Ticket Booking System

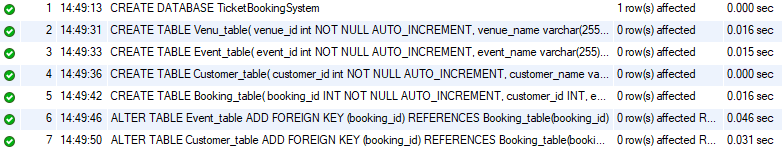
# Database Tables:

## Tasks 1: Database Design:

### Create Database named "TicketBookingSystem":

CREATE DATABASE TicketBookingSystem;

##### Results:



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

#### • Venu:

CREATE TABLE Venu\_table(

venue\_id int NOT NULL AUTO\_INCREMENT,

venue\_name varchar(255),

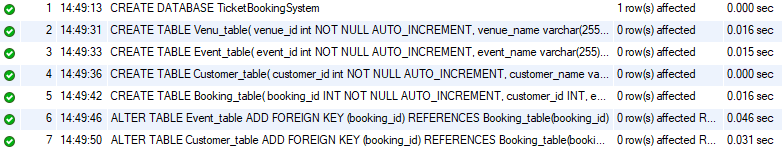
address varchar(255),

UNIQUE(venue\_id),

PRIMARY KEY (venue\_id)

);

##### Results:



#### • Event:

CREATE TABLE Event\_table(

event\_id int NOT NULL AUTO\_INCREMENT,

event\_name varchar(255),

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),

event\_type varchar(7),

CHECK (event\_type IN ('Movie', 'Sports', 'Concert')),

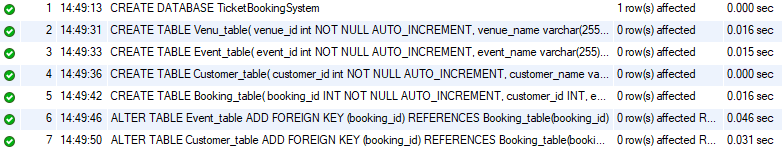
booking\_id int,

PRIMARY KEY (event\_id),

FOREIGN KEY (venue\_id) REFERENCES Venu\_table(venue\_id)

);

##### Results:



#### • Customers:

CREATE TABLE Customer\_table(

customer\_id int NOT NULL AUTO\_INCREMENT,

customer\_name varchar(255),

email varchar(255),

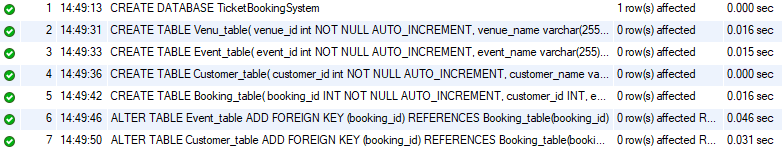
phone\_number bigint,

booking\_id int,

PRIMARY KEY (customer\_id)

);

##### Results:



#### • Booking:

CREATE TABLE Booking\_table(

booking\_id INT NOT NULL AUTO\_INCREMENT,

customer\_id INT,

event\_id INT,

num\_tickets INT DEFAULT 1 ,

total\_cost DECIMAL(10,2),

booking\_date DATE NOT NULL,

UNIQUE(booking\_id),

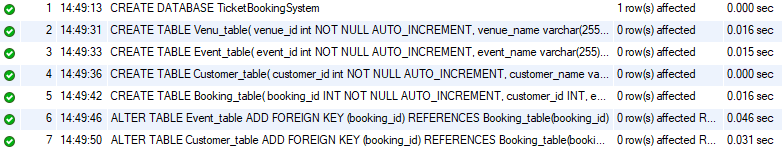
PRIMARY KEY (booking\_id),

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

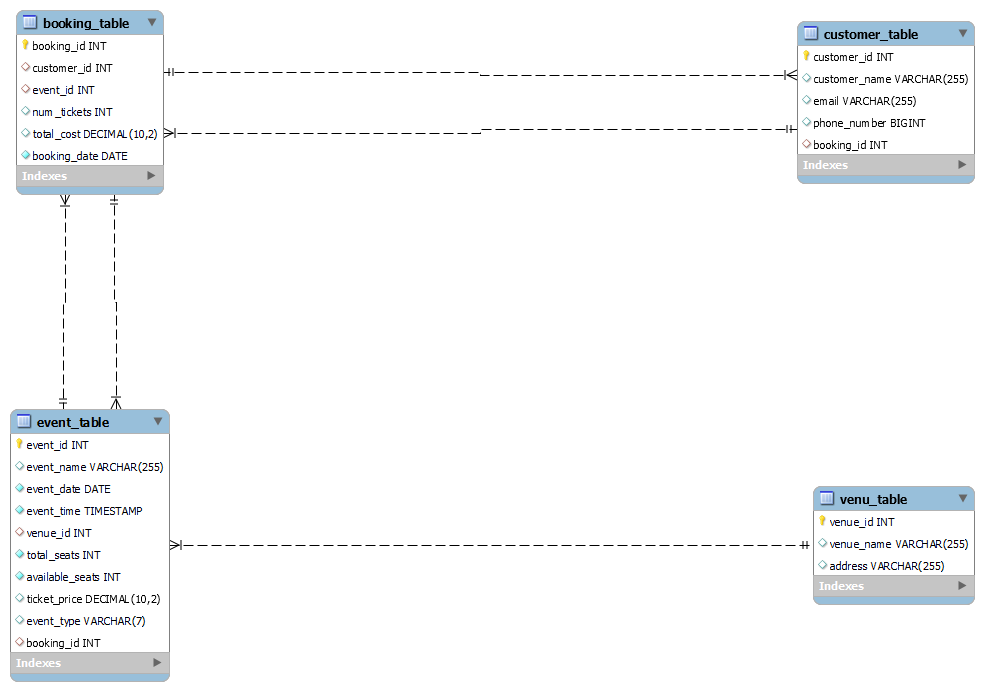
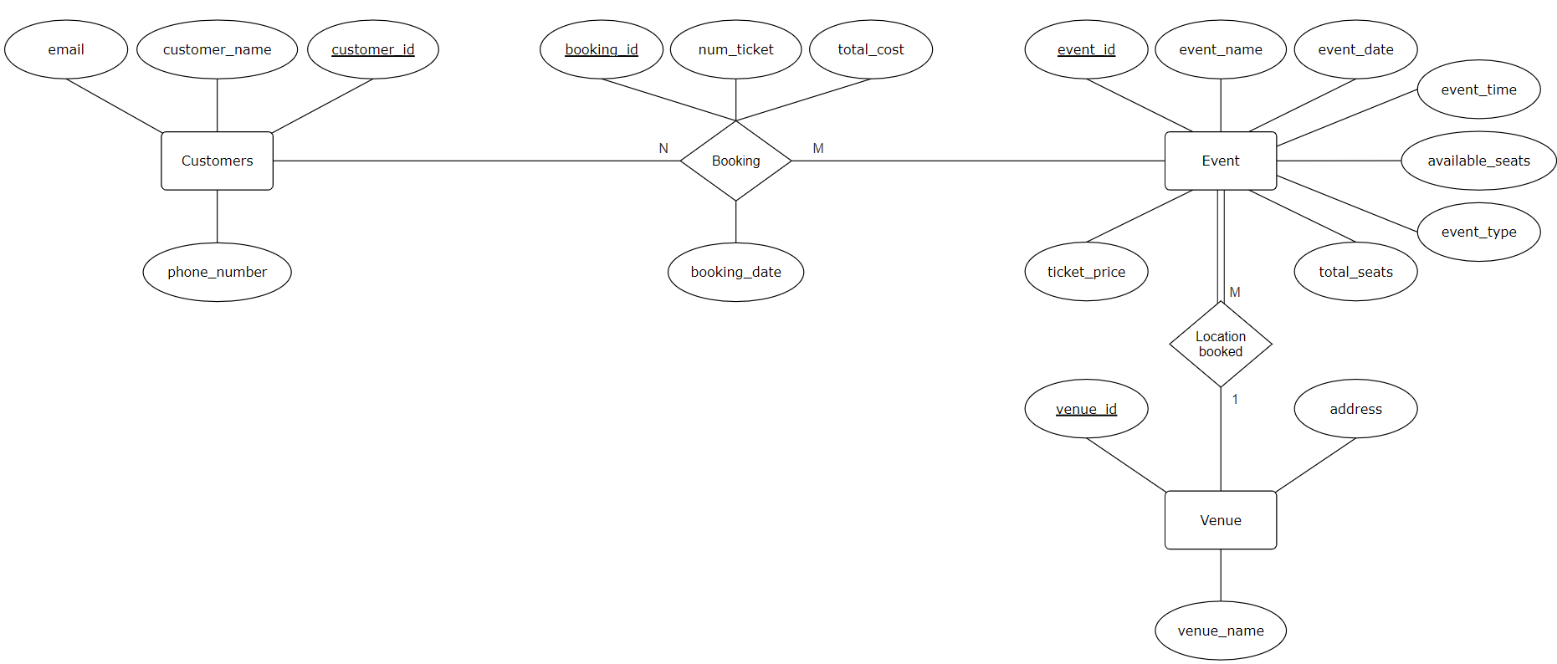
FOREIGN KEY (event\_id) REFERENCES Event\_table(event\_id)

);

##### Results:



### Create an ERD (Entity Relationship Diagram) for the database:



### Create appropriate Primary Key and Foreign Key constraints for referential integrity:

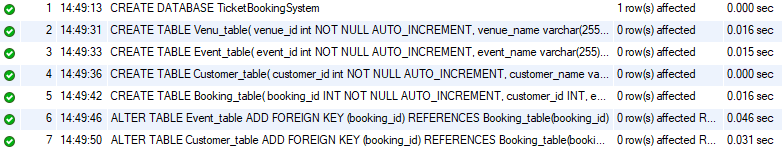
ALTER TABLE Event\_table

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

ALTER TABLE Customer\_table

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

##### Results:



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

### Write a SQL query to insert at least 10 sample records into each table:

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

('EGA Theatre', 'Kilpauk'),

('Chepauk Stadium', 'Chepauk'),

('MGR Theatre', 'Vadapalani'),

('Sathyam Cinemas', 'Royapeta'),

('ECR Beach', 'ECR'),

('Sangam Cinemas', 'Pachiyapaas'),

('Mayajaal Multiplex', 'ECR'),

('PVR', 'Anna Nagar'),

('AGS', 'OMR Navalur'),

('S2', 'Perambur');

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

('Avengers: Endgame', '2019-04-26', '09:30:00', 1, 100, 50, 1500, 'Movie'),

('IPL Cup Final', '2024-05-26', '19:30:00', 2, 15000, 10000, 2000, 'Sports'),

('A.R. Rahman Special', '2023-09-10', '18:00:00', 3, 30000, 0, 2800, 'Concert'),

('Oppenheimer', '2023-07-21', '15:00:00', 4, 300, 200, 2200, 'Movie'),

('Alan Walker SummerSaga', '2024-07-09', '20:00:00', 5, 250, 150, 2300, 'Concert'),

('ICC World Cup Inaugration', '2024-04-28', '10:00:00', 2, 180, 10, 4100, 'Sports'),

('Barbie', '2023-07-21', '18:00:00', 6, 350, 250, 2400, 'Movie'),

('Xtreme Hits Of Sonu Nigam', '2024-04-30', '19:00:00', 5, 20000, 13000, 2000, 'Concert'),

('PKL Match-2', '2023-03-02', '18:00:00', 2, 200, 100, 1900, 'Sports'),

('Kung Fu Panda 4', '2024-04-02', '20:00:00', 8, 150, 50, 1700, 'Movie'),

('Deadpool & Wolverine', '2024-07-24', '10:15:00', 8, 200, 200, 180, 'Movie'),

('The Fall Guy', '2024-05-03', '12:30:00', 9, 400, 400, 210, 'Movie'),

('Avengers: Endgame', '2019-04-30', '10:30:00', 10, 300, 100, 500, 'Movie');

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

('Om', 'om@mail.com', '123456000'),

('Sayooj', 'sayooj@mail.com', '234567010'),

('Atharva', 'atharva@mail.com', '345678000'),

('Vikram', 'vikram@mail.com', '456789010'),

('Rahul', 'rahul@mail.com', '567890000'),

('Jholline', 'jholly@mail.com', '678901010'),

('Srimman', 'srimman@mail.com', '789012000'),

('Jagadeesh', 'jake@mail.com', '890123010'),

('Nischey', 'triggered@mail.com', '901234000'),

('Sonali', 'sonali@mail.com', '012345010'),

('Vaibhav', 'vaibhav@mail.com', '7778889990');

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

(1, 1, 2, 3000, '2019-03-29'),

(2, 2, 3, 6000, '2024-05-03'),

(3, 8, 4, 8000, '2023-02-02'),

(8, 4, 5, 11000, '2023-07-01'),

(5, 5, 2, 4600, '2024-04-22'),

(7, 6, 3, 6300, '2024-03-22'),

(6, 7, 4, 9600, '2023-07-01'),

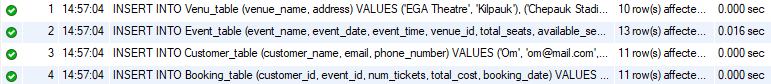
(4, 10, 5, 12500, '2024-04-02'),

(9, 9, 2, 3800, '2024-04-22'),

(10, 3, 3, 8400, '2023-06-18'),

(1, 13, 2, 3000, '2019-03-29');

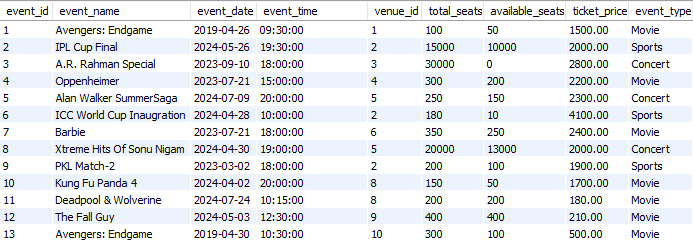
##### Results:



### Write a SQL query to list all Event:

SELECT \* FROM event\_table;

##### Results:

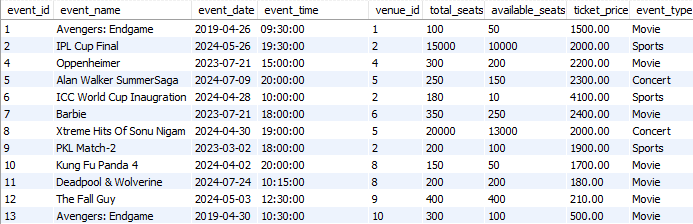


### Write a SQL query to select Event with available tickets:

SELECT \* FROM event\_table

WHERE available\_seats > 0;

##### Results:



### Write a SQL query to select Event name partial match with ‘cup’:

SELECT \* FROM event\_table

WHERE event\_name LIKE '%cup%';

##### Results:

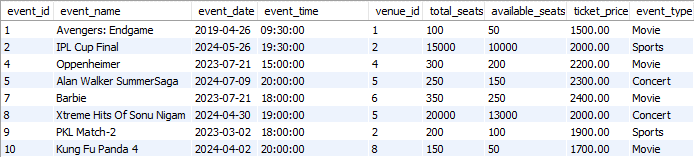


### Write a SQL query to select Event with ticket price range is between 1000 to 2500:

SELECT \* FROM event\_table

WHERE ticket\_price BETWEEN 1000 AND 2500;

##### Results:



### Write a SQL query to retrieve Event with dates falling within a specific range:

SELECT \* FROM event\_table

WHERE event\_date BETWEEN '2023-07-01' AND '2023-07-30';

##### Results:



### Write a SQL query to retrieve Event with available tickets that also have "Concert" in their name:

SELECT \* FROM event\_table

WHERE event\_type = 'Concert'

AND available\_seats > 0;

##### Results:

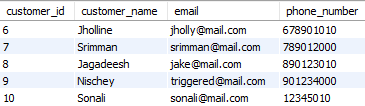


### Write a SQL query to retrieve users in batches of 5, starting from the 6th user:

SELECT \* FROM customer\_table

LIMIT 5 OFFSET 5;

##### Results:



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

SELECT \* FROM booking\_table

WHERE num\_tickets > 4;

##### Results:

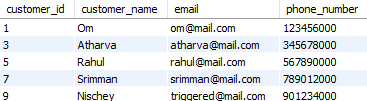


### Write a SQL query to retrieve customer information whose phone number end with ‘000’:

SELECT \* FROM customer\_table

WHERE phone\_number LIKE '%000';

##### Results:



### Write a SQL query to retrieve the Event in order whose seat capacity more than 15000:

SELECT \* FROM event\_table

WHERE total\_seats > 15000

ORDER BY total\_seats;

##### Results:



### Write a SQL query to select Event name not start with ‘x’, ‘y’, ‘z’:

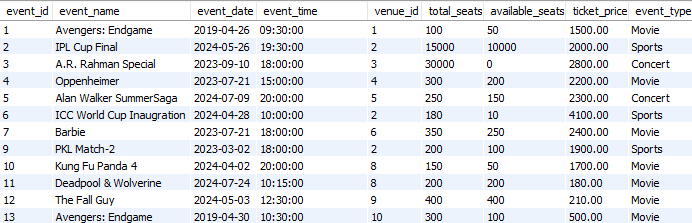
SELECT \* FROM event\_table

WHERE event\_name NOT LIKE 'x%'

AND event\_name NOT LIKE 'y%'

AND event\_name NOT LIKE 'z%';

##### Results:



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

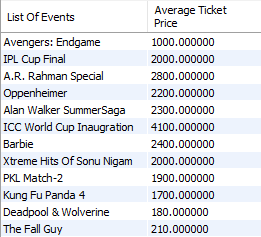
### Write a SQL query to List Events and Their Average Ticket Prices:

SELECT DISTINCT event\_name AS 'List Of Events', AVG(ticket\_price) AS 'Average Ticket Price'

FROM event\_table

GROUP BY event\_name;

##### Results:

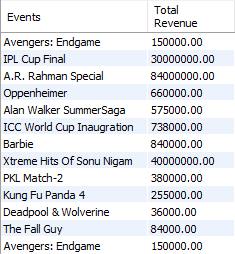


### Write a SQL query to Calculate the Total Revenue Generated by Events:

SELECT event\_name AS 'Events', (total\_seats\*ticket\_price) AS 'Total Revenue'

FROM event\_table;

##### Results:



### Write a SQL query to find the event with the highest ticket sales:

SELECT event\_name AS 'Events', (total\_seats - available\_seats) AS 'Tickets Sold'

FROM event\_table

WHERE (total\_seats - available\_seats) = (SELECT

MAX(total\_seats - available\_seats)

FROM event\_table);

##### Results:

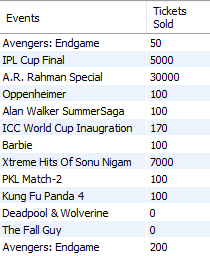


### Write a SQL query to Calculate the Total Number of Tickets Sold for Each Event:

SELECT event\_name AS 'Events', (total\_seats - available\_seats) AS 'Tickets Sold'

FROM event\_table;

##### Results:



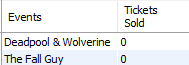
### Write a SQL query to Find Events with No Ticket Sales:

SELECT event\_name AS 'Events', (total\_seats - available\_seats) AS 'Tickets Sold'

FROM event\_table

WHERE total\_seats = available\_seats;

##### Results:



### Write a SQL query to Find the User Who Has Booked the Most Tickets:

SELECT customer\_table.customer\_name AS Customer, booking\_table.num\_tickets AS Tickets

FROM customer\_table

LEFT JOIN booking\_table

ON customer\_table.customer\_id = booking\_table.customer\_id

WHERE booking\_table.num\_tickets = (SELECT

MAX(num\_tickets)

FROM booking\_table);

##### Results:



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

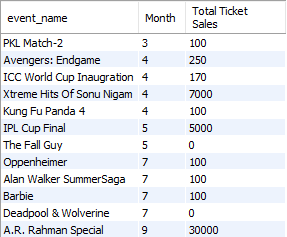
SELECT event\_name, MONTH(event\_date) AS Month, SUM(total\_seats - available\_seats) AS 'Total Ticket Sales'

FROM event\_table

GROUP BY event\_name, Month

ORDER BY Month;

##### Results:



### Write a SQL query to calculate the average Ticket Price for Events in Each Venue:

SELECT v.venue\_name AS Venues, AVG(e.ticket\_price) AS Average\_ticket\_price

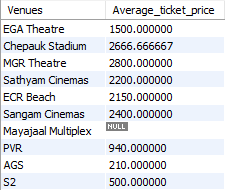
FROM event\_table AS e

RIGHT JOIN venu\_table AS v

ON e.venue\_id = v.venue\_id

GROUP BY v.venue\_name;

##### Results:



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

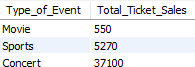
SELECT event\_type AS Type\_of\_Event, SUM(total\_seats - available\_seats) AS Total\_Ticket\_Sales

FROM event\_table

GROUP BY event\_type

ORDER BY Total\_Ticket\_Sales;

##### Results:



### Write a SQL query to calculate the total Revenue Generated by Events in Each Year:

SELECT YEAR(event\_date) AS Year, SUM((total\_seats - available\_seats)\*ticket\_price) AS 'Total Revenue'

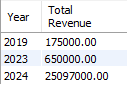
FROM event\_table

WHERE available\_seats <> 0

GROUP BY Year

ORDER BY Year;

##### Results:



### Write a SQL query to list users who have booked tickets for multiple events:

SELECT c.customer\_name AS User, COUNT(b.customer\_id) AS Number\_of\_Bookings

FROM booking\_table AS b

RIGHT JOIN customer\_table AS c

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

GROUP BY c.customer\_name

HAVING (Number\_of\_Bookings > 1);

##### Results:



### Write a SQL query to calculate the Total Revenue Generated by Events for Each User:

SELECT c.customer\_name, SUM(b.num\_tickets\*e.ticket\_price) AS Total\_Revenue

FROM ((event\_table AS e

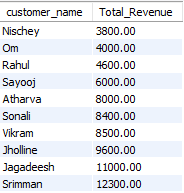
INNER JOIN booking\_table AS b ON e.event\_id = b.event\_id)

INNER JOIN customer\_table AS c ON b.customer\_id = c.customer\_id)

GROUP BY c.customer\_name

ORDER BY Total\_Revenue;

##### Results:



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

SELECT e.event\_type, v.venue\_name AS Venues, AVG(e.ticket\_price) AS Average\_ticket\_price

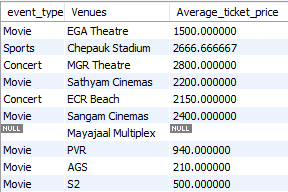
FROM event\_table AS e

RIGHT JOIN venu\_table AS v

ON e.venue\_id = v.venue\_id

GROUP BY Venues, e.event\_type;

##### Results:



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

SELECT customer\_name AS Users, COUNT(customer\_name) AS Number\_of\_Bookings

FROM customer\_table AS c

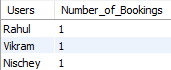
JOIN booking\_table AS b ON b.customer\_id = c.customer\_id

WHERE b.booking\_date BETWEEN DATE\_SUB(CURDATE(), INTERVAL 30 DAY) AND CURDATE()

GROUP BY Users

ORDER BY Number\_of\_Bookings DESC;

##### Results:



## Tasks 4: Subquery and its types:

### Calculate the Average Ticket Price for Events in Each Venue Using a Subquery:

SELECT venue\_name,

(SELECT AVG(ticket\_price)

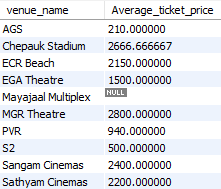
FROM event\_table

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

FROM venu\_table AS v

ORDER BY venue\_name;

##### Results:



### Find Events with More Than 50% of Tickets Sold using subquery:

SELECT DISTINCT a.event\_name AS Events, ((1 - (available\_seats / total\_seats)) \* 100) AS Percent\_Sold

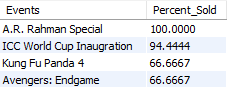
FROM event\_table AS a

WHERE (SELECT ((1 - (available\_seats / total\_seats)) \* 100)

FROM event\_table AS b

WHERE b.event\_id = a.event\_id) > 50;

##### Results:



### Calculate the Total Number of Tickets Sold for Each Event:

SELECT event\_name,

(SELECT num\_tickets

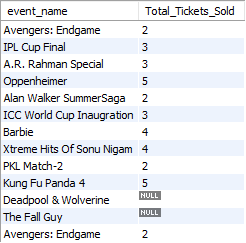
FROM Booking\_table

WHERE event\_id = e.event\_id)

AS Total\_Tickets\_Sold

FROM Event\_table AS e;

##### Results:



### Find Users Who Have Not Booked Any Tickets Using a NOT EXISTS Subquery:

SELECT customer\_name

FROM customer\_table AS c

WHERE NOT EXISTS

(SELECT b.customer\_id

FROM booking\_table

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

##### Results:



### List Events with No Ticket Sales Using a NOT IN Subquery:

SELECT event\_name

FROM event\_table

WHERE event\_name

NOT IN (SELECT event\_name

FROM event\_table

WHERE total\_seats <> available\_seats);

##### Results:



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

SELECT Total\_tickets\_sold, event\_type

FROM (SELECT SUM(b.num\_tickets) AS Total\_tickets\_sold, event\_type

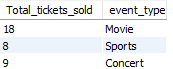
FROM event\_table AS e

JOIN booking\_table AS b

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

GROUP BY event\_type) AS ticket\_sales;

##### Results:



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

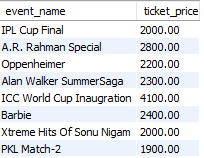
SELECT event\_name, ticket\_price

FROM event\_table

WHERE (SELECT AVG(ticket\_price)

FROM event\_table) < ticket\_price;

##### Results:



### Calculate the Total Revenue Generated by Events for Each User Using a Correlated Subquery:

SELECT customer\_name,

(SELECT SUM(total\_cost)

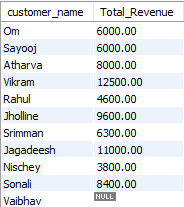
FROM booking\_table

WHERE customer\_id = c.customer\_id

GROUP BY customer\_id) AS Total\_Revenue

FROM customer\_table AS c;

##### Results:



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

SELECT c.customer\_name

FROM Customer\_table AS c

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

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

WHERE e.venue\_id = (SELECT venue\_id

FROM Venu\_table

WHERE venue\_name = 'EGA Theatre');

##### Results:



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

SELECT Total\_tickets\_sold, event\_type

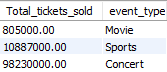
FROM (SELECT

SUM((total\_seats-available\_seats)\*ticket\_price) AS Total\_tickets\_sold, event\_type

FROM event\_table

GROUP BY event\_type) AS ticket\_sales;

##### Results:



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

SELECT customer\_name,

(SELECT DATE\_FORMAT(booking\_date,"%b")

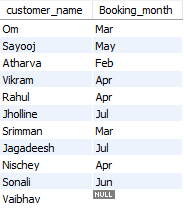
FROM booking\_table

WHERE customer\_id = c.customer\_id

GROUP BY booking\_date) AS Booking\_month

FROM customer\_table AS c;

##### Results:



### Calculate the Average Ticket Price for Events in Each Venue Using a Subquery:

SELECT (SELECT venue\_name

FROM venu\_table

WHERE venue\_id = e.venue\_id) AS venue,

AVG(ticket\_price)

FROM event\_table AS e

GROUP BY venue;

##### Results:

