Tasks 1: Database Design:

1.Create the database named "TicketBookingSystem"

CREATE DATABASE TicketBookingSystem;

USE TicketBookingSystem;

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

• Venue

• Event

• Customers

• Booking

Venue Table:

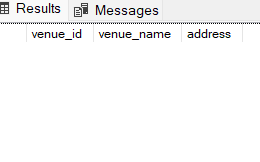
CREATE TABLE Venue (

venue\_id INT PRIMARY KEY,

venue\_name VARCHAR(255),

address VARCHAR(255)

);



Event Table:

CREATE TABLE Event (

event\_id INT PRIMARY KEY,

event\_name VARCHAR(255),

event\_date DATE,

event\_time TIME,

venue\_id INT,

total\_seats INT,

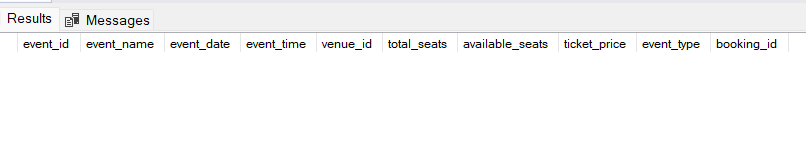
available\_seats INT,

ticket\_price DECIMAL(10, 2),

event\_type VARCHAR(50),

booking\_id INT

);



Customer Table:

CREATE TABLE Customer (

customer\_id INT PRIMARY KEY,

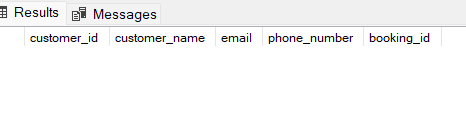
customer\_name VARCHAR(255),

email VARCHAR(255),

phone\_number VARCHAR(20),

booking\_id INT

);



Booking Table:

CREATE TABLE Booking (

booking\_id INT PRIMARY KEY,

customer\_id INT,

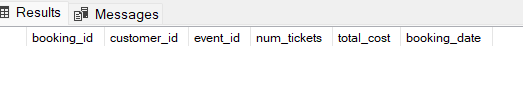
event\_id INT,

num\_tickets INT,

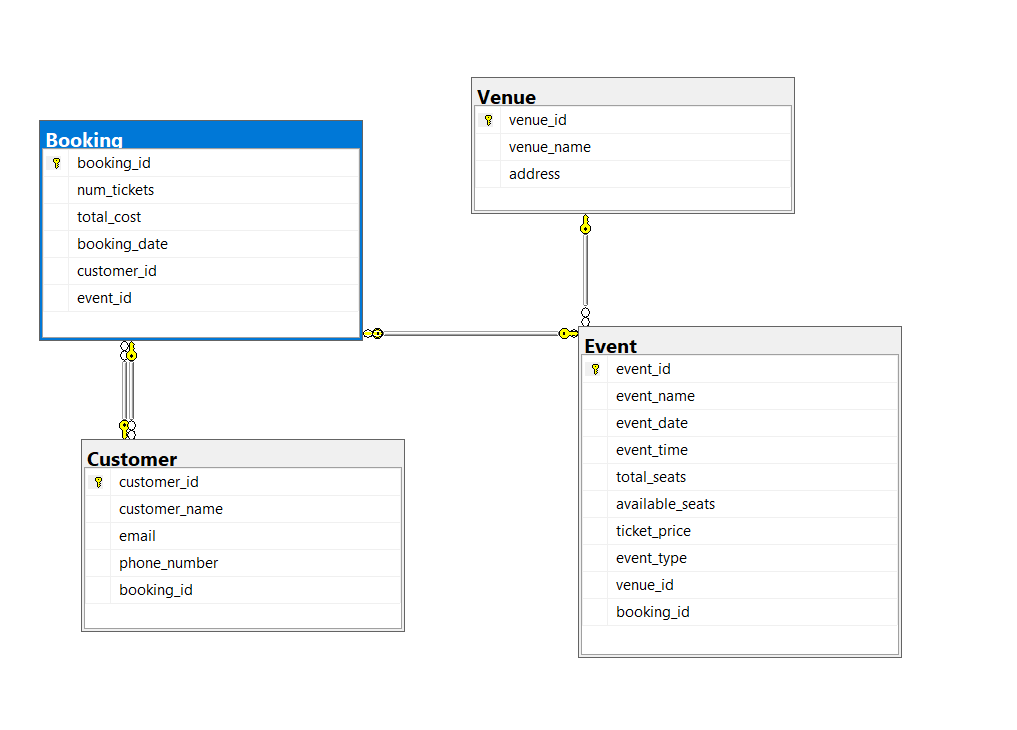
total\_cost DECIMAL(10, 2),

booking\_date DATE

);



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



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

Alter table Event

add venue\_id INT

Alter table Event

add booking\_id INT

Alter table Customer

add booking\_id INT

Alter table booking

add customer\_id INT

Alter table booking

add event\_id INT

ALTER TABLE Event

add constraint FK\_venue\_id FOREIGN KEY (venue\_id) REFERENCES Venue(venue\_id);

ALTER TABLE Event

add constraint FK\_booking\_id FOREIGN KEY (booking\_id) REFERENCES Booking(booking\_id);

ALTER TABLE Customer

add constraint FKey\_booking\_id FOREIGN KEY (booking\_id) REFERENCES Booking(booking\_id);

ALTER TABLE booking

add constraint FK\_customer\_id FOREIGN KEY(customer\_id) references Customer(customer\_id);

ALTER TABLE booking

add constraint FK\_event\_id FOREIGN KEY(event\_id) references Event(event\_id);

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

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

INSERT INTO Venue (venue\_id, venue\_name, address) VALUES

(1, 'Delhi', 'Suite 640 1856 Turner Camp, East Harriett, NJ 34953'),

(2, 'Mumbai', 'F-1/10, Sector 10, Vashi'),

(3, 'Hyderabad', '5-1-459/7, Jam Bagh'),

(4, 'Bangalore', '92, 2nd Flr Mosque Road, Frazer Town'),

(5, 'Ahmedabad', 'Opp Spss Hall Nr Aditya Complex, Navrangpura'),

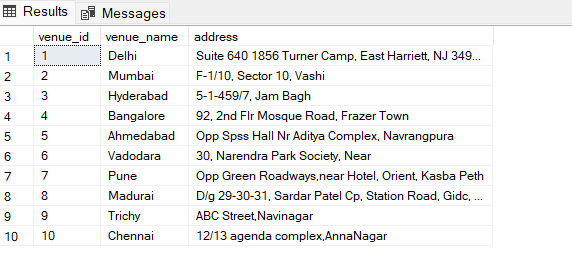
(6, 'Vadodara', '30, Narendra Park Society, Near'),

(7, 'Pune', 'Opp Green Roadways,near Hotel, Orient, Kasba Peth'),

(8, 'Madurai', 'D/g 29-30-31, Sardar Patel Cp, Station Road, Gidc, Ankleshwar'),

(9, 'Trichy', 'ABC Street,Navinagar'),

(10, 'Chennai', '12/13 agenda complex,AnnaNagar');



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

VALUES

(1, 'Fictional', '2023-01-01', '12:00:00', 100, 50, 250.00, 'Movie',1,1),

(2, 'Kabbadi Cup', '2023-02-02', '15:30:00', 1500, 500, 380.00, 'Sports',2,2),

(3, 'Dance Concert', '2023-02-03', '15:00:00', 155, 100, 2000.00, 'Concert',3,3),

(4, 'Horror', '2023-02-04', '11:30:00', 150, 100, 300.00, 'Movie',4,4),

(5, 'singing', '2023-02-05', '12:30:00', 150, 100, 2000.00, 'Concert',5,5),

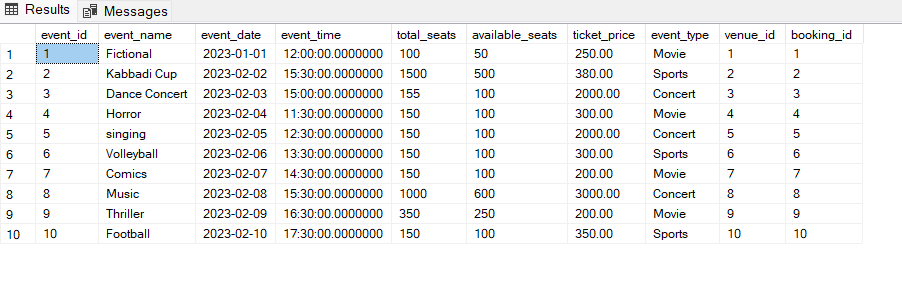
(6, 'Volleyball', '2023-02-06', '13:30:00', 150, 100, 300.00, 'Sports',6,6),

(7, 'Comics', '2023-02-07', '14:30:00', 150, 100, 200.00, 'Movie',7,7),

(8, 'Music', '2023-02-08', '15:30:00', 1000, 600, 3000.00, 'Concert',8,8),

(9, 'Thriller', '2023-02-09', '16:30:00', 350, 250, 200.00, 'Movie',9,9),

(10, 'Football', '2023-02-10', '17:30:00', 150, 100, 350.00, 'Sports',10,10);



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

(1, 'John', 'john@example.com', '123-456-7890',1),

(2, 'Jane', 'jane@example.com', '987-654-3210',2),

(3, 'Bob', 'bob@example.com', '423-456-7890',3),

(4, 'Alice', 'alice@example.com', '523-456-7890',4),

(5, 'Charlie', 'charlie@example.com', '623-456-7890',5),

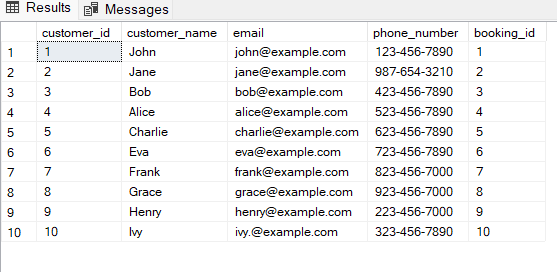
(6, 'Eva', 'eva@example.com', '723-456-7890',6),

(7, 'Frank', 'frank@example.com', '823-456-7000',7),

(8, 'Grace', 'grace@example.com', '923-456-7000',8),

(9, 'Henry', 'henry@example.com', '223-456-7000',9),

(10, 'Ivy', 'ivy.@example.com', '323-456-7890',10);



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

(1, 2, 4000.00, '2023-01-01 10:00:00',1,1),

(2, 3, 900.00, '2023-02-02 12:30:00',2, 2),

(3, 2, 5550.00, '2023-01-03 09:00:00',3, 3),

(4, 2, 4444.00, '2023-01-04 09:00:00', 4, 4),

(5, 1, 4300.00, '2023-01-05 10:00:00', 5, 5),

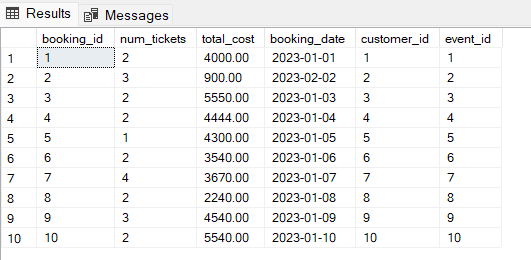
(6, 2, 3540.00, '2023-01-06 11:00:00', 6, 6),

(7, 4, 3670.00, '2023-01-07 11:00:00',7, 7),

(8, 2, 2240.00, '2023-01-08 10:00:00', 8, 8),

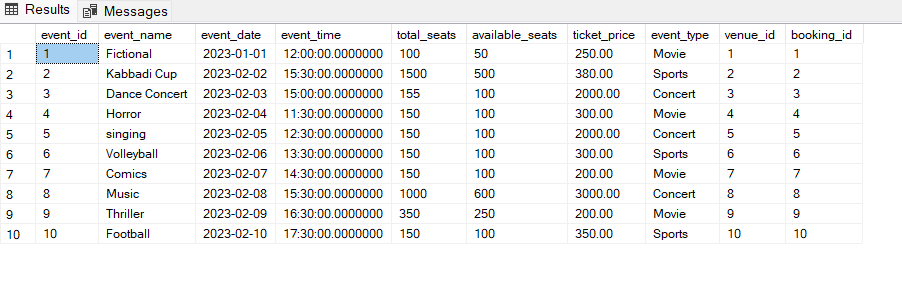
(9, 3, 4540.00, '2023-01-09 09:00:00',9,9),

(10, 2, 5540.00, '2023-01-10 10:00:00',10, 10);



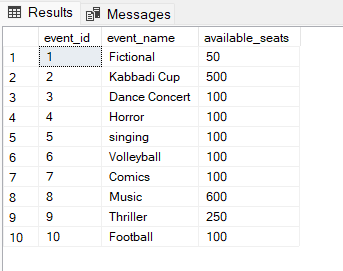
2. Write a SQL query to list all Events.

SELECT \* FROM Event;



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

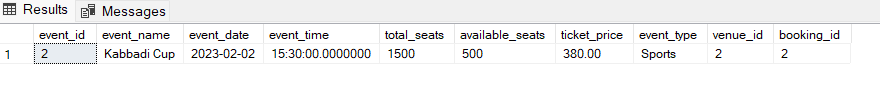
SELECT event\_id,event\_name,available\_seats FROM Event ;



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

SELECT \* FROM Event

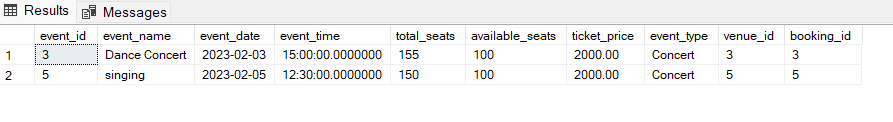
WHERE event\_name LIKE '%cup%';



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

SELECT \* FROM Event

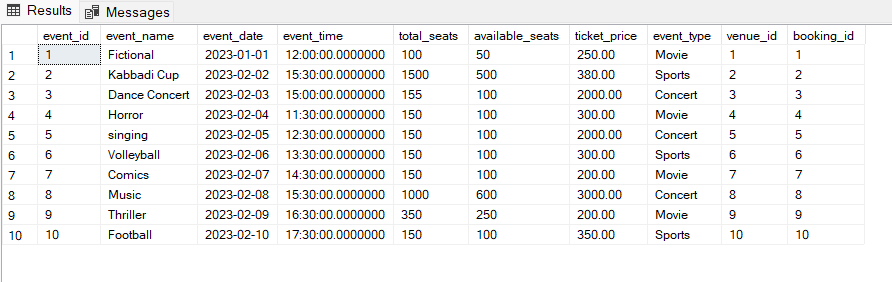
WHERE ticket\_price BETWEEN 1000 AND 2500;



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

SELECT \* FROM Event

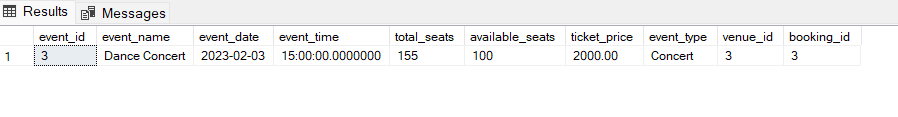
WHERE event\_date BETWEEN '2023-01-01' AND '2023-12-31';



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\_name LIKE '%Concert%';



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

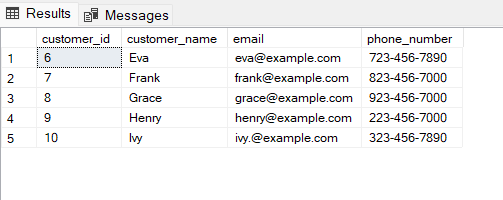
SELECT customer\_id, customer\_name, email, phone\_number

FROM Customer

ORDER BY customer\_id

OFFSET 5 ROWS

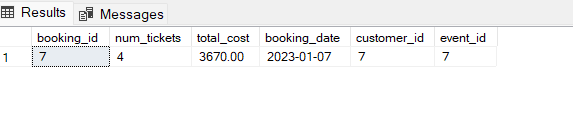
FETCH NEXT 5 ROWS ONLY;



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

SELECT \* FROM Booking

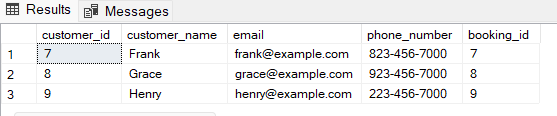
WHERE num\_tickets > =4;



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

SELECT \* FROM Customer

WHERE phone\_number LIKE '%000';

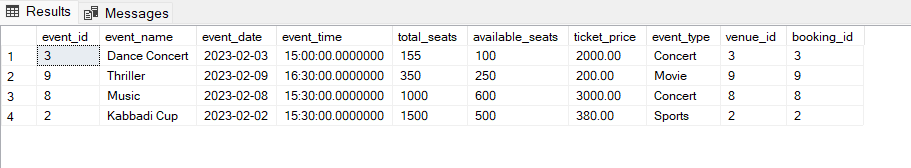


11. Write a SQL query to retrieve the events in order whose seat capacity more than 15000.

SELECT \* FROM Event

WHERE total\_seats > 150

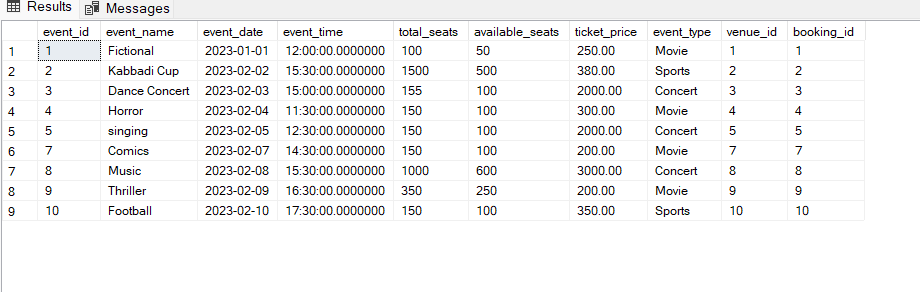
ORDER BY total\_seats;



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 'v%';



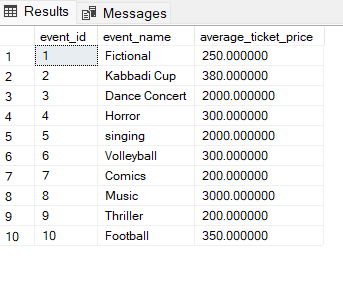
TASK 3: Aggregate functions, Group By and Joins:

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

SELECT e.event\_id, e.event\_name, AVG(e.ticket\_price) AS average\_ticket\_price

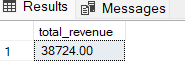
FROM Event e

GROUP BY e.event\_id, e.event\_name;



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

SELECT SUM(B.total\_cost) AS total\_revenue FROM Booking B;



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

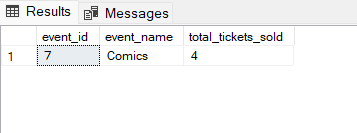
SELECT TOP 1 E.event\_id,E.event\_name,SUM(B.num\_tickets) AS total\_tickets\_sold

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;



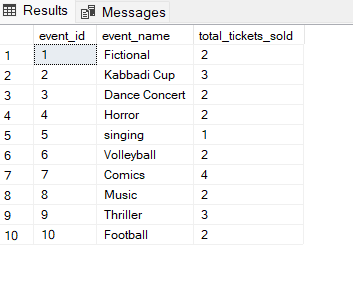
4. Write a SQL query to Calculate the Total Number of Tickets Sold for Each Event.

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;



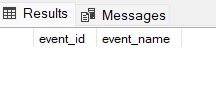
5. Write a SQL query to Find Events with No Ticket Sales.

SELECT e.event\_id, e.event\_name

FROM Event e

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

WHERE b.event\_id IS NULL;



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

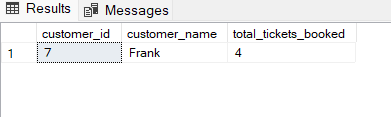
SELECT TOP 1 c.customer\_id, c.customer\_name, SUM(b.num\_tickets) AS total\_tickets\_booked

FROM Customer c

JOIN Booking b ON c.booking\_id = b.booking\_id

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

ORDER BY total\_tickets\_booked DESC;



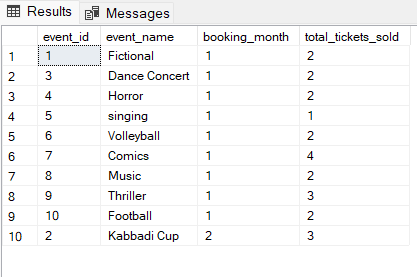
7. Write a SQL query to List Events and the total number of tickets sold for each month.

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

GROUP BY E.event\_id, E.event\_name, MONTH(booking\_date)

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

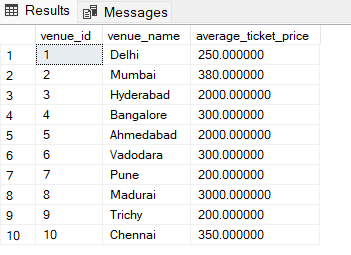


8. Write a SQL query to calculate the average Ticket Price for Events in Each Venue

SELECT V.venue\_id, V.venue\_name, AVG(E.ticket\_price) AS average\_ticket\_price FROM Venue V

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

GROUP BY V.venue\_id, V.venue\_name;

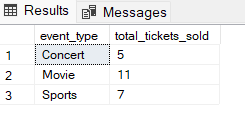


9. Write a SQL query to calculate the total Number of Tickets Sold for Each Event Type.

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

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

GROUP BY e.event\_type;



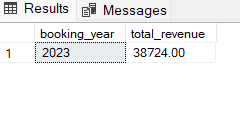
10. Write a SQL query to calculate the total Revenue Generated by Events in Each Year.

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

FROM Booking B

GROUP BY YEAR(B.booking\_date)

ORDER BY booking\_year;



11. List users who have booked tickets for multiple events:

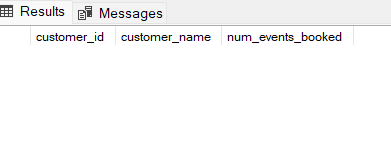
SELECT c.customer\_id, c.customer\_name, COUNT( b.event\_id) AS num\_events\_booked

FROM Customer c

JOIN Booking b ON c.booking\_id = b.booking\_id

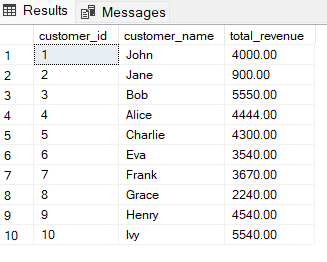
GROUP BY c.customer\_id, c.customer\_name

HAVING COUNT( b.event\_id) > 1;



12. 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 Customer c join Booking b on c.customer\_id = b.customer\_id group by c.customer\_id, c.customer\_name;



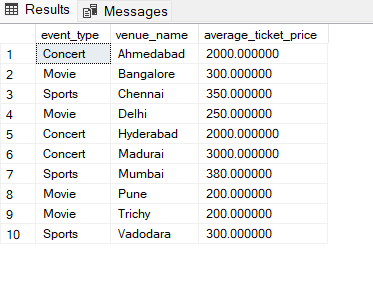
13. Calculate the Average Ticket Price for Events in Each Category and Venue:

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. List Users and the Total Number of Tickets They've Purchased in the Last 30 Days:

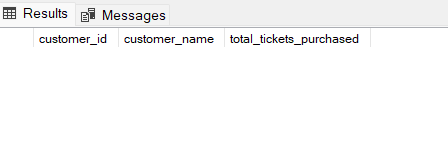
SELECT c.customer\_id, c.customer\_name, COUNT(b.booking\_id) AS total\_tickets\_purchased

FROM Customer c

JOIN Booking b ON c.booking\_id = b.booking\_id

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

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



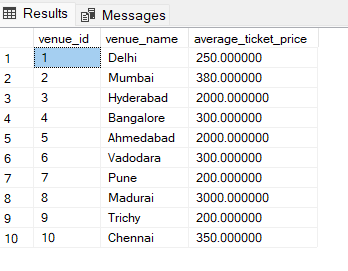
TASK 4 – Subquery and its Types

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

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

(SELECT AVG(e.ticket\_price) FROM Event e WHERE e.venue\_id = v.venue\_id) AS average\_ticket\_price

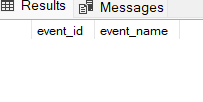
FROM Venue v;



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

SELECT e.event\_id,e.event\_name,e.total\_seats,available\_seats,ticket\_price,event\_type FROM Event e

WHERE (SELECT SUM(num\_tickets) FROM Booking b WHERE b.event\_id = e.event\_id) > 0.5 \* e.total\_seats;

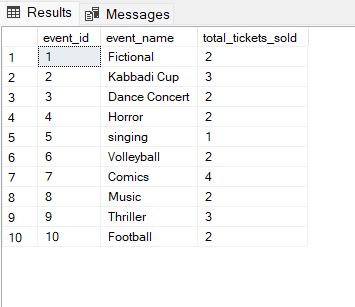


3. Calculate the Total Number of Tickets Sold for Each Event.

SELECT e.event\_id, e.event\_name,

(SELECT SUM(b.num\_tickets) FROM Booking b WHERE b.event\_id = e.event\_id) AS total\_tickets\_sold

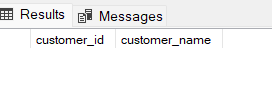
FROM Event e;



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

SELECT c.customer\_id, c.customer\_name FROM Customer c

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

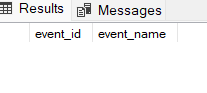


5. List Events with No Ticket Sales Using a NOT IN Subquery

SELECT e.event\_id, e.event\_name

FROM Event e

WHERE e.event\_id NOT IN (SELECT DISTINCT event\_id FROM Booking b);



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

SELECT

e.event\_type,

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

FROM

(SELECT

event\_id,

event\_type

FROM

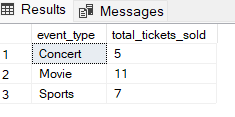
Event) e

JOIN

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

GROUP BY

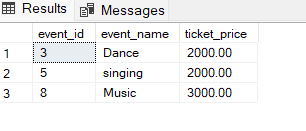
e.event\_type;



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\_name,

(

SELECT

SUM(b.total\_cost)

FROM

Booking b

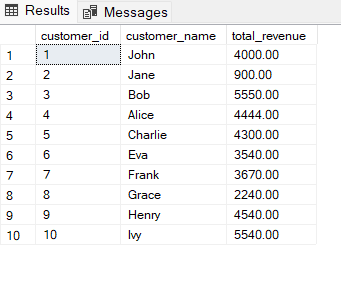
WHERE

b.customer\_id = c.customer\_id

) AS total\_revenue\_generated

FROM

Customer c;

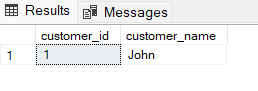


9. List Users Who Have Booked Tickets for Events in a Given Venue Using a Subquery in the WHERE Clause.

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.

SELECT event\_type, SUM(total\_tickets\_sold) AS total\_tickets\_sold

FROM (

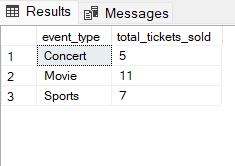
SELECT event\_id, event\_type,

(SELECT SUM(num\_tickets) FROM Booking WHERE Booking.event\_id = Event.event\_id) AS total\_tickets\_sold

FROM Event

) AS Subquery

GROUP BY event\_type;



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

SELECT

c.customer\_id,

c.customer\_name,

FORMAT(booking\_date, 'MM-yyyy') AS booking\_month

FROM

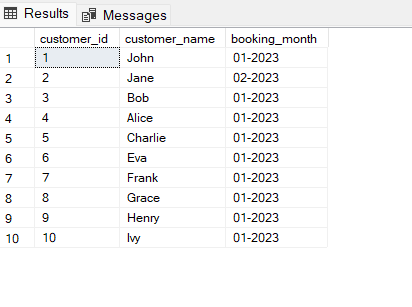
Customer c

JOIN

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

GROUP BY

c.customer\_id, c.customer\_name, FORMAT(booking\_date, 'MM-yyyy');



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

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

(SELECT AVG(e.ticket\_price) FROM Event e WHERE e.venue\_id = v.venue\_id) AS average\_ticket\_price

FROM Venue v;

