**SQL ASSIGNMENT**

**J520 – SRIMATHI R**

**TICKET BOOKING SYSTEM**

**TASK 1: Database Design**

1. Creating database

CREATE DATABASE TicketBookingSystem;

USE TicketBookingSystem;

1. Creating tables:

Venue table:

CREATE TABLE Venue ( venue\_id INT PRIMARY KEY AUTO\_INCREMENT, venue\_name VARCHAR(255) NOT NULL, address TEXT NOT NULL );

Event table:

CREATE TABLE Event (

-> event\_id INT PRIMARY KEY AUTO\_INCREMENT,

-> event\_name VARCHAR(255) 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 CHECK (available\_seats >= 0),

-> ticket\_price DECIMAL(10,2) NOT NULL CHECK (ticket\_price > 0),

-> event\_type ENUM('Movie', 'Sports', 'Concert') NOT NULL,

-> FOREIGN KEY (venue\_id) REFERENCES Venue(venue\_id) ON DELETE CASCADE

-> );

Customer table:

CREATE TABLE Customer (

-> customer\_id INT PRIMARY KEY AUTO\_INCREMENT,

-> customer\_name VARCHAR(255) NOT NULL,

-> email VARCHAR(255) UNIQUE NOT NULL,

-> phone\_number VARCHAR(15) NOT NULL

-> );

Booking Table:

CREATE TABLE Booking (

-> booking\_id INT PRIMARY KEY AUTO\_INCREMENT,

-> customer\_id INT NOT NULL,

-> event\_id INT NOT NULL,

-> num\_tickets INT NOT NULL CHECK (num\_tickets > 0),

-> total\_cost DECIMAL(10,2) NOT NULL CHECK (total\_cost >= 0),

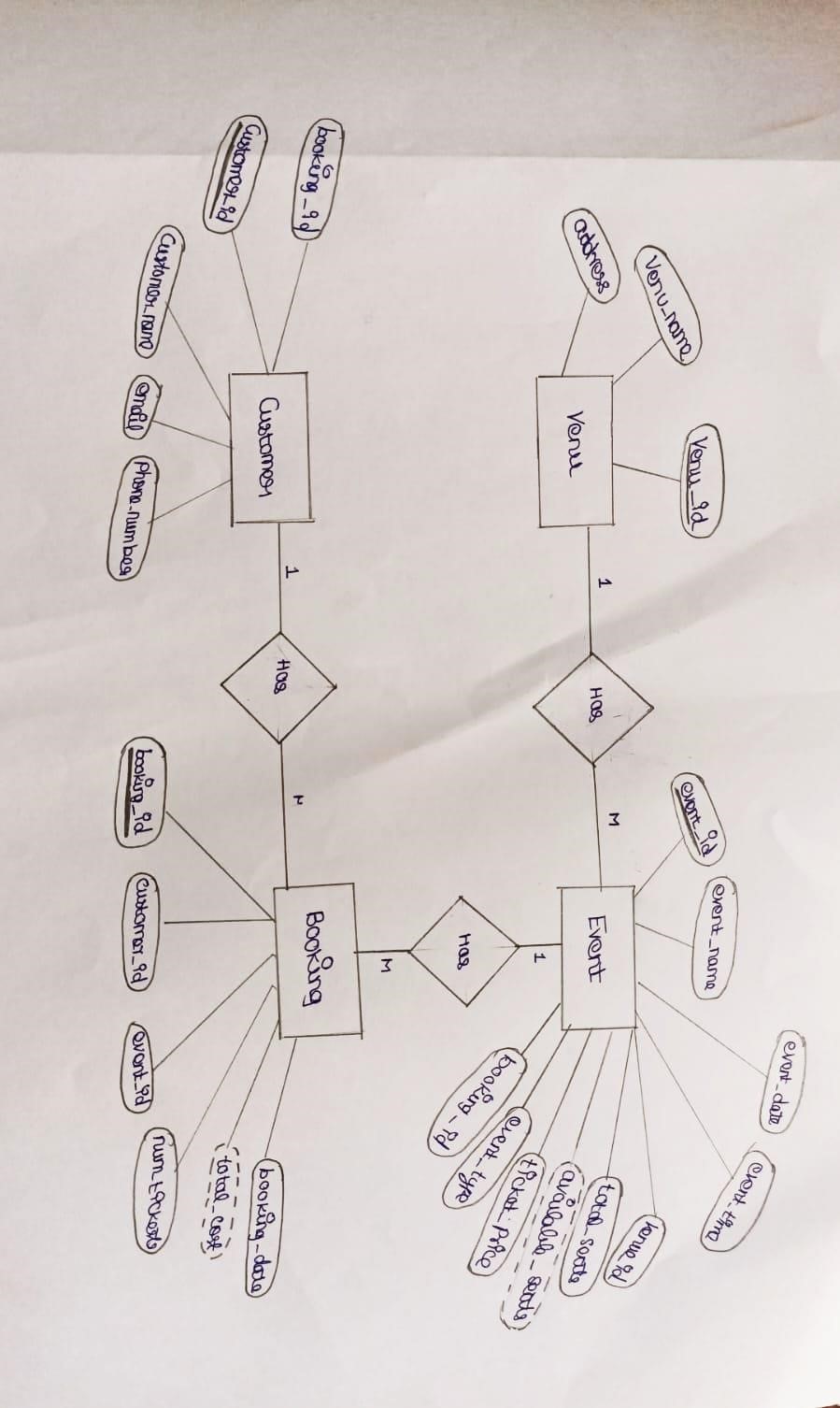
-> booking\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

-> FOREIGN KEY (customer\_id) REFERENCES Customer(customer\_id) ON DELETE CASCADE,

-> FOREIGN KEY (event\_id) REFERENCES Event(event\_id) ON DELETE CASCADE

-> );

3. Entity Relationship Diagram



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

alter table Customer add booking\_id int;

alter table Customer add constraint fk\_booking\_id FOREIGN KEY (booking\_id) REFERENCES Booking(booking\_id);

alter table Event add constraint fk\_book\_id FOREIGN KEY(booking\_id) REFERENCES Booking (booking\_id);

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

1. Inserting values to the table:

Venue table:

INSERT INTO Venue (venue\_name, address) VALUES

-> ('Stadium A', '123 Main St'),

-> ('Concert Hall B', '456 Park Ave'),

-> ('Arena C', '789 Broadway'),

-> ('Theater D', '101 Elm St'),

-> ('Sports Complex E', '202 Maple St'),

-> ('Convention Center F', '303 Oak St'),

-> ('Open Ground G', '404 Pine St'),

-> ('Stadium H', '505 Birch St'),

-> ('Concert Hall I', '606 Cedar St'),

-> ('Arena J', '707 Walnut St');

Event table:

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

-> VALUES

-> ('Football Cup Final', '2025-05-10', '18:00:00', 1, 50000, 20000, 1500.00, 'Sports'),

-> ('Rock Concert Night', '2025-06-15', '20:00:00', 2, 20000, 5000, 2500.00, 'Concert'),

-> ('Comedy Show', '2025-04-20', '19:00:00', 3, 10000, 2000, 1200.00, 'Movie'),

-> ('Basketball Playoffs', '2025-07-05', '17:30:00', 4, 30000, 10000, 1800.00, 'Sports'),

-> ('Cricket World Cup', '2025-03-30', '16:00:00', 5, 60000, 15000, 2000.00, 'Sports'),

-> ('Classical Music Concert', '2025-05-25', '19:45:00', 6, 15000, 5000, 1300.00, 'Concert'),

-> ('Movie Premiere', '2025-06-01', '21:00:00', 7, 8000, 3000, 1100.00, 'Movie'),

-> ('Jazz Night Concert', '2025-07-15', '19:00:00', 8, 12000, 4000, 1400.00, 'Concert'),

-> ('Tennis Grand Slam', '2025-08-01', '15:00:00', 9, 20000, 8000, 1700.00, 'Sports'),

-> ('Music Festival', '2025-09-10', '18:30:00', 10, 50000, 25000, 2200.00, 'Concert');

Customer table:

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

-> ('Arun Kumar', 'arun@gmail.com', '9876543000'),

-> ('Bhuvanesh Kumar', 'bhuvan@gmail.com', '9876543100'),

-> ('Charu Latha', 'charu@gmail.com', '9876543200'),

-> ('Darren Smith', 'darren@gmail.com', '9876543300'),

-> ('Emilie Thomas', 'emil@gmail.com', '9876543400'),

-> ('Karan Luthra', 'karan@gmail.com', '9876543500'),

-> ('Grace Johnson', 'grace@gmail.com', '9876543600'),

-> ('Rishabh Malhotra', 'rishabh@gmail.com', '9876543700'),

-> ('Chandhini Krishnan', 'chand@gmail.com', '9876543800'),

-> ('Aravindh Ramesh', 'aravind@gmail.com', '9876543900');

Booking table:

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

-> (1, 1, 5, 7500.00, '2025-03-01 12:30:00'),

-> (2, 2, 2, 5000.00, '2025-03-02 14:15:00'),

-> (3, 3, 3, 3600.00, '2025-03-03 16:00:00'),

-> (4, 4, 6, 10800.00, '2025-03-04 10:45:00'),

-> (5, 5, 4, 8000.00, '2025-03-05 11:30:00'),

-> (6, 6, 1, 1300.00, '2025-03-06 13:10:00'),

-> (7, 7, 3, 3300.00, '2025-03-07 09:00:00'),

-> (8, 8, 7, 9800.00, '2025-03-08 15:45:00'),

-> (9, 9, 2, 3400.00, '2025-03-09 18:00:00'),

-> (10, 10, 8, 17600.00, '2025-03-10 20:00:00');

1. SELECT \* FROM Event;
2. SELECT \* FROM Event WHERE available\_seats > 0;

1. SELECT \* FROM Event WHERE event\_name LIKE '%cup%';

1. SELECT \* FROM Event WHERE ticket\_price BETWEEN 1000 AND 2500;

1. SELECT \* FROM Event WHERE event\_date BETWEEN '2025-05-01' AND '2025-07-01';

1. SELECT \* FROM Event WHERE available\_seats > 0 AND event\_name LIKE '%Concert%';

1. SELECT \* FROM Customer LIMIT 5 OFFSET 5;

1. SELECT \* FROM Booking WHERE num\_tickets > 4;

1. SELECT \* FROM Customer WHERE phone\_number LIKE '%000';

1. SELECT \* FROM Event WHERE total\_seats > 15000 ORDER BY total\_seats DESC;

1. SELECT \* FROM Event WHERE event\_name NOT LIKE 'x%' AND event\_name NOT LIKE 'y%' AND event\_name NOT LIKE 'z%';

**TASK 3: Aggregate functions, Having, Order by, GroupBy and Joins**

1. SELECT event\_name, AVG(ticket\_price) AS average\_ticket\_price FROM Event GROUP BY event\_name;

1. SELECT e.event\_name, SUM(b.total\_cost) AS total\_revenue FROM Booking b JOIN Event e ON

b.event\_id = e.event\_id GROUP BY e.event\_name;

1. SELECT e.event\_name, SUM(b.num\_tickets) AS total\_tickets\_sold FROM Booking b JOIN Event e ON

b.event\_id = e.event\_id GROUP BY e.event\_name ORDER BY total\_tickets\_sold DESC LIMIT 1;

1. SELECT e.event\_name, SUM(b.num\_tickets) AS total\_tickets\_sold FROM Booking b JOIN Event e ON

b.event\_id = e.event\_id GROUP BY e.event\_name;

1. SELECT e.event\_name FROM Event e LEFT JOIN Booking b ON e.event\_id = b.event\_id WHERE b.booking\_id IS NULL;

1. SELECT c.customer\_name, SUM(b.num\_tickets) AS total\_tickets FROM Booking b JOIN Customer c ON b.customer\_id = c.customer\_id GROUP BY c.customer\_name ORDER BY total\_tickets DESC LIMIT 1;

1. SELECT DATE\_FORMAT(b.booking\_date, '%Y-%m') AS month, e.event\_name, SUM(b.num\_tickets) AS total\_tickets\_sold FROM Booking b JOIN Event e ON b.event\_id = e.event\_id GROUP BY month, e.event\_name ORDER BY month;

1. SELECT v.venue\_name, AVG(e.ticket\_price) AS avg\_ticket\_price FROM Event e JOIN Venue v ON

e.venue\_id = v.venue\_id GROUP BY v.venue\_name;

1. SELECT e.event\_type, SUM(b.num\_tickets) AS total\_tickets\_sold FROM Booking b JOIN Event e ON

b.event\_id = e.event\_id GROUP BY e.event\_type;

1. SELECT YEAR(b.booking\_date) AS year, SUM(b.total\_cost) AS total\_revenue FROM Booking b GROUP BY year ORDER BY year;

1. SELECT c.customer\_name, COUNT(DISTINCT b.event\_id) AS event\_count FROM Booking b JOIN

Customer c ON b.customer\_id = c.customer\_id GROUP BY c.customer\_name HAVING event\_count > 1;

12.SELECT c.customer\_name, SUM(b.total\_cost) AS total\_spent FROM Booking b JOIN Customer c ON b.customer\_id = c.customer\_idGROUP BY c.customer\_name;

1. SELECT e.event\_type, v.venue\_name, AVG(e.ticket\_price) AS avg\_ticket\_price FROM Event e JOIN

Venue v ON e.venue\_id = v.venue\_id GROUP BY e.event\_type, v.venue\_name;

1. SELECT c.customer\_name, SUM(b.num\_tickets) AS total\_tickets FROM Booking b JOIN Customer c ON b.customer\_id = c.customer\_id WHERE b.booking\_date >= NOW() - INTERVAL 30 DAY GROUP BY c.customer\_name;

**TASK 4: Subquery and its Types**

1. SELECT venue\_name, (SELECT AVG(ticket\_price) FROM Event e WHERE e.venue\_id = v.venue\_id) AS avg\_ticket\_price FROM Venue v;

1. SELECT event\_name FROM Event WHERE (total\_seats - available\_seats) > (total\_seats \* 0.5);

1. SELECT event\_name, (SELECT SUM(num\_tickets) FROM Booking b WHERE b.event\_id = e.event\_id) AS total\_tickets\_sold FROM Event e;

1. SELECT customer\_name FROM Customer c WHERE NOT EXISTS (SELECT 1 FROM Booking b WHERE b.customer\_id = c.customer\_id);

1. SELECT event\_name FROM Event WHERE event\_id NOT IN (SELECT DISTINCT event\_id FROM Booking);

1. SELECT event\_type, SUM(total\_sold) AS total\_tickets\_sold FROM (SELECT event\_type, (SELECT SUM(num\_tickets) FROM Booking b WHERE b.event\_id = e.event\_id) AS total\_sold FROM Event e) AS event\_sales GROUP BY event\_type;

1. SELECT event\_name, ticket\_price FROM Event WHERE ticket\_price > (SELECT AVG(ticket\_price) FROM Event);

1. SELECT c.customer\_name, (SELECT SUM(total\_cost) FROM Booking b WHERE b.customer\_id =

c.customer\_id) AS total\_revenue FROM Customer c;

1. SELECT customer\_name FROM Customer WHERE customer\_id IN (SELECT DISTINCT b.customer\_id

FROM Booking b JOIN Event e ON b.event\_id = e.event\_id WHERE e.venue\_id = (SELECT venue\_id FROM Venue WHERE venue\_name = 'Stadium A'));

1. 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;)

1. SELECT customer\_name, (SELECT COUNT(\*) FROM Booking b WHERE b.customer\_id = c.customer\_id

-> AND DATE\_FORMAT(b.booking\_date, '%Y-%m') = '2025-03') AS tickets\_booked FROM Customer c;

1. SELECT venue\_name, (SELECT AVG(ticket\_price) FROM Event e WHERE e.venue\_id = v.venue\_id) AS avg\_ticket\_price FROM Venue v;