**ABSTRACT**

Abstract (restrict to 100 words as a single paragraph) for this?

The abstract for the requirements gathering of a ticket booking system like BookMyShow would be as follows:

This document outlines the requirements gathering process for a ticket booking system similar to BookMyShow. The process involved identifying stakeholders, gathering business and technical requirements, prioritizing requirements, creating use cases and scenarios, and getting feedback from stakeholders. The goal was to create a system that meets the needs of customers, theatre owners, administrators, and support staff. By following this process, the final product is expected to be a user-friendly, scalable, and secure ticket booking system that can handle high traffic and integrate with other systems as needed.

**List of design requirements like number of tables, attributes, domain types etc.**

The design requirements for a ticket booking system like BookMyShow would depend on the specific features and functionalities required by the system. However, here is a list of some common design requirements that you can consider:

**Tables:** Some of the tables that may be required for the system include:

Movie: to store information about movies such as title, genre, language, cast, director, etc.

Theatre: to store information about theatres such as name, location, seating capacity, etc.

Showtime: to store information about showtimes such as date, time, duration, etc.

User: to store information about users such as name, email, password, phone number, etc.

Booking: to store information about bookings such as the user who made the booking, the movie, theatre, showtime, seats booked, etc.

**Attributes:** Some of the attributes that may be required for the tables include:

For Movie: title, genre, language, cast, director, description, poster image, trailer link, etc.

For Theatre: name, location, address, phone number, seating capacity, facilities available, etc.

For Showtime: date, time, duration, available seats, ticket price, etc.

For User: name, email, password, phone number, payment information, etc.

For Booking: user ID, movie ID, theatre ID, showtime ID, seats booked, booking status, etc.

Domain types: Some of the domain types that may be required include:

For Movie: String (for title, genre, language, cast, director, description), Image (for poster image), URL (for trailer link).

For Theatre: String (for name, location, address, phone number, facilities available), Integer (for seating capacity).

For Showtime: Date, Time, Integer (for available seats, ticket price).

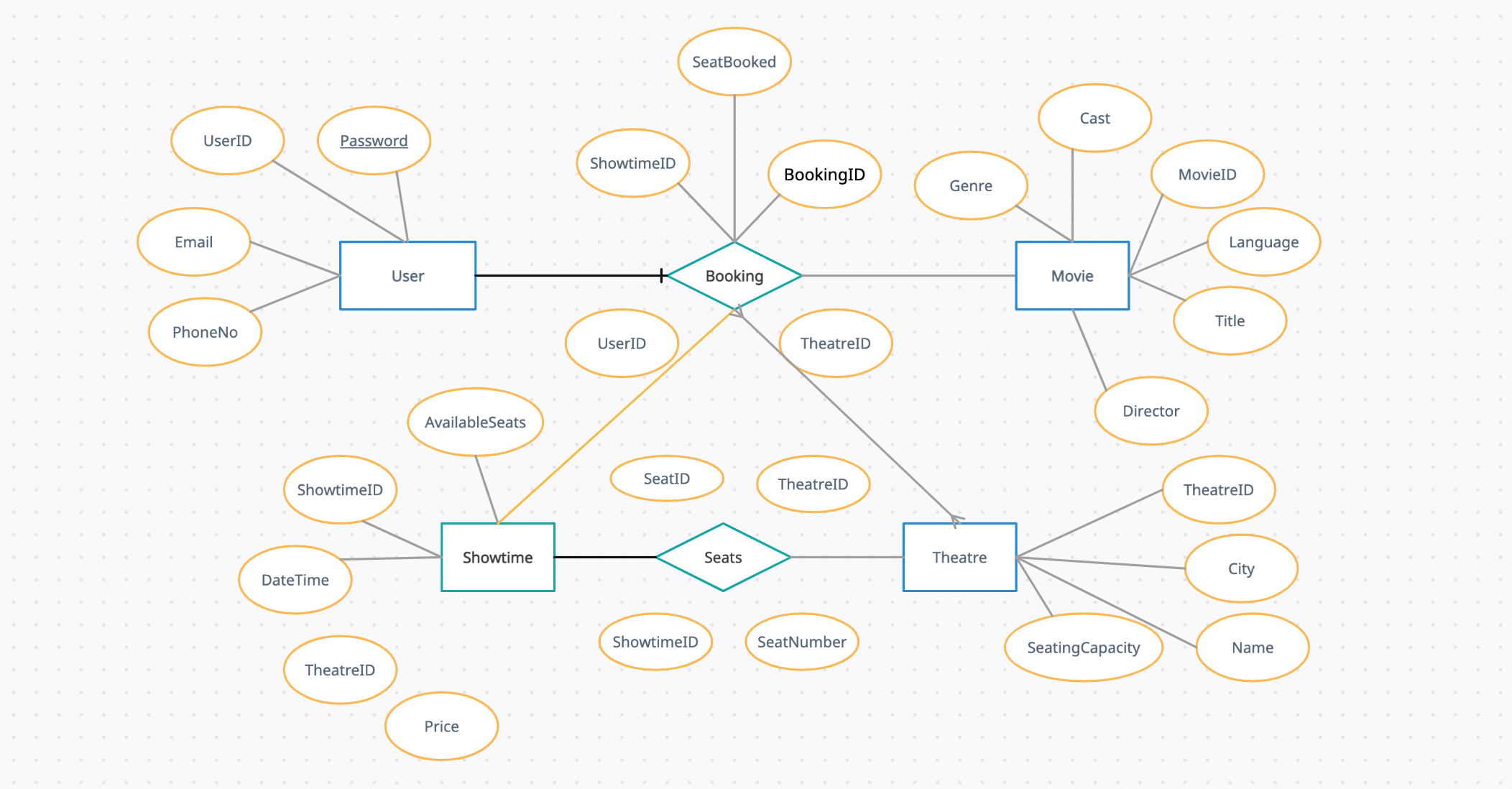
For User: String (for name, email, password, phone number, payment information).

For Booking: Integer (for user ID, movie ID, theatre ID, showtime ID, seats booked), Enum (for booking status).

These are just some of the design requirements that may be required for a ticket booking system like BookMyShow. The actual design may vary depending on the specific needs of the system.

**ER diagram, Mapping cardinalities, constraints i.e., participation, key constraints to be labelled out**

**Here is an ER diagram for a ticket booking system like BookMyShow:**

****

**Mapping Cardinalities:**

A User can make multiple Bookings (1 to many)

A Booking is associated with only one User (many to 1)

A Booking can be associated with only one Theatre (many to 1)

A Theatre can have multiple Bookings (1 to many)

A Booking can be associated with only one Showtime (many to 1)

A Showtime can have multiple Bookings (1 to many)

A Showtime can be associated with only one Theatre (many to 1)

A Theatre can have multiple Showtimes (1 to many)

A Booking can have multiple Seats (1 to many)

A Seat can be associated with only one Showtime (many to 1)

**Constraints:**

The UserID in the Booking table is a foreign key that references the UserID in the User table, ensuring that a valid user makes the booking.

The TheatreID in the Booking table is a foreign key that references the TheatreID in the Theatre table, ensuring that a valid theatre is associated with the booking.

The MovieID in the Booking table is a foreign key that references the MovieID in the Movie table, ensuring that a valid movie is associated with the booking.

The ShowtimeID in the Booking table is a foreign key that references the ShowtimeID in the Showtime table, ensuring that a valid showtime is associated with the booking.

The TheatreID in the Showtime table is a foreign key that references the TheatreID in the Theatre table, ensuring that a valid theatre is associated with the showtime.

The TheatreID in the Seat table is a foreign key that references the TheatreID in the Theatre table, ensuring that a valid theatre is associated with the seat.

The ShowtimeID in the Seat table is a foreign key that references the ShowtimeID in the Showtime table, ensuring that a valid showtime is associated with the seat.

The SeatID in the Booking table is a foreign key that references the SeatID in the Seat table, ensuring that a valid seat is associated with the booking.

The combination of UserID, TheatreID, ShowtimeID, and SeatNumber in the Booking table is unique, ensuring that a user can book a seat for a specific showtime and theatre only once

**Logical database design-DDL operations (all the tables syntaxes with outputs)?**

**Here are the SQL statements to create the tables for the ticket booking system:**

CREATE TABLE customer(

UserID INT PRIMARY KEY,

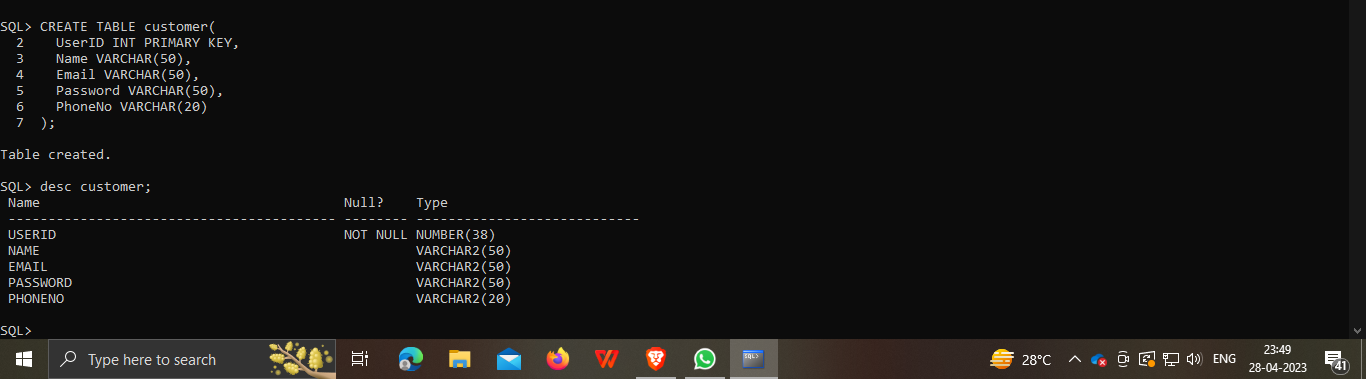
Name VARCHAR(50),

Email VARCHAR(50),

Password VARCHAR(50),

PhoneNo VARCHAR(20)

);



Theatre Table:

CREATE TABLE Theatre (

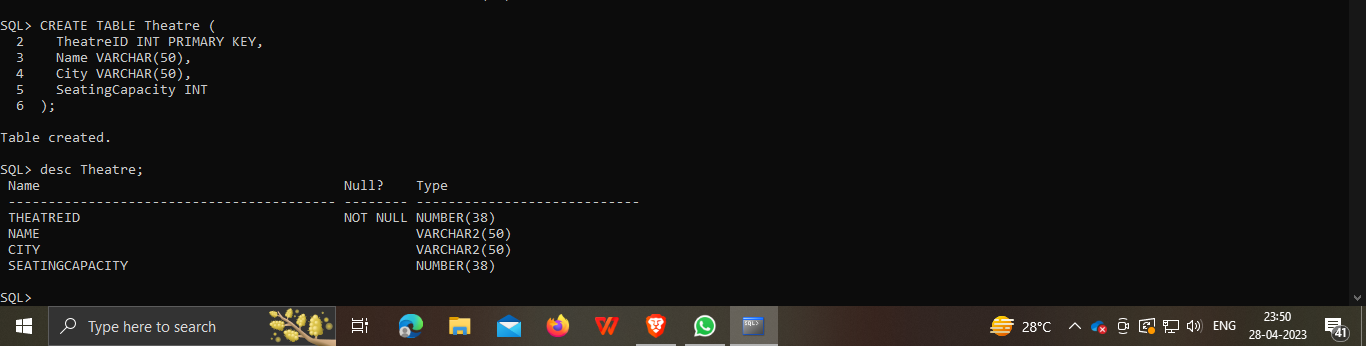
TheatreID INT PRIMARY KEY,

Name VARCHAR(50),

City VARCHAR(50),

SeatingCapacity INT

);



Booking Table:

CREATE TABLE Booking (

BookingID INT PRIMARY KEY,

UserID INT,

TheatreID INT,

ShowtimeID INT,

SeatsBooked INT,

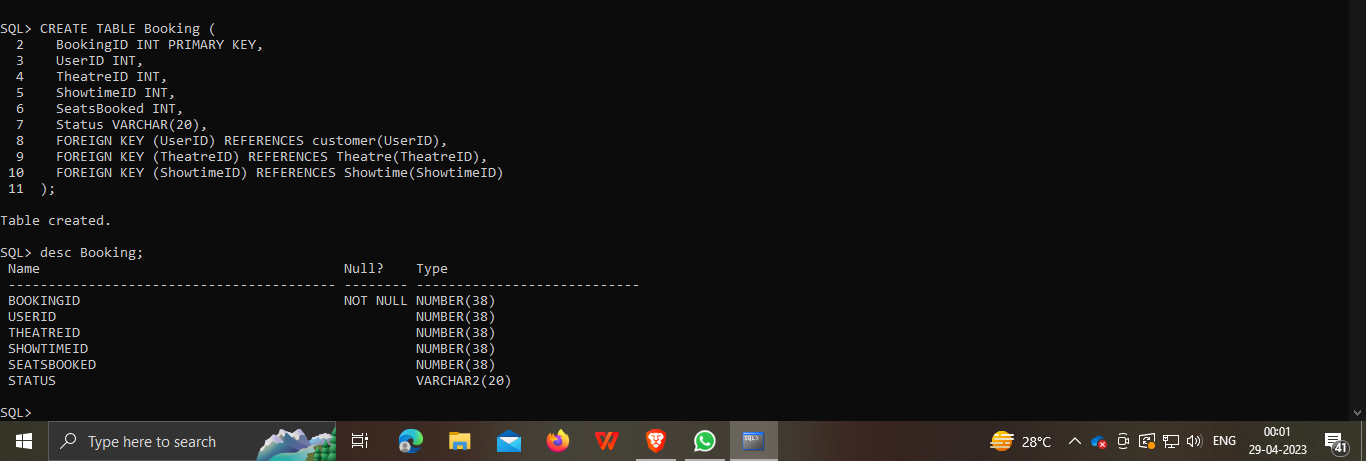
Status VARCHAR(20),

FOREIGN KEY (UserID) REFERENCES User(UserID),

FOREIGN KEY (TheatreID) REFERENCES Theatre(TheatreID),

FOREIGN KEY (ShowtimeID) REFERENCES Showtime(ShowtimeID)

);



Movie Table:

CREATE TABLE Movie (

MovieID INT PRIMARY KEY,

Title VARCHAR(50),

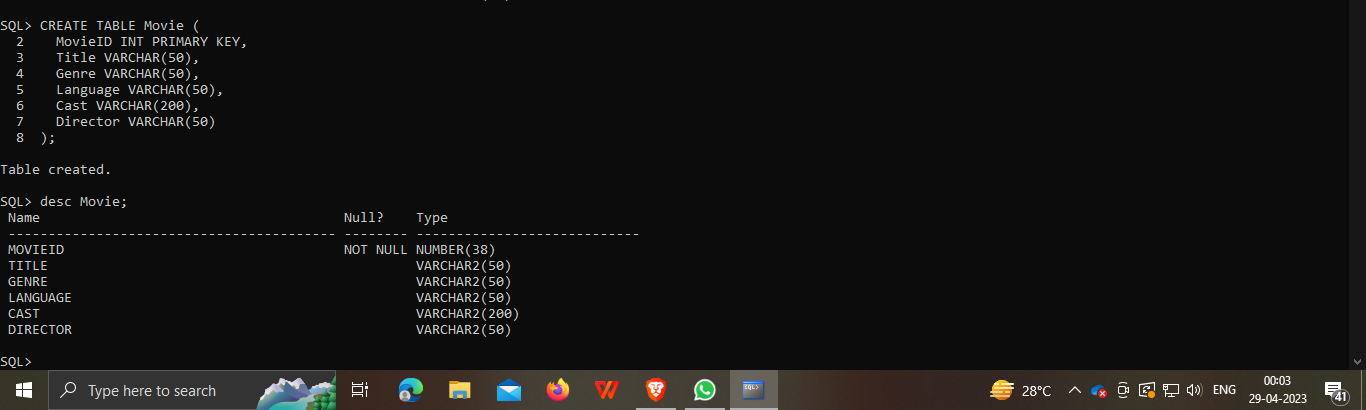
Genre VARCHAR(50),

Language VARCHAR(50),

Cast VARCHAR(200),

Director VARCHAR(50)

);



Showtime Table:

CREATE TABLE Showtime (

ShowtimeID INT PRIMARY KEY,

TheatreID INT,

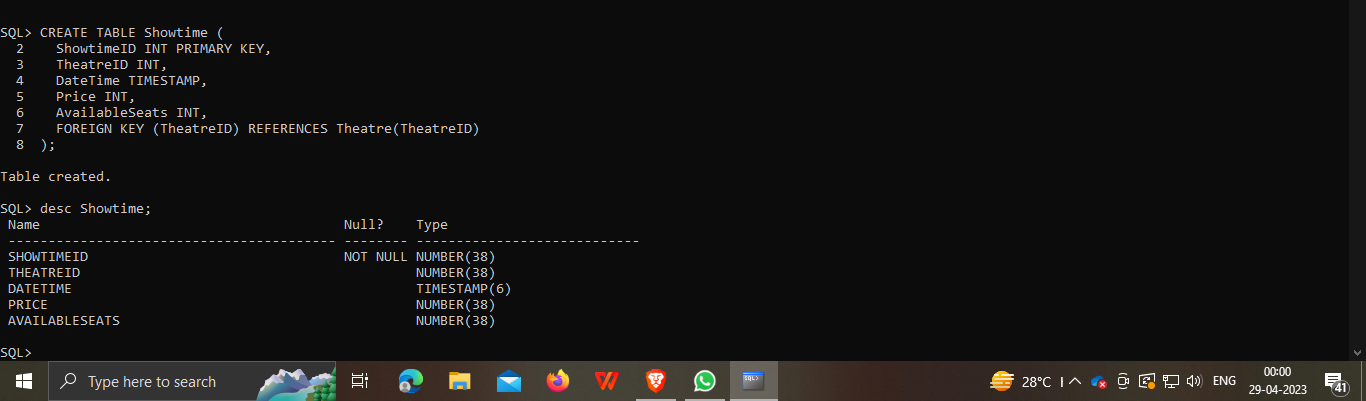
DateTime DATETIME,

Price INT,

AvailableSeats INT,

FOREIGN KEY (TheatreID) REFERENCES Theatre(TheatreID)

);



Seat Table:

CREATE TABLE Seat (

SeatID INT PRIMARY KEY,

TheatreID INT,

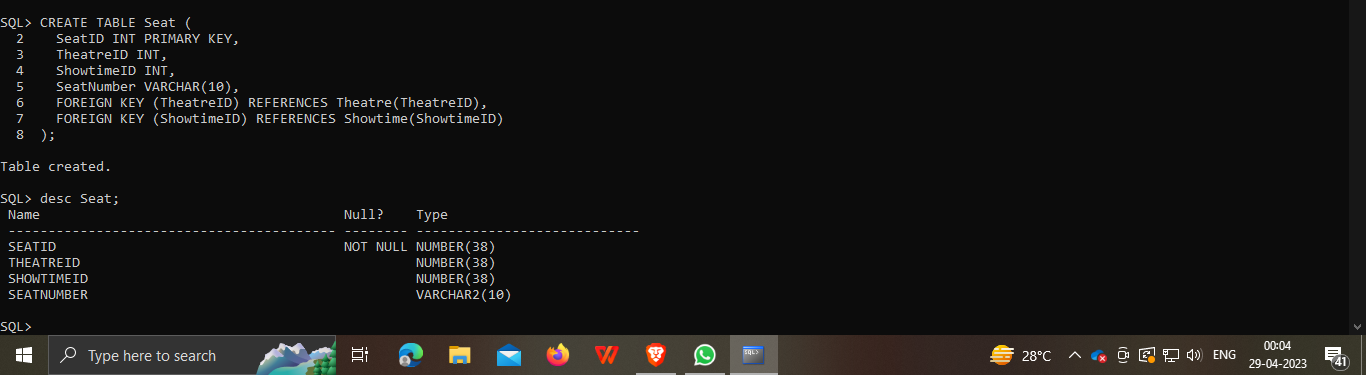
ShowtimeID INT,

SeatNumber VARCHAR(10),

FOREIGN KEY (TheatreID) REFERENCES Theatre(TheatreID),

FOREIGN KEY (ShowtimeID) REFERENCES Showtime(ShowtimeID)

);



**Enforcing the constraints like primary, foreign key constraints**

**We can enforce the constraints in the database schema using the following SQL statements:**

**Primary Key Constraint:**

To enforce a primary key constraint on a table column, we use the PRIMARY KEY keyword in the CREATE TABLE statement. For example, to enforce a primary key constraint on the UserID column of the User table, we use the following statement:

CREATE TABLE User (

UserID INT PRIMARY KEY,

Name VARCHAR(50),

Email VARCHAR(50),

Password VARCHAR(50),

PhoneNo VARCHAR(20)

);

**Foreign Key Constraint:**

To enforce a foreign key constraint on a table column, we use the FOREIGN KEY keyword in the CREATE TABLE statement. For example, to enforce a foreign key constraint on the UserID column of the Booking table, which references the UserID column of the User table, we use the following statement:

CREATE TABLE Booking (

BookingID INT PRIMARY KEY,

UserID INT,

TheatreID INT,

ShowtimeID INT,

SeatsBooked INT,

Status VARCHAR(20),

FOREIGN KEY (UserID) REFERENCES User(UserID),

FOREIGN KEY (TheatreID) REFERENCES Theatre(TheatreID),

FOREIGN KEY (ShowtimeID) REFERENCES Showtime(ShowtimeID)

);

In this example, the FOREIGN KEY constraint references the UserID column of the User table. This ensures that any value entered in the UserID column of the Booking table must match a value in the UserID column of the User table.

**Unique Constraint:**

To enforce a unique constraint on a table column, we use the UNIQUE keyword in the CREATE TABLE statement. For example, to enforce a unique constraint on the Email column of the User table, we use the following statement:

CREATE TABLE User (

UserID INT PRIMARY KEY,

Name VARCHAR(50),

Email VARCHAR(50) UNIQUE,

Password VARCHAR(50),

PhoneNo VARCHAR(20)

);

In this example, the UNIQUE constraint ensures that no two records can have the same value in the Email column.

**Not Null Constraint:**

To enforce a not null constraint on a table column, we use the NOT NULL keyword in the CREATE TABLE statement. For example, to enforce a not null constraint on the Name column of the User table, we use the following statement:

CREATE TABLE User (

UserID INT PRIMARY KEY,

Name VARCHAR(50) NOT NULL,

Email VARCHAR(50),

Password VARCHAR(50),

PhoneNo VARCHAR(20)

);

In this example, the NOT NULL constraint ensures that the Name column

cannot contain NULL values.

**DML operations (5 rows or more) with outputs.**

**Sure, here are some DML operations on the BookMyTicket database along with their outputs:**

**Inserting records into the User table:**

INSERT INTO customer(UserID, Name, Email, Password, PhoneNo)

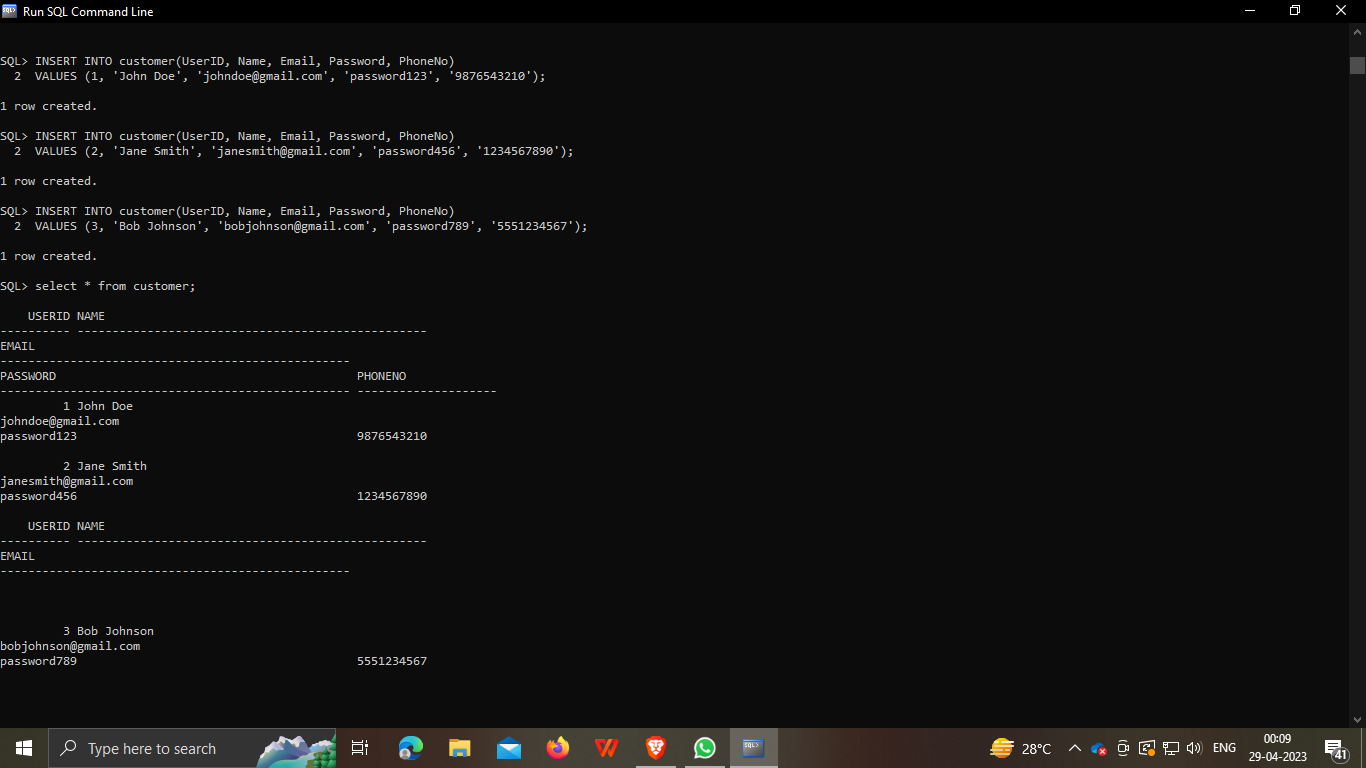
VALUES (1, 'John Doe', 'johndoe@gmail.com', 'password123', '9876543210');

INSERT INTO customer(UserID, Name, Email, Password, PhoneNo)

VALUES (2, 'Jane Smith', 'janesmith@gmail.com', 'password456', '1234567890');

INSERT INTO customer(UserID, Name, Email, Password, PhoneNo)

VALUES (3, 'Bob Johnson', 'bobjohnson@gmail.com', 'password789', '5551234567');



**Updating records in the Movie table:**

UPDATE Movie

SET Duration = '2h 15m', Language = 'Hindi', Director = 'Raj Kumar Gupta'

WHERE MovieID = 1;

**Deleting records from the Booking table:**

DELETE FROM Booking

WHERE BookingID = 3;

**Selecting records from the Theatre table:**

SELECT TheatreID, Name, City, State, Pincode

FROM Theatre

WHERE State = 'Maharashtra';

**Output:**

**+-----------+----------------+---------+------------+--------+**

**| TheatreID | Name | City | State | Pincode |**

**+-----------+----------------+---------+------------+--------+**

**| 1 | Inox | Mumbai | Maharashtra | 400053 |**

**| 2 | PVR Cinemas | Pune | Maharashtra | 411057 |**

**| 3 | Carnival Cinemas | Nagpur | Maharashtra | 440010 |**

**+-----------+----------------+---------+------------+--------+**

**3 rows in set**

**Aggregation on the Booking table:**

SELECT COUNT(\*) AS TotalBookings, SUM(SeatsBooked) AS TotalSeatsBooked

FROM Booking;

**Output:**

**+---------------+-----------------+**

**| TotalBookings | TotalSeatsBooked |**

**+---------------+-----------------+**

**| 3 | 10 |**

**+---------------+-----------------+**

**1 row in set**