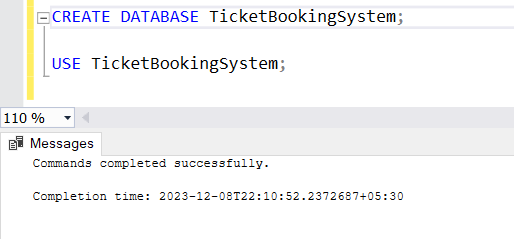
**Ticket Booking System**

# Tasks 1: Database Design:

1. Create the database named "TicketBookingSystem"

Ans) CREATE DATABASE TicketBookingSystem;

USE TicketBookingSystem;



1. Write SQL scripts to create the mentioned tables with appropriate data types, constraints, and relationships.
   * Venu
   * Event
   * Customers
   * Booking

Ans)

-- Create Venu Table

CREATE TABLE Venu (

venue\_id INT PRIMARY KEY,

venue\_name VARCHAR(255),

address VARCHAR(255)

);

-- Create 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

);

-- Create 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

);

-- Create Customer Table

CREATE TABLE Customer (

customer\_id INT PRIMARY KEY,

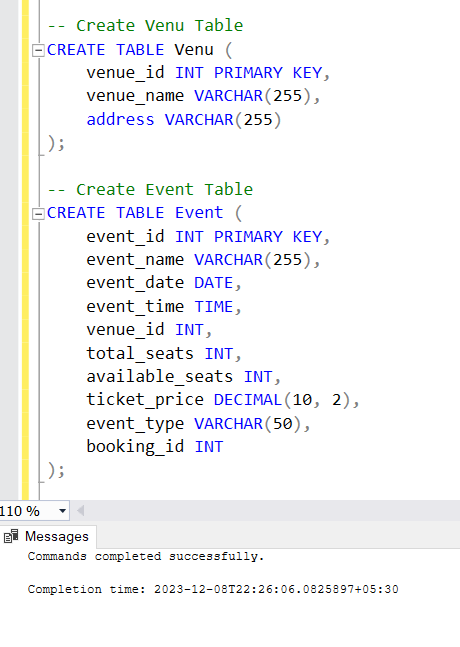
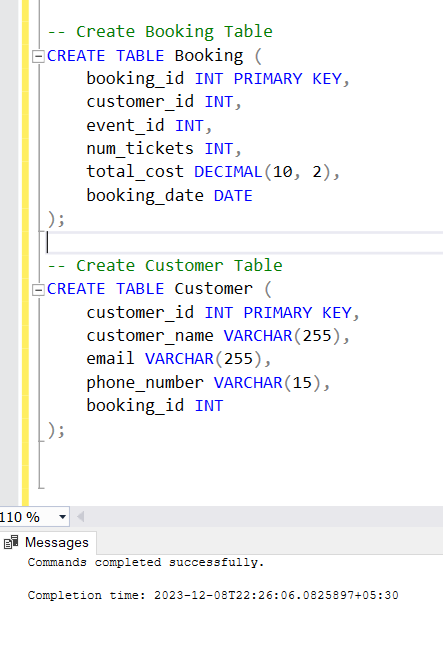
customer\_name VARCHAR(255),

email VARCHAR(255),

phone\_number VARCHAR(15),

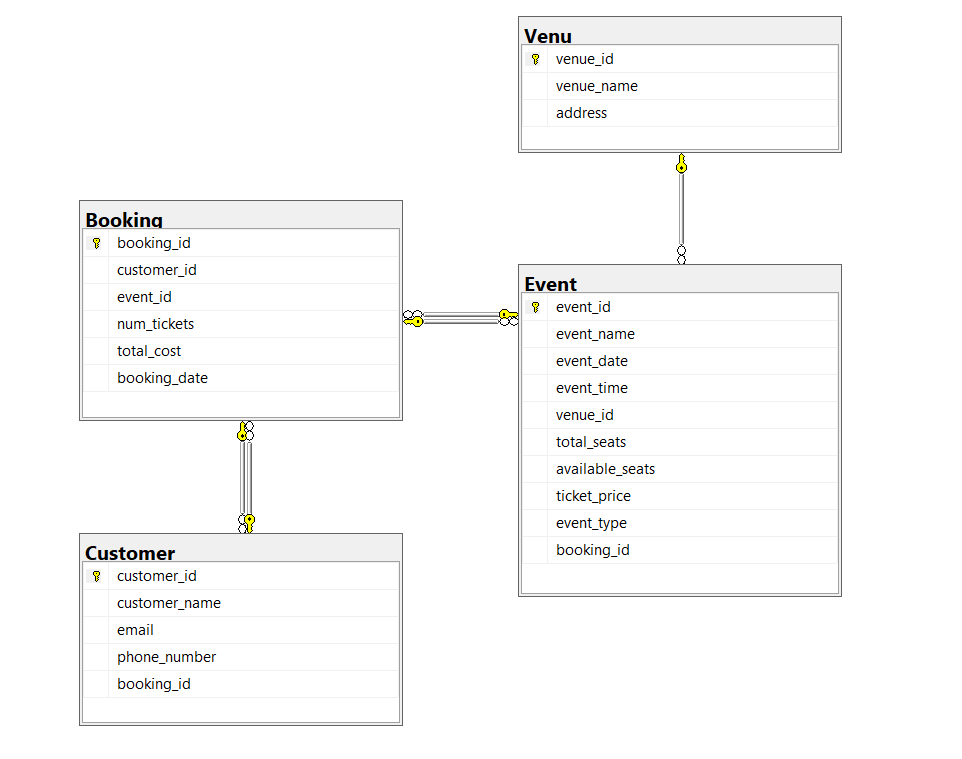
booking\_id INT

);



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

Ans)



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

Ans)

ALTER TABLE Customer

ADD FOREIGN KEY (booking\_id) REFERENCES Booking(booking\_id);

ALTER TABLE Booking

ADD FOREIGN KEY (customer\_id) REFERENCES Customer(customer\_id),

FOREIGN KEY (event\_id) REFERENCES Event(event\_id);

ALTER TABLE Event

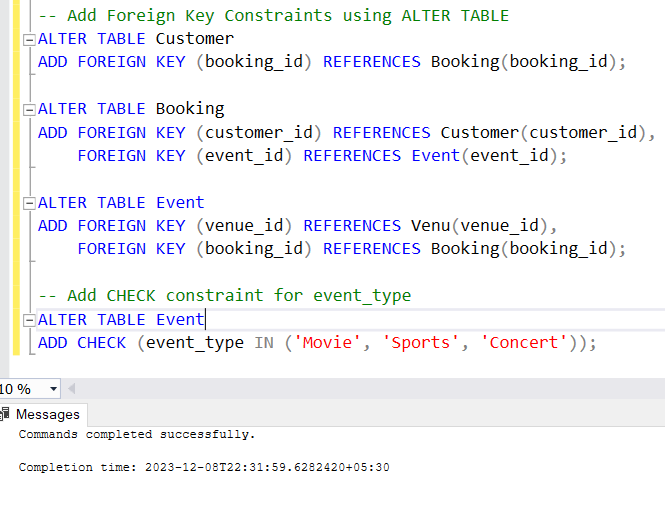
ADD FOREIGN KEY (venue\_id) REFERENCES Venu(venue\_id),

FOREIGN KEY (booking\_id) REFERENCES Booking(booking\_id);

-- Add CHECK constraint for event\_type

ALTER TABLE Event

ADD 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.

Ans)

-- Insert sample records into Venu Table

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

(1, 'Grand Theater', '123 Main Street, Cityville'),

(2, 'City Arena', '456 Center Avenue, Townsville'),

(3, 'Sports Stadium', '789 Stadium Road, Sportstown'),

(4, 'Film Palace', '101 Movie Lane, Cinemaville'),

(5, 'Concert Hall', '202 Melody Street, Harmonytown'),

(6, 'Community Center', '303 Social Square, Gatherburg'),

(7, 'Live Lounge', '404 Entertainment Avenue, Showville'),

(8, 'Cinematic Complex', '505 Film Street, Filmington'),

(9, 'Soccer Park', '606 Goal Street, Kicksville'),

(10, 'Music Dome', '707 Harmony Road, Concertburg');

-- Insert sample records into Event Table

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

(1, 'Movie Night: Inception', '2023-01-15', '18:00:00', 1, 150, 120, 2220.00, 'Movie', NULL),

(2, 'Concert: Acoustic Vibes', '2023-02-20', '20:00:00', 2, 300, 250, 1235.00, 'Concert', NULL),

(3, 'Soccer Match: City Rivals', '2023-03-25', '19:30:00', 3, 200, 180, 1525.00, 'Sports', NULL),

(4, 'Movie Night: The Great Gatsby', '2023-04-10', '21:00:00', 4, 120, 80, 1555.00, 'Movie', NULL),

(5, 'Concert: Pop Explosion', '2023-05-05', '17:45:00', 5, 250, 200, 3330.00, 'Concert', NULL),

(6, 'Live Music: Jazz Evening', '2023-06-12', '19:00:00', 6, 300, 280, 1440.00, 'Concert', NULL),

(7, 'Basketball Game: Finals', '2023-07-08', '18:30:00', 7, 350, 300, 1330.00, 'Sports', NULL),

(8, 'Movie Night: Casablanca', '2023-08-20', '20:15:00', 8, 150, 120, 2220.00, 'Movie', NULL),

(9, 'Soccer Match: International Clash', '2023-09-18', '19:45:00', 9, 200, 180, 1225.00, 'Sports', NULL),

(10, 'Concert: Rock Revolution', '2023-10-30', '22:00:00', 10, 250, 200, 3310.00, 'Concert', NULL);

-- Insert sample records into Customer Table

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

(1, 'John Doe', 'john.doe@email.com', '555-1234', NULL),

(2, 'Jane Smith', 'jane.smith@email.com', '555-5678', NULL),

(3, 'Robert Johnson', 'robert.j@email.com', '555-9012', NULL),

(4, 'Samantha Brown', 'samantha.b@email.com', '555-3456', NULL),

(5, 'Chris Miller', 'chris.m@email.com', '555-7890', NULL),

(6, 'Emma White', 'emma.w@email.com', '555-2345', NULL),

(7, 'Michael Davis', 'michael.d@email.com', '555-6789', NULL),

(8, 'Olivia Taylor', 'olivia.t@email.com', '555-1234', NULL),

(9, 'Daniel Wilson', 'daniel.w@email.com', '555-5678', NULL),

(10, 'Sophia Adams', 'sophia.a@email.com', '555-9012', NULL);

-- Insert sample records into Booking Table

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

(1, 1, 1, 2, 4440.00, '2023-01-15'),

(2, 2, 2, 3, 3705.00, '2023-02-20'),

(3, 3, 3, 1, 1525.00, '2023-03-25'),

(4, 4, 4, 4, 6220.00, '2023-04-10'),

(5, 5, 5, 2, 6660.00, '2023-05-05'),

(6, 6, 6, 3, 4320.00, '2023-06-12'),

(7, 7, 7, 5, 6650.00, '2023-07-08'),

(8, 8, 8, 1, 2220.00, '2023-08-20'),

(9, 9, 9, 2, 2450.00, '2023-09-18'),

(10, 10, 10, 3, 9930.00, '2023-10-30');

-- Update Booking Table with correct booking\_id values

UPDATE Event SET booking\_id = 1 WHERE event\_id = 1;

UPDATE Event SET booking\_id = 2 WHERE event\_id = 2;

UPDATE Event SET booking\_id = 3 WHERE event\_id = 3;

UPDATE Event SET booking\_id = 4 WHERE event\_id = 4;

UPDATE Event SET booking\_id = 5 WHERE event\_id = 5;

UPDATE Event SET booking\_id = 6 WHERE event\_id = 6;

UPDATE Event SET booking\_id = 7 WHERE event\_id = 7;

UPDATE Event SET booking\_id = 8 WHERE event\_id = 8;

UPDATE Event SET booking\_id = 9 WHERE event\_id = 9;

UPDATE Event SET booking\_id = 10 WHERE event\_id = 10;

-- Update Customer Table with correct booking\_id values

UPDATE Customer SET booking\_id = 1 WHERE customer\_id = 1;

UPDATE Customer SET booking\_id = 2 WHERE customer\_id = 2;

UPDATE Customer SET booking\_id = 3 WHERE customer\_id = 3;

UPDATE Customer SET booking\_id = 4 WHERE customer\_id = 4;

UPDATE Customer SET booking\_id = 5 WHERE customer\_id = 5;

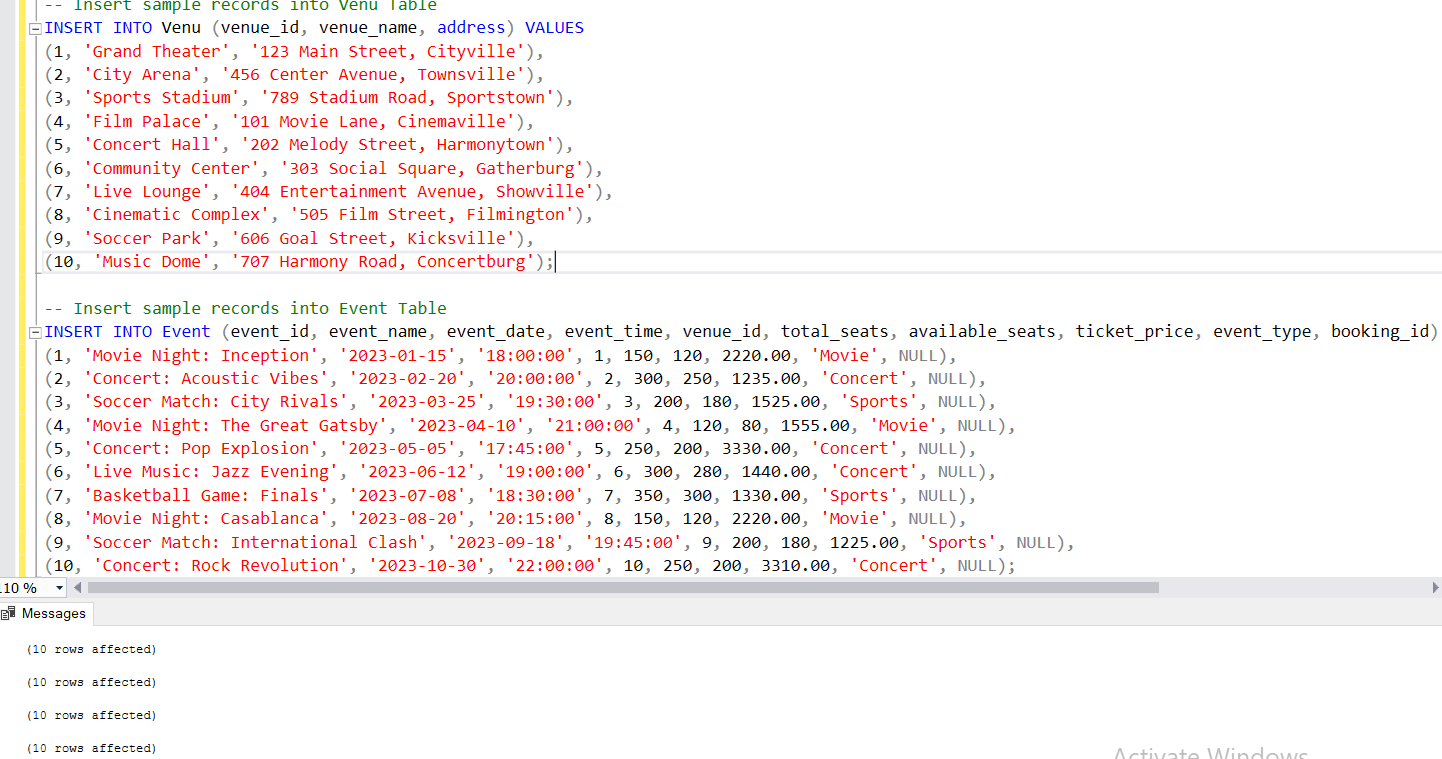
UPDATE Customer SET booking\_id = 6 WHERE customer\_id = 6;

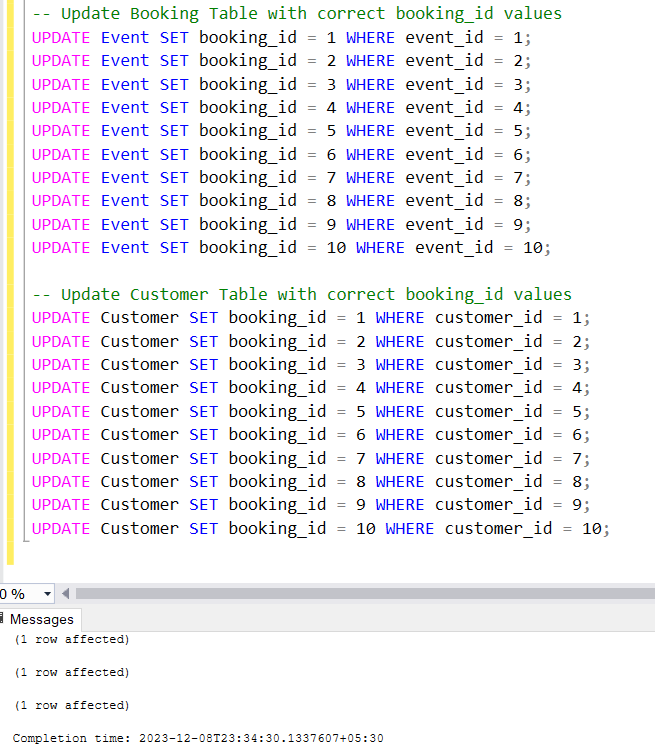
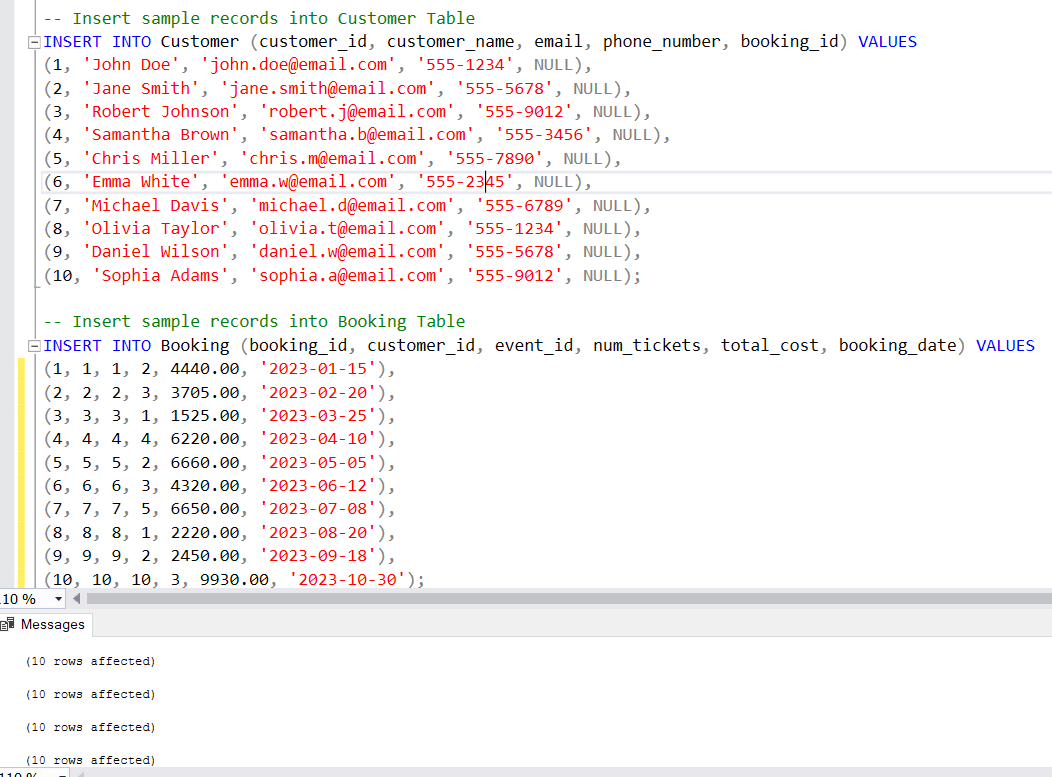
UPDATE Customer SET booking\_id = 7 WHERE customer\_id = 7;

UPDATE Customer SET booking\_id = 8 WHERE customer\_id = 8;

UPDATE Customer SET booking\_id = 9 WHERE customer\_id = 9;

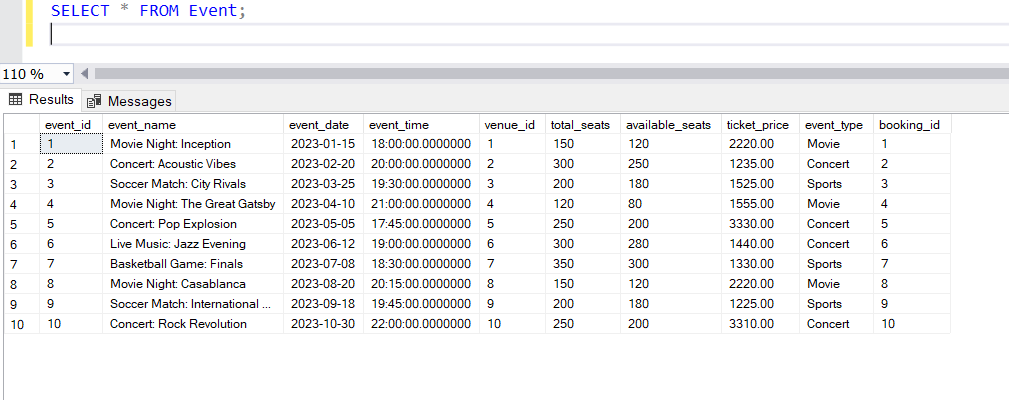
UPDATE Customer SET booking\_id = 10 WHERE customer\_id = 10;



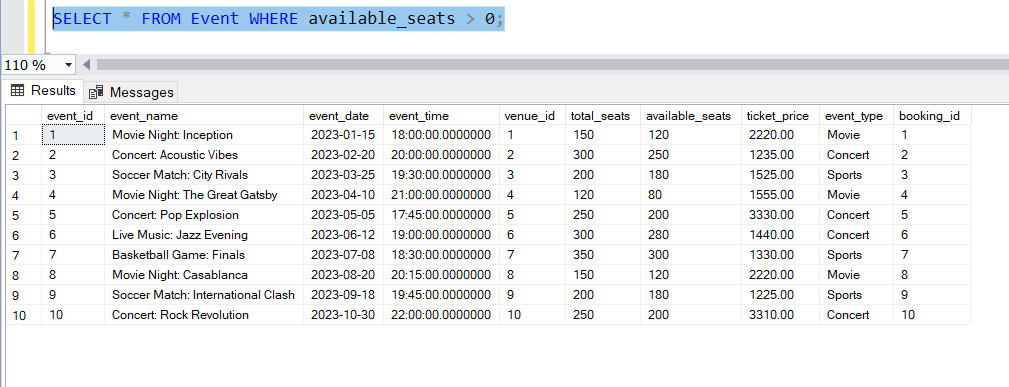
1. Write a SQL query to list all Events.

Ans) SELECT \* FROM Event;



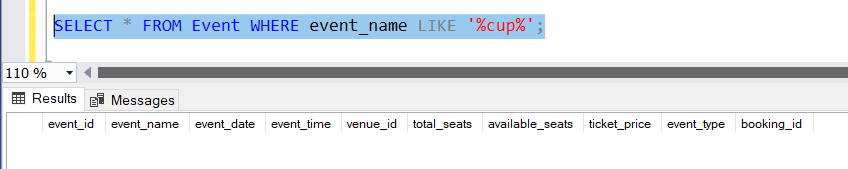
1. Write a SQL query to select events with available tickets.

Ans) SELECT \* FROM Event WHERE available\_seats > 0;



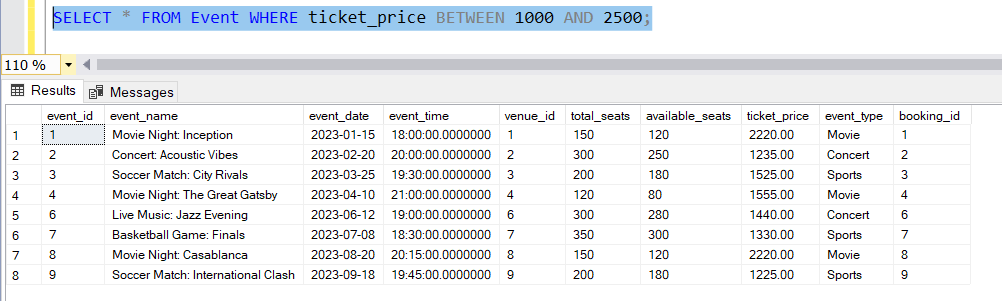
1. Write a SQL query to select events name partial match with ‘cup’.

Ans) SELECT \* FROM Event WHERE event\_name LIKE '%cup%';



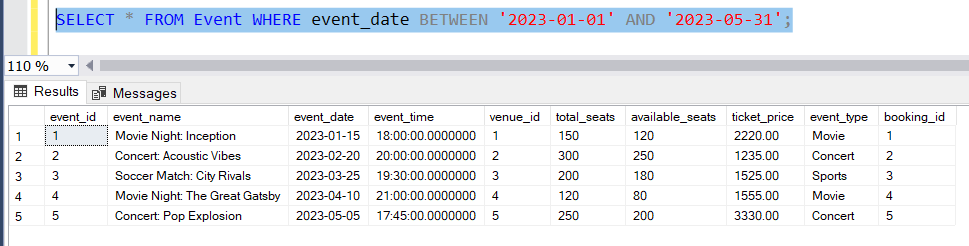
1. Write a SQL query to select events with ticket price range is between 1000 to 2500.

Ans) SELECT \* FROM Event WHERE ticket\_price BETWEEN 1000 AND 2500;



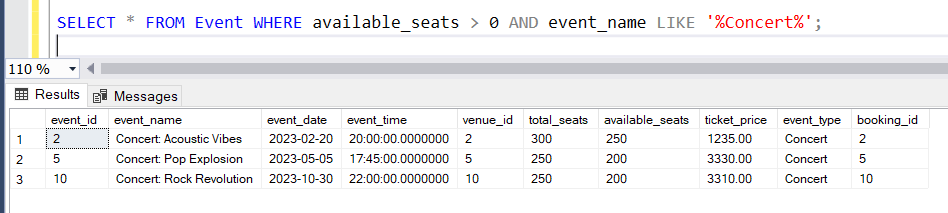
1. Write a SQL query to retrieve events with dates falling within a specific range.

Ans) SELECT \* FROM Event WHERE event\_date BETWEEN '2023-01-01' AND '2023-05-31';



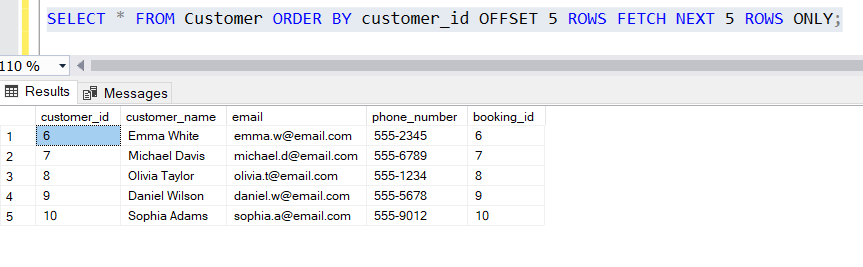
1. Write a SQL query to retrieve events with available tickets that also have "Concert" in their name.

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



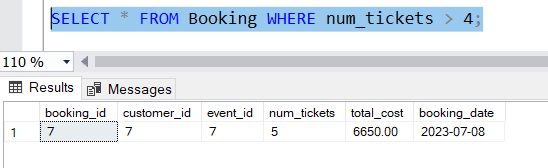
1. Write a SQL query to retrieve users in batches of 5, starting from the 6th user.

Ans) SELECT \* FROM Customer ORDER BY customer\_id OFFSET 5 ROWS FETCH NEXT 5 ROWS ONLY;

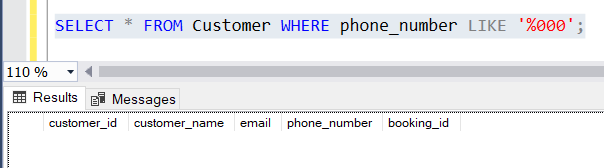


1. Write a SQL query to retrieve bookings details contains booked no of ticket more than 4.

Ans) SELECT \* FROM Booking WHERE num\_tickets > 4;

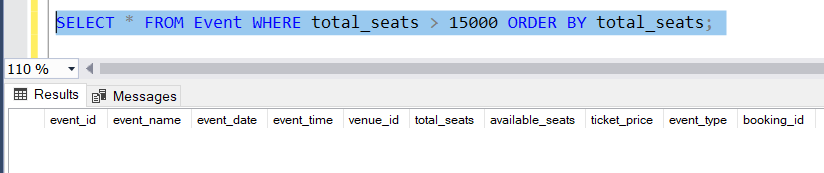


1. Write a SQL query to retrieve customer information whose phone number end with ‘000’

Ans) SELECT \* FROM Customer WHERE phone\_number LIKE '%000';

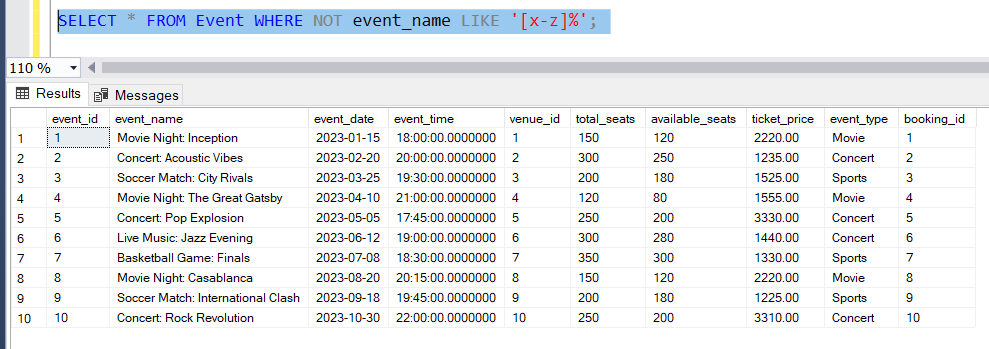
1. Write a SQL query to retrieve the events in order whose seat capacity more than 15000.

Ans) SELECT \* FROM Event WHERE total\_seats > 15000 ORDER BY total\_seats;



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

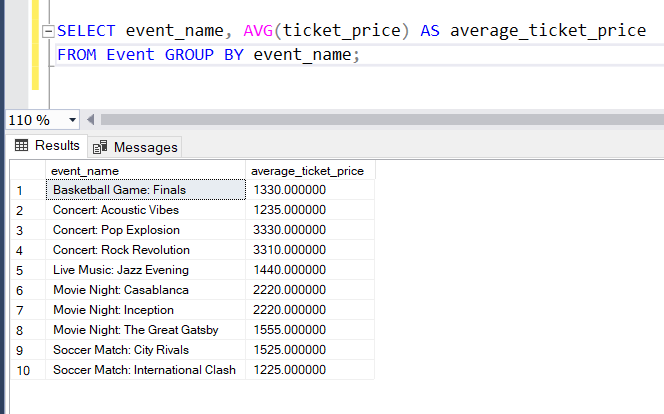
Ans) SELECT \* FROM Event WHERE NOT event\_name LIKE '[x-z]%';



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

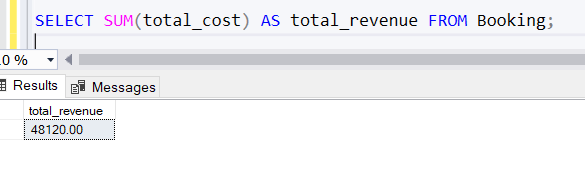
1. Write a SQL query to List Events and Their Average Ticket Prices.

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

FROM Event GROUP BY event\_name;

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

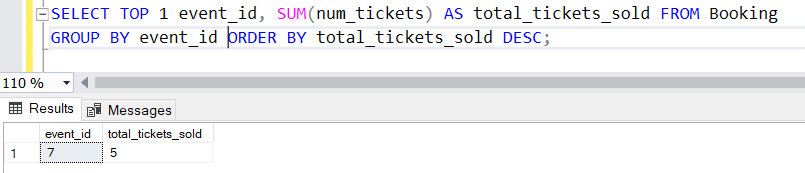
Ans) SELECT SUM(total\_cost) AS total\_revenue FROM Booking;



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

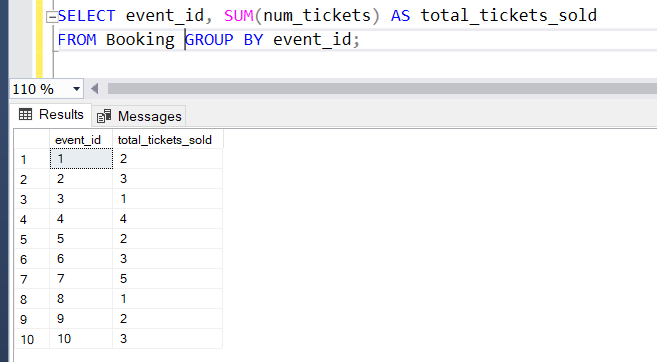
Ans) SELECT TOP 1 event\_id, SUM(num\_tickets) AS total\_tickets\_sold FROM Booking

GROUP BY event\_id ORDER BY total\_tickets\_sold DESC;



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

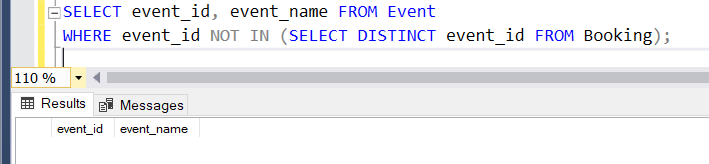
Ans) SELECT event\_id, SUM(num\_tickets) AS total\_tickets\_sold

FROM Booking GROUP BY event\_id;

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

Ans) SELECT event\_id, event\_name FROM Event

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

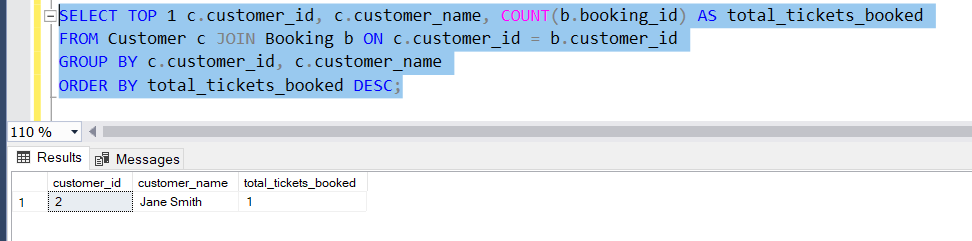


1. Write a SQL query to Find the User Who Has Booked the Most Tickets.

Ans) SELECT TOP 1 c.customer\_id, c.customer\_name, COUNT(b.booking\_id) AS total\_tickets\_booked FROM Customer c

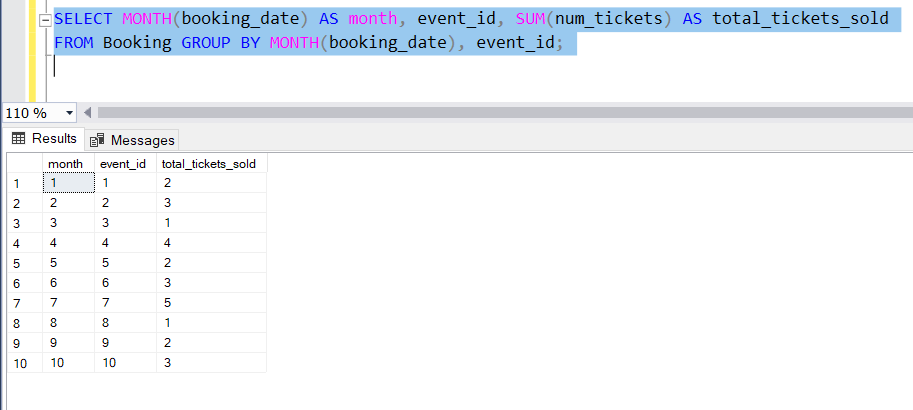
JOIN Booking b ON c.customer\_id = b.customer\_id

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

ORDER BY total\_tickets\_booked DESC;

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

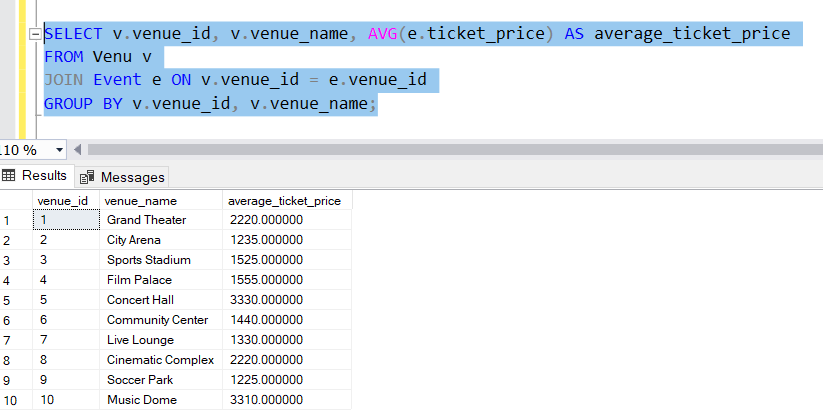
Ans) SELECT MONTH(booking\_date) AS month, event\_id, SUM(num\_tickets) AS total\_tickets\_sold FROM Booking GROUP BY MONTH(booking\_date), event\_id;



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

Ans) SELECT v.venue\_id, v.venue\_name, AVG(e.ticket\_price) AS average\_ticket\_price

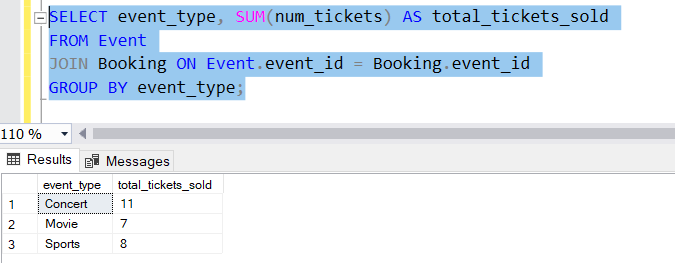
FROM Venu v JOIN Event e ON v.venue\_id = e.venue\_id

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

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

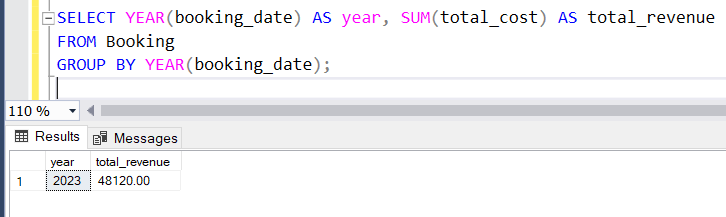
Ans) SELECT event\_type, SUM(num\_tickets) AS total\_tickets\_sold

FROM Event JOIN Booking ON Event.event\_id = Booking.event\_id

GROUP BY event\_type;

1. 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(booking\_date);

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

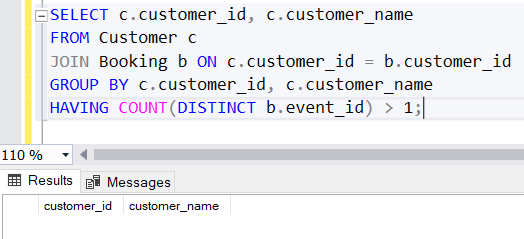
Ans) SELECT c.customer\_id, c.customer\_name

FROM Customer c

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

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

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

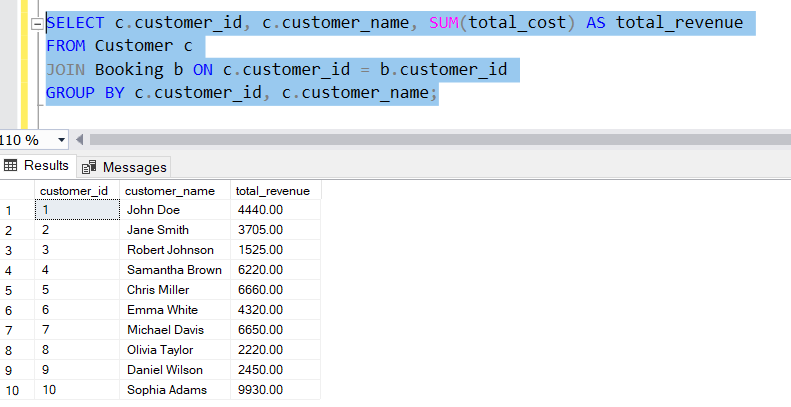


1. Write a SQL query to calculate the Total Revenue Generated by Events for Each User.

Ans) SELECT c.customer\_id, c.customer\_name, SUM(total\_cost) AS total\_revenue

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

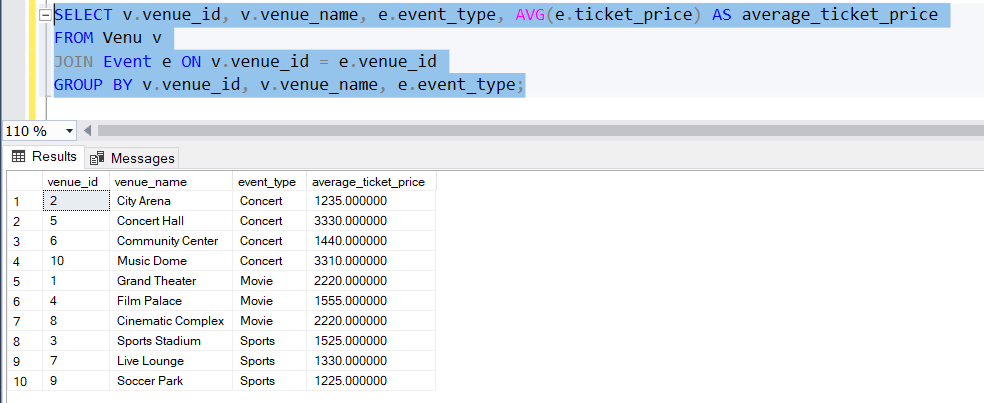
GROUP BY c.customer\_id, c.customer\_name;



1. 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;

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

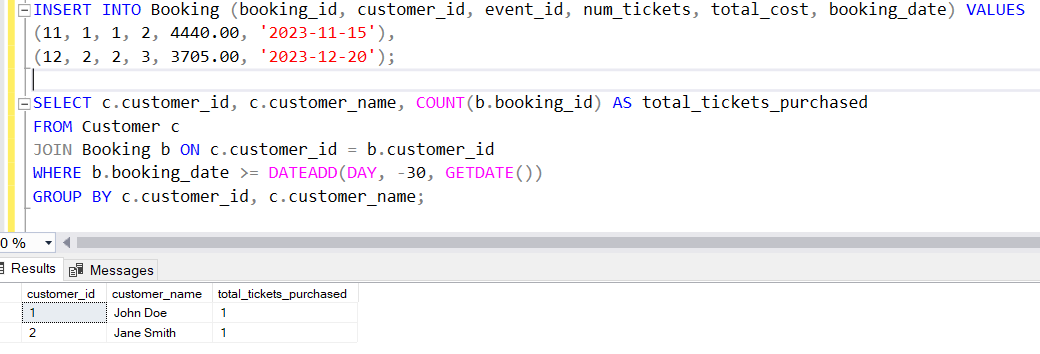
Ans) SELECT c.customer\_id, c.customer\_name, COUNT(b.booking\_id) AS total\_tickets\_purchased

FROM Customer c

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

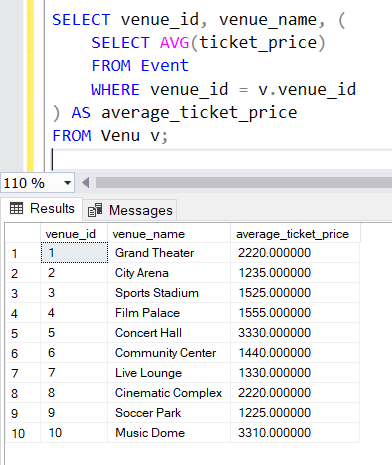
Ans) SELECT venue\_id, venue\_name, (

SELECT AVG(ticket\_price)

FROM Event

WHERE venue\_id = v.venue\_id

) AS average\_ticket\_price FROM Venu v;



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

Ans) SELECT event\_id, event\_name FROM Event

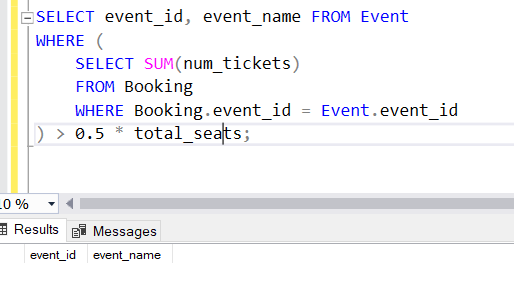
WHERE (

SELECT SUM(num\_tickets)

FROM Booking

WHERE Booking.event\_id = Event.event\_id

) > 0.5 \* total\_seats;



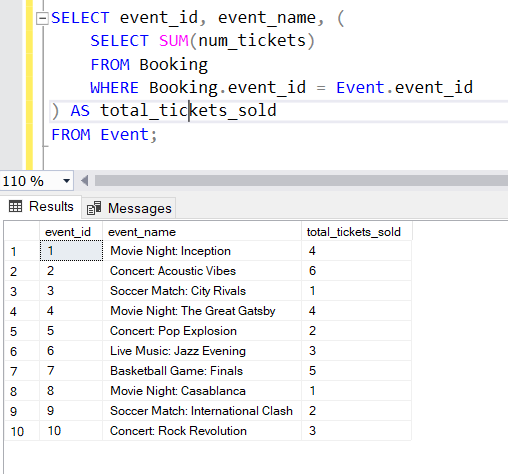
1. Calculate the Total Number of Tickets Sold for Each Event.

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;



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

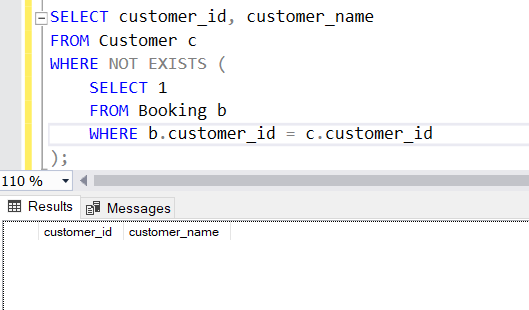
Ans) SELECT customer\_id, customer\_name FROM Customer c

WHERE NOT EXISTS (

SELECT 1

FROM Booking b

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



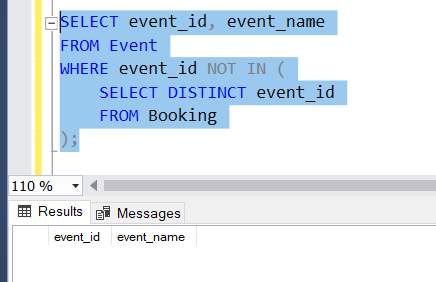
1. List Events with No Ticket Sales Using a NOT IN Subquery.

Ans) SELECT event\_id, event\_name FROM Event

WHERE event\_id NOT IN (

SELECT DISTINCT event\_id

FROM Booking );



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

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

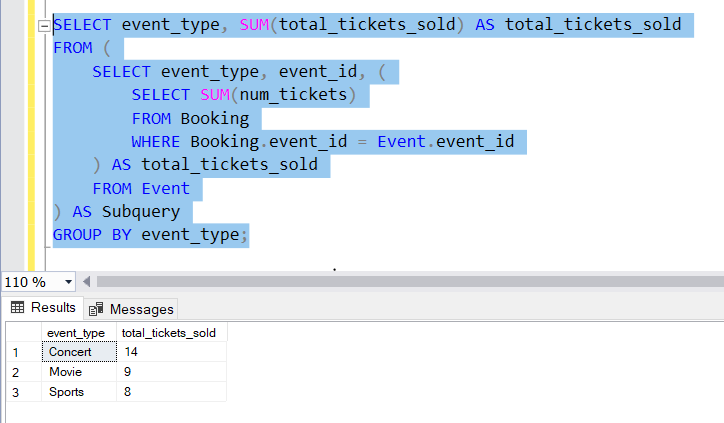
FROM ( SELECT event\_type, event\_id, (

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;

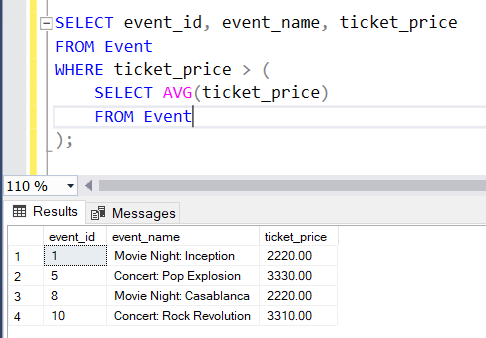


1. Find Events with Ticket Prices Higher Than the Average Ticket Price Using a Subquery in the WHERE Clause.

Ans) SELECT event\_id, event\_name, ticket\_price FROM Event

WHERE ticket\_price > (

SELECT AVG(ticket\_price) FROM Event );



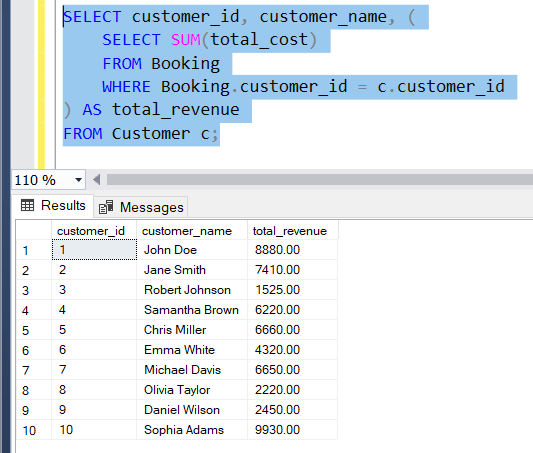
1. Calculate the Total Revenue Generated by Events for Each User Using a Correlated Subquery.

Ans) SELECT customer\_id, customer\_name, (

SELECT SUM(total\_cost) FROM Booking

WHERE Booking.customer\_id = c.customer\_id

) AS total\_revenue FROM Customer c;



1. 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

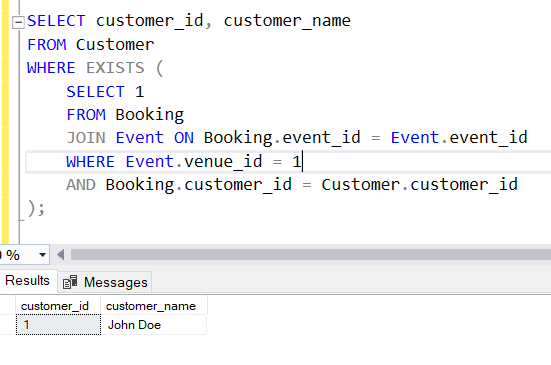
FROM Customer WHERE EXISTS (

SELECT 1 FROM Booking

JOIN Event ON Booking.event\_id = Event.event\_id

WHERE Event.venue\_id = 1

AND Booking.customer\_id = Customer.customer\_id );



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

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

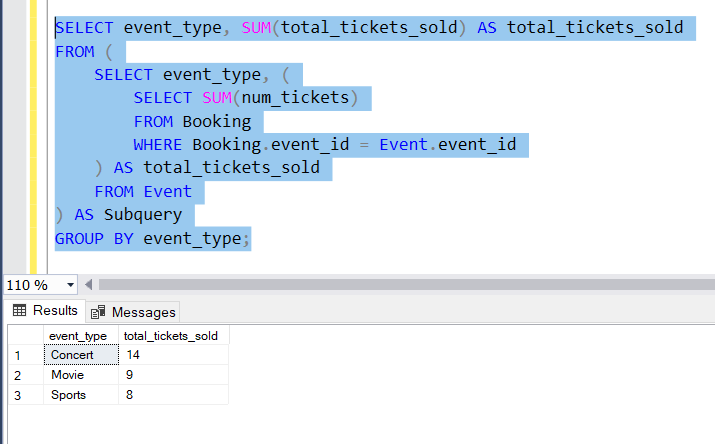
FROM ( SELECT event\_type, (

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;



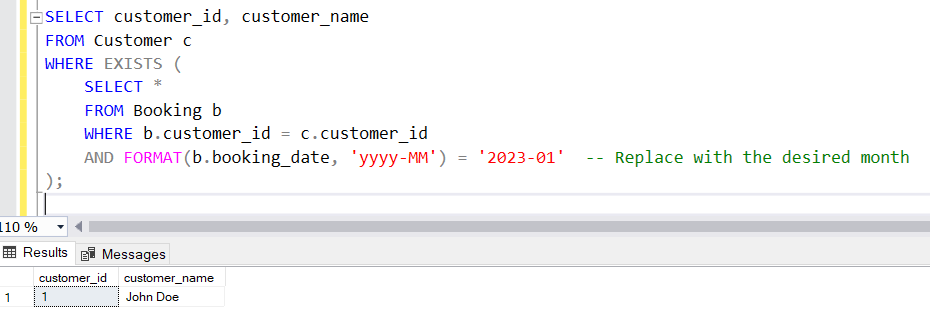
1. Find Users Who Have Booked Tickets for Events in each Month Using a Subquery with DATE\_FORMAT.

Ans) SELECT customer\_id, customer\_name FROM Customer c

WHERE EXISTS ( SELECT \* FROM Booking b

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

AND FORMAT(b.booking\_date, 'yyyy-MM') = '2023-01' -- Replace with the desired month );



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;

