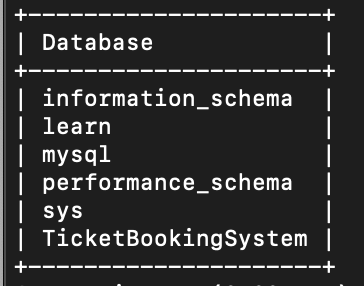
**Assignment 5**

**Tasks 1: Database Design:**

1. Create the database named "TicketBookingSystem"

CREATE DATABASE IF NOT EXISTS TicketBookingSystem;

Screenshot 2024-04-08 at 3.02.42 PM



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

relationships.

• Venue

• Event

• Customers

• Booking

**1. Venue Table**

• **venue\_id (Primary Key)**

• venue\_name,

• address

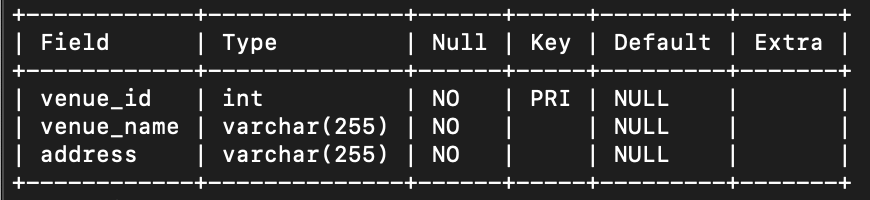
CREATE TABLE IF NOT EXISTS Venue (

venue\_id INT PRIMARY KEY,

venue\_name VARCHAR(255) NOT NULL,

address VARCHAR(255) NOT NULL

);

****

**2. Event Table**

• **event\_id (Primary Key)**

• event\_name,

• event\_date DATE,

• event\_time TIME,

• venue\_id **(Foreign Key)**,

• total\_seats,

• available\_seats,

• ticket\_price DECIMAL,

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

• booking\_id **(Foreign Key)**

CREATE TABLE IF NOT EXISTS Event (

event\_id INT PRIMARY KEY,

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,

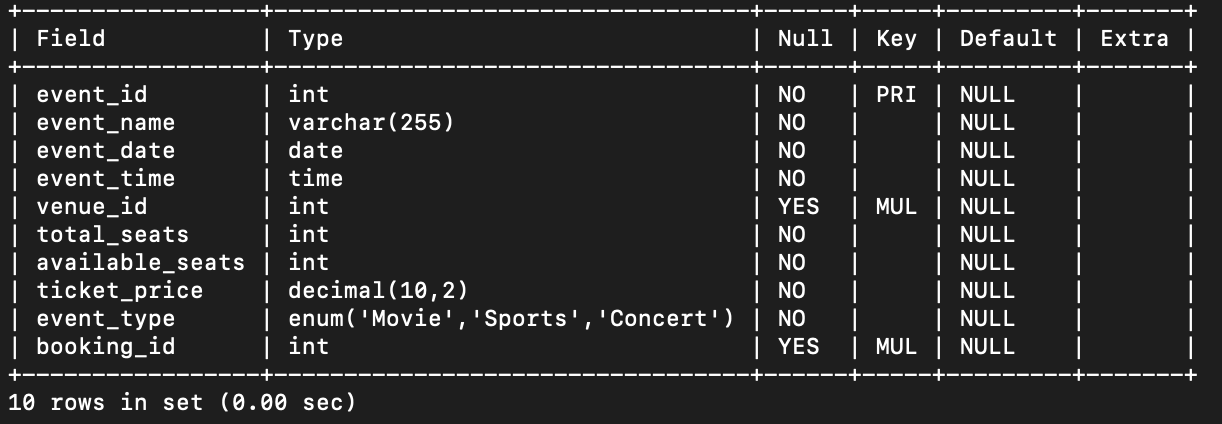
ticket\_price DECIMAL(10, 2) NOT NULL,

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

booking\_id INT

);

**Screenshot 2024-04-08 at 3.23.15 PM**

****

**3. Customer Table**

• **customer\_id (Primary key)**

• customer\_name,

• email,

• phone\_number,

• booking\_id **(Foreign Key)**

CREATE TABLE IF NOT EXISTS Customer (

customer\_id INT PRIMARY KEY,

customer\_name VARCHAR(255) NOT NULL,

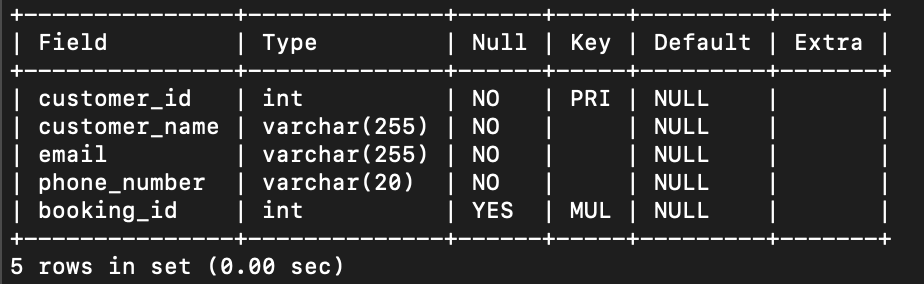
email VARCHAR(255) NOT NULL,

phone\_number VARCHAR(20) NOT NULL,

booking\_id INT

);

**Screenshot 2024-04-08 at 3.24.08 PM**

****

**4. Booking Table**

• booking\_id **(Primary Key)**,

• customer\_id **(Foreign Key)**,

• event\_id **(Foreign Key)**,

• num\_tickets,

• total\_cost,

• booking\_date,

CREATE TABLE IF NOT EXISTS Booking (

booking\_id INT PRIMARY KEY,

customer\_id INT,

event\_id INT,

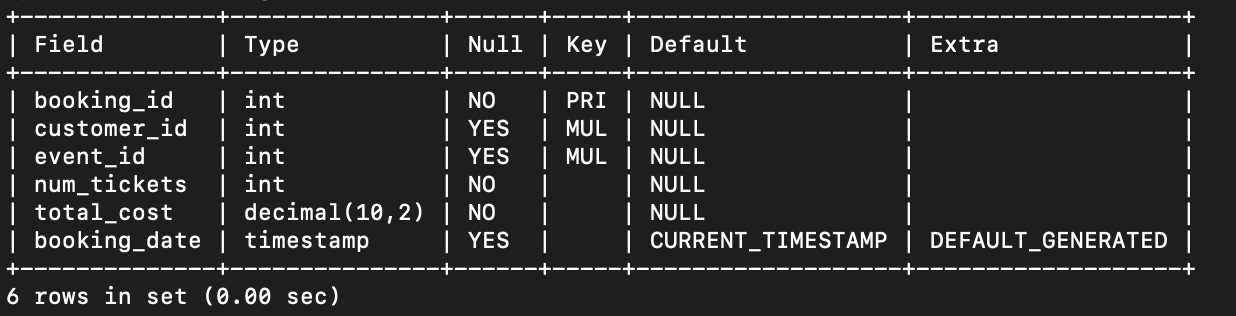
num\_tickets INT NOT NULL,

total\_cost DECIMAL(10, 2) NOT NULL,

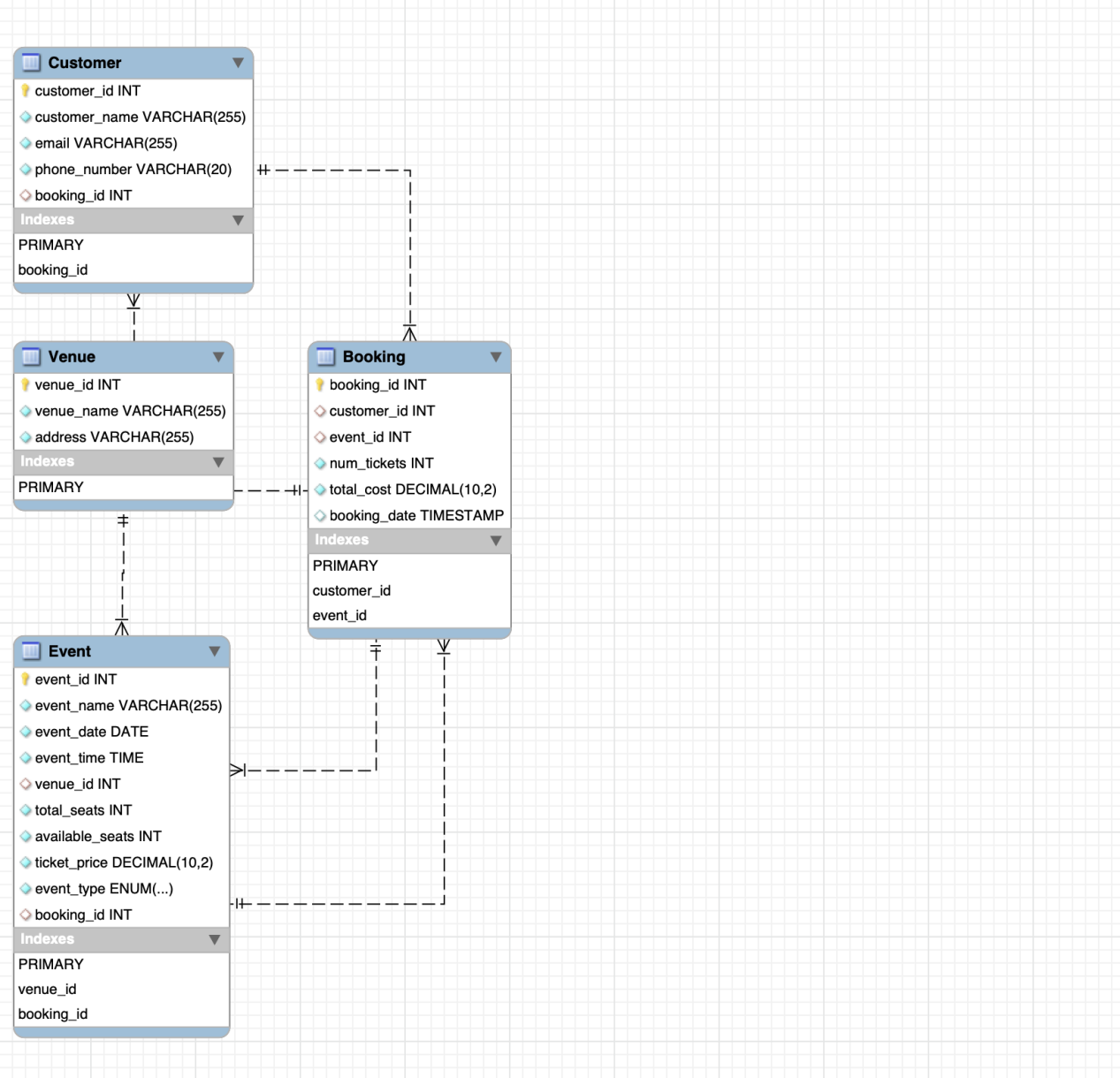
booking\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

Screenshot 2024-04-08 at 3.24.23 PM



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



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

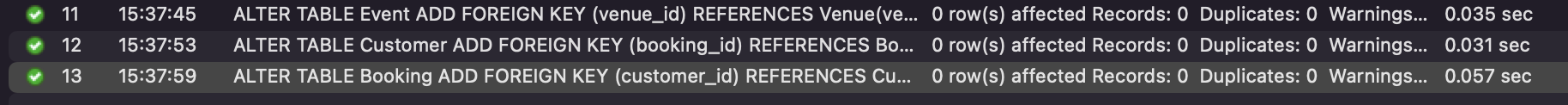
ALTER TABLE Event ADD FOREIGN KEY (venue\_id) REFERENCES Venue(venue\_id);

ALTER TABLE Event ADD FOREIGN KEY (booking\_id) REFERENCES Booking(booking\_id);

ALTER TABLE Customer ADD FOREIGN KEY (booking\_id) REFERENCES Booking(booking\_id);

ALTER TABLE Booking ADD FOREIGN KEY (customer\_id) REFERENCES Customer(customer\_id);

ALTER TABLE Booking ADD 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, 'Venue 1', 'Address 1'),

(2, 'Venue 2', 'Address 2'),

(3, 'Venue 3', 'Address 3'),

(4, 'Venue 4', 'Address 4'),

(5, 'Venue 5', 'Address 5'),

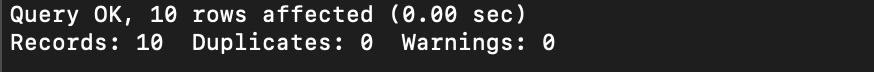
(6, 'Venue 6', 'Address 6'),

(7, 'Venue 7', 'Address 7'),

(8, 'Venue 8', 'Address 8'),

(9, 'Venue 9', 'Address 9'),

(10, 'Venue 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

(1, 'Event 1', '2024-04-10', '12:00:00', 1, 200, 200, 1500.00, 'Concert', NULL),

(2, 'Event 2', '2024-04-11', '14:00:00', 2, 150, 100, 2000.00, 'Movie', NULL),

(3, 'Event 3', '2024-04-12', '15:00:00', 3, 300, 250, 1800.00, 'Sports', NULL),

(4, 'Event 4', '2024-04-13', '18:00:00', 4, 250, 150, 2200.00, 'Concert', NULL),

(5, 'Event 5', '2024-04-14', '20:00:00', 5, 400, 350, 1200.00, 'Concert', NULL),

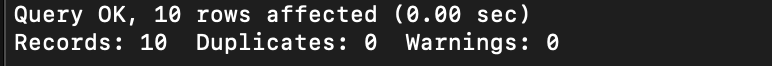
(6, 'Event 6', '2024-04-15', '19:00:00', 6, 350, 300, 1600.00, 'Sports', NULL),

(7, 'Event 7', '2024-04-16', '17:00:00', 7, 200, 100, 2500.00, 'Movie', NULL),

(8, 'Event 8', '2024-04-17', '16:00:00', 8, 300, 200, 1700.00, 'Concert', NULL),

(9, 'Event 9', '2024-04-18', '21:00:00', 9, 500, 450, 1900.00, 'Sports', NULL),

(10, 'Event 10', '2024-04-19', '13:00:00', 10, 450, 400, 2100.00, 'Movie', NULL);



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

(1, 'John Doe', 'john@example.com', '1234567890', NULL),

(2, 'Jane Smith', 'jane@example.com', '9876543210', NULL),

(3, 'Alice Johnson', 'alice@example.com', '4567890123', NULL),

(4, 'Bob Brown', 'bob@example.com', '3216549870', NULL),

(5, 'Charlie Davis', 'charlie@example.com', '7891234560', NULL),

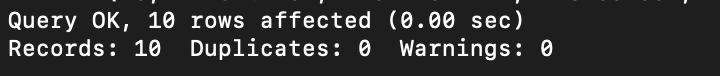
(6, 'Emma Wilson', 'emma@example.com', '6549873210', NULL),

(7, 'David Lee', 'david@example.com', '1472583690', NULL),

(8, 'Olivia Clark', 'olivia@example.com', '2583691470', NULL),

(9, 'James Miller', 'james@example.com', '3698521470', NULL),

(10, 'Sophia Martinez', 'sophia@example.com', '8527419630', NULL);



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

(1, 1, 1, 2, 3000.00, '2024-04-03 09:30:00'),

(2, 2, 2, 3, 6000.00, '2024-04-02 10:45:00'),

(3, 3, 3, 1, 1800.00, '2024-04-01 11:20:00'),

(4, 4, 4, 4, 8800.00, '2024-03-05 12:15:00'),

(5, 5, 5, 2, 2400.00, '2024-04-02 13:00:00'),

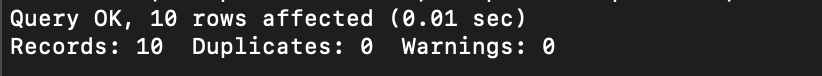
(6, 6, 6, 3, 4800.00, '2024-04-06 14:30:00'),

(7, 7, 7, 1, 2500.00, '2024-04-07 15:10:00'),

(8, 8, 8, 2, 3400.00, '2024-04-07 16:00:00'),

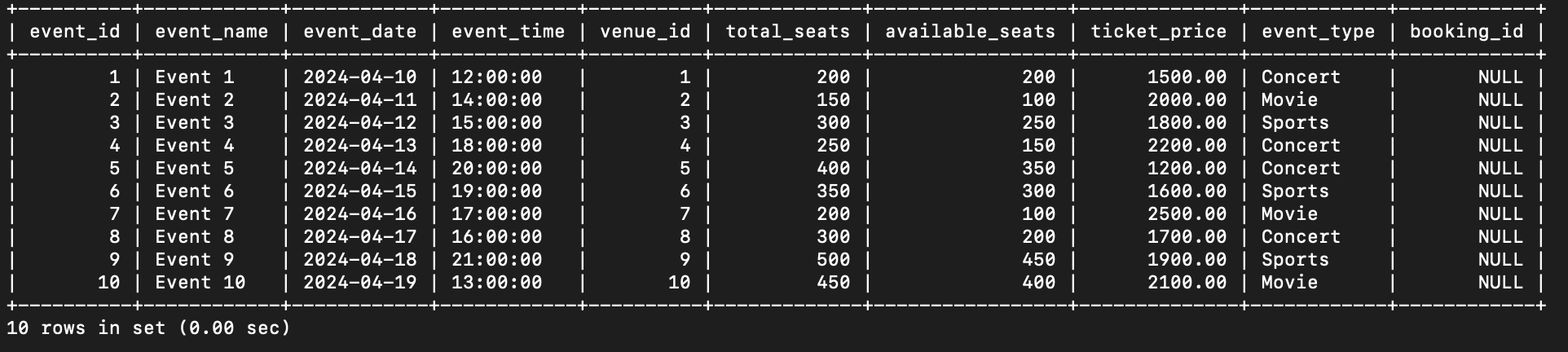
(9, 9, 9, 4, 7600.00, '2024-04-02 17:45:00'),

(10, 10, 10, 3, 6300.00, '2024-03-01 18:20:00');



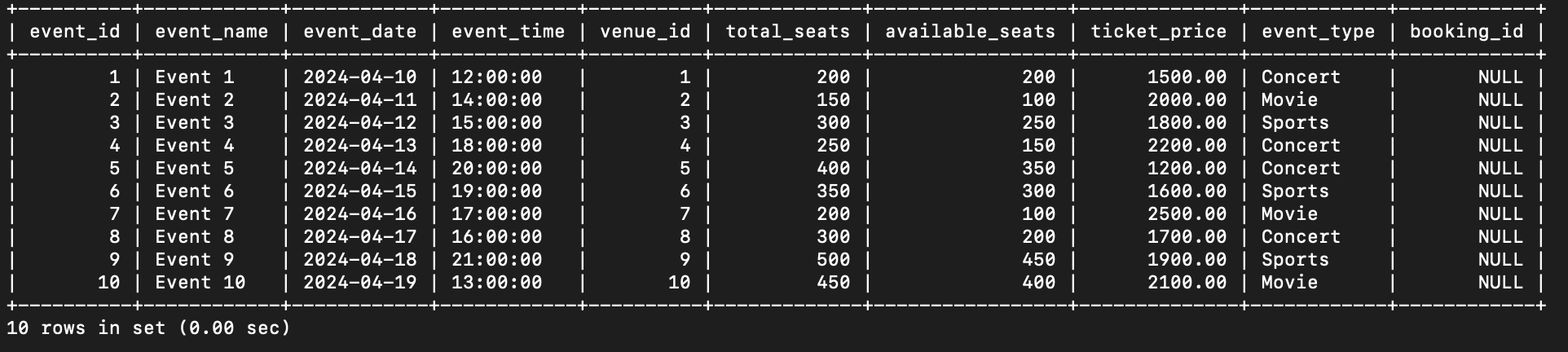
1. Write a SQL query to list all Events.

SELECT \* FROM Event;



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

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



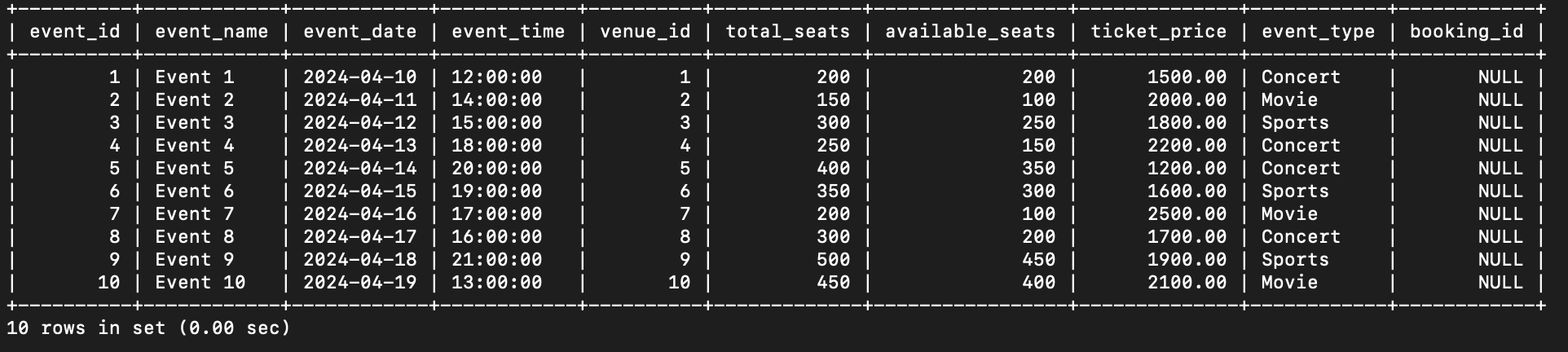
4. Write a SQL query to select events name partial match with ‘cup’.

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

Screenshot 2024-04-09 at 3.19.04 PM

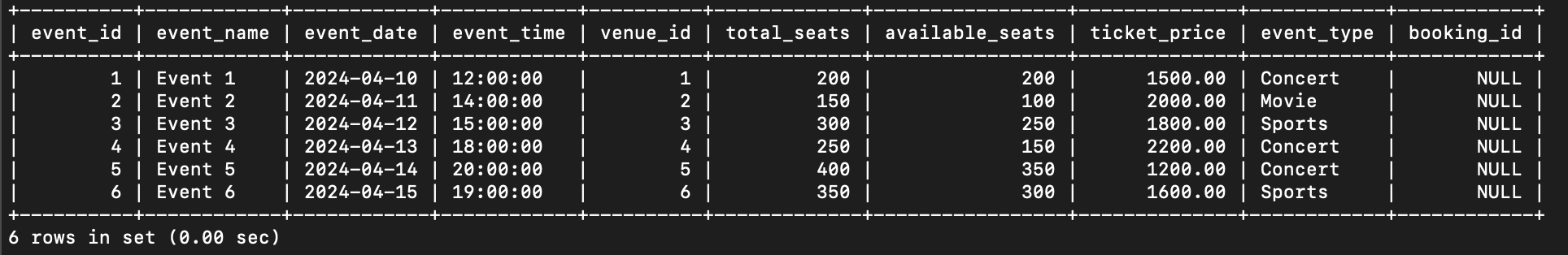
1. Write a SQL query to select events with ticket price range is between 1000 to 2500.

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



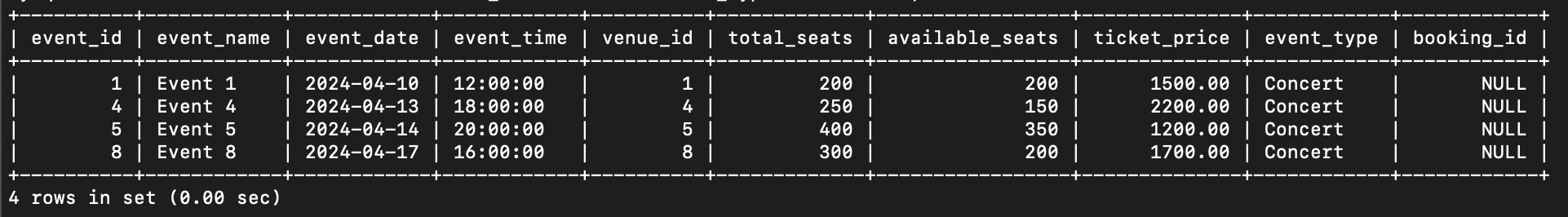
1. Write a SQL query to retrieve events with dates falling within a specific range.

SELECT \* FROM Event WHERE event\_date BETWEEN '2024-04-10' AND '2024-04-15';



1. 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';



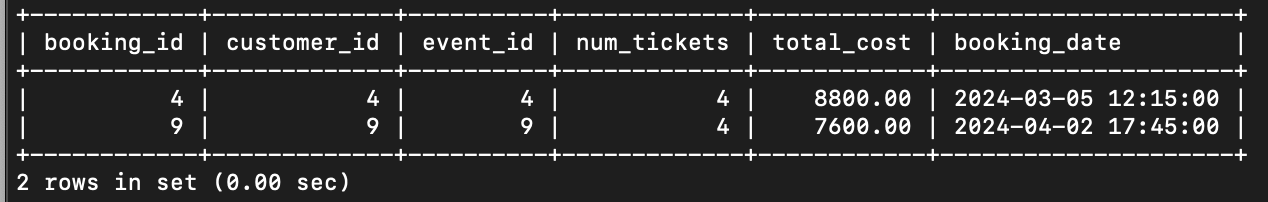
1. Write a SQL query to retrieve users in batches of 5, starting from the 6th user.

SELECT \* FROM Customer LIMIT 5 OFFSET 5;



1. Write a SQL query to retrieve bookings details contains booked no of ticket more than 3.

SELECT \* FROM Booking WHERE num\_tickets > 3;



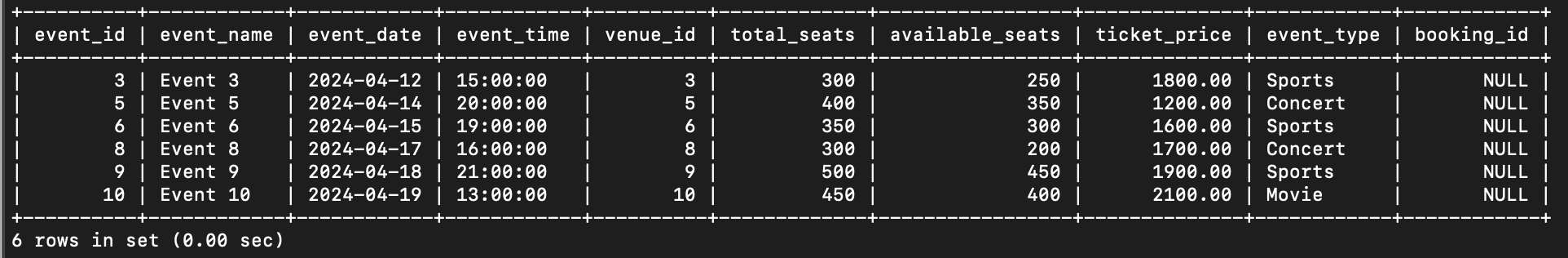
10. Write a SQL query to retrieve customer information whose phone number end with ‘000’

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

Screenshot 2024-04-09 at 3.38.48 PM

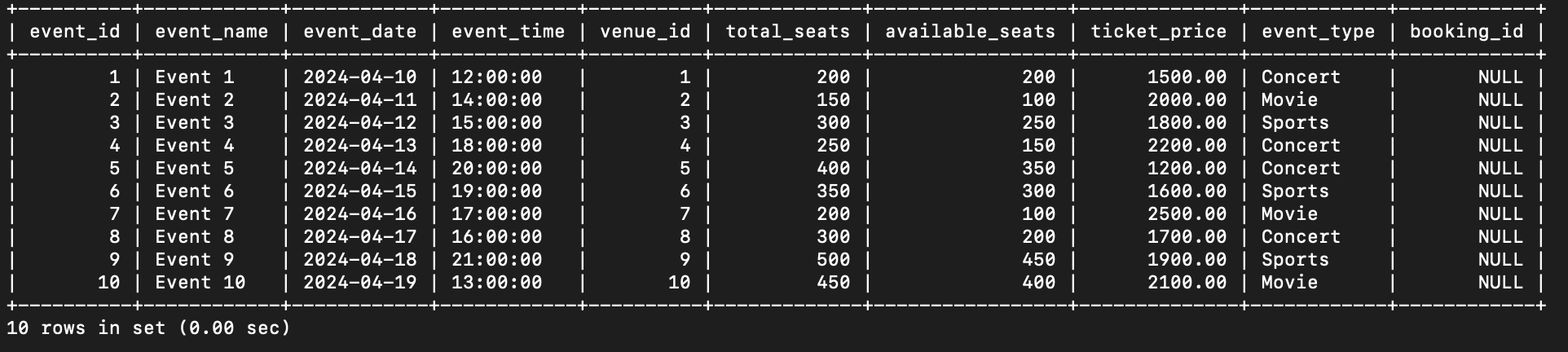
11. Write a SQL query to retrieve the events in order whose seat capacity more than 15000.

SELECT \* FROM Event WHERE total\_seats > 250;



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\_type, AVG(ticket\_price) AS avg\_ticket\_price FROM Event GROUP BY event\_type;



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.

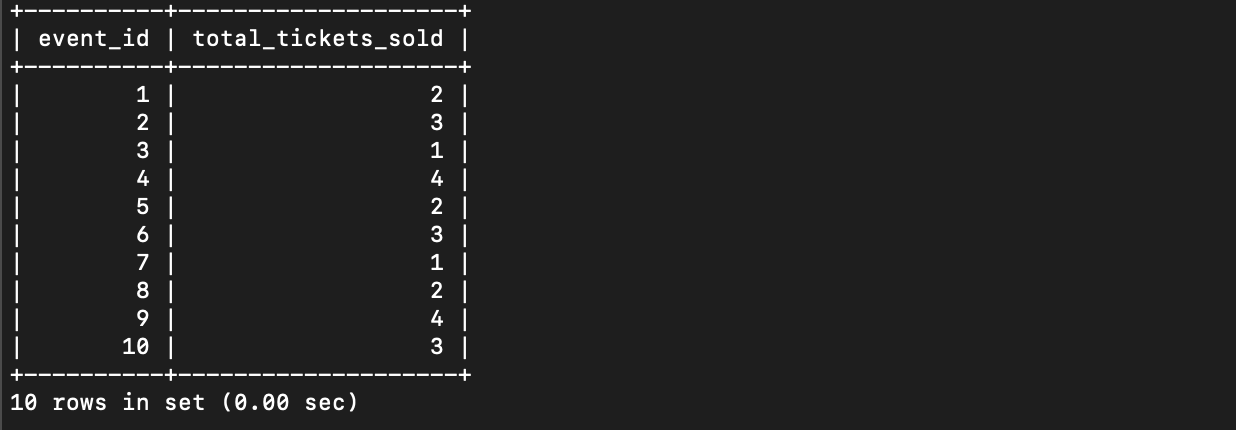
SELECT event\_id, SUM(num\_tickets) AS total\_tickets\_sold, SUM(total\_cost) AS total\_revenue FROM Booking GROUP BY event\_id ORDER BY total\_tickets\_sold DESC, total\_revenue DESC LIMIT 1;



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

Each Event.

SELECT event\_id, SUM(num\_tickets) AS total\_tickets\_sold FROM Booking GROUP BY event\_id;



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

SELECT event\_id, event\_name FROM Event WHERE event\_id NOT IN (SELECT event\_id FROM Booking);

Screenshot 2024-04-10 at 2.51.53 PM

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

SELECT customer\_id, SUM(num\_tickets) AS total\_tickets\_booked

FROM Booking

GROUP BY customer\_id

HAVING SUM(num\_tickets) = (

SELECT MAX(total\_tickets)

FROM (

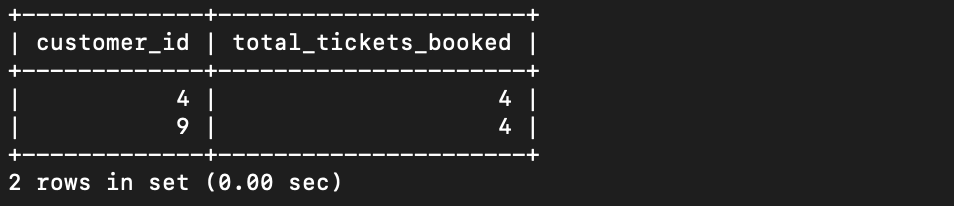
SELECT SUM(num\_tickets) AS total\_tickets

FROM Booking

GROUP BY customer\_id

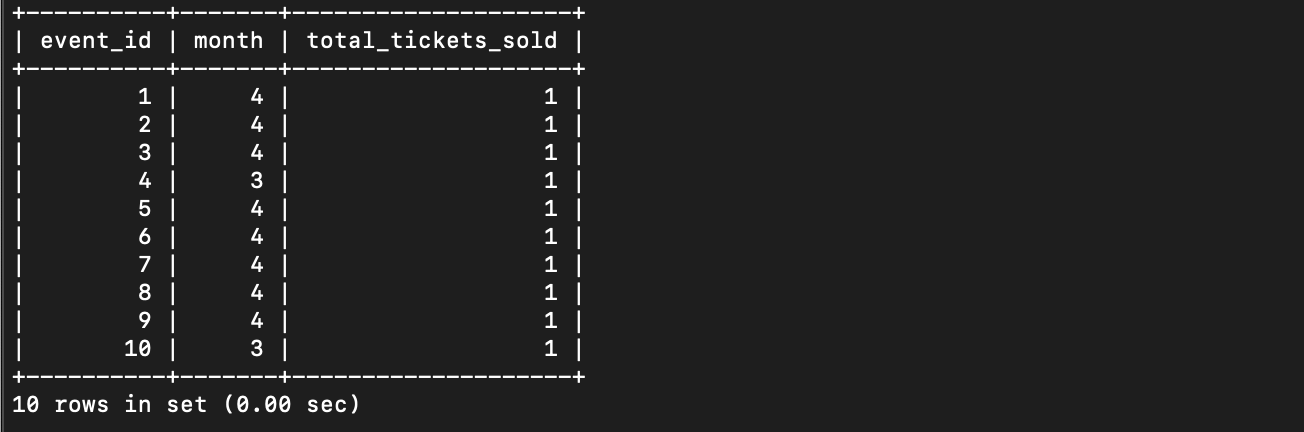
) AS subquery

);



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

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



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

Each Venue.

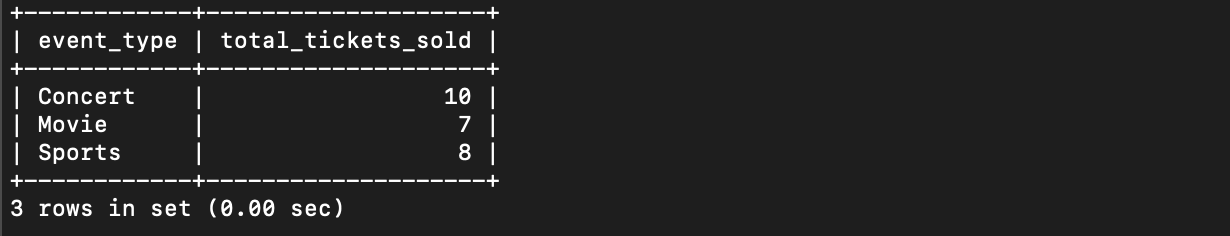
SELECT venue\_id, AVG(ticket\_price) AS avg\_ticket\_price FROM Event GROUP BY venue\_id;



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

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;



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

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



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

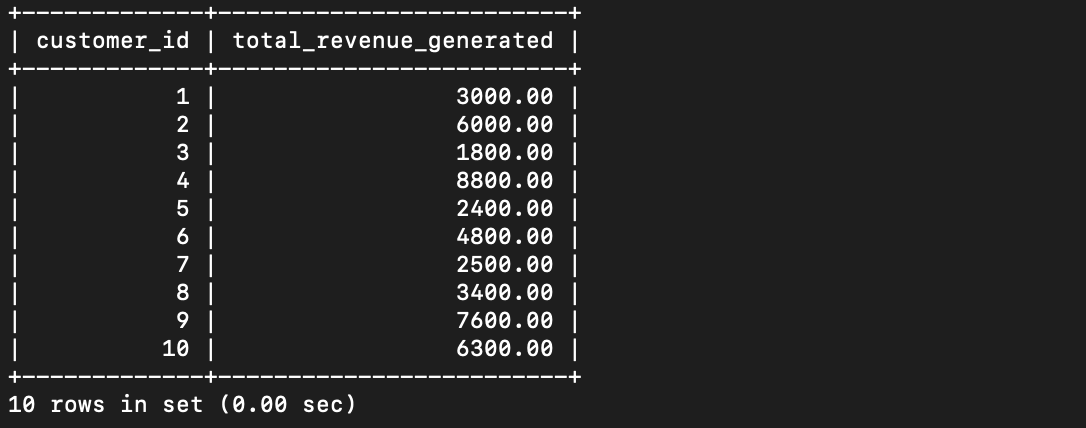
SELECT customer\_id FROM Booking GROUP BY customer\_id HAVING COUNT(DISTINCT event\_id) > 1;

Screenshot 2024-04-10 at 2.51.53 PM

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

SELECT customer\_id, SUM(total\_cost) AS total\_revenue\_generated FROM Booking

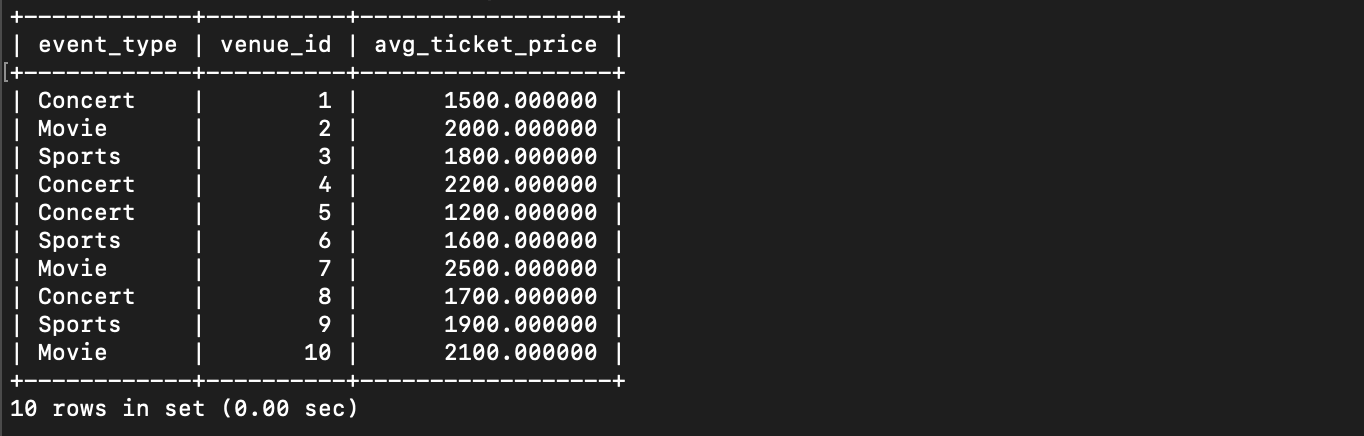
GROUP BY customer\_id;



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 avg\_ticket\_price

FROM Event GROUP BY event\_type, venue\_id;



1. Write a SQL query to list Users and the Total Number of Tickets

They've Purchased in the Last 30 Days.

SELECT customer\_id, COUNT(\*) AS total\_tickets\_purchased\_last\_30\_days FROM Booking

WHERE booking\_date >= DATE\_SUB(CURRENT\_DATE(), INTERVAL 30 DAY)

GROUP BY customer\_id;



**Tasks 4: Subquery and its types**

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

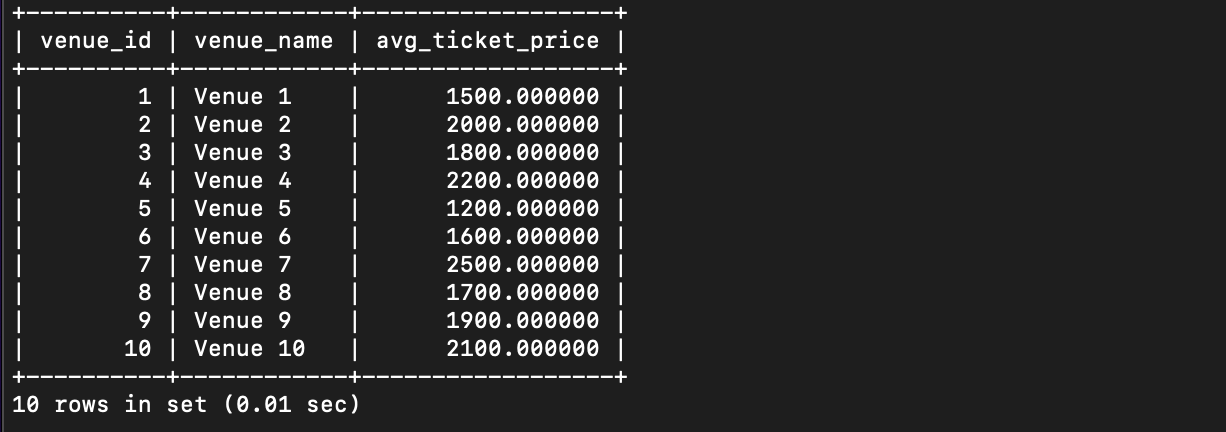
SELECT v.venue\_id, v.venue\_name,

(SELECT AVG(ticket\_price)

FROM Event

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

FROM Venue v;



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

SELECT event\_id, event\_name

FROM Event

WHERE (SELECT SUM(num\_tickets)

FROM Booking

WHERE Booking.event\_id = Event.event\_id) > (total\_seats / 2);

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

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.

SELECT customer\_id, customer\_name

FROM Customer c

WHERE NOT EXISTS (

SELECT \*

FROM Booking

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

);

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

SELECT event\_id, event\_name

FROM Event

WHERE 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(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;

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 customer\_id, customer\_name,

(SELECT SUM(total\_cost)

FROM Booking

WHERE Booking.customer\_id = Customer.customer\_id) AS total\_revenue\_generated

FROM Customer;

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 = 'given\_venue\_id'

)

);

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 each Month Using a Subquery with DATE\_FORMAT.

SELECT customer\_id, customer\_name,

DATE\_FORMAT(booking\_date, '%Y-%m') AS booking\_month

FROM Booking

JOIN Customer ON Booking.customer\_id = Customer.customer\_id

GROUP BY customer\_id, booking\_month;

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

SELECT venue\_id, venue\_name,

(SELECT AVG(ticket\_price)

FROM Event

WHERE Event.venue\_id = Venue.venue\_id) AS avg\_ticket\_price

FROM Venue;