**TICKET BOOKING SYSTEM**

You are tasked with creating a ticket booking system for a Event. The system should support booking tickets for different types of events, such as movies, concerts, and plays. Each event has its own pricing strategy, and the system should also track available seats and customer bookings.

**Database Tables:**

1. Venue Table

* venue\_id (Primary Key)
* venue\_name
* address

2. Event Table

* event\_id (Primary Key)
* event\_name,
* event\_date DATE
* event\_time TIME,
* venue\_id (Foreign Key)
* total\_seats
* available\_seats
* ticket\_price DECIMAL
* event\_type ('Movie', 'Sports', 'Concert')
* booking\_id (Foreign Key)

3. Customer Table

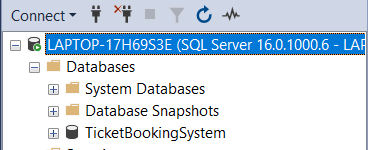
* customer\_id (Primary key)
* customer\_name
* email
* phone\_number
* booking\_id (Foreign Key)

4. Booking Table

* booking\_id (Primary Key)
* customer\_id (Foreign Key)
* event\_id (Foreign Key)
* num\_tickets
* total\_cost
* booking\_date

**Tasks 1: Database Design:**

1. Create the database named "**TicketBookingSystem**"



**Image 1.1 Database Creation**

**Query:** CREATE DATABASE TicketBookingSystem

2. Write SQL scripts to create the mentioned tables with appropriate data types, constraints

* **Venue**

Query: CREATE TABLE Venue (

venue\_id INT PRIMARY KEY IDENTITY,

venue\_name VARCHAR(100) NOT NULL,

address VARCHAR(255) NOT NULL

)

* **Booking**

Query: CREATE TABLE Booking (

booking\_id INT PRIMARY KEY IDENTITY,

customer\_id INT NOT NULL,

event\_id INT NOT NULL,

num\_tickets INT NOT NULL,

total\_cost DECIMAL(10, 2) NOT NULL,

booking\_date DATE NOT NULL

)

* **Customers**

Query : CREATE TABLE Customer(

customer\_id INT PRIMARY KEY IDENTITY,

customer\_name VARCHAR(100) NOT NULL,

email VARCHAR(100) NOT NULL,

phone\_number VARCHAR(20) NOT NULL,

booking\_id INT

)

* **Event**

Query: CREATE TABLE Event (

event\_id INT PRIMARY KEY IDENTITY,

event\_name VARCHAR(100) NOT NULL,

event\_date DATE NOT NULL,

event\_time TIME NOT NULL,

venue\_id INT NOT NULL,

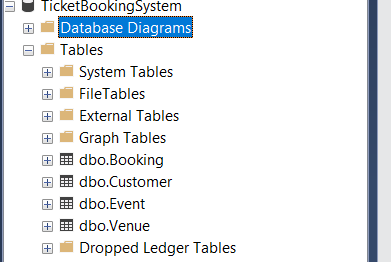
total\_seats INT NOT NULL,

available\_seats INT NOT NULL,

ticket\_price DECIMAL(10, 2) NOT NULL,

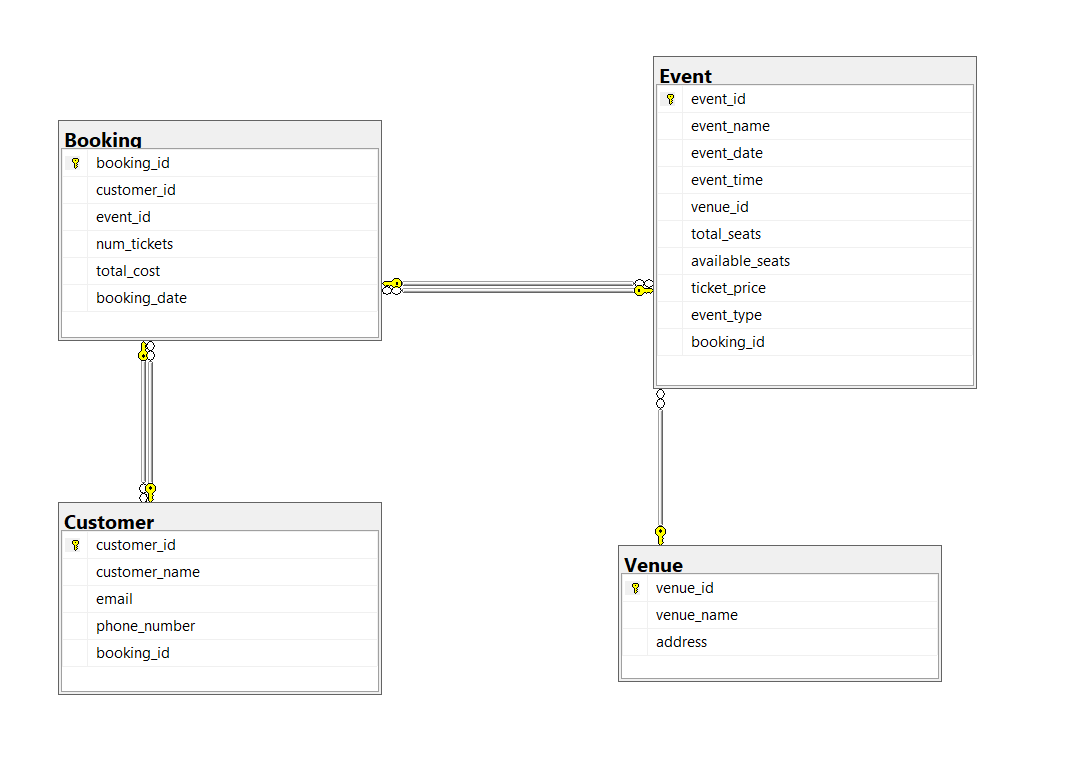
event\_type VARCHAR(50) NOT NULL,

booking\_id INT)

****

**Image 1.2 Table Creation**

3. Create an ERD (Entity Relationship Diagram) for the database.



**Image 1.3 Entity-Relationship Diagram**

4. Create appropriate Primary Key and Foreign Key constraints for referential integrity.

**--For Event Table:**

ALTER TABLE Event

ADD CONSTRAINT FK\_Event\_Venue FOREIGN KEY (venue\_id) REFERENCES Venue(venue\_id),

CONSTRAINT FK\_Event\_Booking FOREIGN KEY (booking\_id) REFERENCES Booking(booking\_id)

**--For Customer Table:**

ALTER TABLE Customer

ADD CONSTRAINT FK\_Customer\_Booking FOREIGN KEY (booking\_id) REFERENCES Booking(booking\_id)

**--For Booking Table:**

ALTER TABLE Booking

ADD CONSTRAINT FK\_Booking\_Customer FOREIGN KEY (customer\_id) REFERENCES Customer(customer\_id),

CONSTRAINT FK\_Booking\_Event FOREIGN KEY (event\_id) REFERENCES Event(event\_id)

**--Using check constraint we can allow specific values :**

ALTER TABLE Event

ADD CONSTRAINT CK\_Event\_EventType CHECK (event\_type IN ('Movie', 'Sports', 'Concert'))

**Tasks 2: Select, Where, Between, AND, LIKE:**

1. Write a SQL query to insert at least 10 sample records into each table

Inserting values into tables using ‘INSERT’ along with its columns and respected values

-- Inserting data into Venue Table:

INSERT INTO Venue (venue\_name, address)

VALUES('Heavenly Palace', ' Walajah Road, Near T Nagar Bus Stand, Chennai, Tamil Nadu 600017')

INSERT INTO Venue (venue\_name, address)

VALUES

('Paradise Gardens', 'SIDCO Industrial Estate, Vandalur Road, Chennai, Tamil Nadu 600127'),

('Atlantis', 'Besant Nagar, Chennai, Tamil Nadu 600014'),

('The Blue Fin', 'Chepauk, Chennai, Tamil Nadu 600001'),

('The Greenhouse', '1, Mount Road, Guindy, Chennai, Tamil Nadu 600032'),

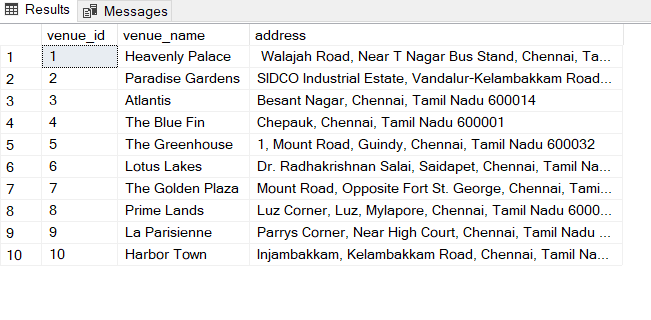
('Lotus Lakes', 'Dr. Radhakrishnan Salai, Saidapet, Chennai, Tamil Nadu 600035'),

('The Golden Plaza', 'Mount Road, Opposite Fort St. George, Chennai, Tamil Nadu 600001'),

('Prime Lands', 'Luz Corner, Luz, Mylapore, Chennai, Tamil Nadu 600004'),

('La Parisienne', 'Parrys Corner, Near High Court, Chennai, Tamil Nadu 600001.'),

('Harbor Town', 'Injambakkam, Kelambakkam Road, Chennai, Tamil Nadu 600127.')



**Image 2.1 Venue Table**

-- Inserting data into Event Table:

INSERT INTO Event (event\_name, event\_date, event\_time, venue\_id, total\_seats, available\_seats, ticket\_price, event\_type)

VALUES

('Event 1', '2024-04-01', '10:00', 1, 100, 100, 150.00, 'Movie'),

('Event 2', '2024-04-03', '11:00', 2, 150, 150, 1600.00, 'Sports'),

('Event 3', '2024-04-03', '12:00', 3, 200, 200, 700.00, 'Concert'),

('Event 4', '2024-04-06', '13:00', 1, 120, 120, 200.00, 'Movie'),

('Event 5', '2024-04-07', '14:00', 6, 180, 180, 650.00, 'Sports'),

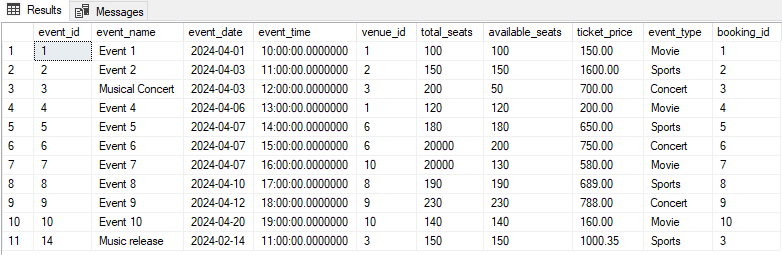
('Event 6', '2024-04-07', '15:00', 6, 220, 220, 750.00, 'Concert'),

('Event 7', '2024-04-07', '16:00', 10, 130, 130, 580.00, 'Movie'),

('Event 8', '2024-04-10', '17:00', 8, 190, 190, 689.00, 'Sports'),

('Event 9', '2024-04-12', '18:00', 9, 230, 230, 788.00, 'Concert'),

('Event 10', '2024-04-20', '19:00', 10, 140, 140, 160.00, 'Movie')



**Image 2.2 Event Table**

-- Inserting data into Customer Table:

INSERT INTO Customer (customer\_name, email, phone\_number)

VALUES

('Priya Sharma', 'priya.sharma@gmail.com', '9876543210'),

('Rahul Patel', 'rahul.patel@gmail.com', '8765432109'),

('Neha Singh', 'neha.singh@gmail.com', '7654321098'),

('Rohit Kumar', 'rohit.kumar@gmail.com', '6543210987'),

('Meera Gupta', 'meera.gupta@gmail.com', '9432109876'),

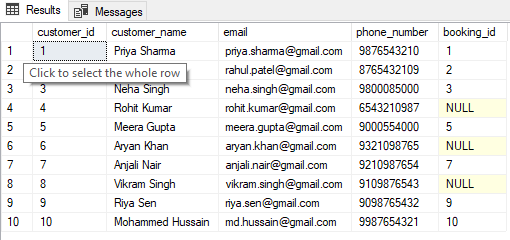
('Aryan Khan', 'aryan.khan@gmail.com', '9321098765'),

('Anjali Nair', 'anjali.nair@gmail.com', '9210987654'),

('Vikram Singh', 'vikram.singh@gmail.com', '9109876543'),

('Riya Sen', 'riya.sen@gmail.com', '9098765432'),

('Mohammed Hussain', 'md.hussain@gmail.com', '9987654321')



**Image 2.3 Customer Table**

--Inserting data into Booking Table:

INSERT INTO Booking (customer\_id, event\_id, num\_tickets, total\_cost, booking\_date)

VALUES

(1, 1, 2, 100.00, '2024-02-28'),

(2, 2, 3, 180.00, '2024-02-29'),

(3, 3, 1, 70.00, '2024-03-01'),

(4, 4, 4, 220.00, '2024-03-02'),

(5, 5, 2, 130.00, '2024-03-03'),

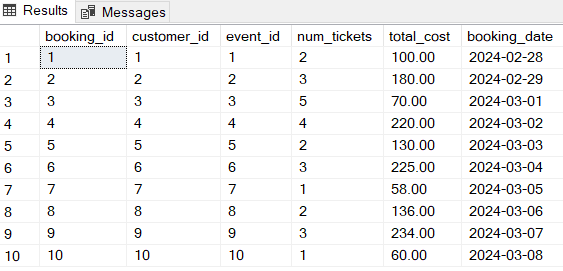
(6, 6, 3, 225.00, '2024-03-04'),

(7, 7, 1, 58.00, '2024-03-05'),

(8, 8, 2, 136.00, '2024-03-06'),

(9, 9, 3, 234.00, '2024-03-07'),

(10, 10, 1, 60.00, '2024-03-08')



**Image 2.4 Customer Table**

2. Write a SQL query to list all Events

**Query:** SELECT \* FROM Event

3. Write a SQL query to select events with available tickets.

**Query:** SELECT \* FROM Event WHERE available\_seats > 0

4. Write a SQL query to select events name partial match with ‘cup’.

**Query:** SELECT \* FROM Event WHERE event\_name LIKE '%cup%'

5. Write a SQL query to select events with ticket price range is between 1000 to 2500.

**Query:** SELECT \* FROM Event WHERE ticket\_price BETWEEN 1000 AND 2500

6. Write a SQL query to retrieve events with dates falling within a specific range.

**Query:** SELECT \* FROM Event WHERE event\_date BETWEEN '2024-04-03' AND '2024-04-10'

7. Write a SQL query to retrieve events with available tickets that also have "Concert" in their name.

**Query:** SELECT \* FROM Event WHERE available\_seats > 0 AND event\_name LIKE '%Concert%'

8. Write a SQL query to retrieve users in batches of 5, starting from the 6th user.

**Query:** SELECT \*

FROM Customer

ORDER BY customer\_id

OFFSET 5 ROWS FETCH NEXT 5 ROWS ONLY

9. Write a SQL query to retrieve bookings details contains booked no of ticket more than 4.

**Query:** SELECT \* FROM Booking WHERE num\_tickets > 4

10. Write a SQL query to retrieve customer information whose phone number end with ‘000’

**Query:** SELECT \* FROM Customer WHERE RIGHT(phone\_number, 3) = '000'

11. Write a SQL query to retrieve the events in order whose seat capacity more than 15000.

**Query:** SELECT \* FROM Event WHERE total\_seats > 15000

12. Write a SQL query to select events name not start with ‘x’, ‘y’, ‘z’.

**Query:** SELECT event\_name

FROM Event

WHERE LEFT(event\_name, 1) != 'x'

AND LEFT(event\_name, 1) != 'y'

AND LEFT(event\_name, 1) != 'z'

**Task 3. Aggregate functions, Having, Order By, GroupBy and Joins**

1. Write a SQL query to List Events and Their Average Ticket Prices.

**Query:** SELECT event\_id, event\_name, AVG(ticket\_price) AS average\_price

FROM Event

GROUP BY event\_id, event\_name

2. Write a SQL query to Calculate the Total Revenue Generated by Events.

**Query:** SELECT SUM(total\_seats \* ticket\_price) AS total\_revenue

FROM Event;

SELECT \* FROM Customer

3. Write a SQL query to find the event with the highest ticket sales.

**Query:** SELECT TOP 1 \*, (total\_seats-available\_seats) AS Tickets\_Sold

FROM Event

ORDER BY (total\_seats-available\_seats) Desc

SELECT \* FROM Event

4. Write a SQL query to Calculate the Total Number of Tickets Sold for Each Event.

**Query:** SELECT \*, (total\_seats-available\_seats) AS [Tickets Sold]

FROM Event

UPDATE Event

SET available\_seats = 50

WHERE event\_id = 3

5. Write a SQL query to Find Events with No Ticket Sales.

**Query:** SELECT \*, (total\_seats-available\_seats) AS [Tickets Sold]

FROM Event

WHERE (total\_seats-available\_seats) = 0

6. Write a SQL query to Find the User Who Has Booked the Most Tickets.

**Query:**SELECT \* FROM Booking

ORDER BY num\_tickets DESC

OR

**Query:** SELECT TOP 1 customer\_id, num\_tickets

FROM Booking

ORDER BY num\_tickets DESC

7. Write a SQL query to List Events and the total number of tickets sold for each month

**Query:** SELECT

YEAR(event\_date) AS [Booking Year],

DATENAME(MONTH, event\_date) AS [Booking Month],

event\_id,

event\_name,

(total\_seats-available\_seats) AS [Tickets Sold]

FROM Event

GROUP BY

YEAR(event\_date),

DATENAME(MONTH, event\_date),

event\_id, event\_name, (total\_seats-available\_seats)

ORDER BY [Booking Year],[Booking Month]

8. Write a SQL query to calculate the average Ticket Price for Events in Each Venue.

**Query:** SELECT

v.venue\_id,

v.venue\_id,

AVG(e.ticket\_price) AS [Average Ticket Price]

FROM VENUE AS v

JOIN Event AS e ON v.venue\_id = e.venue\_id

GROUP BY V.venue\_id, E.event\_id

9. Write a SQL query to calculate the total Number of Tickets Sold for Each Event Type

**Query:** SELECT

e.event\_type,

SUM(b.num\_tickets) AS total\_tickets\_sold

FROM Event e

JOIN Booking b ON e.event\_id = b.event\_id

GROUP BY e.event\_type

10. Write a SQL query to calculate the total Revenue Generated by Events in Each Year.

**Query:** SELECT

YEAR(b.booking\_date) AS booking\_year,

SUM(b.num\_tickets \* e.ticket\_price) AS total\_revenue

FROM Event e

JOIN Booking b ON e.event\_id = b.event\_id

GROUP BY YEAR(b.booking\_date)

11. Write a SQL query to list users who have booked tickets for multiple events

**Query:**SELECT customer\_id, COUNT(DISTINCT event\_id) AS num\_events\_booked

FROM Booking

GROUP BY customer\_id

HAVING COUNT(DISTINCT event\_id) > 1

12. Write a SQL query to calculate the Total Revenue Generated by Events for Each User.

**Query:** SELECT

b.customer\_id,

c.customer\_name,

SUM(b.num\_tickets \* e.ticket\_price) AS total\_revenue

FROM Booking b

JOIN Event e ON b.event\_id = e.event\_id

JOIN Customer c ON b.customer\_id = c.customer\_id

GROUP BY b.customer\_id, c.customer\_name

13. Write a SQL query to calculate the Average Ticket Price for Events in Each Category and Venue

**Query:** SELECT

e.event\_type,

v.venue\_name,

AVG(e.ticket\_price) AS average\_ticket\_price

FROM Event e

JOIN Venue v ON e.venue\_id = v.venue\_id

GROUP BY e.event\_type, v.venue\_name

14. Write a SQL query to list Users and the Total Number of Tickets They've Purchased in the Last 30 Days.

**Query:** SELECT

c.customer\_id,

c.customer\_name,

SUM(b.num\_tickets) AS total\_tickets\_purchased

FROM Booking b

JOIN Customer c ON b.customer\_id = c.customer\_id

WHERE b.booking\_date >= DATEADD(day, -30, GETDATE())

GROUP BY c.customer\_id, c.customer\_name

**Tasks 4: Subquery and its types**

1. Calculate the Average Ticket Price for Events in Each Venue Using a Subquery.

SELECT

V.venue\_id,

(SELECT AVG(ticket\_price)

FROM Event

WHERE venue\_id=v.venue\_id) AS [Avg Ticket Price]

FROM Venue v

2. Find Events with More Than 50% of Tickets Sold using subquery.

SELECT

event\_id,

event\_name,

(

SELECT SUM(num\_tickets) \* 100.0 / total\_seats

FROM Booking

WHERE event\_id = e.event\_id

) AS percentage\_tickets\_sold

FROM

Event e

WHERE

(

SELECT SUM(num\_tickets) \* 100.0 / total\_seats

FROM Booking

WHERE event\_id = e.event\_id

) > 50

3. Calculate the Total Number of Tickets Sold for Each Event.

SELECT

event\_id,

event\_name,

IsNull(

(Select SUM(b.num\_tickets)

From Booking b

WHERE b.event\_id = e.event\_id ) , 0)As [Total Tickets Sold]

FROM Event e

Group By event\_id, event\_name

4. Find Users Who Have Not Booked Any Tickets Using a NOT EXISTS Subquery.

SELECT \*

FROM Customer C

WHERE NOT EXISTS (

SELECT 1

FROM Booking b

WHERE b.customer\_id = c.customer\_id

)

5. List Events with No Ticket Sales Using a NOT IN Subquery.

SELECT event\_id, event\_name

FROM Event

WHERE event\_id Not In(

SELECT DISTINCT event\_id FROM Booking)

6. Calculate the Total Number of Tickets Sold for Each Event Type Using a Subquery in the FROM Clause.

SELECT event\_type, SUM(num\_tickets) AS [Total Tickets]

FROM (

SELECT e.event\_type, b.num\_tickets

FROM EVENT e

JOIN Booking b

ON b.event\_id = e.event\_id

) As Subquery

Group By event\_type

7. Find Events with Ticket Prices Higher Than the Average Ticket Price Using a Subquery in the WHERE Clause.

SELECT event\_id, event\_name, ticket\_price

FROM Event

WHERE ticket\_price>(

SELECT AVG(ticket\_price)

FROM Event

)

8. Calculate the Total Revenue Generated by Events for Each User Using a Correlated Subquery.

SELECT

c.customer\_id,

c.customer\_id,

IsNull((

SELECT SUM(total\_cost)

FROM Booking b

WHERE b.customer\_id = c.customer\_id

), 0) AS [Total revenue]

FROM Customer c

9. List Users Who Have Booked Tickets for Events in a Given Venue Using a Subquery in the WHERE Clause.

DECLARE @venueId int = 3

SELECT customer\_id, customer\_name

FROM Customer c

WHERE EXISTS(

SELECT 1

FROM Booking b

JOIN Event e ON e.event\_id = b.booking\_id

WHERE

b.customer\_id = c.customer\_id AND e.venue\_id =@venueId

)

SELECT\* FROM Booking

10. Calculate the Total Number of Tickets Sold for Each Event Category Using a Subquery with GROUP BY.

SELECT

event\_type,

SUM(num\_tickets) AS total\_tickets\_sold

FROM(

SELECT e.event\_id, e.event\_type, b.num\_tickets

FROM

Event e

INNER JOIN Booking b ON e.event\_id = b.event\_id

) AS subquery

GROUP BY

event\_type;

11. Find Users Who Have Booked Tickets for Events in each Month Using a Subquery with DATE\_FORMAT

SELECT DISTINCT

c.customer\_id,

c.customer\_name,

DATENAME (month, b.booking\_date) AS booking\_month

FROM

Customer c

INNER JOIN

Booking b ON c.customer\_id = b.customer\_id

GROUP BY

c.customer\_id,

c.customer\_name,

DATENAME (month, b.booking\_date)

12. Calculate the Average Ticket Price for Events in Each Venue Using a Subquery

SELECT

venue\_id,

venue\_name,

IsNULL((

SELECT AVG(e.ticket\_price)

FROM Event e

WHERE e.venue\_id = v.venue\_id

), 0) AS [Avg Ticket Price]

FROM Venue v

**----- Documentation Until SQL Part (05/03/2024) -----**