**TICKET BOOKING SYSYTEM**

**TASKS 1: DATABASE DESIGN**

1. Create the database named "TicketBookingSystem"

Create database TicketBookingSystem;

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

• Venu

• Event

• Customers

• Booking

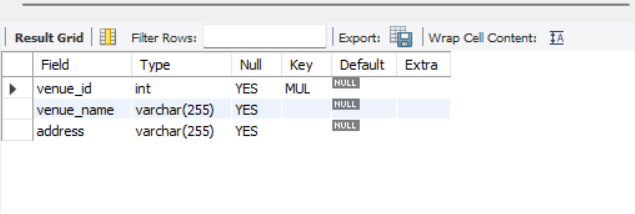
**--CREATE VENUE TABLE**

create table venue (

venue\_id int,

venue\_name varchar(255),

address varchar(255));



**--CREATE EVENT TABLE**

create table event (

event\_id int,

event\_name varchar(225),

event\_date date,

event\_time time,

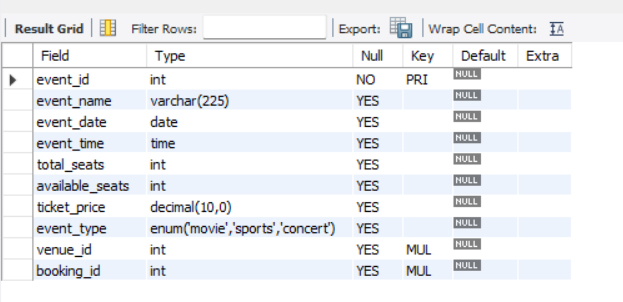
total\_seats int,

available\_seats int,

ticket\_price decimal,

event\_type enum('movie', 'sports', 'concert'),

primary key (event\_id));

****

**--CREATE CUSTOMER TABLE**

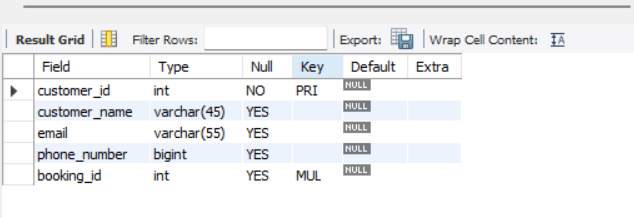
create table booking(

booking\_id int,

num\_tickets int,

total\_cost decimal,

booking\_date date);



**--CREATE BOOKING TABLE**

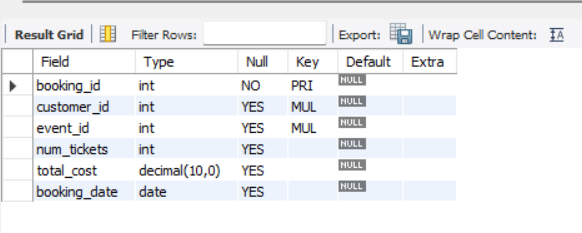
create table booking(

booking\_id int,

num\_tickets int,

total\_cost decimal,

booking\_date date);



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

-- Alter the Event Table to add Foreign Key Constraint for venue\_id

ALTER TABLE event ADD CONSTRAINT fk\_venue\_id FOREIGN KEY (venue\_id) REFERENCES venue(venue\_id);

-- Alter the Event Table to add Foreign Key Constraint for booking\_id

ALTER TABLE event ADD CONSTRAINT fk1\_booking\_id FOREIGN KEY (booking\_id) REFERENCES booking(booking\_id);

-- Alter the Customer Table to add Foreign Key Constraint for booking\_id

ALTER TABLE customer ADD CONSTRAINT fk2\_booking\_id FOREIGN KEY (booking\_id) REFERENCES booking(booking\_id);

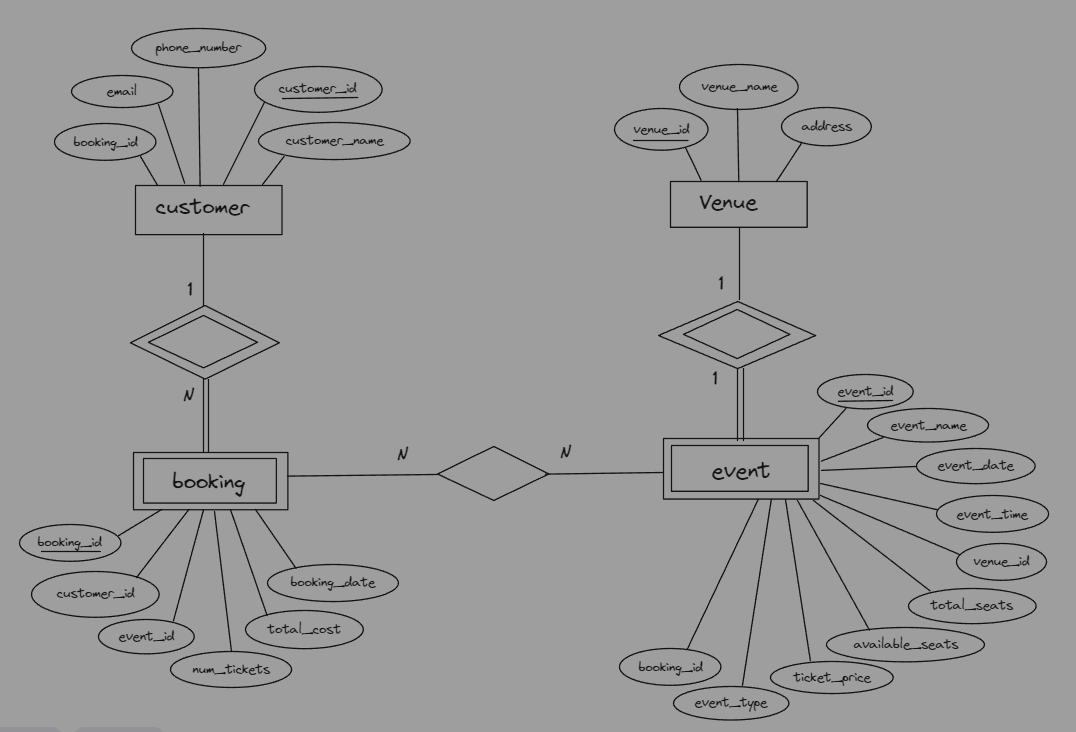
-- Alter the Booking Table to add Foreign Key Constraint for customer\_id

ALTER TABLE booking ADD CONSTRAINT fk\_customer\_id FOREIGN KEY (customer\_id) REFERENCES customer(customer\_id);

-- Alter the Booking Table to add Foreign Key Constraint for event\_id

ALTER TABLE booking ADD CONSTRAINT fk\_event\_id FOREIGN KEY (event\_id) REFERENCES event(event\_id);

**Entity Relationship Diagram**



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

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

*--Inserting Values To Venue Table*

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

(101,'alpha city', '123 Alpha Avenue'),

(102,'beta street', '456 Beta Street'),

(103,'came street', '789 Came Street'),

(104,'delta park', '101 Delta Park Lane'),

(105,'epsilon avenue', '202 Epsilon Avenue'),

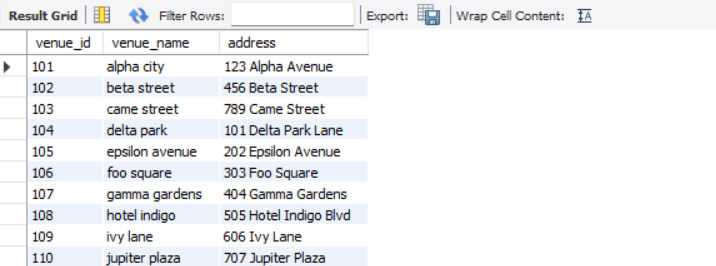
(106,'foo square', '303 Foo Square'),

(107,'gamma gardens', '404 Gamma Gardens'),

(108,'hotel indigo', '505 Hotel Indigo Blvd'),

(109,'ivy lane', '606 Ivy Lane'),

(110,'jupiter plaza', '707 Jupiter Plaza');



*-- Inserting Values In Event Table*

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, 'Movie Night', '2024-04-15', '19:00:00', 100, 90, 800.00, 'movie', 101, 111),

(2, 'Football Match', '2024-04-16', '15:30:00', 200, 180, 900.00, 'sports', 102, 222),

(3, 'Concert Under the Stars', '2024-04-17', '20:00:00', 150, 140, 1000.00, 'concert', 103, 333),

(4, 'Comedy Show', '2024-04-18', '18:00:00', 120, 110, 1200.00, 'concert', 104, 444),

(5, 'Basketball Game', '2024-04-19', '17:45:00', 180, 160, 1400.00, 'sports', 105, 555),

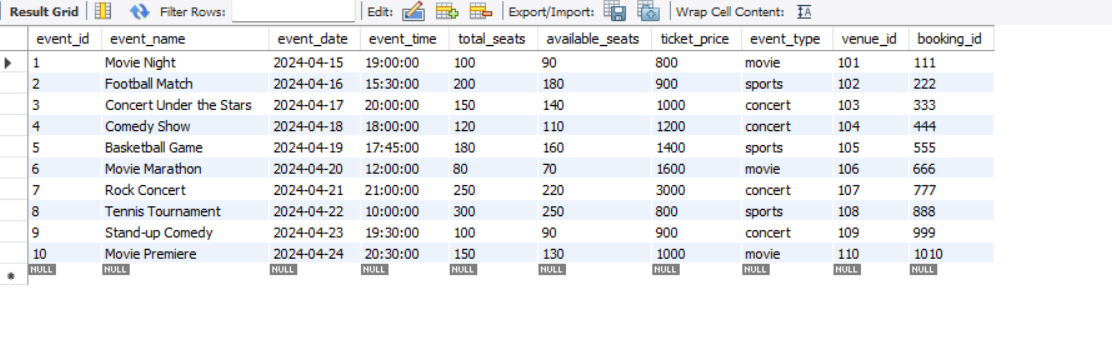
(6, 'Movie Marathon', '2024-04-20', '12:00:00', 80, 70, 1600.00, 'movie', 106, 666),

(7, 'Rock Concert', '2024-04-21', '21:00:00', 250, 220, 3000.00, 'concert', 107, 777),

(8, 'Tennis Tournament', '2024-04-22', '10:00:00', 300, 250, 800.00, 'sports', 108, 888),

(9, 'Stand-up Comedy', '2024-04-23', '19:30:00', 100, 90, 900.00, 'concert', 109, 999),

(10, 'Movie Premiere', '2024-04-24', '20:30:00', 150, 130, 1000.00, 'movie', 110, 1010);

**

*-- Inserting Rows To Customer Table*

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

(1, 'John Doe', 'john@example.com', 11111111, 111),

(2, 'Alice Smith', 'alice@example.com', 287654302, 222),

(3, 'Bob Johnson', 'bob@example.com', 32222114553, 333),

(4, 'Emily Davis', 'emily@example.com', 4151512314, 444),

(5, 'Michael Wilson', 'michael@example.com',5222211115, 555),

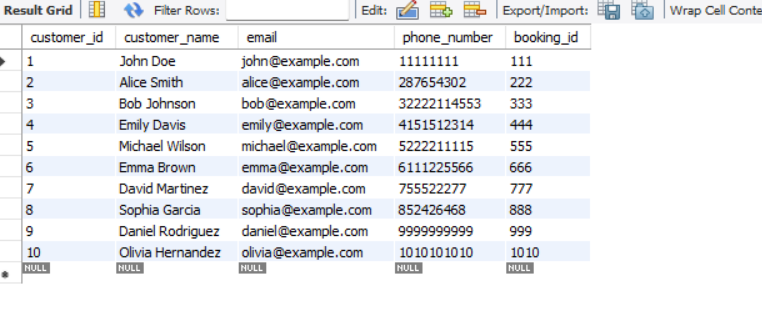
(6, 'Emma Brown', 'emma@example.com', 6111225566, 666),

(7, 'David Martinez', 'david@example.com', '755522277', 777),

(8, 'Sophia Garcia', 'sophia@example.com', 852426468, 888),

(9, 'Daniel Rodriguez', 'daniel@example.com', '9999999999', 999),

(10, 'Olivia Hernandez', 'olivia@example.com', '1010101010', 1010);



*--Inserting Rows in Booking Table*

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

VALUES

(111, 5, 1, 2, 2000.00, '2024-04-11'),

(222, 9, 2, 4, 2300.00, '2024-04-12'),

(333, 2, 3, 1, 1000.00, '2024-04-13'),

(444, 10, 4, 3, 2900.00, '2024-04-14'),

(555, 1, 5, 2, 3020.00, '2024-04-15'),

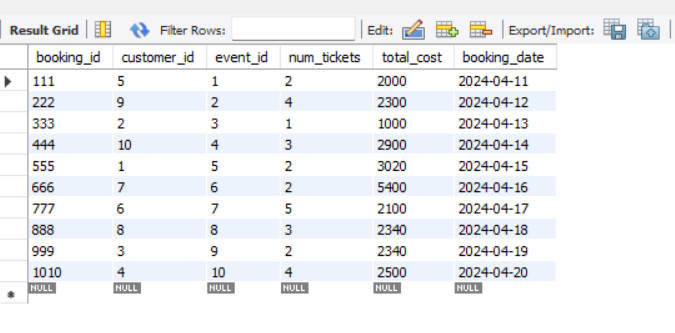
(666, 7, 6, 2, 5400.00, '2024-04-16'),

(777, 6, 7, 5, 2100.00, '2024-04-17'),

(888, 8, 8, 3, 2340.00, '2024-04-18'),

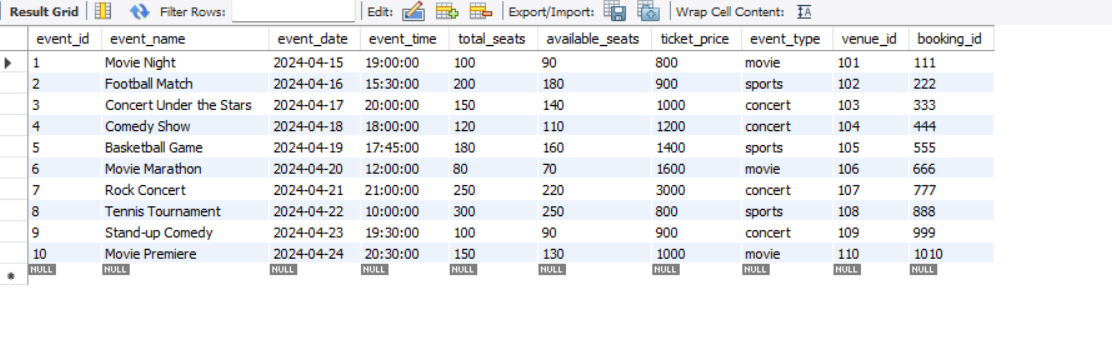
(999, 3, 9, 2, 2340.00, '2024-04-19'),

(1010*, 4, 10, 4, 2500.00, '2024-04-20');*

**

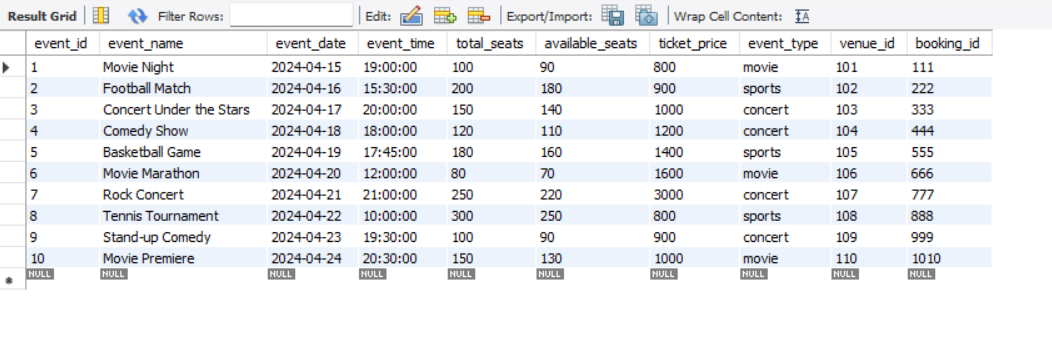
2. Write a SQL query to list all Events.

Select \* FROM event;

**

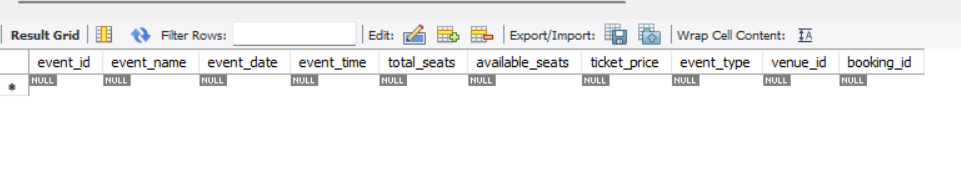
3. Write a SQL query to select events with available tickets.

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



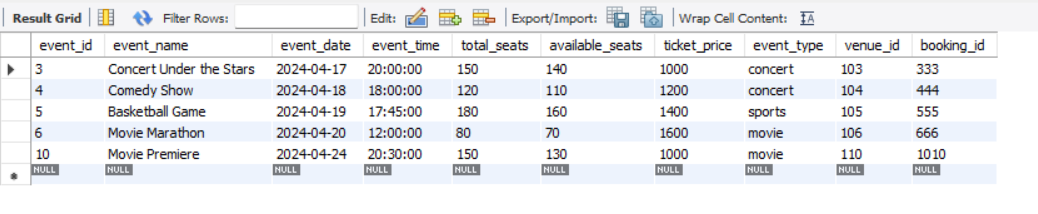
4. Write a SQL query to select events name partial match with ‘cup’.

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



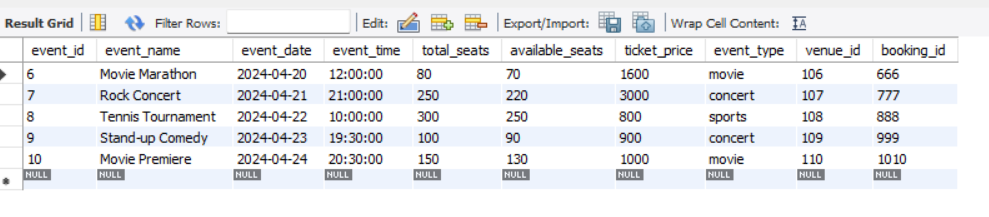
5. Write a SQL query to select events with ticket price range is between 1000 to 2500.

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



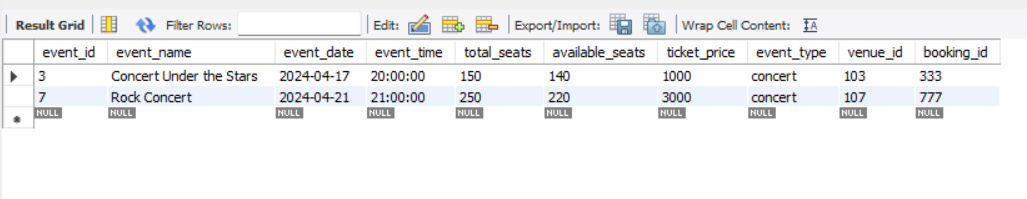
6. Write a SQL query to retrieve events with dates falling within a specific range.

SELECT \* FROM event WHERE event\_date BETWEEN '2024-04-20' AND '2024-04-30';



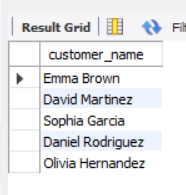
7. Write a SQL query to retrieve events with available tickets that also have "Concert" in name.

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



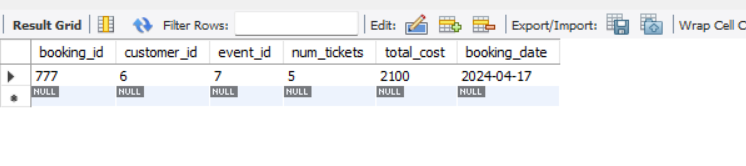
8. Write a SQL query to retrieve users in batches of 5, starting from the 6th user.

SELECT customer\_name FROM customer ORDER BY customer\_id LIMIT 5 OFFSET 5;



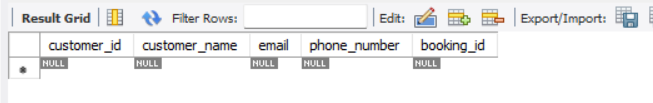
9. Write a SQL query to retrieve bookings details contains booked no of ticket more than 4.

SELECT \* FROM booking WHERE num\_tickets > 4;



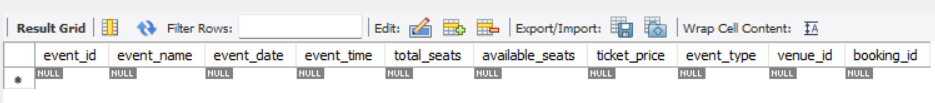
10. Write a SQL query to retrieve customer information whose phone number end with ‘000’.

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



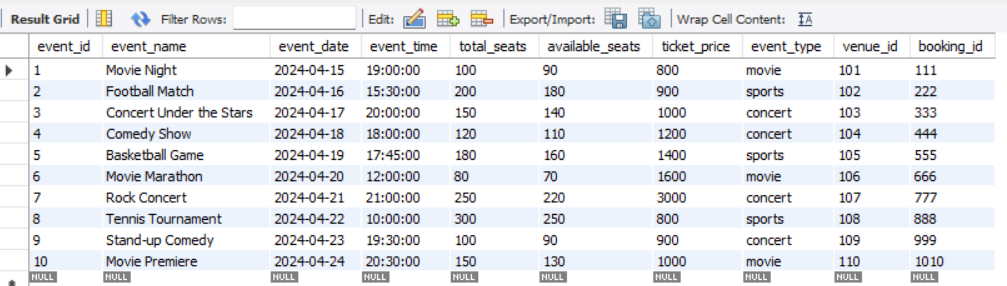
11. Write a SQL query to retrieve the events in order whose seat capacity more than 15000.

SELECT \* FROM event WHERE total\_seats > 15000 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 'z%';



**TASKS 3: AGGREGATE FUNCTIONS, HAVING, ORDER BY, GROUPBY , JOINS:**

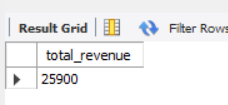
1. Write a SQL query to List Events and Their Average Ticket Prices.

SELECT event\_name, AVG(ticket\_price) AS avg\_TicketPrice FROM event GROUP BY event\_name;



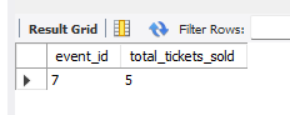
2. Write a SQL query to Calculate the Total Revenue Generated by Events.

SELECT SUM(total\_cost) AS total\_revenue FROM booking;



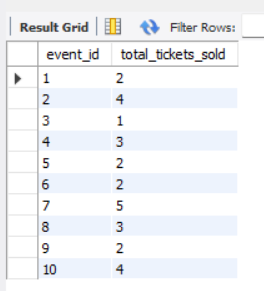
3. Write a SQL query to find the event with the highest ticket sales.

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



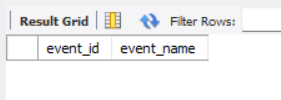
4. Write a SQL query to Calculate the Total Number of Tickets Sold for Each Event.

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



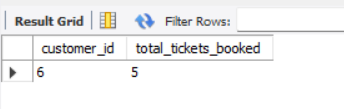
5. Write a SQL query to Find Events with No Ticket Sales.

SELECT event.event\_id, event.event\_name FROM event LEFT JOIN booking ON event.event\_id = booking.event\_id WHERE booking.booking\_id IS NULL;



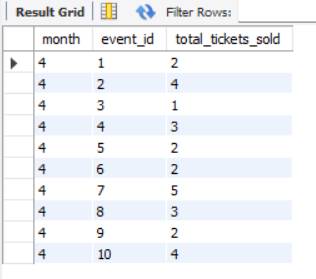
6. Write a SQL query to Find the User Who Has Booked the Most Tickets.

SELECT customer\_id, SUM(num\_tickets) AS total\_tickets\_booked FROM booking GROUP BY customer\_id ORDER BY total\_tickets\_booked DESC LIMIT 1;



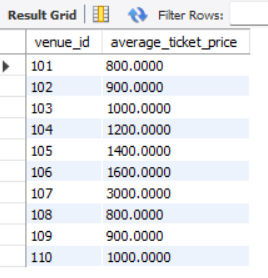
7. Write a SQL query to List Events and the total number of tickets sold for each month.

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



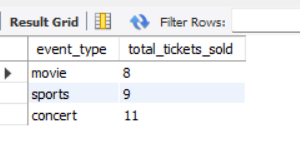
8. Write a SQL query to calculate the average Ticket Price for Events in Each Venue.

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



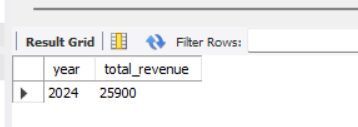
9. Write a SQL query to calculate the total Number of Tickets Sold for Each Event Type.

SELECT 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;



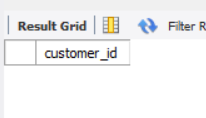
10. Write a SQL query to calculate the total Revenue Generated by Events in Each Year.

SELECT YEAR(b.booking\_date) AS year, SUM(total\_cost) AS total\_revenue FROM booking b JOIN event e ON b.event\_id = e.event\_id GROUP BY YEAR(b.booking\_date);



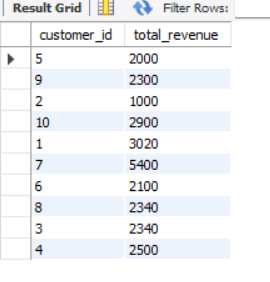
11. Write a SQL query to list users who have booked tickets for multiple events.

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



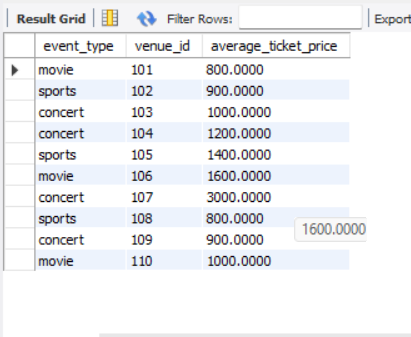
12. Write a SQL query to calculate the Total Revenue Generated by Events for Each User.

SELECT b.customer\_id, SUM(total\_cost) AS total\_revenue FROM booking b JOIN event e ON b.event\_id = e.event\_id GROUP BY b.customer\_id;



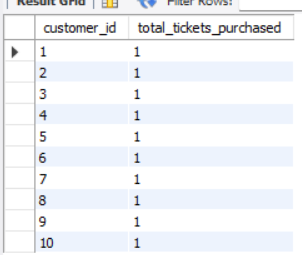
13. Write a SQL query to calculate the Average Ticket Price for Events in Each Category and Venue.

SELECT e.event\_type, e.venue\_id, AVG(e.ticket\_price) AS average\_ticket\_price FROM event e GROUP BY e.event\_type, e.venue\_id;



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

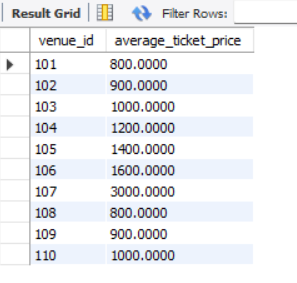
Select customer\_id, count(\*) as total\_tickets\_purchased from booking where booking\_date >= date\_sub(current\_date(), interval 30 day) group by customer\_id;



**TASKS 4: SUBQUERY AND ITS TYPES**

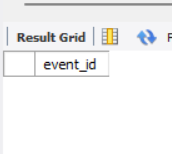
1. Calculate the Average Ticket Price for Events in Each Venue Using a Subquery.

SELECT venue\_id, (SELECT AVG(ticket\_price) FROM event WHERE venue\_id=v.venue\_id) AS average\_ticket\_price FROM venue v;



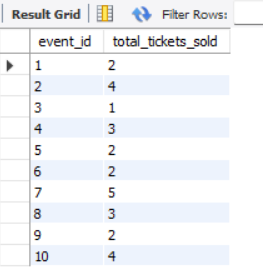
2. Find Events with More Than 50% of Tickets Sold using subquery.

SELECT event\_id FROM booking GROUP BY event\_id HAVING SUM(num\_tickets) > (SELECT 0.5 \* total\_seats FROM event WHERE event.event\_id = booking.event\_id);



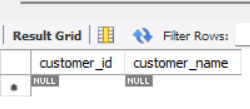
3. Calculate the Total Number of Tickets Sold for Each Event.

SELECT event\_id,(SELECT SUM(num\_tickets) FROM booking WHERE event\_id = e.event\_id) AS total\_tickets\_sold FROM event e;



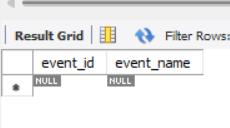
4. Find Users Who Have Not Booked Any Tickets Using a NOT EXISTS Subquery.

SELECT customer\_id, customer\_name FROM customer c WHERE NOT EXISTS (SELECT \* FROM booking WHERE customer\_id = c.customer\_id);



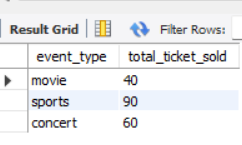
5. List Events with No Ticket Sales Using a NOT IN Subquery.

SELECT event\_id, event\_name FROM event WHERE event\_id NOT IN (SELECT DISTINCT event\_id FROM booking);



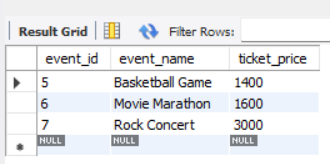
6. Calculate the Total Number of Tickets Sold for Each Event Type Using a Subquery in the FROM Clause.

SELECT event\_type, SUM(total\_tickets\_sold) AS total\_tickets\_sold FROM (SELECT e.event\_type, (SELECT SUM(num\_tickets) FROM booking b WHERE b.event\_id = e.event\_id) AS total\_tickets\_sold FROM event e) AS subquery GROUP BY event\_type;



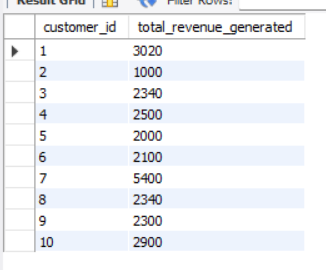
7. Find Events with Ticket Prices Higher Than the Average Ticket Price Using a Subquery in the WHERE Clause.

SELECT event\_id, event\_name, ticket\_price FROM event WHERE ticket\_price > (SELECT AVG(ticket\_price) FROM event);



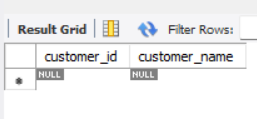
8. Calculate the Total Revenue Generated by Events for Each User Using a Correlated Subquery.

SELECT customer\_id, (SELECT SUM(total\_cost) FROM booking b WHERE b.customer\_id = c.customer\_id) AS total\_revenue\_generated FROM customer c;



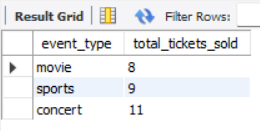
9. List Users Who Have Booked Tickets for Events in a Given Venue Using a Subquery in the WHERE Clause.

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 = (select venue\_id from Venue where venue\_name = 'Venue A')));



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

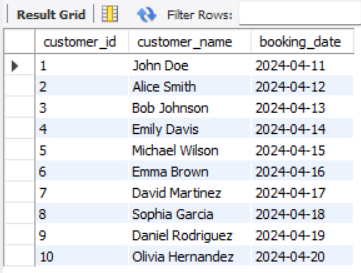
select event\_type, sum(total\_tickets\_sold) as total\_tickets\_sold from (select event\_type, event\_id, (Select (num\_tickets) from Booking where Booking.event\_id = Event.event\_id) as total\_tickets\_sold from event) as EventsByCategory group by event\_type;



11. Find Users Who Have Booked Tickets for Events in each Month Using a Subquery with

DATE\_FORMAT.

select c.customer\_id, c.customer\_name, (select b.booking\_date from booking b where c.booking\_id = b.booking\_id) as booking\_date from customer c order by booking\_date;



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

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

