ASSIGNMENT – SQL – Ticket Booking System

ALOK MISHRA – ALOKMISHRA2112@GMAIL.COM

JAVA BATCH – “5”

Tasks 1: Database Design:

1. Create the database named "TicketBookingSystem".

ANS.

```
CREATE DATABASE TicketBookingSystem;
USE TicketBookingSystem; (Using this database for further query.)
```

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

- **Venu**
- **Event**
- **Customers**
- **Booking**

ANS.

```
CREATE TABLE Venu (
   venue_id INT PRIMARY KEY NOT NULL,
   venue_name VARCHAR(50) NOT NULL,
   address VARCHAR(50) NOT NULL
);

CREATE TABLE Event (
   event_id INT PRIMARY KEY NOT NULL,
   event_name VARCHAR(255) NOT NULL,
   event_date DATE NOT NULL,
   event_time TIME NOT NULL,
   venue_id INT,
   total_seats INT NOT NULL,
   available_seats INT NOT NULL,
   ticket_price FLOAT(8,2) NOT NULL,
   event_type ENUM('Movie', 'Sports', 'Concert'),
   booking_id INT
);
```

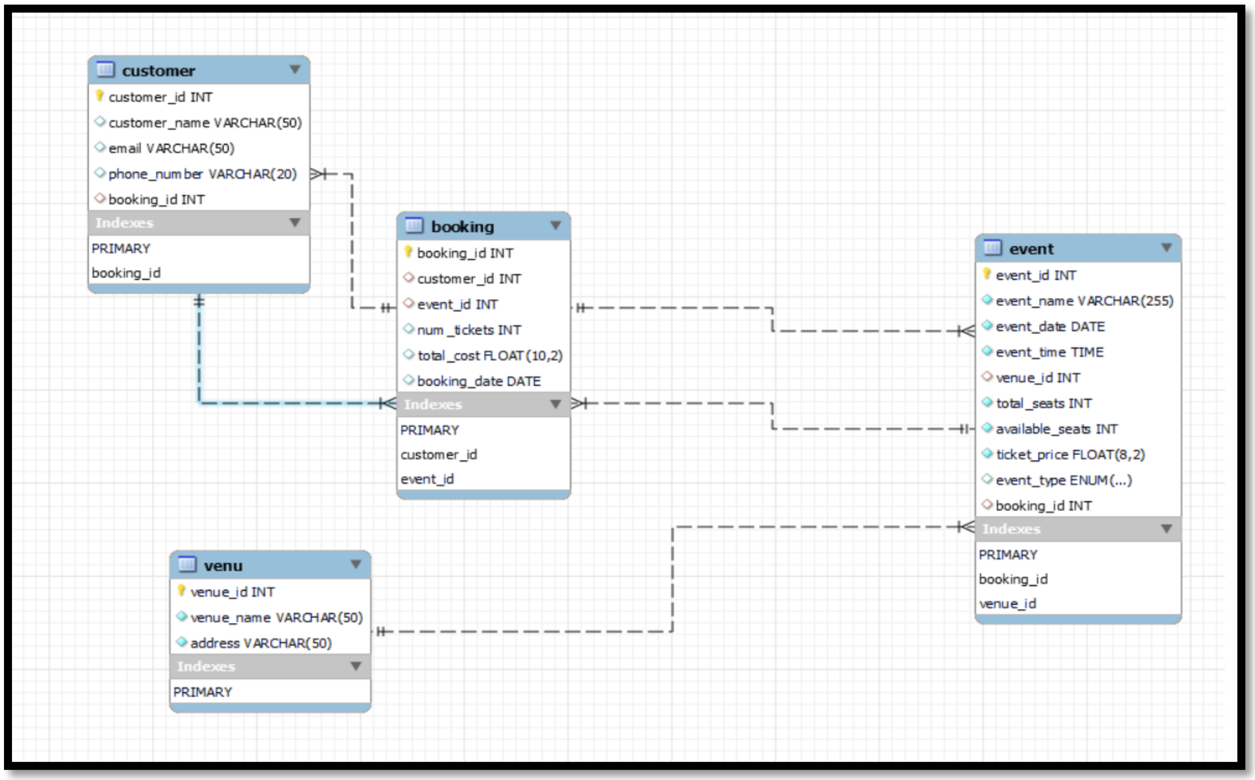
```
CREATE TABLE Customer (
   customer_id INT PRIMARY KEY,
   customer_name VARCHAR(50),
   email VARCHAR(50),
   phone_number VARCHAR(20),
   booking_id INT
);

CREATE TABLE Booking (
   booking_id INT PRIMARY KEY,
   customer_id INT,
   event_id INT,
   num_tickets INT,
   total_cost FLOAT(10, 2),
   booking_date DATE,
   FOREIGN KEY (customer_id) REFERENCES Customer(customer_id),
   FOREIGN KEY (event_id) REFERENCES Event(event_id)
);
```

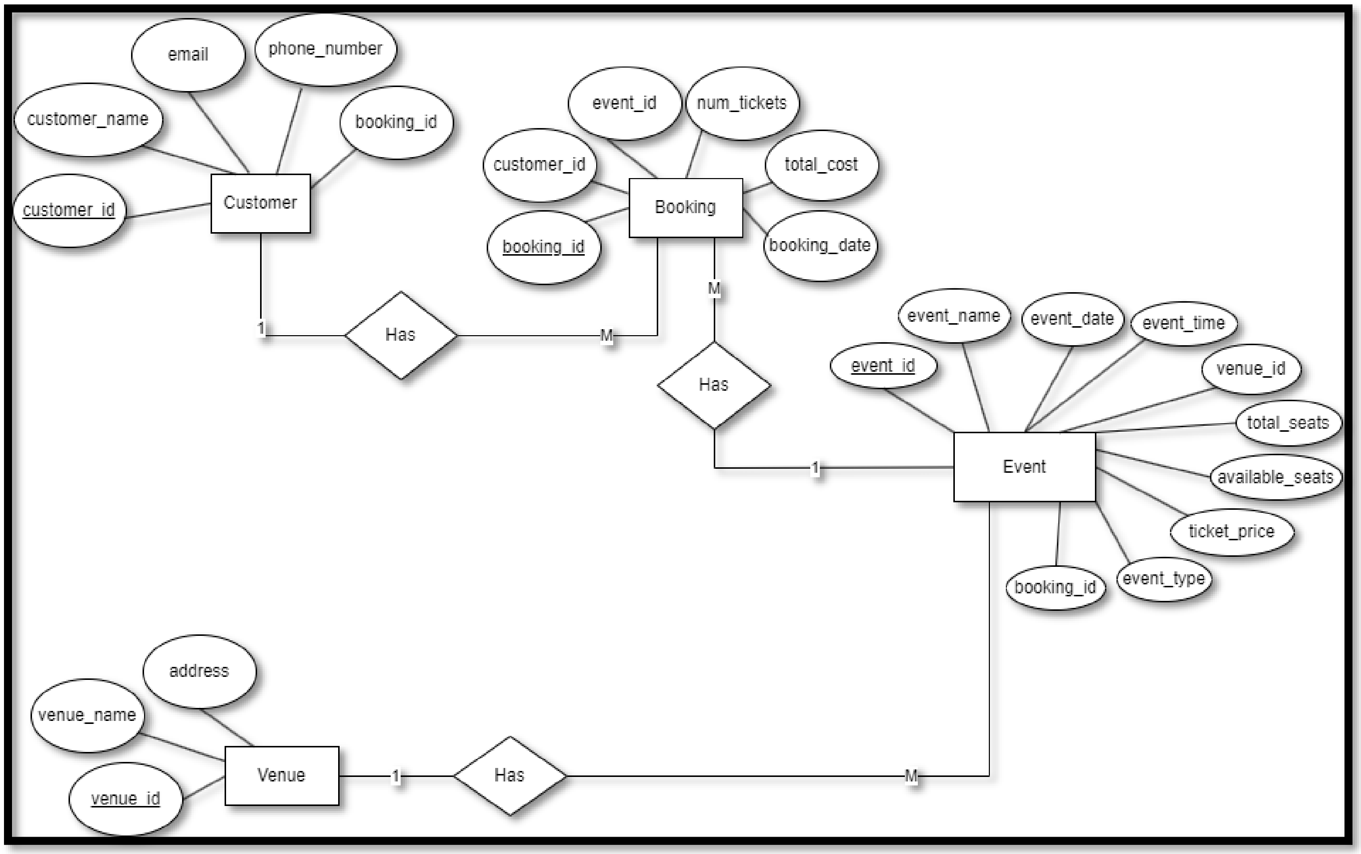
3. Create an ERD (Entity Relationship Diagram) for the database.

ANS.

Schema Diagram:



ER Diagram:



4. Create appropriate Primary Key and Foreign Key constraints for referential integrity.
ANS.
Some of the Primary Key and Foreign Key constraints are already included in the initial SQL statements, r
rest are included below.
Source code:

```
  ALTER Table event  ADD CONSTRAINT  foreign key (booking_id) REFERENCES Booking(booking_id);

  ALTER Table event  ADD CONSTRAINT  FOREIGN KEY (venue_id) REFERENCES Venu(venue_id);

  ALTER Table customer  ADD CONSTRAINT  foreign key (booking_id) REFERENCES Booking(booking_id);
```

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

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

```
INSERT INTO Venu (venue_id, venue_name, address) VALUES
(1,'Hall 1','Address 1'),
(2,'Hall 2','Address 2'),
(3,'Hall 3','Address 3'),
(4,'Hall 4','Address 4'),
(5,'Hall 5','Address 5'),
(6,'Hall 6','Address 6'),
(7,'Hall 7','Address 7'),
(8,'Hall 8','Address 8'),
(9,'Hall 9','Address 9'),
(10,'Hall 10','Address 10');

INSERT INTO Event (event_id, event_name, event_date, event_time, venue_id, total_seats,
available_seats, ticket_price, event_type, booking_id) VALUES
(101, 'Youthopia', '2023-04-11', '18:20:00', 1, 650, 80, 1500.00, 'Concert', 21),
(102, 'Bullet_Train', '2023-08-21', '15:20:00', 2, 300, 10, 400.00, 'Movie', 22),
(103, 'Nirvana', '2023-10-01', '17:30:00', 3, 1000, 350, 800.00, 'Concert', 23),
(104, 'Badminton_cup1', '2023-05-27', '10:20:00', 4, 60, 20, 50.00, 'Sports', 24),
(105, '1917', '2023-01-11', '12:20:00', 5, 350, 30, 400.00, 'Movie', 25),
(106, 'The_Imitation_Game', '2023-08-17', '9:00:00', 6, 500, 80, 1600.00, 'Movie', 26),
(107, 'Creed', '2023-05-11', '12:20:00', 7, 350, 100, 400.00, 'Movie', 27),
(108, 'Chess_Blitz', '2023-12-15', '10:00:00', 8, 100, 20, 100.00, 'Sports', 28),
(109, 'Dale_jamboree', '2023-12-01', '17:00:00', 9, 20000, 350, 1200.00, 'Concert', 29),
(110, 'Treasure_Hunt', '2023-11-07', '11:00:00', 10, 80, 40, 200.00, 'Sports', 30);

INSERT INTO Customer (customer_id, customer_name, email, phone_number, booking_id) VALUES
(11, 'Alok', 'alok1@gmail.com', '1234567890', 21),
(12, 'Tushar', 'tusharr1@gmail.com', '1234567000', 22),
(13, 'Kamal', 'kpandey007@gmail.com', '1234569890', 23),
(14, 'Vidhaan', 'vidhaan78@gmail.com', '1234467890', 24),
(15, 'Mayank', 'heymayank@gmail.com', '1234547890', 25),
(16, 'Aditya', 'bhattad1@gamil.com', '1234562890', 26),
(17, 'Chetan', 'chetan77@gmail.com', '1234561890', 27),
(18, 'Shorya', 'sure17@gmail.com', '1234567830', 28),
(19, 'Nikhil', 'nikhil22@gmail.com', '1234564890', 29),
(20, 'Hrishabh', 'hris1@gmail.com', '1234567190', 30);

INSERT INTO Booking (booking_id, customer_id, event_id, num_tickets, total_cost, booking_date)
VALUES
(21, 11, 101, 2, 3000.00, '2022-03-10'),
(22, 12, 102, 1, 400.00, '2022-03-11'),
(23, 13, 103, 2, 1600.00, '2022-04-12'),
(24, 14, 104, 4, 200.00, '2022-03-13'),
(25, 20, 108, 4, 400.00, '2022-05-14'),
(26, 16, 106, 1, 1600.00, '2022-05-15'),
(27, 17, 107, 2, 800.00, '2022-07-16'),
(28, 18, 108, 1, 100.00, '2022-06-17'),
(29, 19, 109, 3, 3600.00, '2022-09-18'),
(30, 20, 110, 5, 1000.00, '2023-11-29');
```

2. Write a SQL query to list all Events.
ANS.
```
Select * from Event;
```

3. Write a SQL query to select events with available tickets.
ANS.
```
SELECT * FROM event WHERE available_seats>0;
```

4. Write a SQL query to select events name partial match with 'cup'.
ANS.
```
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.
ANS.
```
SELECT * FROM event WHERE ticket_price  BETWEEN 1000 AND 2000;
```

6. Write a SQL query to retrieve events with dates falling within a specific range.
ANS.
```
SELECT * FROM event WHERE event_date  BETWEEN '2023-01-01' AND '2023-06-30';
```

7. Write a SQL query to retrieve events with available tickets that also have "Concert" in their name.
ANS.
```
SELECT * FROM event WHERE available_seats>0 AND event_type='Concert';
```

8. Write a SQL query to retrieve users in batches of 5, starting from the 6th user.
ANS.
```
SELECT * FROM customer LIMIT 5,5;
```

9. Write a SQL query to retrieve bookings details contains booked no of ticket more than 4.
ANS.
```
SELECT * FROM booking WHERE num_tickets>4;
```

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

ANS.

```
SELECT * FROM customer WHERE phone_number LIKE '%000';
```

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

ANS.

```
SELECT * FROM event WHERE total_seats>15000 ORDER BY total_seats;
```

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

ANS.

```
SELECT * FROM event WHERE event_name NOT LIKE 'x%' AND event_name NOT LIKE 'y%' AND
event_name NOT LIKE 'z%';
```

Tasks 3: Aggregate functions, Having, Order By, GroupBy and Joins:

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

ANS.

```
SELECT event_id, event_name, AVG(ticket_price) AS Average_Ticket_Price
FROM event
GROUP BY event_id, event_name;
```

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

ANS.

```
SELECT SUM(total_cost) AS total_revenue
FROM booking;
```

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

ANS.

```
SELECT E.event_id, E.event_name, 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_id, E.event_name
ORDER BY total_tickets_sold DESC
LIMIT 1;
```

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

ANS.

```
SELECT E.event_id, E.event_name, SUM(B.num_tickets) AS total_tickets_sold
FROM event E
```

```sql
JOIN booking B ON E.event_id = B.event_id
GROUP BY E.event_id, E.event_name;
```

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

ANS.

```sql
SELECT E.event_id, E.event_name
FROM event E
LEFT JOIN booking ON E.event_id = booking.event_id
WHERE booking.booking_id IS NULL;
```

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

ANS.

```sql
SELECT C.customer_id, C.customer_name, SUM(B.num_tickets) AS total_tickets_booked
FROM customer C
JOIN booking B ON C.customer_id = B.customer_id
GROUP BY C.customer_id, C.customer_name
ORDER BY total_tickets_booked DESC
LIMIT 1;
```

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

ANS.

```sql
SELECT MONTH(b1.booking_date) AS month, e1.event_id, e1.event_name, SUM(b1.num_tickets) AS
total_tickets_sold
FROM event e1
JOIN booking b1 ON e1.event_id = b1.event_id
GROUP BY month, e1.event_id, e1.event_name
ORDER BY month, e1.event_id;
```

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

ANS.

```sql
SELECT v0.venue_id, v0.venue_name, AVG(E.ticket_price) AS average_ticket_price
FROM venu v0
JOIN event E ON v0.venue_id = E.venue_id
GROUP BY v0.venue_id, v0.venue_name;
```

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

ANS.

```sql
SELECT event_type, SUM(num_tickets) AS total_tickets_sold
FROM event
INNER JOIN booking ON event.event_id = booking.event_id
```

```
GROUP BY event_type;
```

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

ANS.

```
SELECT YEAR(booking_date) AS Year, SUM(total_cost) AS Total_Revenue
FROM booking
GROUP BY year;
```

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

ANS.

```
SELECT C1.customer_id, C1.customer_name
FROM customer C1
INNER JOIN booking ON C1.customer_id = booking.customer_id
GROUP BY customer_id, customer_name
HAVING COUNT(DISTINCT booking.event_id) > 1;
```

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

ANS.

```
SELECT C.customer_id, C.customer_name, SUM(total_cost) AS total_revenue
FROM booking
INNER JOIN customer C ON booking.customer_id = C.customer_id
GROUP BY customer_id, customer_name;
```

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

ANS.

```
SELECT V.venue_id, V.venue_name, E.event_type, AVG(E.ticket_price) AS average_ticket_price
FROM Venu V
JOIN Event E ON V.venue_id = E.venue_id
GROUP BY V.venue_id, V.venue_name, E.event_type;
```

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

ANS.

```
SELECT C.customer_id, C.customer_name, SUM(num_tickets) AS total_tickets_purchased
FROM booking
INNER JOIN customer C ON booking.customer_id = C.customer_id
WHERE booking_date >= CURDATE() - INTERVAL 30 DAY
GROUP BY customer_id, customer_name;
```

Tasks 4: Subquery and its types

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

ANS.

```
SELECT venue_id, venue_name,
    (SELECT AVG(ticket_price) FROM event WHERE venue_id = V.venue_id) AS average_ticket_price
FROM venu V;
```

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

ANS.

```
SELECT event_id, event_name
FROM Event
WHERE (SELECT SUM(num_tickets) FROM Booking WHERE Booking.event_id = Event.event_id) > (0.5 *
total_seats);
```

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

ANS.

```
SELECT event_id, event_name,
    (SELECT SUM(num_tickets) FROM Booking WHERE Booking.event_id = Event.event_id) AS
total_tickets_sold
FROM Event;
```

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

ANS.

```
SELECT customer_id, customer_name
FROM Customer C
WHERE NOT EXISTS (SELECT 1 FROM Booking WHERE Booking.customer_id = C.customer_id);
```

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

ANS.

```
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.

ANS.

```
SELECT event_type, SUM(num_tickets) AS total_tickets_sold
FROM (SELECT B.event_id, B.num_tickets, E.event_type FROM Booking B JOIN Event E 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.

ANS.

```
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.

ANS.

```
SELECT customer_id, customer_name,
(SELECT SUM(total_cost) FROM Booking WHERE Booking.customer_id = Customer.customer_id) AS
total_revenue
FROM Customer;
```

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

ANS.

```
SELECT customer_id, customer_name
FROM Customer
WHERE customer_id IN (SELECT DISTINCT customer_id FROM Booking WHERE event_id IN (SELECT
event_id FROM Event WHERE venue_id = 1));
```

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

ANS.

```
SELECT event_type, SUM(num_tickets) AS total_tickets_sold
FROM Event E
JOIN Booking B ON E.event_id = B.event_id
GROUP BY event_type;
```

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

ANS.

FOR ONE MONTH:

```
SELECT customer_id, customer_name
FROM Customer
WHERE customer_id IN (
   SELECT DISTINCT customer_id
   FROM Booking
   WHERE DATE_FORMAT(booking_date, '%Y-%m') = '2022-05'
);
```

FOR ALL MONTHS:

```
SELECT customer_id, customer_name
FROM Customer
WHERE customer_id IN (
   SELECT DISTINCT customer_id
   FROM Booking
   WHERE DATE_FORMAT(booking_date, '%Y-%m') IN (
      SELECT DISTINCT DATE_FORMAT(booking_date, '%Y-%m')
      FROM Booking
   )
);
```

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

ANS.

```
SELECT venue_id, venue_name,
     (SELECT AVG(ticket_price) FROM Event WHERE venue_id = V.venue_id) AS average_ticket_price
FROM Venu V;
```