Name: Nishanth J

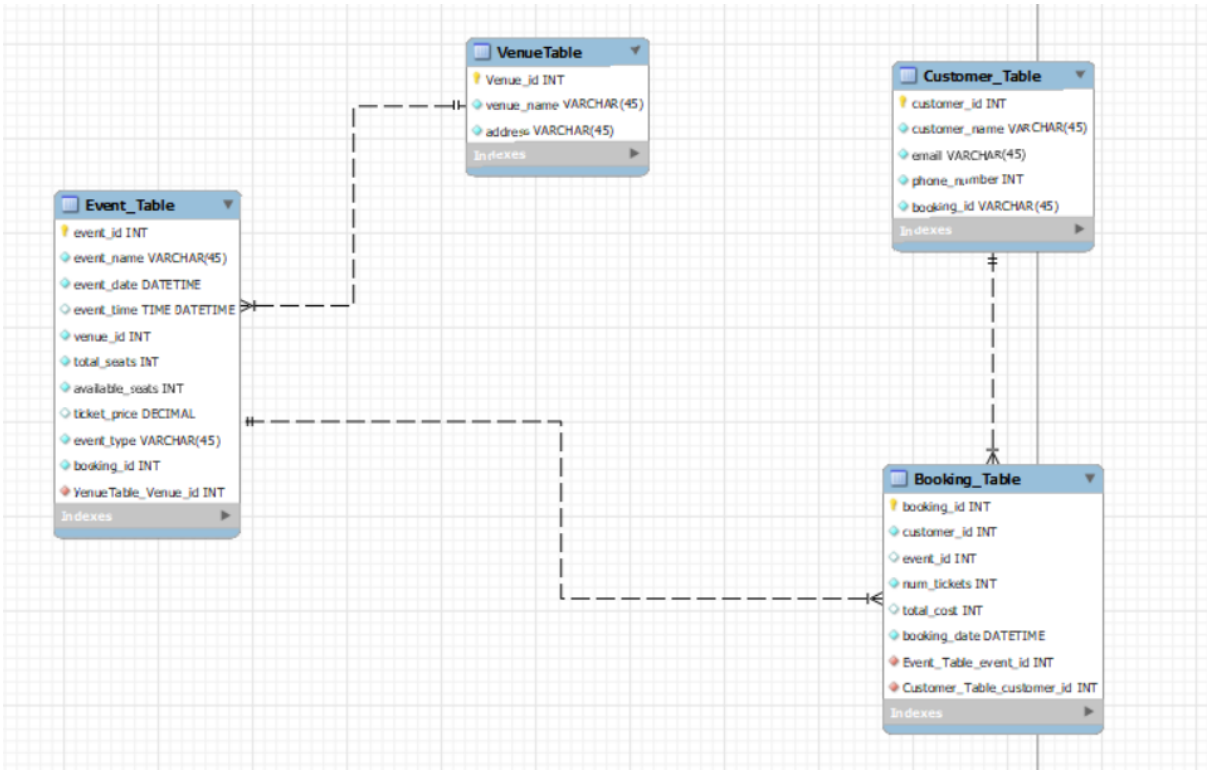
ASSIGNMENT TICKET BOOKING SYSTEM

Tasks 1: Database Design

1. **Create the database named "TicketBookingSystem"**
   Create database TicketBookingSystem;
   Use TicketBookingSystem;
2. **Write SQL scripts to create the mentioned tables with appropriate data types, constraints, and relationships.**
   - **Venu**
   - **Event**
   - **Customers**
   - **Booking**

```
CREATE TABLE VenueTable(
Venue_id INT PRIMARY KEY,
Venue_name VARCHAR(255),
Address Text);
CREATE TABLE EventTable(
Event_id INT PRIMARY KEY,
Event_name VARCHAR(255),
Event_date DATE,
Event_time TIME,
Venue_id INT,
FOREIGN KEY (Venue_id) REFERENCES VenueTable(Venue_id),
Total_seats INT,
Available_seats INT,
Ticket_price DECIMAL,
Event_Type VARCHAR(30),
booking_id INT,
);
CREATE TABLE CustomerTable(
Customer_id INT PRIMARY KEY,
Customer_name VARCHAR(30),
Email VARCHAR(60),
Phone_number INT,
booking_id INT,
);
CREATE TABLE BookingTable(
booking_id INT PRIMARY KEY,
Customer_id INT,
FOREIGN KEY (Customer_id) REFERENCES CustomerTable(Customer_id),
Event_id INT,
FOREIGN KEY (Event_id) REFERENCES EventTable(Event_id),
Num_tickets INT,
Total_cost INT,
Booking_Date DATE);
```

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



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

ALTER TABLE EventTable

ADD FOREIGN KEY (booking_id) REFERENCES BookingTable(booking_id);

ALTER TABLE CustomerTable

ADD FOREIGN KEY (booking_id) REFERENCES BookingTable(booking_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 VenueTable (venue_name, address) VALUES
   ('Venue 1', 'Address 1'),
   ('Venue 2', 'Address 2'),
   ('Venue 3', 'Address 3'),
   ('Venue 4', 'Address 4'),
   ('Venue 5', 'Address 5'),
   ('Venue 6', 'Address 6'),
   ('Venue 7', 'Address 7'),
   ('Venue 8', 'Address 8'),
   ('Venue 9', 'Address 9'),
   ('Venue 10', 'Address 10');

```
INSERT INTO Event_Table (event_name, event_date, event_time, venue_id, total_seats,
available_seats, ticket_price, event_type) VALUES
  ('Event 1', '2024-03-10', '18:00:00', 1, 200, 200, 25.00, 'Movie'),
  ('Event 2', '2024-03-12', '19:30:00', 2, 150, 150, 30.00, 'Concert'),
  ('Event 3', '2024-03-15', '15:00:00', 3, 100, 100, 20.00, 'Sports'),
  ('Event 4', '2024-03-18', '20:00:00', 4, 300, 300, 40.00, 'Movie'),
  ('Event 5', '2024-03-20', '17:30:00', 5, 250, 250, 35.00, 'Concert'),
  ('Event 6', '2024-03-22', '14:00:00', 6, 180, 180, 28.00, 'Sports'),
  ('Event 7', '2024-03-25', '21:00:00', 7, 220, 220, 45.00, 'Movie'),
  ('Event 8', '2024-03-28', '16:45:00', 8, 170, 170, 32.00, 'Concert'),
  ('Event 9', '2024-03-30', '19:00:00', 9, 120, 120, 22.00, 'Sports'),
  ('Event 10', '2024-04-02', '18:30:00', 10, 280, 280, 38.00, 'Movie');

INSERT INTO Customer_Table (customer_name, email, phone_number) VALUES
  ('John Doe', 'john.doe@example.com', '1234567890'),
  ('Alice Smith', 'alice.smith@example.com', '4567890123'),
  ('Bob Johnson', 'bob.johnson@example.com', '7890123456'),
  ('Emily Brown', 'emily.brown@example.com', '0123456789'),
  ('Michael Davis', 'michael.davis@example.com', '2345678901'),
  ('Sarah Wilson', 'sarah.wilson@example.com', '5678901234'),
  ('David Miller', 'david.miller@example.com', '8901234567'),
  ('Emma Martinez', 'emma.martinez@example.com', '3216549870'),
  ('James Taylor', 'james.taylor@example.com', '6549870123'),
  ('Olivia Garcia', 'olivia.garcia@example.com', '9870123456');

INSERT INTO Booking_Table (customer_id, event_id, num_tickets, total_cost, booking_date)
VALUES
  (1, 1, 2, 50.00, NOW()),
  (2, 2, 3, 90.00, NOW()),
  (3, 3, 1, 20.00, NOW()),
  (4, 4, 4, 160.00, NOW()),
  (5, 5, 2, 70.00, NOW()),
  (6, 6, 3, 84.00, NOW()),
  (7, 7, 1, 45.00, NOW()),
  (8, 8, 2, 64.00, NOW()),
  (9, 9, 3, 66.00, NOW()),
  (10, 10, 4, 152.00, NOW());
```

2. Write a SQL query to list all Events.

```
SELECT *
FROM Event_Table;
SELECT event_id, event_name, event_date, event_time, venue_id, total_seats,
available_seats, ticket_price, event_type
FROM Event_Table;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

event_id	event_name	event_date	event_time	venue_id	total_seats	available_seats	ticket_price	event_type
1	Event 1	2024-03-10	18:00:00	1	200	200	25.00	Movie
2	Event 2	2024-03-12	19:30:00	2	150	150	30.00	Concert
3	Event 3	2024-03-15	15:00:00	3	100	100	20.00	Sports
4	Event 4	2024-03-18	20:00:00	4	300	300	40.00	Movie
5	Event 5	2024-03-20	17:30:00	5	250	250	35.00	Concert
6	Event 6	2024-03-22	14:00:00	6	180	180	28.00	Sports
7	Event 7	2024-03-25	21:00:00	7	220	220	45.00	Movie
8	Event 8	2024-03-28	16:45:00	8	170	170	32.00	Concert
9	Event 9	2024-03-30	19:00:00	9	120	120	22.00	Sports
10	Event 10	2024-04-02	18:30:00	10	280	280	38.00	Movie
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

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

```
SELECT *
FROM Event_Table
WHERE available_seats > 0;
SELECT event_id, event_name, event_date, event_time, venue_id, total_seats,
available_seats, ticket_price, event_type
FROM Event_Table
WHERE available_seats > 0;
```

event_id	event_name	event_date	event_time	venue_id	total_seats	available_seats	ticket_price	event_type
1	Event 1	2024-03-10	18:00:00	1	200	200	25.00	Movie
2	Event 2	2024-03-12	19:30:00	2	150	150	30.00	Concert
3	Event 3	2024-03-15	15:00:00	3	100	100	20.00	Sports
4	Event 4	2024-03-18	20:00:00	4	300	300	40.00	Movie
5	Event 5	2024-03-20	17:30:00	5	250	250	35.00	Concert
6	Event 6	2024-03-22	14:00:00	6	180	180	28.00	Sports
7	Event 7	2024-03-25	21:00:00	7	220	220	45.00	Movie
8	Event 8	2024-03-28	16:45:00	8	170	170	32.00	Concert
9	Event 9	2024-03-30	19:00:00	9	120	120	22.00	Sports
10	Event 10	2024-04-02	18:30:00	10	280	280	38.00	Movie
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

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

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

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

event_id	event_name	event_date	event_time	venue_id	total_seats	available_seats	ticket_price	event_type
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

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

```
SELECT *
FROM Event_Table
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
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

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

```
SELECT *
FROM Event_Table
WHERE event_date BETWEEN '2024-03-01' AND '2024-03-31';
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

event_id	event_name	event_date	event_time	venue_id	total_seats	available_seats	ticket_price	event_type
1	Event 1	2024-03-10	18:00:00	1	200	200	25.00	Movie
2	Event 2	2024-03-12	19:30:00	2	150	150	30.00	Concert
3	Event 3	2024-03-15	15:00:00	3	100	100	20.00	Sports
4	Event 4	2024-03-18	20:00:00	4	300	300	40.00	Movie
5	Event 5	2024-03-20	17:30:00	5	250	250	35.00	Concert
6	Event 6	2024-03-22	14:00:00	6	180	180	28.00	Sports
7	Event 7	2024-03-25	21:00:00	7	220	220	45.00	Movie
8	Event 8	2024-03-28	16:45:00	8	170	170	32.00	Concert
9	Event 9	2024-03-30	19:00:00	9	120	120	22.00	Sports
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

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

```
SELECT *
FROM Event_Table
WHERE event_type = 'Concert'
AND available_seats > 0
AND event_name LIKE '%Concert%';
```

event_id	event_name	event_date	event_time	venue_id	total_seats	available_seats	ticket_price	event_type
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

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

```
SELECT *
FROM Customer_Table
ORDER BY customer_id
LIMIT 5 OFFSET 5;
```

customer_id	customer_name	email	phone_number	booking_id
6	Sarah Wilson	sarah.wilson@example.com	2147483647	
7	David Miller	david.miller@example.com	2147483647	
8	Emma Martinez	emma.martinez@example.com	2147483647	
9	James Taylor	james.taylor@example.com	2147483647	
10	Olivia Garcia	olivia.garcia@example.com	2147483647	
NULL	NULL	NULL	NULL	NULL

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

```
SELECT *
FROM Booking_Table
WHERE num_tickets > 4;
```

event_id	event_name	event_date	event_time	venue_id	total_seats	available_seats	ticket_price	event_type
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

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

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

customer_id	customer_name	email	phone_number	booking_id
NULL	NULL	NULL	NULL	NULL

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

```
SELECT *
FROM Event_Table
WHERE total_seats > 15000
ORDER BY total_seats ASC;
```

event_id	event_name	event_date	event_time	venue_id	total_seats	available_seats	ticket_price	event_type
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

11. Write a SQL query to select events 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%';
```

event_id	event_name	event_date	event_time	venue_id	total_seats	available_seats	ticket_price	event_type
1	Event 1	2024-03-10	18:00:00	1	200	200	25.00	Movie
2	Event 2	2024-03-12	19:30:00	2	150	150	30.00	Concert
3	Event 3	2024-03-15	15:00:00	3	100	100	20.00	Sports
4	Event 4	2024-03-18	20:00:00	4	300	300	40.00	Movie
5	Event 5	2024-03-20	17:30:00	5	250	250	35.00	Concert
6	Event 6	2024-03-22	14:00:00	6	180	180	28.00	Sports
7	Event 7	2024-03-25	21:00:00	7	220	220	45.00	Movie
8	Event 8	2024-03-28	16:45:00	8	170	170	32.00	Concert
9	Event 9	2024-03-30	19:00:00	9	120	120	22.00	Sports
10	Event 10	2024-04-02	18:30:00	10	280	280	38.00	Movie
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

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

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

```
SELECT
   e.event_id,
   e.event_name,
   (SELECT AVG(num_tickets * ticket_price) FROM Booking_Table b WHERE b.event_id =
e.event_id) AS average_ticket_price
FROM
   Event_Table e;
```

	event_id	event_name	average_ticket_price
▶	1	Event 1	50.000000
	2	Event 2	90.000000
	3	Event 3	20.000000
	4	Event 4	160.000000
	5	Event 5	70.000000
	6	Event 6	84.000000
	7	Event 7	45.000000
	8	Event 8	64.000000
	9	Event 9	66.000000
	10	Event 10	152.000000

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

```
SELECT
  SUM(booking_revenue) AS total_revenue
FROM
  (SELECT
    SUM(num_tickets * ticket_price) AS booking_revenue
  FROM
    Booking_Table
  JOIN
    Event_Table ON Booking_Table.event_id = Event_Table.event_id
  GROUP BY
    Booking_Table.event_id) AS event_revenues;
```

	total_revenue
▶	801.00

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

```
SELECT
  e.event_id,
  e.event_name,
  SUM(b.num_tickets) AS total_ticket_sales
FROM
  Event_Table e
JOIN
  Booking_Table b ON e.event_id = b.event_id
GROUP BY
  e.event_id, e.event_name
ORDER BY
  total_ticket_sales DESC
LIMIT 1;
```

	event_id	event_name	total_ticket_sales
▶	10	Event 10	4

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

```
SELECT
  e.event_id,
  e.event_name,
  SUM(b.num_tickets) AS total_tickets_sold
FROM
  Event_Table e
JOIN
  Booking_Table b ON e.event_id = b.event_id
GROUP BY
  e.event_id, e.event_name;
```

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	3
10	Event 10	4

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

```
SELECT
  e.event_id,
  e.event_name
FROM
  Event_Table e
LEFT JOIN
  Booking_Table b ON e.event_id = b.event_id
WHERE
  b.booking_id IS NULL;
```

event_id	event_name

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

```
SELECT
  c.customer_id,
  c.customer_name,
  SUM(b.num_tickets) AS total_tickets_booked
FROM
  Customer_Table c
JOIN
  Booking_Table b ON c.customer_id = b.customer_id
GROUP BY
  c.customer_id, c.customer_name
ORDER BY
  total_tickets_booked DESC
LIMIT 1;
```

	customer_id	customer_name	total_tickets_booked
▶	10	Olivia Garcia	4

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

```
SELECT
   YEAR(event_date) AS year,
   MONTH(event_date) AS month,
   e.event_id,
   e.event_name,
   SUM(b.num_tickets) AS total_tickets_sold
FROM
   Event_Table e
JOIN
   Booking_Table b ON e.event_id = b.event_id
GROUP BY
   YEAR(event_date),
   MONTH(event_date),
   e.event_id,
   e.event_name
ORDER BY
   YEAR(event_date),
   MONTH(event_date);
```

	year	month	event_id	event_name	total_tickets_sold
▶	2024	3	1	Event 1	2
	2024	3	2	Event 2	3
	2024	3	3	Event 3	1
	2024	3	4	Event 4	4
	2024	3	5	Event 5	2
	2024	3	6	Event 6	3
	2024	3	7	Event 7	1
	2024	3	8	Event 8	2
	2024	3	9	Event 9	3
	2024	4	10	Event 10	4

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

```
SELECT
   v.venue_id,
   v.venue_name,
   AVG(e.ticket_price) AS average_ticket_price
FROM
   VenueTable v
JOIN
   Event_Table e ON v.venue_id = e.venue_id
GROUP BY
   v.venue_id,
   v.venue_name;
```

	venue_id	venue_name	average_ticket_price
▶	1	Venue 1	25.000000
	2	Venue 2	30.000000
	3	Venue 3	20.000000
	4	Venue 4	40.000000
	5	Venue 5	35.000000
	6	Venue 6	28.000000
	7	Venue 7	45.000000
	8	Venue 8	32.000000
	9	Venue 9	22.000000
	10	Venue 10	38.000000

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_Table e
JOIN
  Booking_Table b ON e.event_id = b.event_id
GROUP BY
  e.event_type;
```

	event_type	total_tickets_sold
▶	Concert	7
	Movie	11
	Sports	7

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

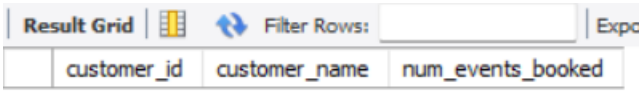
```
SELECT
  YEAR(b.booking_date) AS year,
  SUM(b.num_tickets * e.ticket_price) AS total_revenue
FROM
  Event_Table e
JOIN
  Booking_Table b ON e.event_id = b.event_id
GROUP BY
  YEAR(b.booking_date);
```

	year	total_revenue
▶	2024	801.00

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

```
SELECT
  c.customer_id,
  c.customer_name,
  COUNT(DISTINCT b.event_id) AS num_events_booked
FROM
  Customer_Table c
JOIN
  Booking_Table b ON c.customer_id = b.customer_id
GROUP BY
  c.customer_id,
```

```
    c.customer_name
HAVING
    COUNT(DISTINCT b.event_id) > 1;
```



customer_id	customer_name	num_events_booked

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

```
SELECT
    c.customer_id,
    c.customer_name,
    SUM(b.num_tickets * e.ticket_price) AS total_revenue
FROM
    Customer_Table c
JOIN
    Booking_Table b ON c.customer_id = b.customer_id
JOIN
    Event_Table e ON b.event_id = e.event_id
GROUP BY
    c.customer_id,
    c.customer_name;
```

customer_id	customer_name	total_revenue
1	John Doe	50.00
2	Alice Smith	90.00
3	Bob Johnson	20.00
4	Emily Brown	160.00
5	Michael Davis	70.00
6	Sarah Wilson	84.00
7	David Miller	45.00
8	Emma Martinez	64.00
9	James Taylor	66.00
10	Olivia Garcia	152.00

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

```
SELECT
    e.event_type,
    v.venue_name,
    AVG(e.ticket_price) AS average_ticket_price
FROM
    Event_Table e
JOIN
    VenueTable v ON e.venue_id = v.venue_id
GROUP BY
    e.event_type,
    v.venue_name;
```

event_type	venue_name	average_ticket_price
Concert	Venue 2	30.000000
Concert	Venue 5	35.000000
Concert	Venue 8	32.000000
Movie	Venue 1	25.000000
Movie	Venue 10	38.000000
Movie	Venue 4	40.000000
Movie	Venue 7	45.000000
Sports	Venue 3	20.000000
Sports	Venue 6	28.000000
Sports	Venue 9	22.000000

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

```
SELECT
   c.customer_id,
   c.customer_name,
   SUM(b.num_tickets) AS total_tickets_purchased
FROM
   Customer_Table c
JOIN
   Booking_Table b ON c.customer_id = b.customer_id
WHERE
   b.booking_date >= DATE_SUB(CURRENT_DATE(), INTERVAL 30 DAY)
GROUP BY
   c.customer_id,
   c.customer_name;
```

customer_id	customer_name	total_tickets_purchased
1	John Doe	2
2	Alice Smith	3
3	Bob Johnson	1
4	Emily Brown	4
5	Michael Davis	2
6	Sarah Wilson	3
7	David Miller	1
8	Emma Martinez	2
9	James Taylor	3
10	Olivia Garcia	4

Tasks 4: Subquery and its types

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

```
SELECT
   v.venue_name,
   (SELECT AVG(e.ticket_price)
    FROM Event_Table e
    WHERE e.venue_id = v.venue_id) AS average_ticket_price
FROM
   VenueTable v;
```

venue_name	average_ticket_price
Venue 1	25.000000
Venue 2	30.000000
Venue 3	20.000000
Venue 4	40.000000
Venue 5	35.000000
Venue 6	28.000000
Venue 7	45.000000
Venue 8	32.000000
Venue 9	22.000000
Venue 10	38.000000

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

```
SELECT
   e.event_id,
   e.event_name
FROM
   Event_Table e
WHERE
   (SELECT COUNT(*) FROM Booking_Table b WHERE b.event_id = e.event_id) >
   (SELECT 0.5 * total_seats FROM Event_Table WHERE event_id = e.event_id);
```

event_id	event_name
NULL	NULL

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

```
SELECT
   e.event_id,
   e.event_name,
   SUM(b.num_tickets) AS total_tickets_sold
FROM
   Event_Table e
JOIN
   Booking_Table b ON e.event_id = b.event_id
GROUP BY
   e.event_id,
   e.event_name;
```

event_id	event_name	total_tickets_sold
1	Event 1	2
2	Event 2	3
3	Event 3	1
4	Event 4	4
5	Eve	2
6	Event 6	3
7	Event 7	1
8	Event 8	2
9	Event 9	3
10	Event 10	4

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

```
SELECT
  c.customer_id,
  c.customer_name
FROM
  Customer_Table c
WHERE
  NOT EXISTS (
    SELECT 1
    FROM Booking_Table b
    WHERE b.customer_id = c.customer_id
  );
```

customer_id	customer_name
NULL	NULL

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

```
SELECT
  event_id,
  event_name
FROM
  Event_Table
WHERE
  event_id NOT IN (
    SELECT DISTINCT event_id
    FROM Booking_Table
  );
```

event_id	event_name
NULL	NULL

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

```
SELECT
  event_type,
  COALESCE(total_tickets_sold, 0) AS total_tickets_sold
FROM
  (SELECT
    e.event_type,
    SUM(b.num_tickets) AS total_tickets_sold
```

```
FROM
    Event_Table e
LEFT JOIN
    Booking_Table b ON e.event_id = b.event_id
GROUP BY
    e.event_type) AS subquery;
```

event_type	total_tickets_sold
Concert	7
Movie	11
Sports	7

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_Table
WHERE
    ticket_price > (
        SELECT AVG(ticket_price)
        FROM Event_Table
    );
```

event_id	event_name	ticket_price
4	Event 4	40.00
5	Event 5	35.00
7	Event 7	45.00
8	Event 8	32.00
10	Event 10	38.00
NULL	NULL	NULL

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

```
SELECT
    c.customer_id,
    c.customer_name,
    (
        SELECT SUM(b.num_tickets * e.ticket_price)
        FROM Booking_Table b
        JOIN Event_Table e ON b.event_id = e.event_id
        WHERE b.customer_id = c.customer_id
    ) AS total_revenue
FROM
    Customer_Table c;
```

customer_id	customer_name	total_revenue
1	John Doe	50.00
2	Alice Smith	90.00
3	Bob Johnson	20.00
4	Emily Brown	160.00
5	Michael Davis	70.00
6	Sarah Wilson	84.00
7	David Miller	45.00
8	Emma Martinez	64.00
9	James Taylor	66.00
10	Olivia Garcia	152.00

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

```
SELECT
   DISTINCT c.customer_id,
   c.customer_name
FROM
   Customer_Table c
WHERE
   c.customer_id IN (
      SELECT
         b.customer_id
      FROM
         Booking_Table b
      WHERE
         b.event_id IN (
            SELECT
               event_id
            FROM
               Event_Table
            WHERE
               venue_id = (SELECT venue_id FROM VenueTable WHERE venue_name = 'Your
Venue Name')
         )
   );
```

customer_id	customer_name
NULL	NULL

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

```
SELECT
   e.event_type,
   COALESCE(total_tickets_sold, 0) AS total_tickets_sold
FROM
   (SELECT
      event_id,
      SUM(num_tickets) AS total_tickets_sold
   FROM
```

```
    Booking_Table
  GROUP BY
    event_id) AS subquery
RIGHT JOIN
  Event_Table e ON subquery.event_id = e.event_id
GROUP BY
  e.event_type;
```

event_type	total_tickets_sold
Concert	3
Movie	2
Sports	1

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,
  DATE_FORMAT(b.booking_date, '%Y-%m') AS booking_month
FROM
  Customer_Table c
JOIN
  Booking_Table b ON c.customer_id = b.customer_id
GROUP BY
  c.customer_id,
  c.customer_name,
  DATE_FORMAT(b.booking_date, '%Y-%m');
```

customer_id	customer_name	booking_month
1	John Doe	2024-03
2	Alice Smith	2024-03
3	Bob Johnson	2024-03
4	Emily Brown	2024-03
5	Michael Davis	2024-03
6	Sarah Wilson	2024-03
7	David Miller	2024-03
8	Emma Martinez	2024-03
9	James Taylor	2024-03
10	Olivia Garcia	2024-03

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

```
SELECT
  v.venue_name,
  COALESCE(avg_ticket_price, 0) AS average_ticket_price
FROM
  VenueTable v
LEFT JOIN
  (SELECT
    venue_id,
    AVG(ticket_price) AS avg_ticket_price
   FROM
    Event_Table
   GROUP BY
```

venue_id) AS subquery ON v.venue_id = subquery.venue_id;

venue_name	average_ticket_price
Venue 1	25.000000
Venue 2	30.000000
Venue 3	20.000000
Venue 4	40.000000
Venue 5	35.000000
Venue 6	28.000000
Venue 7	45.000000
Venue 8	32.000000
Venue 9	22.000000
Venue 10	38.000000