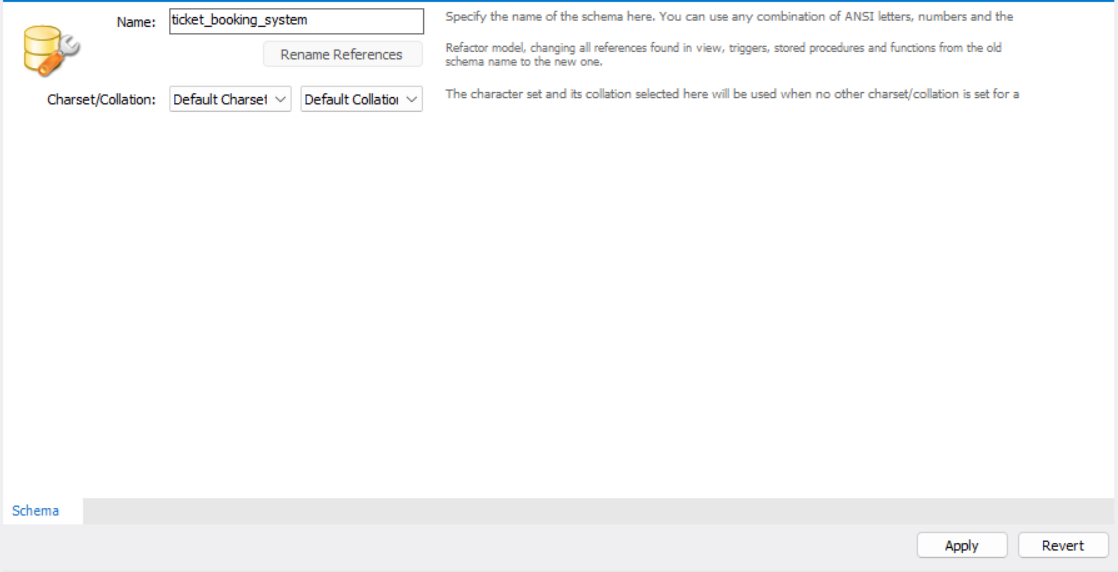
**TICKET MANAGEMENT SYSTEM**

**Task 1: Database Design:**

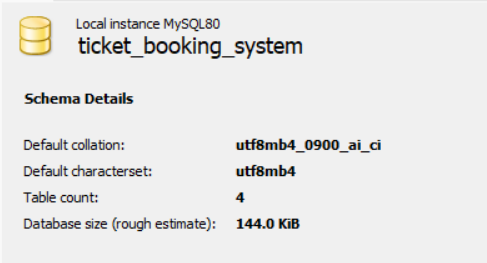
**1. Create the database named "TicketBookingSystem"**

**QUERY:**

CREATE DATABASE IF NOT EXISTS ticket\_booking\_system;



**OUTPUT:**



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

**relationships.**

• Venu

• Event

• Customers

• Booking

**A) VENUE TABLE:**

**QUERY:**

CREATE TABLE Venue (

venue\_id INT AUTO\_INCREMENT PRIMARY KEY,

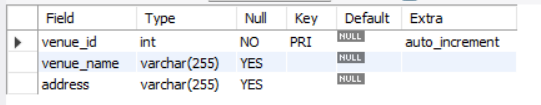
venue\_name VARCHAR(255),

address VARCHAR(255)

);

DESC Venue;

**OUTPUT:**

****

**B) EVENT Table:**

**QUERY:**

CREATE TABLE Event (

event\_id INT AUTO\_INCREMENT PRIMARY KEY,

event\_name VARCHAR(255),

event\_date DATE,

event\_time TIME,

venue\_id INT,

total\_seats INT,

available\_seats INT,

ticket\_price DECIMAL(10, 2),

event\_type ENUM('Movie', 'Sports', 'Concert'),

-- FOREIGN KEY (venue\_id) REFERENCES ticket\_booking\_system.Venue(venue\_id)

);

DESC Event;

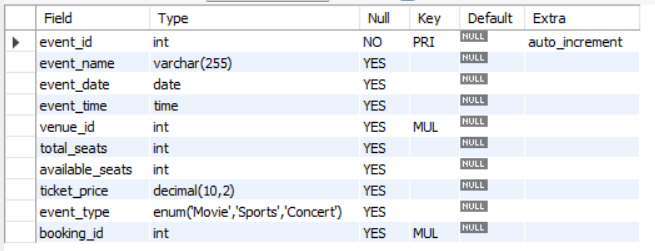
ALTER TABLE ticket\_booking\_system.Event

ADD COLUMN booking\_id INT,

ADD CONSTRAINT fk\_event\_booking\_id FOREIGN KEY (booking\_id) REFERENCES ticket\_booking\_system.Booking(booking\_id);

DESC Event;

**OUTPUT:**



**C) CUSTOMERS Table:**

**QUERY:**

CREATE TABLE Customer (

customer\_id INT AUTO\_INCREMENT PRIMARY KEY,

customer\_name VARCHAR(255),

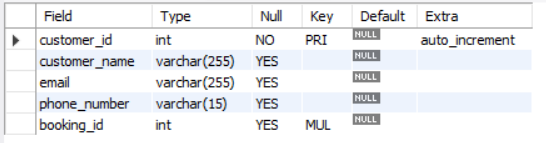
email VARCHAR(255),

phone\_number VARCHAR(15)

--FOREIGN KEY (booking\_id) REFERENCES ticket\_booking\_system.Venue(booking\_id)

);

**OUTPUT:**



**D) BOOKING Table:**

**QUERY:**

CREATE TABLE Booking (

booking\_id INT AUTO\_INCREMENT PRIMARY KEY,

customer\_id INT,

event\_id INT,

num\_tickets INT,

total\_cost DECIMAL(10, 2),

booking\_date DATE,

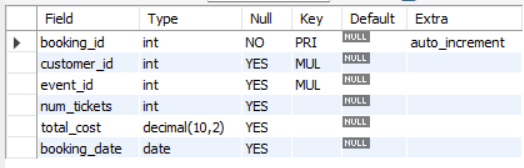
FOREIGN KEY (customer\_id) REFERENCES ticket\_booking\_system.Customer(customer\_id),

FOREIGN KEY (event\_id) REFERENCES ticket\_booking\_system.Event(event\_id)

);

DESC customer;

**OUTPUT**:



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

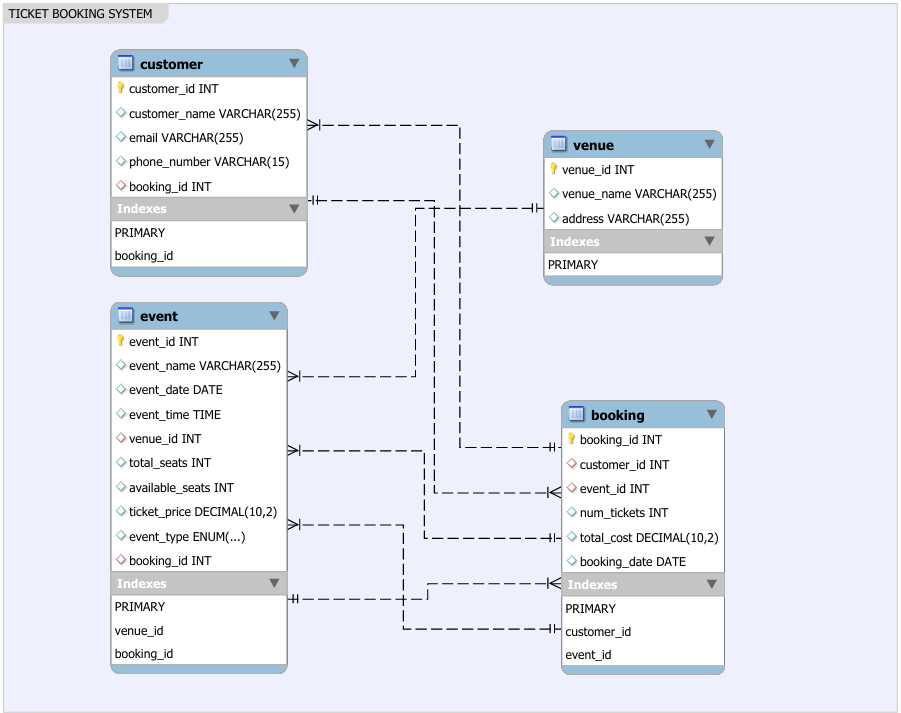
****

Fig.ER-diagram Ticket Management System

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

**a) Event Table:**

CREATE TABLE Event (

event\_id INT AUTO\_INCREMENT PRIMARY KEY,

FOREIGN KEY (venue\_id) REFERENCES ticket\_booking\_system.Venue(venue\_id)

* To add booking\_id as foreign key from Booking Table in Event Table.

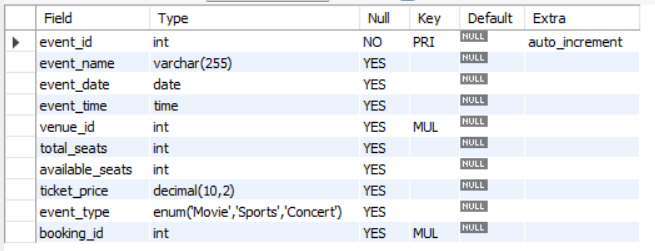
ALTER TABLE ticket\_booking\_system.Event

ADD CONSTRAINT fk\_event\_booking\_id

FOREIGN KEY (booking\_id)

REFERENCES ticket\_booking\_system.Booking(booking\_id);

**OUTPUT:**



**b) Customers Table:**

To add booking\_id as foreign key from booking table in Customers Table

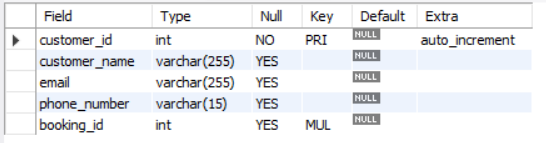
ALTER TABLE ticket\_booking\_system.Customer

ADD FOREIGN KEY (booking\_id)

REFERENCES ticket\_booking\_system.Booking(booking\_id);

DESC Customer;

**OUTPUT:**



**c) Booking Table:**

CREATE TABLE Booking (

booking\_id INT AUTO\_INCREMENT PRIMARY KEY,

.

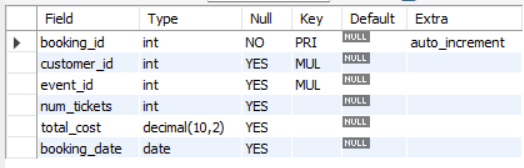
.

FOREIGN KEY (customer\_id) REFERENCES ticket\_booking\_system.Customer(customer\_id),

FOREIGN KEY (event\_id) REFERENCES ticket\_booking\_system.Event(event\_id)

);

OUTPUT:



**TASK 2: SELECT, WHERE, BETWEEN, AND, LIKE**

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

**a) Venue Table**

INSERT INTO Venue (venue\_name, address)

VALUES

('ABC Theater', '123 Main Street'),

('XYZ Stadium', '456 Park Avenue'),

('City Hall', '789 Elm Road'),

('Sunset Arena', '101 Sunset Boulevard'),

('Moonlight Hall', '222 Star Avenue'),

('Ocean View Park', '333 Ocean Drive'),

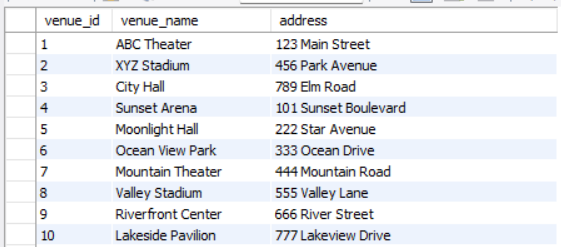
('Mountain Theater', '444 Mountain Road'),

('Valley Stadium', '555 Valley Lane'),

('Riverfront Center', '666 River Street'),

('Lakeside Pavilion', '777 Lakeview Drive');

**OUTPUT:**



**b) Event Table**

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

VALUES

('Movie Night', '2024-04-20', '18:00:00', 1, 100, 100, 15.00, 'Movie'),

('Rock Concert', '2024-04-22', '20:00:00', 2, 500, 500, 50.00, 'Concert'),

('Football Match', '2024-04-25', '15:00:00', 3, 10000, 9500, 25.00, 'Sports'),

('Drama Play', '2024-04-28', '19:30:00', 4, 200, 180, 20.00, 'Sports'),

('Live Comedy Show', '2024-04-30', '21:00:00', 5, 300, 290, 30.00, 'Concert'),

('Basketball Game', '2024-05-02', '16:30:00', 6, 8000, 7500, 35.00, 'Sports'),

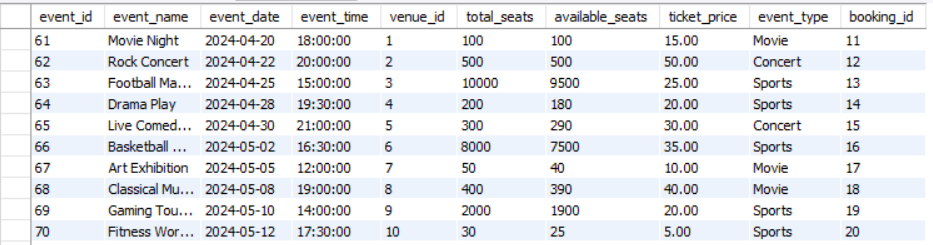
('Art Exhibition', '2024-05-05', '12:00:00', 7, 50, 40, 10.00, 'Movie'),

('Classical Music Concert', '2024-05-08', '19:00:00', 8, 400, 390, 40.00, 'Movie'),

('Gaming Tournament', '2024-05-10', '14:00:00', 9, 2000, 1900, 20.00, 'Sports'),

('Fitness Workshop', '2024-05-12', '17:30:00', 10, 30, 25, 5.00, 'Sports');

**OUTPUT:**



**c) Customer Table**

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

VALUES

('Alice Johnson', 'alice@example.com', '123-456-7890'),

('Bob Smith', 'bob@example.com', '987-654-3210'),

('John Doe', 'john.doe@example.com', '111-222-3333'),

('Jane Doe', 'jane.doe@example.com', '444-555-6666'),

('Michael Brown', 'michael.brown@example.com', '777-888-9999'),

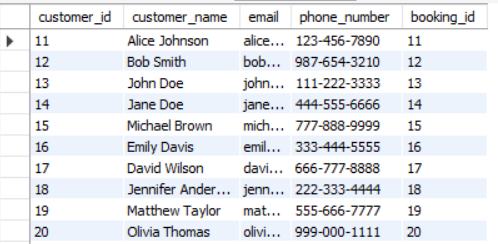
('Emily Davis', 'emily.davis@example.com', '333-444-5555'),

('David Wilson', 'david.wilson@example.com', '666-777-8888'),

('Jennifer Anderson', 'jennifer.anderson@example.com', '222-333-4444'),

('Matthew Taylor', 'matthew.taylor@example.com', '555-666-7777');

**OUTPUT:**



**d) Booking Table**

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

VALUES

(11, 61, 2, 40.00, '2024-04-25'),

(12, 62, 3, 150.00, '2024-04-26'),

(13, 63, 1, 20.00, '2024-04-27'),

(14, 64, 4, 80.00, '2024-04-28'),

(15, 65, 2, 30.00, '2024-04-29'),

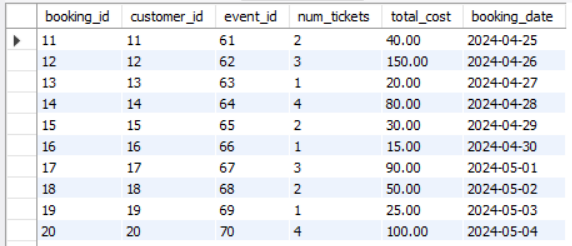
(16, 66, 1, 15.00, '2024-04-30'),

(17, 67, 3, 90.00, '2024-05-01'),

(18, 68, 2, 50.00, '2024-05-02'),

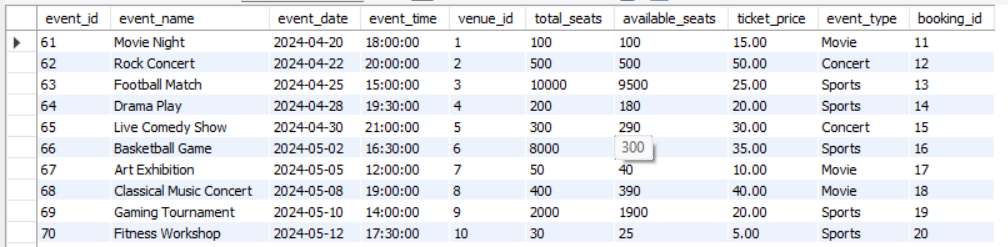
(19, 69, 1, 25.00, '2024-05-03'),

(20, 70, 4, 100.00, '2024-05-04');



**2. Write a SQL query to list all Events.**

> SELECT \* FROM event;

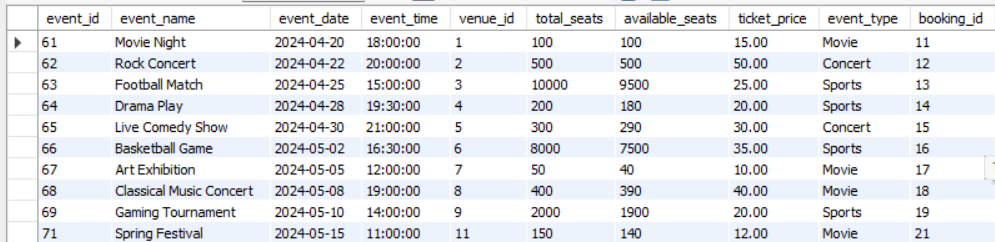


**3. Write a SQL query to select events with available tickets.**

> SELECT \*

FROM Event

WHERE available\_seats > 0;

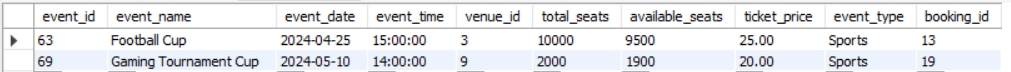


**4. Write a SQL query to select events name partial match with ‘cup’.**

> SELECT \*

FROM event

WHERE event\_name LIKE '%cup%';

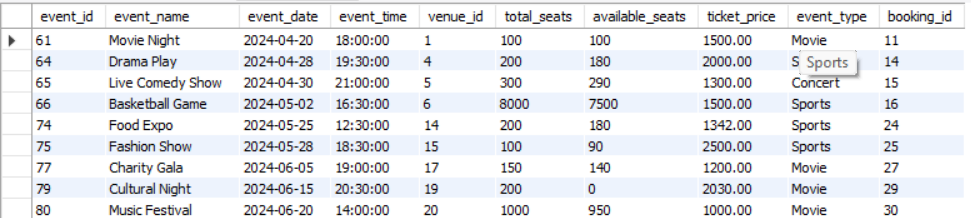
****

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

> SELECT \*

FROM event

WHERE ticket\_price BETWEEN 1000 AND 2500;

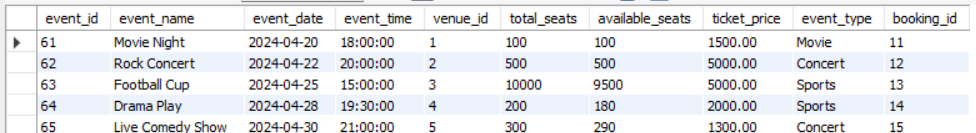


**6. Write a SQL query to retrieve events with dates falling within a specific range.**

> SELECT \*

FROM event

WHERE event\_date BETWEEN '2024-04-01' AND '2024-04-30';



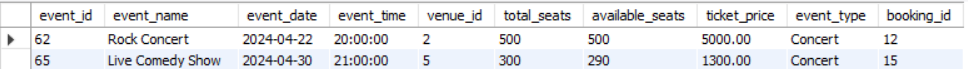
**7. Write a SQL query to retrieve events with available tickets that also have "Concert" in their**

**name.**

> 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.**

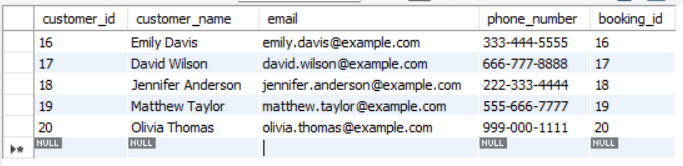
> SELECT \*

FROM Customer

Where customer\_id BETWEEN 16 AND 20;

OR

>SELECT \* FROM Customer LIMIT 5 OFFSET 5;

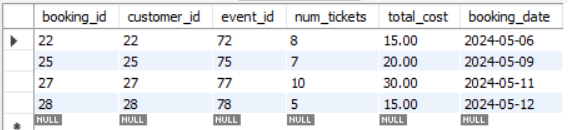


**9. Write a SQL query to retrieve bookings details contains booked no of ticket more than 4.**

> SELECT \*

FROM booking

WHERE num\_tickets > 4;

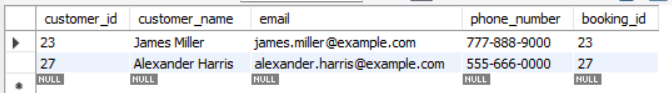


**10. Write a SQL query to retrieve customer information whose phone number end with ‘000’**

> SELECT \*

FROM customer

WHERE phone\_number LIKE '%000';

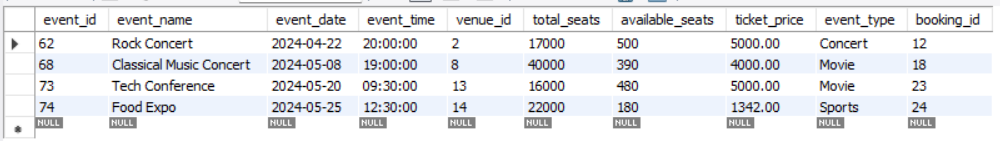


**11. Write a SQL query to retrieve the events in order whose seat capacity more than 15000.**

> SELECT \*

FROM event

WHERE total\_seats > 15000;

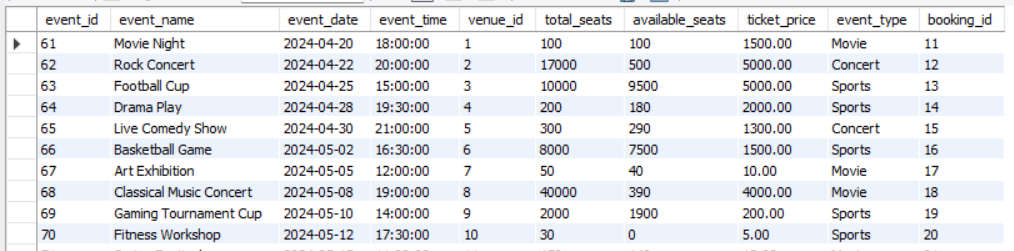


**12. Write a SQL query to select events name not start with ‘x’, ‘y’, ‘z’**

> SELECT \*

FROM event

WHERE event\_name NOT LIKE 'x%' AND event\_name NOT LIKE 'y%' AND event\_name NOT LIKE 'z%';



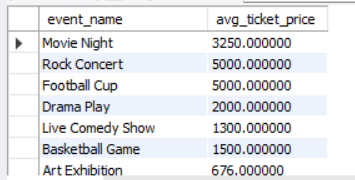
**Tasks 3: Aggregate functions, Having, Order By, GroupBy and Joins**

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

SELECT event\_name, AVG(ticket\_price) AS avg\_ticket\_price

FROM event

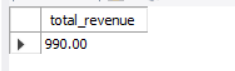
GROUP BY event\_name;



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

SELECT SUM(total\_cost) AS total\_revenue

FROM Booking;



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

> SELECT E.event\_id, E.event\_name, SUM(B.num\_tickets) AS total\_tickets\_sold

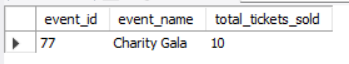
FROM Booking B

Join Event E ON B.event\_id = E.event\_id

GROUP BY event\_id

ORDER BY total\_tickets\_sold DESC

LIMIT 1;



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

SELECT E.event\_id, E.event\_name, SUM(num\_tickets) AS total\_tickets\_sold

FROM Event E

LEFT JOIN Booking B ON B.event\_id = E.event\_id

GROUP BY event\_id

ORDER BY total\_tickets\_sold DESC;



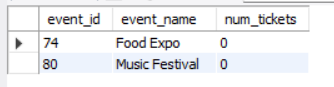
**5. Write a SQL query to Find Events with No Ticket Sales.**

SELECT E.event\_id, E.event\_name, B.num\_tickets

FROM Event E

LEFT JOIN Booking B ON B.event\_id = E.event\_id

WHERE B.num\_tickets = 0;



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

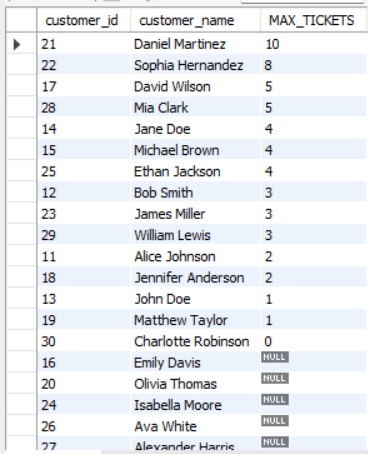
SELECT C.customer\_id, C.customer\_name, MAX(B.num\_tickets) AS MAX\_TICKETS

FROM Booking B

RIGHT JOIN Customer C ON B.customer\_id = C.customer\_id

GROUP BY C.customer\_id

ORDER BY MAX\_TICKETS DESC;

****

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

SELECT E.event\_id, E.event\_name, MONTH(B.booking\_date) AS booking\_month,

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

FROM Event E

LEFT JOIN Booking B ON B.event\_id = E.event\_id

GROUP BY booking\_month, E.event\_id;

****

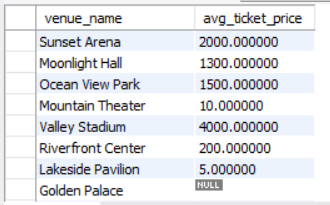
**8. Write a SQL query to calculate the average Ticket Price for Events in Each Venue.**

SELECT V.venue\_name, AVG(E.ticket\_price) AS avg\_ticket\_price

FROM Venue V

LEFT JOIN Event E ON E.venue\_id = V.venue\_id

GROUP BY V.venue\_id;



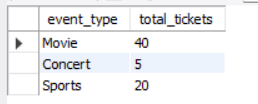
**9. Write a SQL query to calculate the total Number of Tickets Sold for Each Event Type.**

SELECT E.event\_type, SUM(B.num\_tickets) AS total\_tickets

FROM Booking B

JOIN Event E ON E.event\_id = B.event\_id

GROUP BY event\_type;



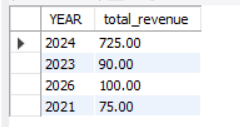
**10. Write a SQL query to calculate the total Revenue Generated by Events in Each Year.**

SELECT YEAR(B.booking\_date) AS YEAR, SUM(B.total\_cost) AS total\_revenue

FROM Booking B

JOIN Event E ON E.booking\_id = B.booking\_id

GROUP BY YEAR(B.booking\_date);



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

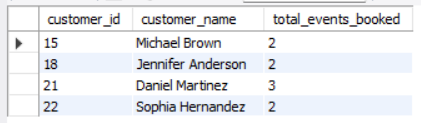
SELECT B.customer\_id, C.customer\_name, COUNT(DISTINCT B.event\_id) AS total\_events\_booked

FROM Booking B

JOIN Customer C ON C.customer\_id = B.customer\_id

GROUP BY B.customer\_id

HAVING total\_events\_booked > 1;



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

SELECT C.customer\_id, C.customer\_name, SUM(B.total\_cost) AS Total\_Revenue

FROM Booking B

RIGHT JOIN Customer C ON B.customer\_id = C.customer\_id

GROUP BY C.customer\_id, C.customer\_name;



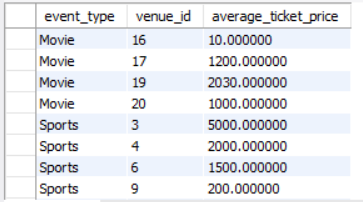
**13. Write a SQL query to calculate the Average Ticket Price for Events in Each Category and Venue.**

SELECT event\_type, venue\_id, AVG(ticket\_price) AS average\_ticket\_price

FROM Event

GROUP BY event\_type, venue\_id

ORDER BY event\_type;

****

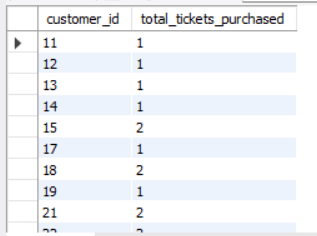
**14. Write a SQL query to list Users and the Total Number of Tickets They've Purchased in the Last 30 Days.**

SELECT B.customer\_id, C. customer\_name, COUNT(\*) AS total\_tickets\_purchased

FROM Booking B

WHERE B.booking\_date >= DATE\_SUB(CURDATE(), INTERVAL 30 DAY)

GROUP BY B.customer\_id,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,

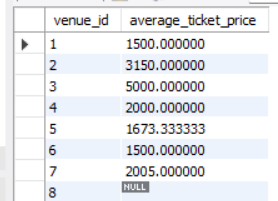
(SELECT AVG(E.ticket\_price)

FROM Event E

WHERE E.venue\_id = V.venue\_id

) AS average\_ticket\_price

FROM Venue V;



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

SELECT event\_id, event\_name

FROM Event e

WHERE event\_id IN (

SELECT b.event\_id

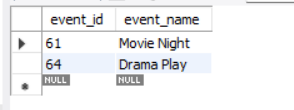
FROM Booking b

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

GROUP BY b.event\_id

HAVING SUM(b.num\_tickets) > e.total\_seats \* 0.5

);



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

SELECT event\_id, event\_name,

(SELECT SUM(b.num\_tickets)

FROM Booking b

WHERE b.event\_id = e.event\_id) AS total\_tickets\_sold

FROM Event e;

****

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

SELECT \*

FROM Customer c

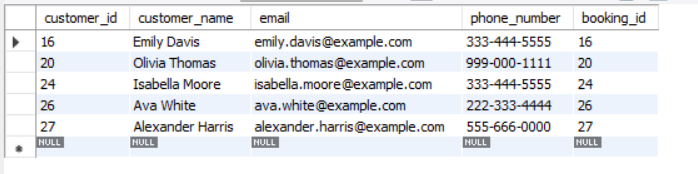
WHERE NOT EXISTS (

SELECT \*

FROM Booking b

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

);



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

SELECT \*

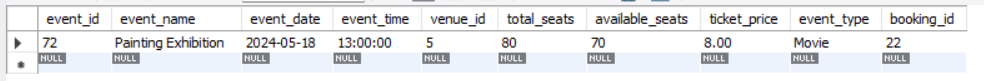
FROM Event e

WHERE e.event\_id NOT IN (

SELECT 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

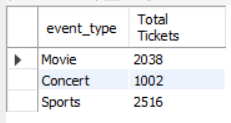
FROM EVENT e

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 \*

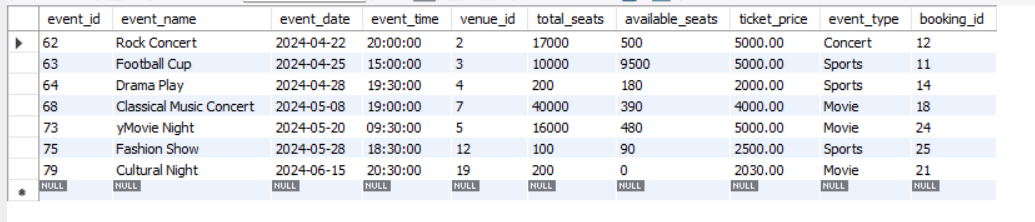
FROM Event e

WHERE e.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\_name,

(

SELECT SUM(b.num\_tickets \* e.ticket\_price)

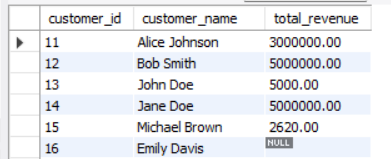
FROM Booking b

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

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

) 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.**

SELECT \*

FROM Customer c

WHERE EXISTS (

SELECT \*

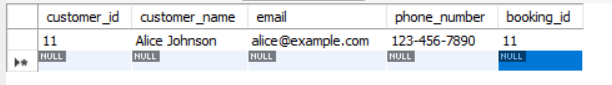
FROM Booking b

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

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

AND e.venue\_id = 1

);



**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 (

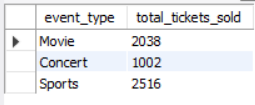
SELECT e.event\_type, b.num\_tickets

FROM Event e

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**

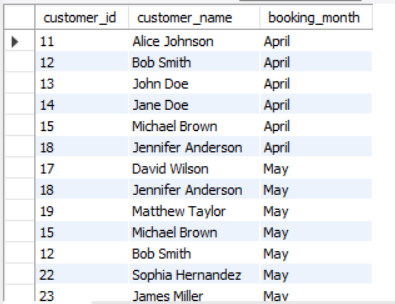
**DATE\_FORMAT.**

SELECT DISTINCT c.customer\_id, c.customer\_name, MONTHNAME(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, MONTHNAME(b.booking\_date);



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

SELECT v.venue\_id, v.venue\_name,

(SELECT AVG(ticket\_price)

FROM Event

WHERE venue\_id = v.venue\_id) AS avg\_ticket\_price

FROM Venue v;

