

This is where there will be a set of application user interfaces with illustrations. In the Movie Theater Database, you are given 5 options to pick from in the Interaction Hub area. You can either Add a movie, Remove a Customer, Update a Movie Rating, Add an Employee, or Update a Movie Start Time. Seen in cli.py, the backend for inserting, updating, and deleting is coded there.



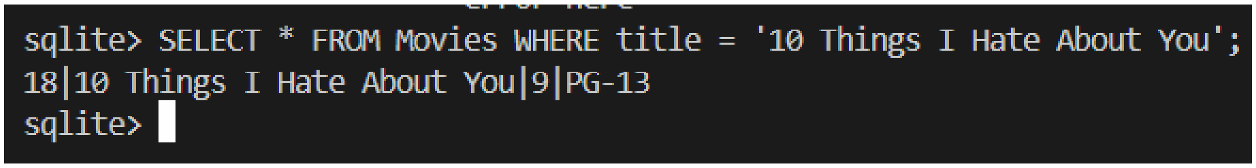
```
PS C:\Users\seren\OneDrive\Documents\databaseproj> python cli.py

Welcome to the Interaction Hub!
1. Add Movie
2. Remove Customer
3. Update Movie Rating
4. Add Employee
5. Update Movie Start Time
6. Exit
Choose an option:
```

First, run the ‘python cli.py’ script to go to the Interaction Hub. You can pick a number from here based on what you want to do.

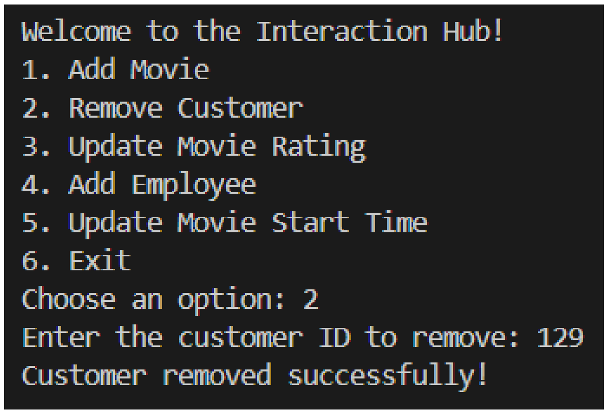


```
Welcome to the Interaction Hub!
1. Add Movie
2. Remove Customer
3. Update Movie Rating
4. Add Employee
5. Update Movie Start Time
6. Exit
Choose an option: 1
Enter movie title: 10 Things I Hate About You
Enter genre ID: 9
Enter movie rating (e.g., PG, PG-13): PG-13
Movie added successfully!
```

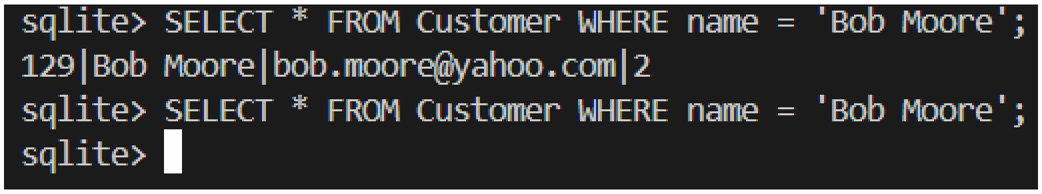
In this example, we are adding a movie, 10 Things I Hate About You. The movie gets successfully added here. You can run a query to make sure that the movie we added in this example is there.



```
sqlite> SELECT * FROM Movies WHERE title = '10 Things I Hate About You';
18|10 Things I Hate About You|9|PG-13
sqlite> 
```

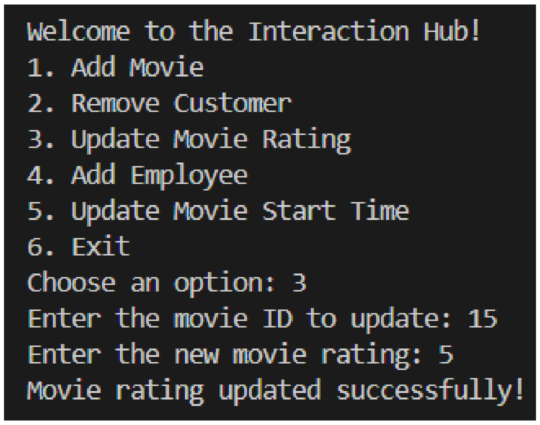


```
Welcome to the Interaction Hub!
1. Add Movie
2. Remove Customer
3. Update Movie Rating
4. Add Employee
5. Update Movie Start Time
6. Exit
Choose an option: 2
Enter the customer ID to remove: 129
Customer removed successfully!
```



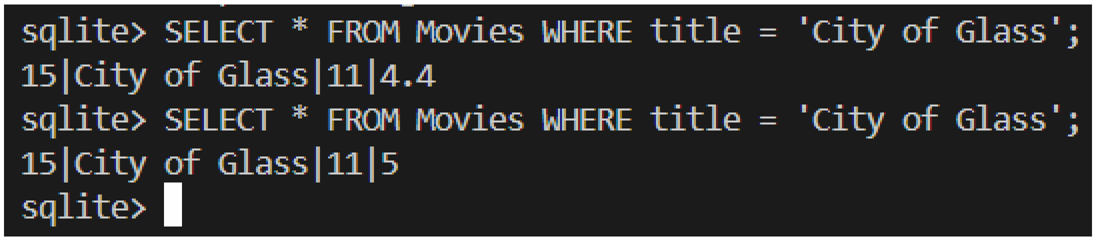
```
sqlite> SELECT * FROM Customer WHERE name = 'Bob Moore';
129|Bob Moore|bob.moore@yahoo.com|2
sqlite> SELECT * FROM Customer WHERE name = 'Bob Moore';
sqlite> 
```

Here, we are removing a customer. If you look at the first query I ran, we have Bob Moore present in the database. We then removed Bob and ran the query again, returning empty.

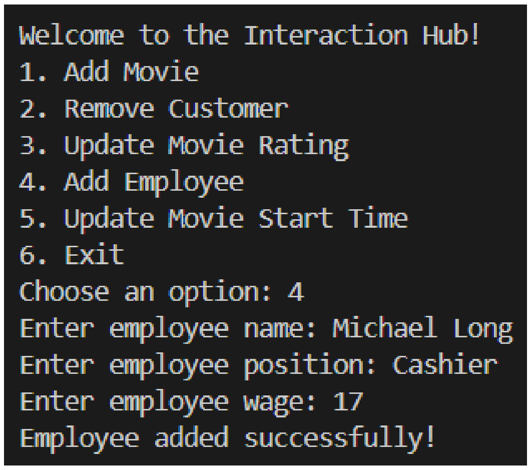


```
Welcome to the Interaction Hub!
1. Add Movie
2. Remove Customer
3. Update Movie Rating
4. Add Employee
5. Update Movie Start Time
6. Exit
Choose an option: 3
Enter the movie ID to update: 15
Enter the new movie rating: 5
Movie rating updated successfully!
```

Here, we are updating a movie's rating. You can see in the first query that the rating is 4.4 for City of Glass, after updating and rerunning the query, it is now a 5.

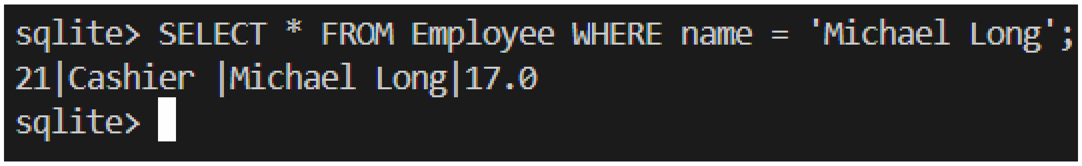


```
sqlite> SELECT * FROM Movies WHERE title = 'City of Glass';
15|City of Glass|11|4.4
sqlite> SELECT * FROM Movies WHERE title = 'City of Glass';
15|City of Glass|11|5
sqlite>
```

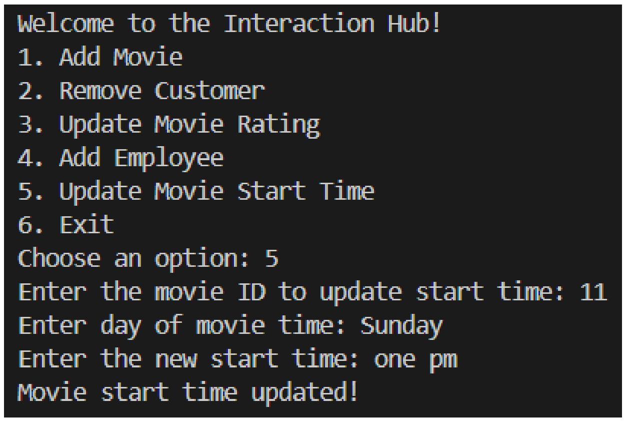


```
Welcome to the Interaction Hub!
1. Add Movie
2. Remove Customer
3. Update Movie Rating
4. Add Employee
5. Update Movie Start Time
6. Exit
Choose an option: 4
Enter employee name: Michael Long
Enter employee position: Cashier
Enter employee wage: 17
Employee added successfully!
```

I am adding an Employee here named Michael Long. Ran this query below to make sure it got added to the database.

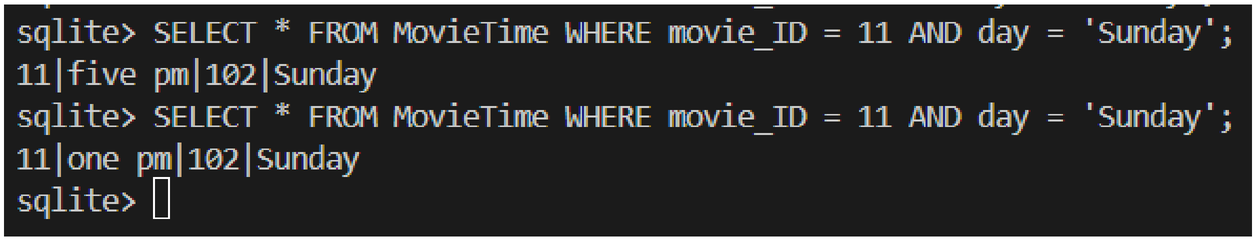


```
sqlite> SELECT * FROM Employee WHERE name = 'Michael Long';
21|Cashier |Michael Long|17.0
sqlite>
```



```
Welcome to the Interaction Hub!
1. Add Movie
2. Remove Customer
3. Update Movie Rating
4. Add Employee
5. Update Movie Start Time
6. Exit
Choose an option: 5
Enter the movie ID to update start time: 11
Enter day of movie time: Sunday
Enter the new start time: one pm
Movie start time updated!
```

In the last example here, we are updating the movie time. I also ran a before and after query, similar process to what's also going on in the other interactions!



```
sqlite> SELECT * FROM MovieTime WHERE movie_ID = 11 AND day = 'Sunday';
11|five pm|102|Sunday
sqlite> SELECT * FROM MovieTime WHERE movie_ID = 11 AND day = 'Sunday';
11|one pm|102|Sunday
sqlite>
```

Participation

Serene Qasem: Developed the code for creating the database schema (setup_database.py) and developed the command line interface (cli.py) for user interaction. Created the SQL file according to the Movie Theater's file data (create_theater.sql). Created and tested the 5 queries as well as the 5 user application options, and included their illustrations in this report.
Adam Azmi: Developed the population of the Movie Theater's data on Excel (MovieDataBase.xlsx) with table names as individual sheets and their variables. Created our E-R Diagram and put it in this report.
Skyler Dowling: Developed the code for the population of the database with data, inserting data from the Excel file into the database (generate_data.py).
Chloe Gray: Wrote up enterprise to the report.
Joseph Aycock: Contributed to the Excel file creation and relational schema
Brett Bradley: Built Relational schema that is also included in the report.