# CINEMA TICKET BOOKING SYSTEM DATABASE DESIGN

ELO, FREDRICK ADUDUORIAGHE

B. Eng (Uniport)
M. Eng (Uniport)
G2023/PHD/CITE/FT/001

TO

DR. UGOCHI ADAKU OKENGWU

ASSOCIATE PROFESSOR (READER)

UNIVERSITY OF PORT HARCOURT

AG HOD, COMPUTER SCIENCE DEPARTMENT

CENTER FOR INFORMATION AND TELECOMMUNICATION ENGINEERING (CITE)
SCHOOL OF GRADUATE STUDIES,
UNIVERSITY OF PORT HARCOURT,
CHOBA, RIVERS STATE, NIGERIA

## 1. Introduction

This technical report details the design and implementation of a Cinema Ticket Booking System Database. The system manages movie schedules, ticket reservations, customer information, and payment records. Additionally, it provides real-time reports for available seats, customer booking history, and most-watched movies.

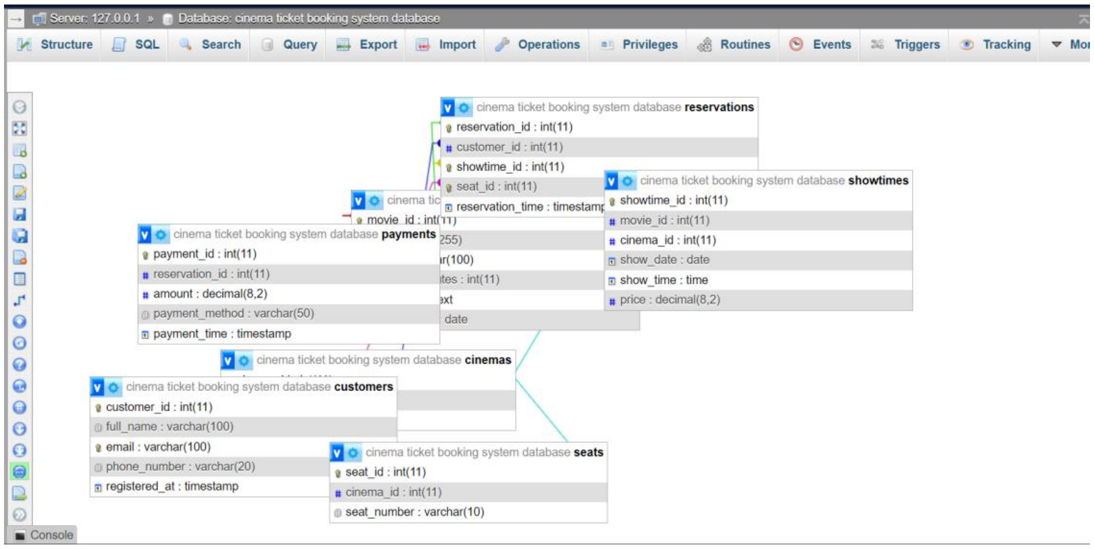
## 2. Objectives

- Store and manage customer data securely.
- Manage movies, showtimes, and seat reservations.
- Automate bookings using stored procedures.
- Track and manage payments.
- Provide efficient data retrieval through SQL queries.

## 3. System Requirements

Database Server: MySQL 8.x (XAMPP stack)
Backend Tools: phpMyAdmin
Environment: Localhost (XAMPP)

## 4. Database Design



cinema ticket booking system database **reservations**
reservation_id : int(11)
customer_id : int(11)
showtime_id : int(11)
seat_id : int(11)
reservation_time : timestamp

cinema ticket booking system database **showtimes**
showtime_id : int(11)
movie_id : int(11)
cinema_id : int(11)
show_date : date
show_time : time
price : decimal(8,2)

cinema tic
movie_id : int(11)
255)
r(100)
tes : int(11)
xt
date

cinema ticket booking system database **payments**
payment_id : int(11)
reservation_id : int(11)
amount : decimal(8,2)
payment_method : varchar(50)
payment_time : timestamp

cinema ticket booking system database **cinemas**

cinema ticket booking system database **customers**
customer_id : int(11)
full_name : varchar(100)
email : varchar(100)
phone_number : varchar(20)
registered_at : timestamp

cinema ticket booking system database **seats**
seat_id : int(11)
cinema_id : int(11)
seat_number : varchar(10)

Entities
- Customers
- Movies
- Cinemas
- Showtimes
- Seats

- Reservations
- Payments

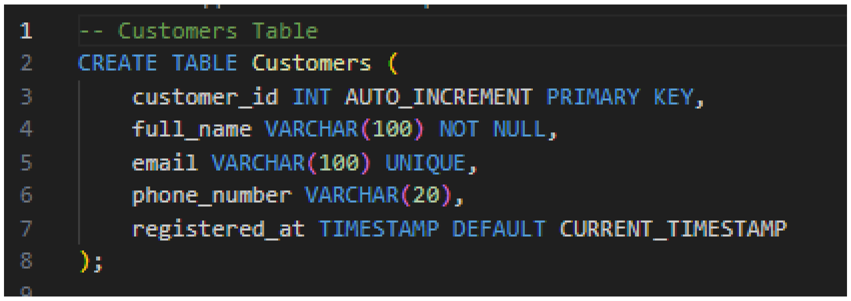
Relationships
- Customers make Reservations.
- Reservations are linked to Showtimes and Seats.
- Showtimes are linked to Movies and Cinemas.
- Payments are linked to Reservations.

## 5. SQL Code Explanation

A breakdown of the major tables and procedures used for the system is included in the attached SQL files. Each table is normalized and maintains foreign key relationships for data integrity. The stored procedure `BookSeat` automates reservation and payment processing while ensuring no seat is double-booked.

**Table Structures**

- **Customers Table**

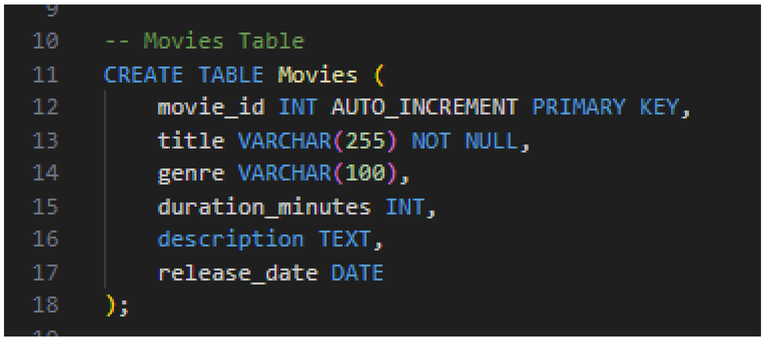


```sql
-- Customers Table
CREATE TABLE Customers (
    customer_id INT AUTO_INCREMENT PRIMARY KEY,
    full_name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE,
    phone_number VARCHAR(20),
    registered_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

**Explanation**:

- Unique customer ID, name, email, phone number.
- Timestamp for registration date.

- **Movies Table**

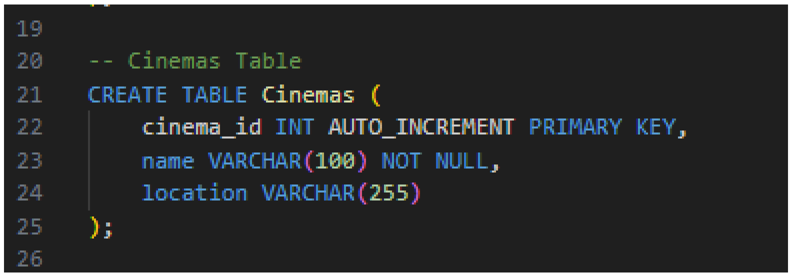


```sql
-- Movies Table
CREATE TABLE Movies (
    movie_id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    genre VARCHAR(100),
    duration_minutes INT,
    description TEXT,
    release_date DATE
);
```

**Explanation**:

- Stores movie details including title, genre, duration, description, and release date.

➢ **Cinemas Table**

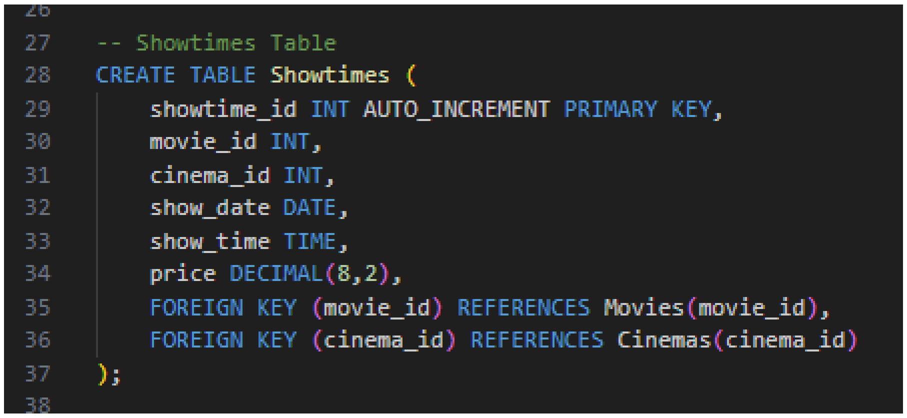


```sql
-- Cinemas Table
CREATE TABLE Cinemas (
    cinema_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    location VARCHAR(255)
);
```

**Explanation**:

- Each cinema has a name and a location.

➢ **Showtimes Table**

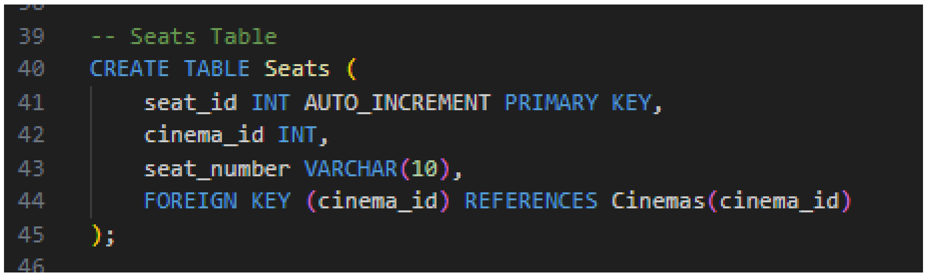


```sql
-- Showtimes Table
CREATE TABLE Showtimes (
    showtime_id INT AUTO_INCREMENT PRIMARY KEY,
    movie_id INT,
    cinema_id INT,
    show_date DATE,
    show_time TIME,
    price DECIMAL(8,2),
    FOREIGN KEY (movie_id) REFERENCES Movies(movie_id),
    FOREIGN KEY (cinema_id) REFERENCES Cinemas(cinema_id)
);
```

**Explanation**:

- Defines when and where a movie is shown.
- Price is linked to each showtime.
- Foreign keys reference Movies and Cinemas.

➢ **Seats Table**

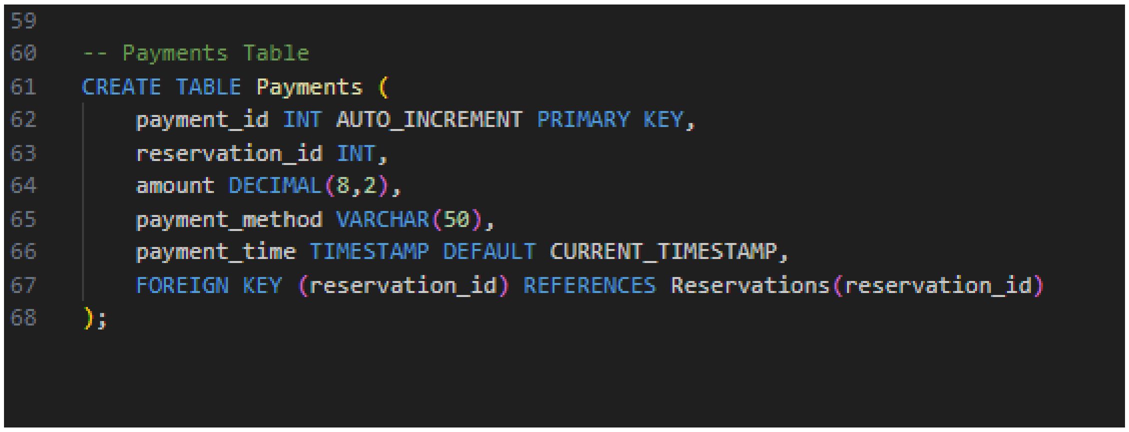


```sql
-- Seats Table
CREATE TABLE Seats (
    seat_id INT AUTO_INCREMENT PRIMARY KEY,
    cinema_id INT,
    seat_number VARCHAR(10),
    FOREIGN KEY (cinema_id) REFERENCES Cinemas(cinema_id)
);
```

**Explanation**:

- Seats belong to a cinema.
- Each seat has a unique number.

➢ **Reservations Table**

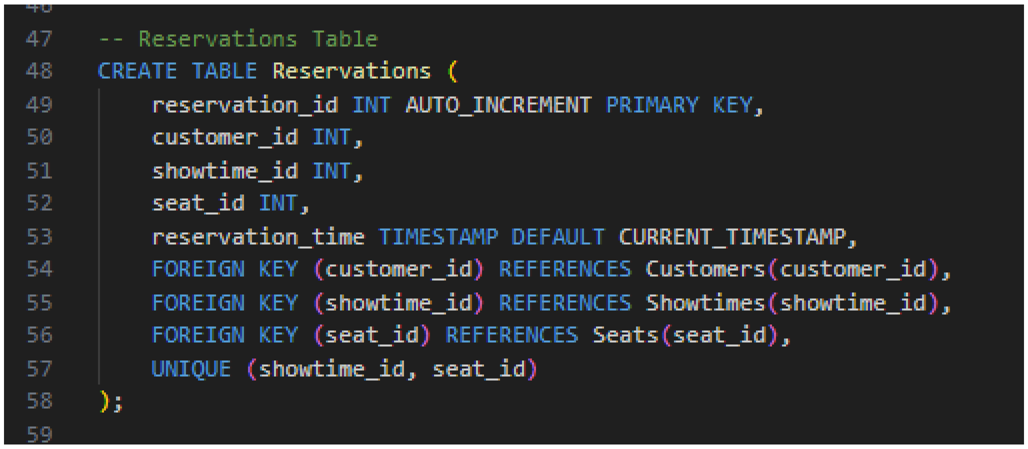


```sql
-- Payments Table
CREATE TABLE Payments (
    payment_id INT AUTO_INCREMENT PRIMARY KEY,
    reservation_id INT,
    amount DECIMAL(8,2),
    payment_method VARCHAR(50),
    payment_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (reservation_id) REFERENCES Reservations(reservation_id)
);
```

**Explanation**:

- Links customers to booked seats and showtimes.
- Prevents double booking with UNIQUE constraint.

➢ **Payments Table**



```sql
-- Reservations Table
CREATE TABLE Reservations (
    reservation_id INT AUTO_INCREMENT PRIMARY KEY,
    customer_id INT,
    showtime_id INT,
    seat_id INT,
    reservation_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (customer_id) REFERENCES Customers(customer_id),
    FOREIGN KEY (showtime_id) REFERENCES Showtimes(showtime_id),
    FOREIGN KEY (seat_id) REFERENCES Seats(seat_id),
    UNIQUE (showtime_id, seat_id)
);
```

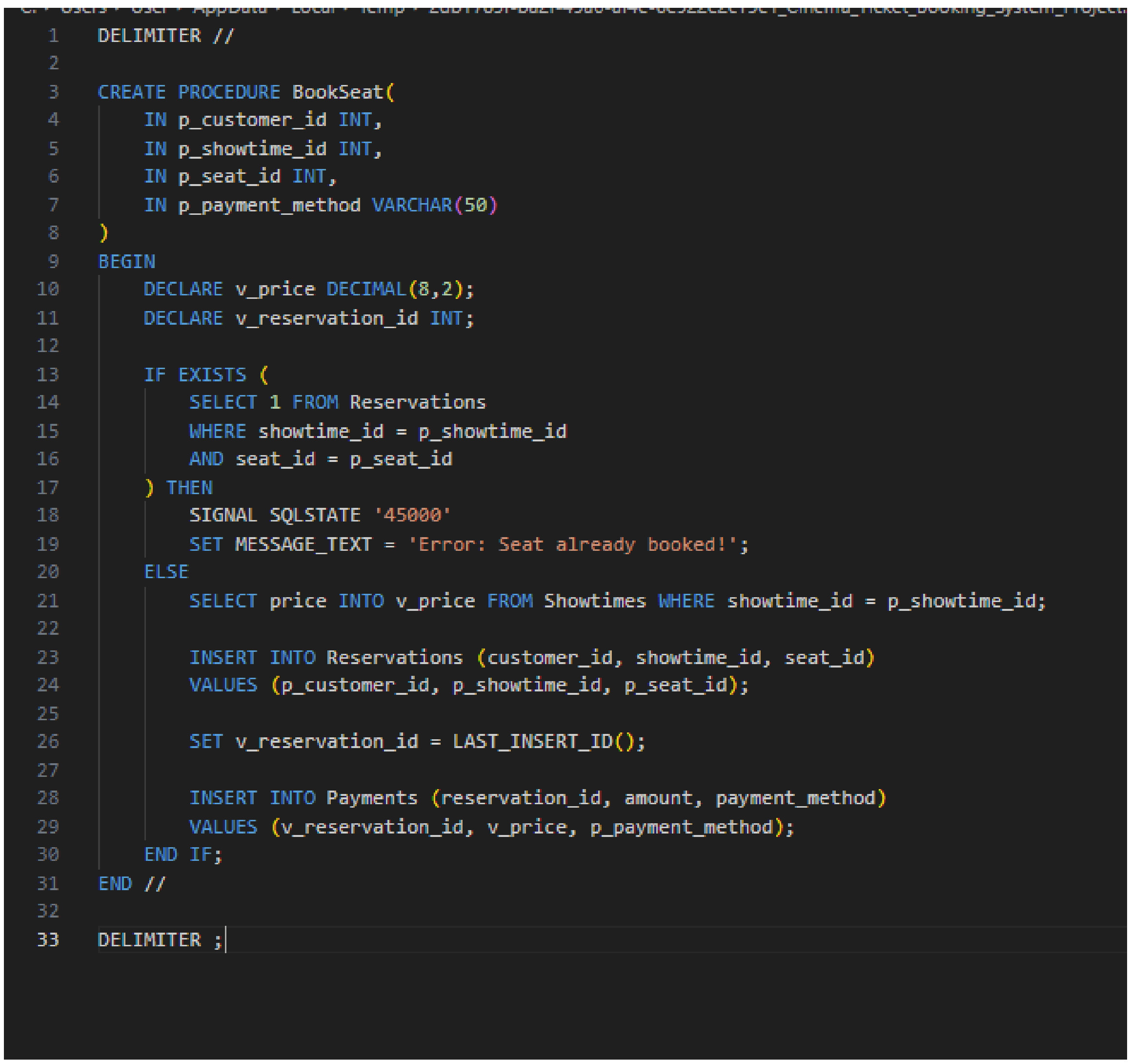
**Explanation**:

- Payments tied directly to reservations.
- Records amount and payment method.

## 6. Stored Procedures

Stored procedure `BookSeat` accepts customer ID, showtime ID, seat ID, and payment method. It checks availability, inserts reservation, and logs payment.

- **BookSeat Stored Procedure**



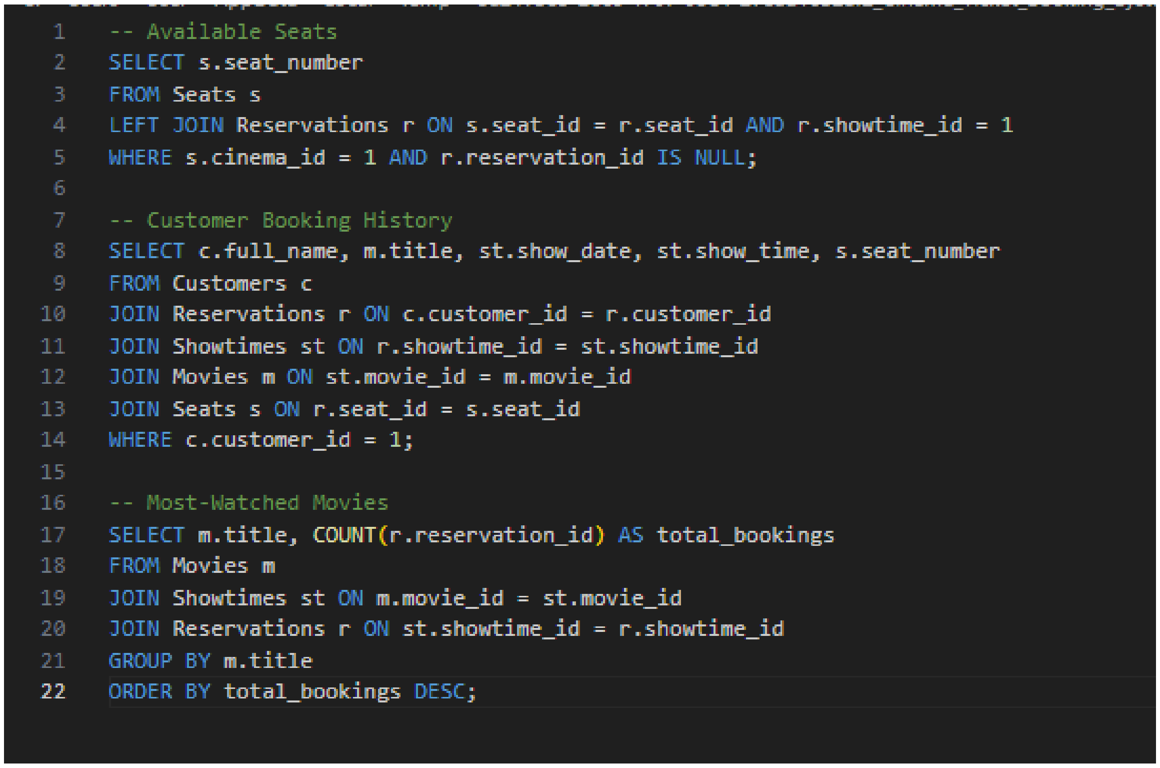
```
DELIMITER //

CREATE PROCEDURE BookSeat(
    IN p_customer_id INT,
    IN p_showtime_id INT,
    IN p_seat_id INT,
    IN p_payment_method VARCHAR(50)
)
BEGIN
    DECLARE v_price DECIMAL(8,2);
    DECLARE v_reservation_id INT;

    IF EXISTS (
        SELECT 1 FROM Reservations
        WHERE showtime_id = p_showtime_id
        AND seat_id = p_seat_id
    ) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Error: Seat already booked!';
    ELSE
        SELECT price INTO v_price FROM Showtimes WHERE showtime_id = p_showtime_id;

        INSERT INTO Reservations (customer_id, showtime_id, seat_id)
        VALUES (p_customer_id, p_showtime_id, p_seat_id);

        SET v_reservation_id = LAST_INSERT_ID();

        INSERT INTO Payments (reservation_id, amount, payment_method)
        VALUES (v_reservation_id, v_price, p_payment_method);
    END IF;
END //

DELIMITER ;
```

**Explanation**:

- Checks if the seat is already booked.
- Retrieves showtime price.
- Creates a reservation if seat is available.
- Records payment details automatically.

## 7. SQL Queries for Information Retrieval

Queries include:
- Available Seats for a Showtime
- Customer Booking History
- Most-Watched Movies
Each query uses optimized joins and filters for real-time reporting.



```sql
-- Available Seats
SELECT s.seat_number
FROM Seats s
LEFT JOIN Reservations r ON s.seat_id = r.seat_id AND r.showtime_id = 1
WHERE s.cinema_id = 1 AND r.reservation_id IS NULL;

-- Customer Booking History
SELECT c.full_name, m.title, st.show_date, st.show_time, s.seat_number
FROM Customers c
JOIN Reservations r ON c.customer_id = r.customer_id
JOIN Showtimes st ON r.showtime_id = st.showtime_id
JOIN Movies m ON st.movie_id = m.movie_id
JOIN Seats s ON r.seat_id = s.seat_id
WHERE c.customer_id = 1;

-- Most-Watched Movies
SELECT m.title, COUNT(r.reservation_id) AS total_bookings
FROM Movies m
JOIN Showtimes st ON m.movie_id = st.movie_id
JOIN Reservations r ON st.showtime_id = r.showtime_id
GROUP BY m.title
ORDER BY total_bookings DESC;
```

➢ **Available Seats**

**Explanation**:

- Lists all unreserved seats for a particular showtime.

➢ **Customer Booking History**

**Explanation**:

- Displays all past bookings of a customer.

➢ **Most-Watched Movies**

**Explanation**:

- Lists movies ranked by the number of reservations.

## 8. Conclusion

The Cinema Ticket Booking System Database effectively manages movie schedules, reservations, customers, and payments. It uses best practices like:

- Data normalization,
- Foreign keys for relational integrity,
- Stored procedures for automation,
- Efficient queries for real-time reporting.