Assignment 5

Tasks 1: Database Design:

1. Create the database named "TicketBookingSystem"

```
CREATE DATABASE IF NOT EXISTS TicketBookingSystem;
```

```
2    15:02:01    CREATE DATABASE IF NOT EXISTS TicketBoo...  1 row(s) affected    0.0075 sec
```

```
+--------------------+
| Database           |
+--------------------+
| information_schema |
| learn              |
| mysql              |
| performance_schema |
| sys                |
| TicketBookingSystem |
+--------------------+
6 rows in set (0.03 sec)
```

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
);
```

```
+------------+--------------+------+-----+---------+-------+
| Field      | Type         | Null | Key | Default | Extra |
+------------+--------------+------+-----+---------+-------+
| venue_id   | int          | NO   | PRI | NULL    |       |
| venue_name | varchar(255) | NO   |     | NULL    |       |
| address    | varchar(255) | NO   |     | 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)**

```sql
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
);
```

```
+----------------+---------------------------------+------+-----+---------+-------+
| Field          | Type                            | Null | Key | Default | Extra |
+----------------+---------------------------------+------+-----+---------+-------+
| event_id       | int                             | NO   | PRI | NULL    |       |
| event_name     | varchar(255)                    | NO   |     | NULL    |       |
| event_date     | date                            | NO   |     | NULL    |       |
| event_time     | time                            | NO   |     | NULL    |       |
| venue_id       | int                             | YES  | MUL | NULL    |       |
| total_seats    | int                             | NO   |     | NULL    |       |
| available_seats | int                            | NO   |     | NULL    |       |
| ticket_price   | decimal(10,2)                   | NO   |     | NULL    |       |
| event_type     | enum('Movie','Sports','Concert') | NO   |     | NULL    |       |
| booking_id     | int                             | YES  | MUL | NULL    |       |
+----------------+---------------------------------+------+-----+---------+-------+
10 rows in set (0.00 sec)
```

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
);
```

```
9  15:23:45  CREATE TABLE IF NOT EXISTS Customer (  customer_id INT AUTO_INCR...  0 row(s) affected  0.013 sec
```

```
+---------------+-------------+------+-----+---------+-------+
| Field         | Type        | Null | Key | Default | Extra |
+---------------+-------------+------+-----+---------+-------+
| customer_id   | int         | NO   | PRI | NULL    |       |
| customer_name | varchar(255) | NO   |     | NULL    |       |
| email         | varchar(255) | NO   |     | NULL    |       |
| phone_number  | varchar(20) | NO   |     | NULL    |       |
| booking_id    | int         | YES  | MUL | NULL    |       |
+---------------+-------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

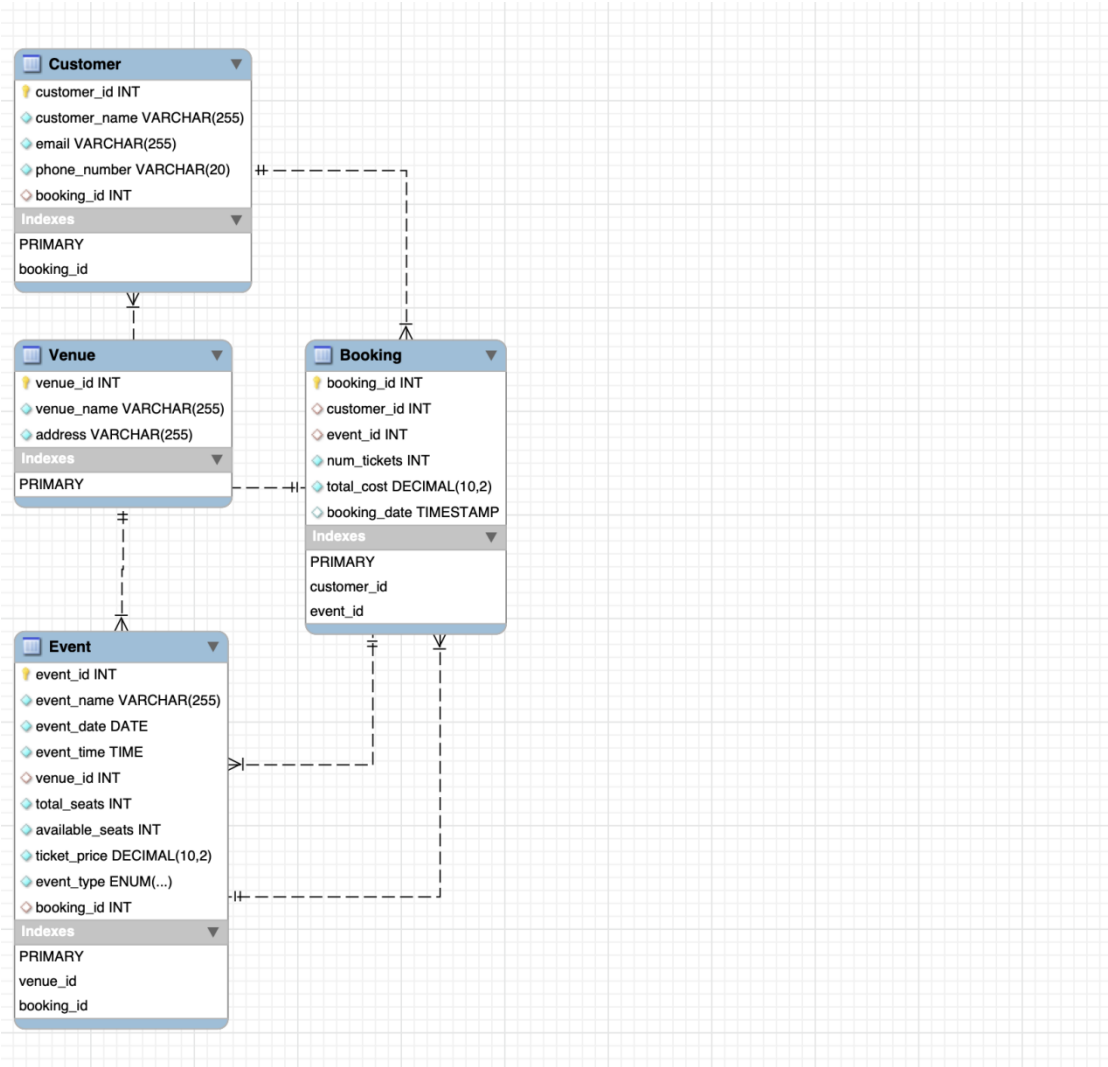
4. Booking Table

• 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
);
```


```
+--------------+---------------+------+-----+-------------------+-------------------+
| Field        | Type          | Null | Key | Default           | Extra             |
+--------------+---------------+------+-----+-------------------+-------------------+
| booking_id   | int           | NO   | PRI | NULL              |                   |
| customer_id  | int           | YES  | MUL | NULL              |                   |
| event_id     | int           | YES  | MUL | NULL              |                   |
| num_tickets  | int           | NO   |     | NULL              |                   |
| total_cost   | decimal(10,2) | NO   |     | NULL              |                   |
| booking_date | timestamp     | YES  |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED |
+--------------+---------------+------+-----+-------------------+-------------------+
6 rows in set (0.00 sec)
```

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



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

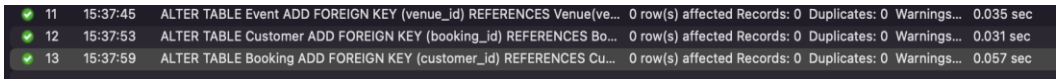
```
ALTER TABLE Event ADD FOREIGN KEY (venue_id) REFERENCES
Venue(venue_id);

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');
```

```
Query OK, 10 rows affected (0.00 sec)
Records: 10  Duplicates: 0  Warnings: 0
```

```
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);
```

```
Query OK, 10 rows affected (0.00 sec)
Records: 10  Duplicates: 0  Warnings: 0
```

```
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);
```

```
Query OK, 10 rows affected (0.00 sec)
Records: 10  Duplicates: 0  Warnings: 0
```

```
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');
```

```
Query OK, 10 rows affected (0.01 sec)
Records: 10  Duplicates: 0  Warnings: 0
```

2. Write a SQL query to list all Events.

```
SELECT * FROM Event;
```

event_id	event_name	event_date	event_time	venue_id	total_seats	available_seats	ticket_price	event_type	booking_id
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

10 rows in set (0.00 sec)

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

```
SELECT * FROM Event WHERE available_seats > 0;
```

event_id	event_name	event_date	event_time	venue_id	total_seats	available_seats	ticket_price	event_type	booking_id
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

10 rows in set (0.00 sec)

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

```
SELECT * FROM Event WHERE event_name LIKE '%cup%';
```

```
Empty set (0.00 sec)
```

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;
```

event_id	event_name	event_date	event_time	venue_id	total_seats	available_seats	ticket_price	event_type	booking_id
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

10 rows in set (0.00 sec)

6. 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';
```

event_id	event_name	event_date	event_time	venue_id	total_seats	available_seats	ticket_price	event_type	booking_id
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

6 rows in set (0.00 sec)

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_type = 'Concert';
```

event_id	event_name	event_date	event_time	venue_id	total_seats	available_seats	ticket_price	event_type	booking_id
1	Event 1	2024-04-10	12:00:00	1	200	200	1500.00	Concert	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
8	Event 8	2024-04-17	16:00:00	8	300	200	1700.00	Concert	NULL

4 rows in set (0.00 sec)

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

```
SELECT * FROM Customer LIMIT 5 OFFSET 5;
```

customer_id	customer_name	email	phone_number	booking_id
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

5 rows in set (0.00 sec)

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

```
SELECT * FROM Booking WHERE num_tickets > 3;
```

booking_id	customer_id	event_id	num_tickets	total_cost	booking_date
4	4	4	4	8800.00	2024-03-05 12:15:00
9	9	9	4	7600.00	2024-04-02 17:45:00

2 rows in set (0.00 sec)

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

```
SELECT * FROM Customer WHERE phone_number LIKE '%000';
```

```
Empty set (0.00 sec)
```

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

```
SELECT * FROM Event WHERE total_seats > 250;
```

event_id	event_name	event_date	event_time	venue_id	total_seats	available_seats	ticket_price	event_type	booking_id
3	Event 3	2024-04-12	15:00:00	3	300	250	1800.00	Sports	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
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

6 rows in set (0.00 sec)

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%';
```

event_id	event_name	event_date	event_time	venue_id	total_seats	available_seats	ticket_price	event_type	booking_id
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

10 rows in set (0.00 sec)

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;
```

event_type	avg_ticket_price
Concert	1650.000000
Movie	2200.000000
Sports	1766.666667

3 rows in set (0.00 sec)

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

```
SELECT SUM(total_cost) AS total_revenue FROM Booking;
```

total_revenue
46600.00

1 row in set (0.01 sec)

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

```sql
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;
```

```
+----------+--------------------+---------------+
| event_id | total_tickets_sold | total_revenue |
+----------+--------------------+---------------+
|        4 |                  4 |       8800.00 |
+----------+--------------------+---------------+
1 row in set (0.00 sec)
```

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

```sql
SELECT event_id, SUM(num_tickets) AS total_tickets_sold
FROM Booking GROUP BY event_id;
```

```
+----------+--------------------+
| event_id | total_tickets_sold |
+----------+--------------------+
|        1 |                  2 |
|        2 |                  3 |
|        3 |                  1 |
|        4 |                  4 |
|        5 |                  2 |
|        6 |                  3 |
|        7 |                  1 |
|        8 |                  2 |
|        9 |                  4 |
|       10 |                  3 |
+----------+--------------------+
10 rows in set (0.00 sec)
```

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

```sql
SELECT event_id, event_name FROM Event WHERE event_id NOT
IN (SELECT event_id FROM Booking);
```

```
Empty set (0.00 sec)
```

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
);
```

```
+-------------+----------------------+
| customer_id | total_tickets_booked |
+-------------+----------------------+
|           4 |                    4 |
|           9 |                    4 |
+-------------+----------------------+
2 rows in set (0.00 sec)
```

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);
```

```
+----------+-------+--------------------+
| event_id | month | total_tickets_sold |
+----------+-------+--------------------+
|        1 |     4 |                  1 |
|        2 |     4 |                  1 |
|        3 |     4 |                  1 |
|        4 |     3 |                  1 |
|        5 |     4 |                  1 |
|        6 |     4 |                  1 |
|        7 |     4 |                  1 |
|        8 |     4 |                  1 |
|        9 |     4 |                  1 |
|       10 |     3 |                  1 |
+----------+-------+--------------------+
10 rows in set (0.00 sec)
```

8. 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;
```

```
+----------+------------------+
| venue_id | avg_ticket_price |
+----------+------------------+
|        1 |      1500.000000 |
|        2 |      2000.000000 |
|        3 |      1800.000000 |
|        4 |      2200.000000 |
|        5 |      1200.000000 |
|        6 |      1600.000000 |
|        7 |      2500.000000 |
|        8 |      1700.000000 |
|        9 |      1900.000000 |
|       10 |      2100.000000 |
+----------+------------------+
10 rows in set (0.01 sec)
```

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;
```

```
+------------+--------------------+
| event_type | total_tickets_sold |
+------------+--------------------+
| Concert    |                 10 |
| Movie      |                  7 |
| Sports     |                  8 |
+------------+--------------------+
3 rows in set (0.00 sec)
```

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);
```

```
+------+---------------+
| year | total_revenue |
+------+---------------+
| 2024 |      46600.00 |
+------+---------------+
1 row in set (0.00 sec)
```

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;
```

```
Empty set (0.00 sec)
```

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;
```

```
+-------------+-------------------------+
| customer_id | total_revenue_generated |
+-------------+-------------------------+
|           1 |                 3000.00 |
|           2 |                 6000.00 |
|           3 |                 1800.00 |
|           4 |                 8800.00 |
|           5 |                 2400.00 |
|           6 |                 4800.00 |
|           7 |                 2500.00 |
|           8 |                 3400.00 |
|           9 |                 7600.00 |
|          10 |                 6300.00 |
+-------------+-------------------------+
10 rows in set (0.00 sec)
```

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;
```

event_type	venue_id	avg_ticket_price
Concert	1	1500.000000
Movie	2	2000.000000
Sports	3	1800.000000
Concert	4	2200.000000
Concert	5	1200.000000
Sports	6	1600.000000
Movie	7	2500.000000
Concert	8	1700.000000
Sports	9	1900.000000
Movie	10	2100.000000

10 rows in set (0.00 sec)

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

```sql
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;
```

customer_id	total_tickets_purchased_last_30_days
1	1
2	1
3	1
5	1
6	1
7	1
8	1
9	1

8 rows in set (0.00 sec)

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;
```

```
+----------+------------+------------------+
| venue_id | venue_name | avg_ticket_price |
+----------+------------+------------------+
|        1 | Venue 1    |      1500.000000 |
|        2 | Venue 2    |      2000.000000 |
|        3 | Venue 3    |      1800.000000 |
|        4 | Venue 4    |      2200.000000 |
|        5 | Venue 5    |      1200.000000 |
|        6 | Venue 6    |      1600.000000 |
|        7 | Venue 7    |      2500.000000 |
|        8 | Venue 8    |      1700.000000 |
|        9 | Venue 9    |      1900.000000 |
|       10 | Venue 10   |      2100.000000 |
+----------+------------+------------------+
10 rows in set (0.01 sec)
```

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);
```

```
Empty set (0.00 sec)
```

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;
```

```
+----------+------------+--------------------+
| event_id | event_name | total_tickets_sold |
+----------+------------+--------------------+
|        1 | Event 1    |                  2 |
|        2 | Event 2    |                  3 |
|        3 | Event 3    |                  1 |
|        4 | Event 4    |                  4 |
|        5 | Event 5    |                  2 |
|        6 | Event 6    |                  3 |
|        7 | Event 7    |                  1 |
|        8 | Event 8    |                  2 |
|        9 | Event 9    |                  4 |
|       10 | Event 10   |                  3 |
+----------+------------+--------------------+
10 rows in set (0.00 sec)
```

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
);
```

```
Empty set (0.00 sec)
```

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
);
```

```
Empty set (0.00 sec)
```

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;
```

```
+------------+--------------------+
| event_type | total_tickets_sold |
+------------+--------------------+
| Concert    |                 10 |
| Movie      |                  7 |
| Sports     |                  8 |
+------------+--------------------+
3 rows in set (0.01 sec)
```

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

```sql
SELECT event_id, event_name, ticket_price
FROM Event
WHERE ticket_price > (
SELECT AVG(ticket_price)
FROM Event
);
```

```
+----------+------------+--------------+
| event_id | event_name | ticket_price |
+----------+------------+--------------+
|        2 | Event 2    |      2000.00 |
|        4 | Event 4    |      2200.00 |
|        7 | Event 7    |      2500.00 |
|        9 | Event 9    |      1900.00 |
|       10 | Event 10   |      2100.00 |
+----------+------------+--------------+
5 rows in set (0.00 sec)
```

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

```sql
SELECT customer_id, customer_name,
(SELECT SUM(total_cost)
FROM Booking
WHERE Booking.customer_id = Customer.customer_id) AS
total_revenue_generated
FROM Customer;
```

```
+-------------+---------------+-------------------------+
| customer_id | customer_name | total_revenue_generated |
+-------------+---------------+-------------------------+
|           1 | John Doe      |                 3000.00 |
|           2 | Jane Smith    |                 6000.00 |
|           3 | Alice Johnson |                 1800.00 |
|           4 | Bob Brown     |                 8800.00 |
|           5 | Charlie Davis |                 2400.00 |
|           6 | Emma Wilson   |                 4800.00 |
|           7 | David Lee     |                 2500.00 |
|           8 | Olivia Clark  |                 3400.00 |
|           9 | James Miller  |                 7600.00 |
|          10 | Sophia Martinez |               6300.00 |
+-------------+---------------+-------------------------+
10 rows in set (0.00 sec)
```

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 = 2
)
);
```

```
+-------------+---------------+
| customer_id | customer_name |
+-------------+---------------+
|           2 | Jane Smith    |
+-------------+---------------+
1 row in set (0.00 sec)
```

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;
```

```
+------------+--------------------+
| event_type | total_tickets_sold |
+------------+--------------------+
| Concert    |                 10 |
| Movie      |                  7 |
| Sports     |                  8 |
+------------+--------------------+
3 rows in set (0.01 sec)
```

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

--

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

```sql
SELECT venue_id, venue_name,
(SELECT AVG(ticket_price)
FROM Event
WHERE Event.venue_id = Venue.venue_id) AS
avg_ticket_price
FROM Venue;
```

```
+----------+------------+------------------+
| venue_id | venue_name | avg_ticket_price |
+----------+------------+------------------+
|        1 | Venue 1    |      1500.000000 |
|        2 | Venue 2    |      2000.000000 |
|        3 | Venue 3    |      1800.000000 |
|        4 | Venue 4    |      2200.000000 |
|        5 | Venue 5    |      1200.000000 |
|        6 | Venue 6    |      1600.000000 |
|        7 | Venue 7    |      2500.000000 |
|        8 | Venue 8    |      1700.000000 |
|        9 | Venue 9    |      1900.000000 |
|       10 | Venue 10   |      2100.000000 |
+----------+------------+------------------+
10 rows in set (0.01 sec)
```