جامعة عفت
EFFAT UNIVERSITY

CS 2071 - Database Systems

Database Project Report - Spring 2022

Group Members:
Jana Abu Hantash (Section 1)
Ahmad Elmaamoun (Section 2)
Flowra Almufadda (Section 2)

Instructor: **Dr. Zain Balfagih**

Date Last Edited: January 1, 2024

Contents

1 Introduction

Our database aims to make an e-commerce platform to sell online tickets for a movie theater. Our database helps movie theaters to store information about movies, their showtimes, user accounts, employees, tickets, booking information, and more. To design this database, we first decided on the main entities in our database, and then we worked on their attributes and created relations between them. Next, we started building the tables associated with those entities. Finally, we analyzed our database for errors and adjusted the design of the ER diagram and schema table.

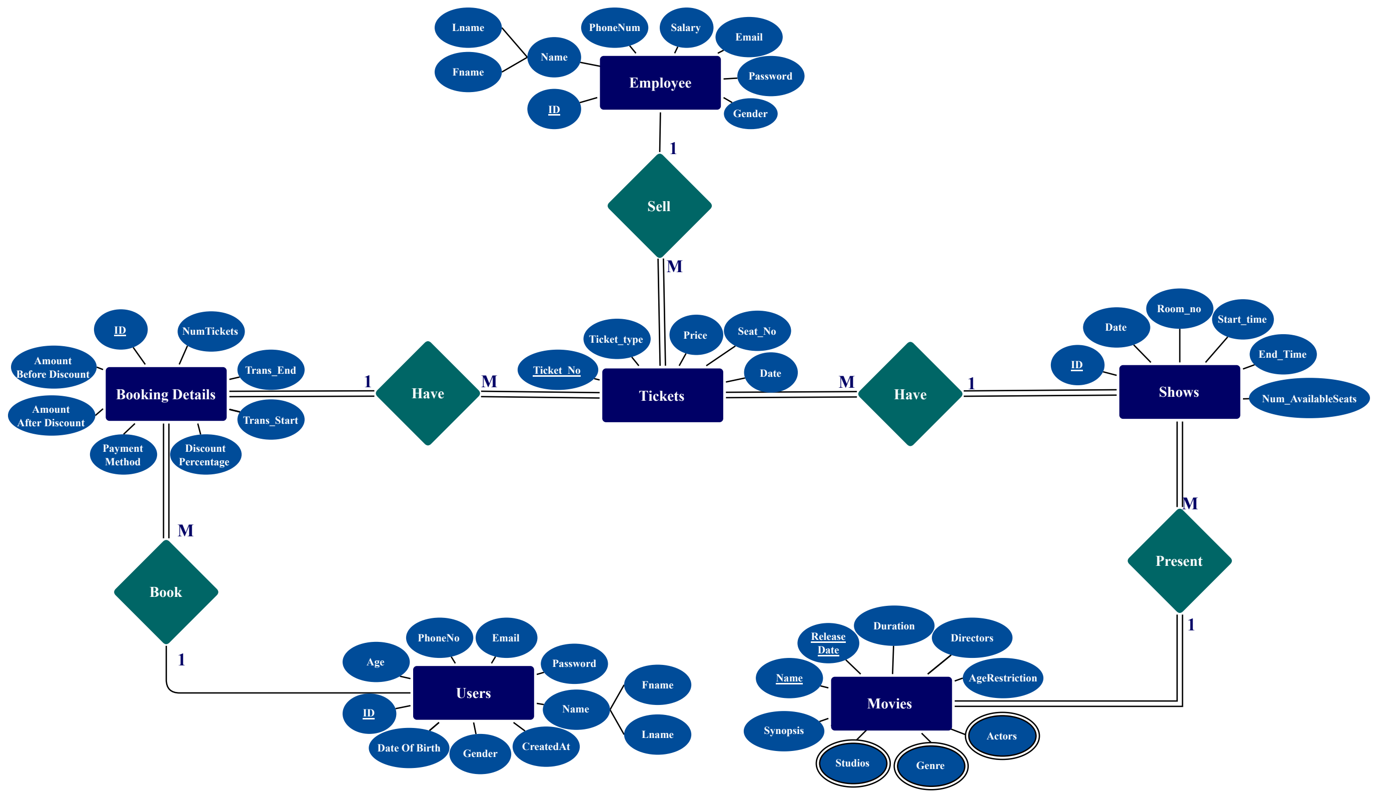
2 Database

2.1 Description of the ER Diagram

We created a movie theater database to record data about movies, their showtimes, user accounts, employees, tickets, booking information, and more. The Data required for our movie theater database are summarized as follows:

- Every employee is responsible for selling tickets to users. Employees are uniquely identified by their ID. For each employee in our movie theater, we will store their first name, last name, email, phone number, salary, password, and gender in our database. Each employee can sell more than one ticket, and each ticket is sold by one employee.
- Every employee is responsible for selling tickets to users. Employees are uniquely identified by their ID. For each employee in our movie theater, we will store their first name, last name, email, phone number, salary, password, and gender in our database. Each employee can sell more than one ticket, and each ticket is sold by one employee.
- Furthermore, our movie theater database will record in the users' accounts: their unique ID, email, first name, last names, date of birth, age, password, phone number, gender, and time the account was created.
- To save all user payments, we designed the booking details entity. The attributes we used to store the booking details are unique ID, the amount before discount, the amount after discount, number of tickets bought, when the transaction starts and ends, payment method, and percentage of the discount. Every Booking detail will record information about one user, and users can have more than one booking information.
- In our database, we created an entity responsible for saving all showtimes of the different movies presented. Every showtime record holds a unique id, date, room number, start time, end time, and the number of available seats.
- Employees sell tickets for movies having different showtimes. Every ticket has specific showtime, and every showtime can have more than one ticket. Information saved about the tickets sold to users includes its unique id, price, date, seat number, and ticket type.
- We will also save details about all movies presented in our movie theater at different showtimes. The information stored for movies in our movie theater database: movie name, year released, duration, directors, age restriction, synopsis genre, actors, and studios. Every movie is uniquely identified by its name and the year it was released.

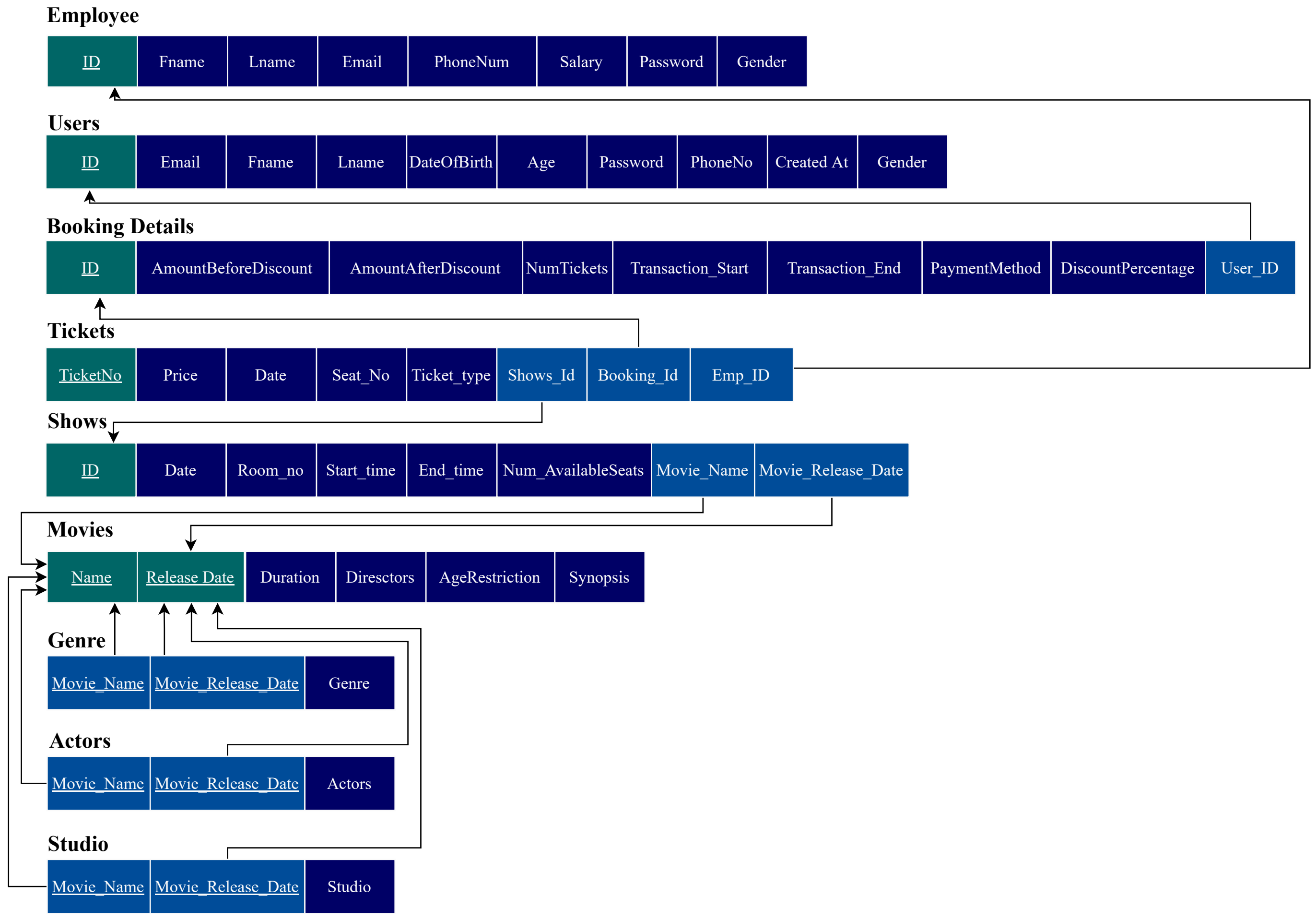
2.2 ER Diagram



2.3 Normalization

Our database's tables are for employees, user accounts, booking details, tickets, shows, movies, genres, actors, and studios. Since genres, actors, and studios are multi-valued attributes, we created separate tables for them. To avoid violating the first normal form, we ensured we didn't have any composite or multi-valued attributes. Our database doesn't violate the second normal forms since all non-prime attributes are fully functionally dependent on each primary key. Our database doesn't violate the third normal form since all non-prime attributes are not transitively dependent on their primary keys. For example, all the employee entity attributes functionally depend on the determinant employee ID. Since we didn't violate the first, second, or third normal form, our database tables are in the third normal form.

2.4 Relational Schema



3 Functionality

3.1 Basic Functions

The basic functions we used in our database include SQL commands that create tables, insert values, and alter tables. We first started by creating all the tables we specified in the schema table, and for each table, we made sure we identified all the primary keys and the foreign keys. We then inserted appropriate values into every table.

3.2 Tables Created in our Movie Theater Database

```sql
CREATE TABLE Employee(
ID INT,
Fname VARCHAR(20) ,
Lname VARCHAR(20),
Gender VARCHAR(2) ,
Email VARCHAR(40),
PhoneNum INT,
Salary INT,
Password VARCHAR(20),
PRIMARY KEY (ID));
```

```sql
CREATE TABLE Users(
User_ID INT,
Email VARCHAR(30),
Fname VARCHAR(20) ,
Lname VARCHAR(20),
Gender VARCHAR(2) ,
Date_of_Birth DATE,
Age INT,
Password VARCHAR(20),
PhoneNum INT,
PRIMARY KEY (User_ID));
```

```sql
CREATE TABLE Booking_details(
Booking_ID INT,
AmountBeforeDiscount INT,
AmountAfterDiscount INT,
Number_Of_Tickets INT,
Transaction_Start DATE,
Transaction_End DATE,
Payment_Method VARCHAR(20),
User_ID INT,
DiscountPercentage decimal(3,2),
PRIMARY KEY (Booking_ID),
FOREIGN KEY (User_ID) REFERENCES Users(User_ID));
```

```sql
CREATE TABLE Shows(
Show_ID INT,
Date DATE,
Room_Num VARCHAR(5),
Start_Time TIME,
End_Time TIME,
Number_Available_Seats INT,
Movie_Name VARCHAR(50),
Movie_Release_Date DATE,
PRIMARY KEY(Show_ID),
FOREIGN KEY (Movie_Name, Movie_Release_Date)
    REFERENCES Movies(Name, Release_Date));
```

```sql
CREATE TABLE Tickets(
Ticket_No INT,
Price INT,
Date DATE,
Seat_No VARCHAR(2),
Ticket_Type VARCHAR(20),
Show_ID INT,
Booking_ID INT,
Emp_ID INT,
PRIMARY KEY (Ticket_No),
FOREIGN KEY (Show_ID) REFERENCES shows(Show_ID),
FOREIGN KEY (Booking_ID) REFERENCES Booking_details(Booking_ID),
FOREIGN KEY (Emp_ID) REFERENCES Employee(ID));
```

```sql
CREATE TABLE Movies(
Name VARCHAR(50),
Release_Date DATE,
Duration TIME,
Directors VARCHAR(30),
Age_Restriction VARCHAR(10),
Synopsis VARCHAR(350),
PRIMARY KEY (Name, Release_Date));
```

```sql
CREATE TABLE Genres(
Movie_Name VARCHAR(50),
Movie_Release_Date DATE,
Genre VARCHAR(20),
PRIMARY KEY (Movie_Name, Movie_Release_Date),
FOREIGN KEY (Movie_Name, Movie_Release_Date) REFERENCES Movies(Name, Release_Date));


CREATE TABLE Actors(
Movie_Name VARCHAR(50),
Movie_Release_Date DATE,
Actor VARCHAR(30),
PRIMARY KEY (Movie_Name, Movie_Release_Date),
FOREIGN KEY (Movie_Name, Movie_Release_Date) REFERENCES Movies(Name, Release_Date));


CREATE TABLE Studios(
Movie_Name VARCHAR(50),
Movie_Release_Date DATE,
Studio VARCHAR(30),
PRIMARY KEY (Movie_Name, Movie_Release_Date),
FOREIGN KEY (Movie_Name, Movie_Release_Date) REFERENCES Movies(Name, Release_Date));
```

3.3 Advanced Functions

The advanced functions we used in our database include SQL commands that created views and triggers. The Views created for our movie theater database are summarized as follows:

- The first view we wrote demonstrates all showtimes available for a specific movie. This view will list the movie name, movie release date, room number, duration, start time, end time, age restriction, and synopsis for a specific movie.

```
CREATE VIEW TheBatmanShows AS
SELECT Name AS 'Movie Name', Date, Room_Num AS 'Room Number', Duration, Start_time,
    End_time, Age_Restriction, Synopsis
FROM Movies, Shows
WHERE name = Movie_Name AND Movie_Name LIKE 'The Batman';
```

  :) hi

- We have views that list movies for every particular genre or every particular age restrictions. This view will list the movie name, movie release date, duration, synopsis, and a specific genre according to the name of the view.

```
CREATE VIEW MysteryMovies AS
SELECT Movie_Name, Movie_Release_Date, Duration, Synopsis, Genre
FROM Movies, Genre
WHERE name = Movie_Name AND Genre LIKE 'Mystery';
```

- We also created a view that demonstrates how many tickets employees sold per day and their revenue. This view will list the date, the number of tickets sold, and the sum of the amount after discount as the revenue.

```
CREATE VIEW Tickets_Sold AS
SELECT Transaction_Start AS 'Date', SUM(Number_of_Tickets) AS 'Number of Tickets',
    CONCAT(SUM(AmountAfterDiscount), ' SR' ) AS 'Revenue'
FROM Booking_Details
GROUP BY Transaction_Start;
```

- Finally, we created a view that illustrates all user transactions we have in our movie theater database. This view will list all user names, the date of the user's transaction, and the number of tickets.

```
CREATE VIEW UserTransactions AS
SELECT CONCAT( Fname, ' ', Lname) AS 'Name', Transaction_Start AS 'Date',
    Number_Of_Tickets AS 'Number of Tickets'
FROM Users LEFT JOIN Booking_details
ON Users.User_ID = Booking_details.User_ID
ORDER BY Transaction_Start;
```

The Triggers we created for our movie theater database are summarized as follows:

- We created triggers that will modify the seat number in the shows table after adding or removing a ticket in the ticket table. After inserting values into the tickets table, this trigger will increment the value of the number of available seats in the shows table.

```
CREATE TRIGGER BookTicket
AFTER INSERT ON Tickets
FOR EACH ROW
UPDATE Shows
SET Number_Available_Seats = Number_Available_Seats - 1
WHERE NEW.Shows_ID = Show_ID;

-----------------------------------------------------------------------------
CREATE TRIGGER RemoveTicket
AFTER DELETE ON Tickets
FOR EACH ROW
UPDATE Shows
SET Number_Available_Seats = Number_Available_Seats + 1
WHERE OLD.Shows_ID = Show_ID;
```

- We also created a trigger to update information in the booking details table if the user bought a new ticket for the same movie and showtime. This trigger will modify the booking details by adding the new ticket price to the amount before the discount column and increments the number of tickets by one. Then, we used the SQL CASE statement to set the value of the discount percentage column depending on the value in the number of tickets column. Finally, we calculated the amount after the discount using the updated values before the discount column and discount percentage column.

```sql
CREATE TRIGGER UpdateTicketPrice
AFTER INSERT ON Tickets
FOR EACH ROW
UPDATE Booking_details
SET
AmountBeforeDiscount = AmountBeforeDiscount + NEW.Price,
Number_Of_Tickets = Number_Of_Tickets + 1,
DiscountPercentage =
CASE
WHEN Number_of_Tickets = 1 THEN 0.00
WHEN Number_of_Tickets = 2 THEN 0.05
WHEN Number_of_Tickets = 3 THEN 0.10
ELSE DiscountPercentage
END,
AmountAfterDiscount = AmountBeforeDiscount- (AmountBeforeDiscount * DiscountPercentage)
WHERE NEW.Booking_ID = booking_details.booking_id;

------------------------------------------------------------------------------

CREATE TRIGGER DeleteTicketPrice
AFTER DELETE ON Tickets
FOR EACH ROW
UPDATE Booking_details
SET
AmountBeforeDiscount = AmountBeforeDiscount - OLD.Price,
Number_Of_Tickets = Number_Of_Tickets - 1,
DiscountPercentage =
CASE
WHEN Number_of_Tickets = 1 THEN 0.00
WHEN Number_of_Tickets = 2 THEN 0.05
WHEN Number_of_Tickets = 3 THEN 0.10
ELSE DiscountPercentage
END,
AmountAfterDiscount = AmountBeforeDiscount- (AmountBeforeDiscount * DiscountPercentage)
WHERE OLD.Booking_ID = booking_details.booking_id;
```

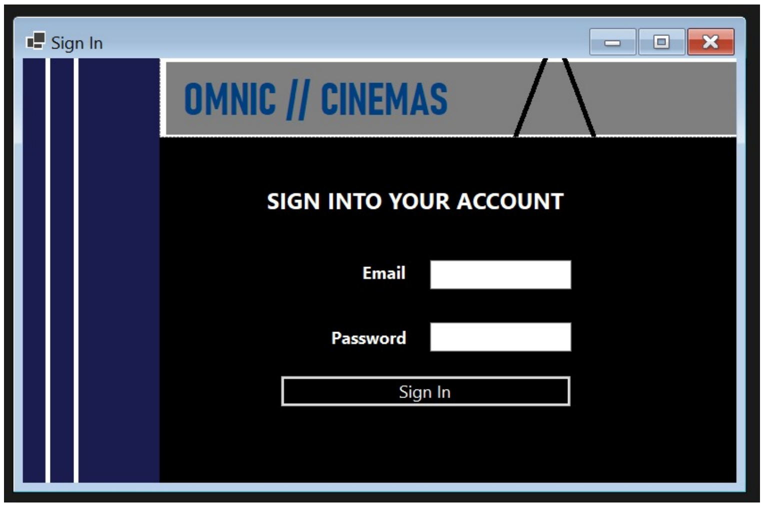
4 Implementation

The front-end of the project was an application form using VB.net as the primary language of programming, using Visual Studio as the program to design the form with the offered tools and functionalities listed within the application. The relation between the front-end and back-end is that the back-end provides data of movies, times, and seats for the customer to see, while the employee has direct access to the database to alter if desired. The functions the application contains are the following:

- Signing in (as customer or employee)
- Display of movies and selecting them
- Display of movie information and selection of timings
- Display of seats and payment page with discount section
- Display of customer account info of previous purchases
- The ability to alter the database (For employees only)

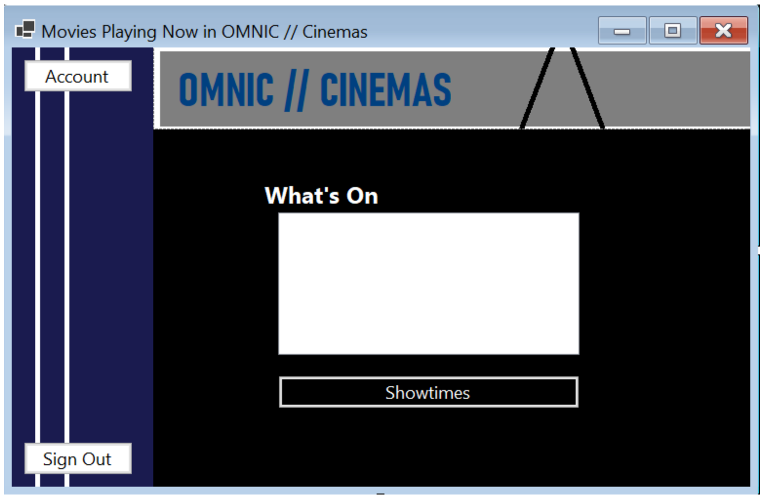
4.1 Signing In

The first form displayed to the user is the sign in form, where both the employee and customer use it to access to their own designated sections. The requirements of the sign in page is for the user to enter their email used for this service and the password accompanied by it. If the system detects that the input information is correct, the user will then be moved to a different page/form. Otherwise an error will pop up, informing the user that the input information is incorrect.



4.2 Movies

This form displays the list of movies that is being played right now on the cinema. This page offers a range of all movies for the customer to choose from. And on every form excluding the sign in form, there will be a button on the lower left corner for the user to sign out from, and on the higher left corner is a button for the customer to open their profile.

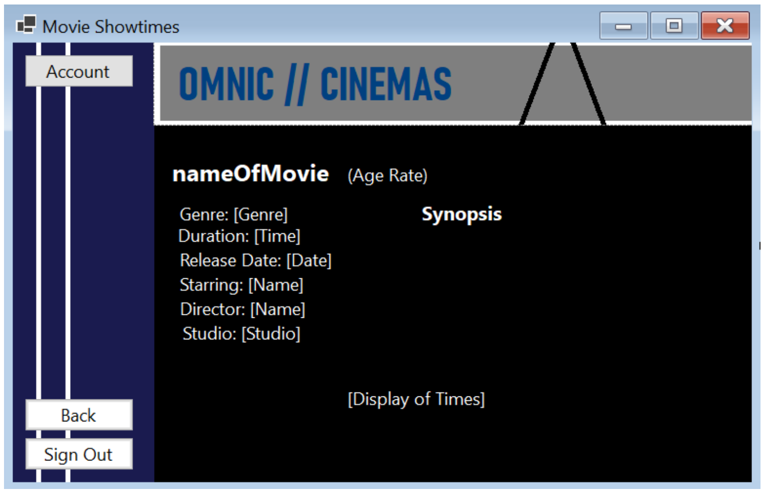


4.3 Movie Information

Once the customer selects the desired movie, a list of information will be displayed for the user to read from and know more about the movie, such listed information

- Age Rate
- Genre
- Synopsis
- Duration
- Release Date
- Actors
- Director
- Studio

On the bottom of it displays the list of times when the movie is on.



nameOfMovie (Age Rate)

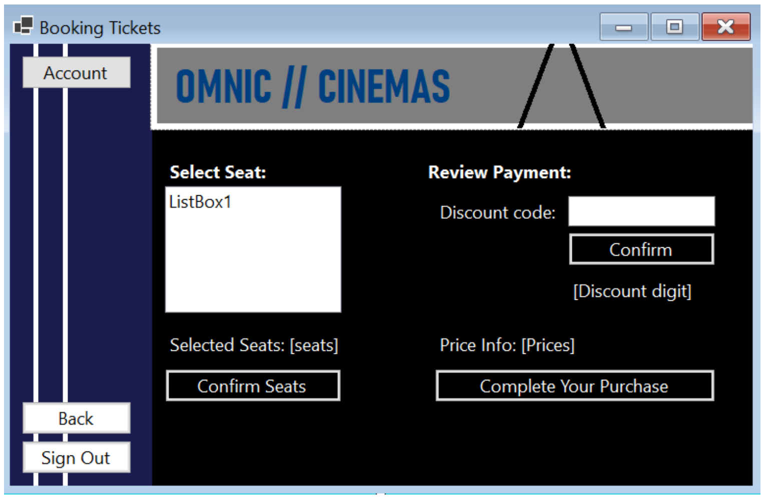
Genre: [Genre]
Duration: [Time]
Release Date: [Date]
Starring: [Name]
Director: [Name]
Studio: [Studio]

Synopsis

[Display of Times]

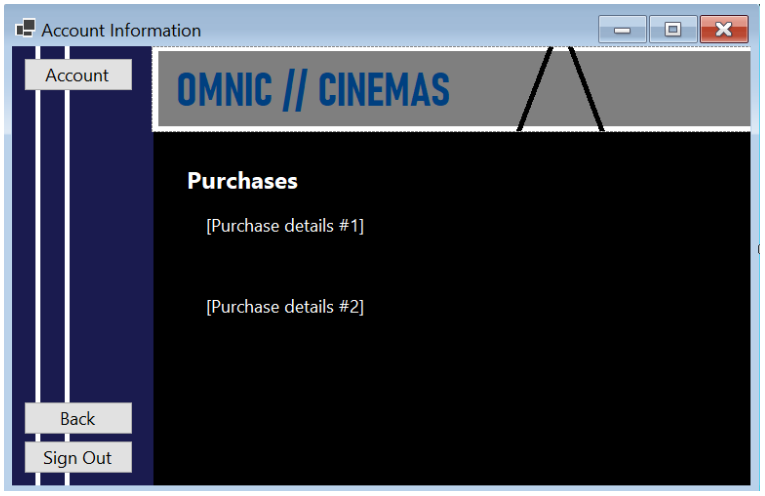
4.4 Booking Ticket

On this page, it is split into two sections, a seat selection and a payment process section. The first section displays the seats that are available and taken with a label under it displaying the selected seats for the user to confirm. While the payment section allows the customer to review their payment by offering a input box for the customer to put a discount code, with a confirmation button.



4.5 Exclusive Role Pages

When the user signs into their account, they are led to one of the two forms depending on what the user's role is. If the user is an employee, this will take them to a page where the employee can alter the database/back-end however they see fit or needed. While for the customer, they have access to their own profile that displays a list of previous purchases/booking tickets.



5 Outcome

In this project, we practiced creating our database from scratch. We first started by creating our ER diagram and the relational schema. Then when we began implementing the SQL to insert values, we noticed some basic diagram errors. So we went back and fixed them. For example, when we started inserting values into the movies table, we noticed that we would have multiple genres, studios, and actors. So we went back to the schema to create new tables for the multi-valued attributes we had.

After we inserted values into all tables, we started making views and triggers. We faced difficulties creating an effective trigger between the booking details table and the tickets table. We wanted to update the booking details table when a specific user books another ticket at the exact showtime. To fix that, we used the SQL case statement inside the trigger we created to first update the discount percentage depending on the number of tickets.

To extend our project to be more advanced in the future, we would like to add a feedback session where we would ask the user about their experience in our movie theater. Moreover, one of the main concerns people face in movie theaters is that they would have to stand in long lines to order food, resulting in them being late for their movie showtime. To improve that, we can add an option for the user to order their food earlier so that when they arrive at the movie theater, their order will be ready.