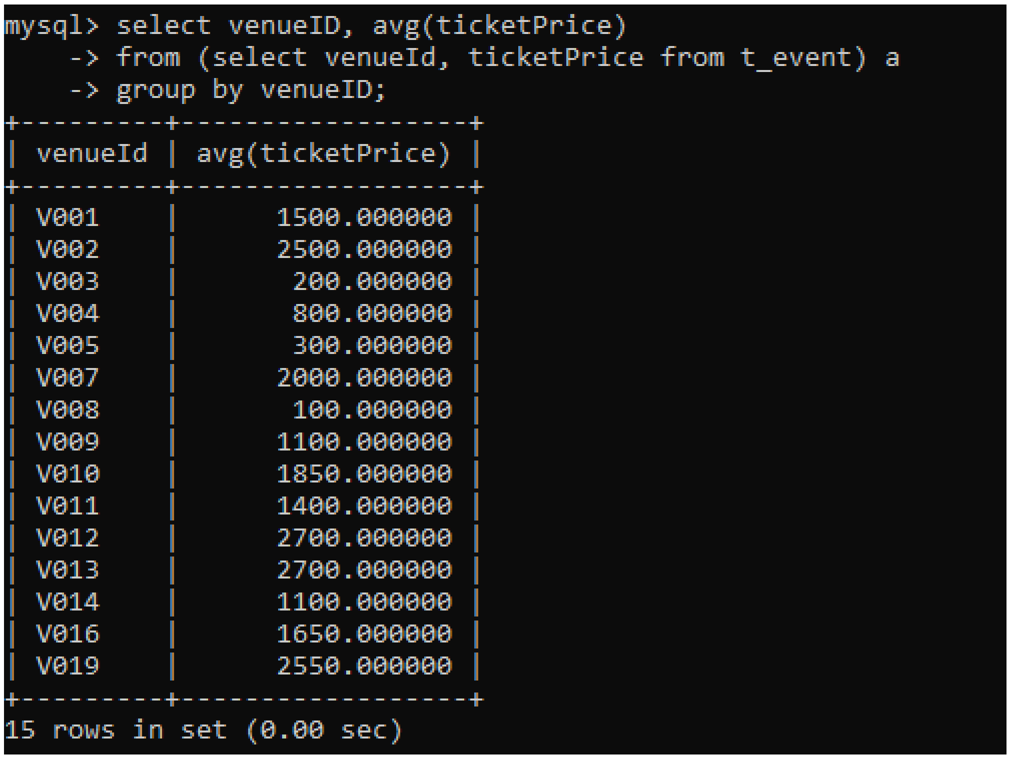
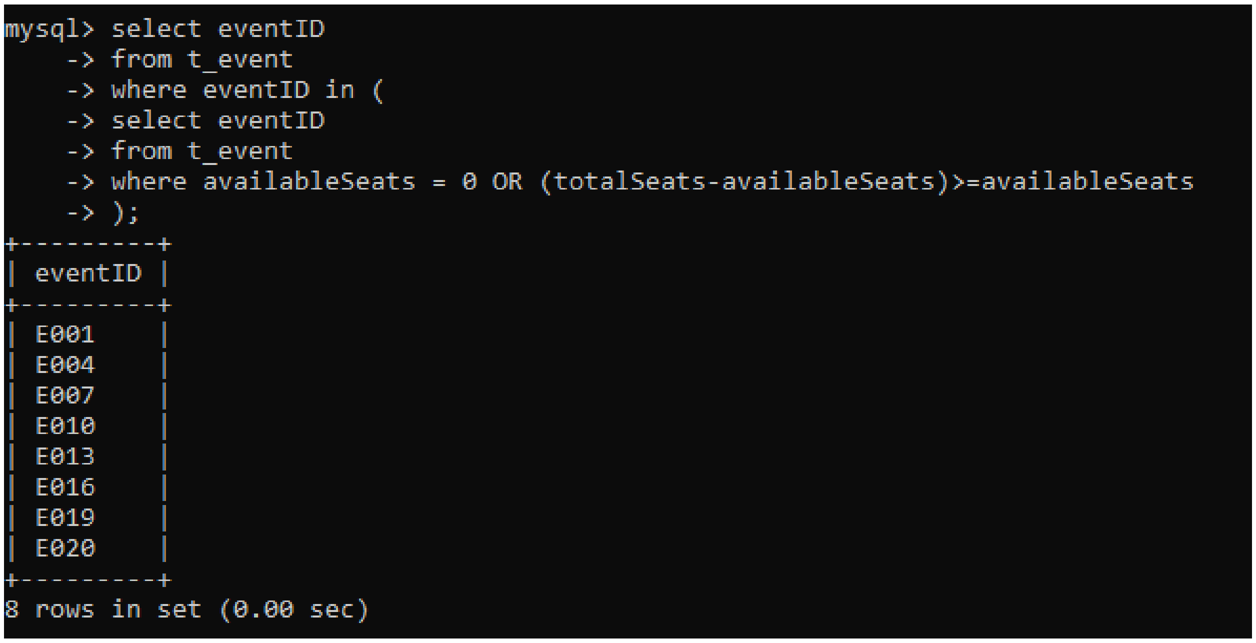
Tasks 4: Subquery and its types

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



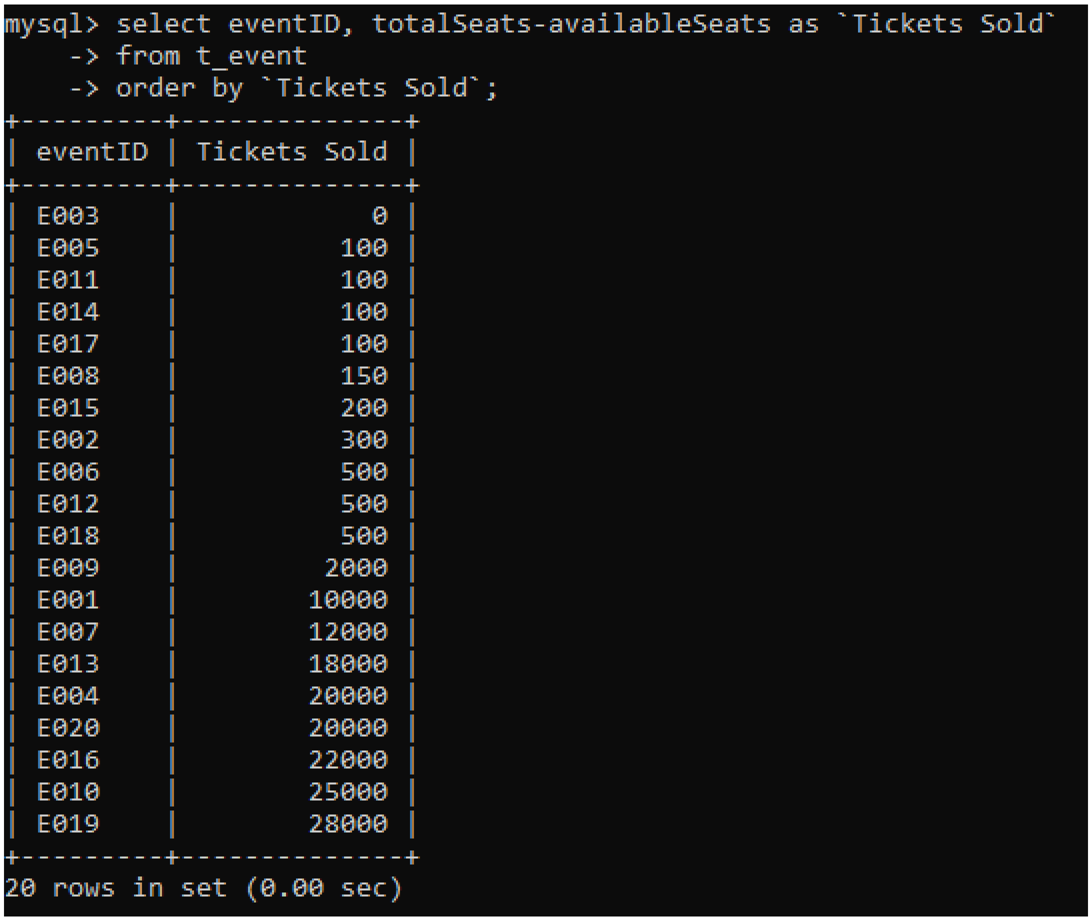
```
mysql> select venueID, avg(ticketPrice)
    -> from (select venueId, ticketPrice from t_event) a
    -> group by venueID;
+---------+------------------+
| venueId | avg(ticketPrice) |
+---------+------------------+
| V001    |      1500.000000 |
| V002    |      2500.000000 |
| V003    |       200.000000 |
| V004    |       800.000000 |
| V005    |       300.000000 |
| V007    |      2000.000000 |
| V008    |       100.000000 |
| V009    |      1100.000000 |
| V010    |      1850.000000 |
| V011    |      1400.000000 |
| V012    |      2700.000000 |
| V013    |      2700.000000 |
| V014    |      1100.000000 |
| V016    |      1650.000000 |
| V019    |      2550.000000 |
+---------+------------------+
15 rows in set (0.00 sec)
```

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



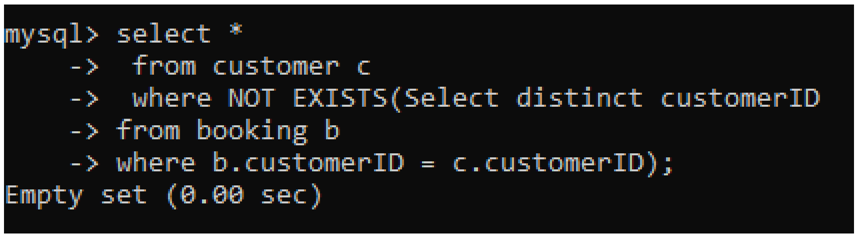
```
mysql> select eventID
    -> from t_event
    -> where eventID in (
    -> select eventID
    -> from t_event
    -> where availableSeats = 0 OR (totalSeats-availableSeats)>=availableSeats
    -> );
+---------+
| eventID |
+---------+
| E001    |
| E004    |
| E007    |
| E010    |
| E013    |
| E016    |
| E019    |
| E020    |
+---------+
8 rows in set (0.00 sec)
```

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



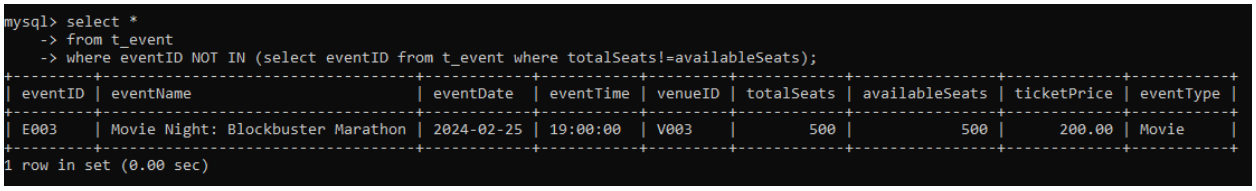
```
mysql> select eventID, totalSeats-availableSeats as `Tickets Sold`
    -> from t_event
    -> order by `Tickets Sold`;
+---------+--------------+
| eventID | Tickets Sold |
+---------+--------------+
| E003    |            0 |
| E005    |          100 |
| E011    |          100 |
| E014    |          100 |
| E017    |          100 |
| E008    |          150 |
| E015    |          200 |
| E002    |          300 |
| E006    |          500 |
| E012    |          500 |
| E018    |          500 |
| E009    |         2000 |
| E001    |        10000 |
| E007    |        12000 |
| E013    |        18000 |
| E004    |        20000 |
| E020    |        20000 |
| E016    |        22000 |
| E010    |        25000 |
| E019    |        28000 |
+---------+--------------+
20 rows in set (0.00 sec)
```

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



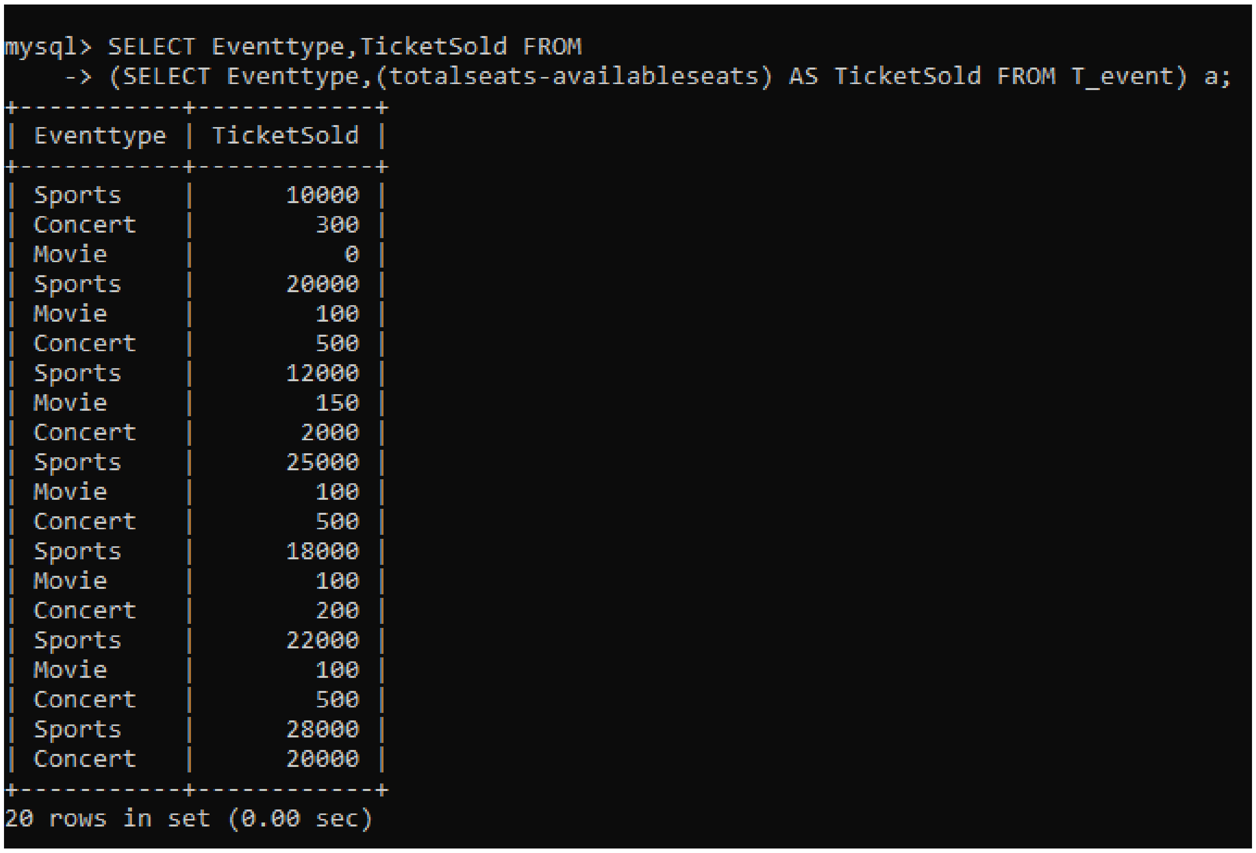
```
mysql> select *
    ->  from customer c
    ->  where NOT EXISTS(Select distinct customerID
    -> from booking b
    -> where b.customerID = c.customerID);
Empty set (0.00 sec)
```

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



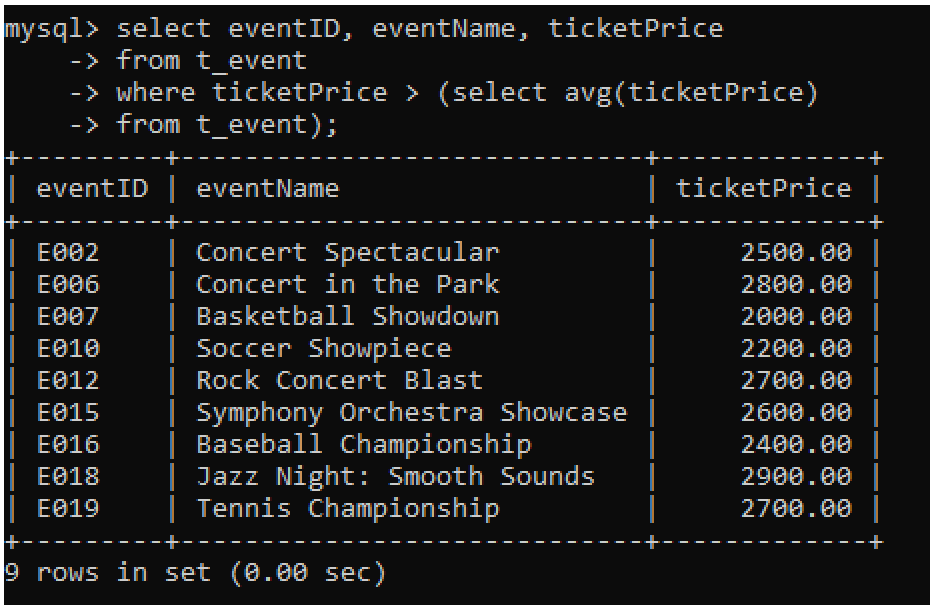
```
mysql> select *
    -> from t_event
    -> where eventID NOT IN (select eventID from t_event where totalSeats!=availableSeats);
+---------+----------------------------------+------------+-----------+---------+------------+----------------+-------------+-----------+
| eventID | eventName                        | eventDate  | eventTime | venueID | totalSeats | availableSeats | ticketPrice | eventType |
+---------+----------------------------------+------------+-----------+---------+------------+----------------+-------------+-----------+
| E003    | Movie Night: Blockbuster Marathon | 2024-02-25 | 19:00:00  | V003    |        500 |            500 |      200.00 | Movie     |
+---------+----------------------------------+------------+-----------+---------+------------+----------------+-------------+-----------+
1 row in set (0.00 sec)
```

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



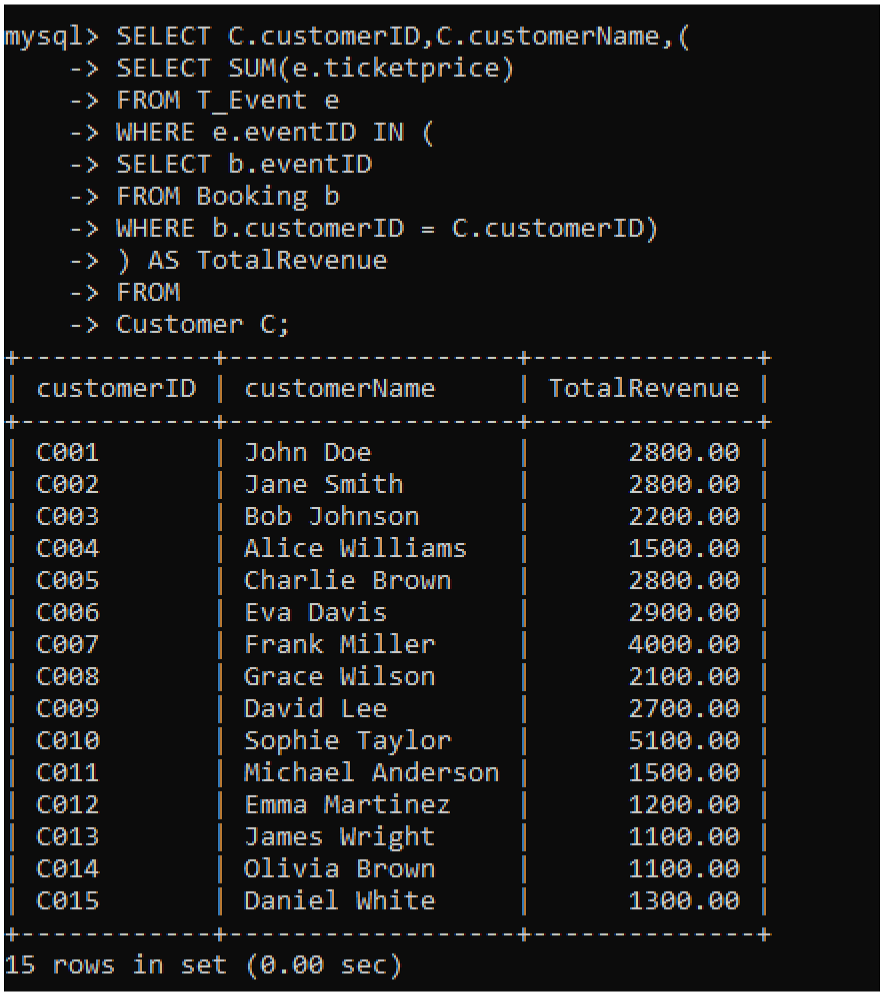
```
mysql> SELECT Eventtype,TicketSold FROM
    -> (SELECT Eventtype,(totalseats-availableseats) AS TicketSold FROM T_event) a;
+-----------+------------+
| Eventtype | TicketSold |
+-----------+------------+
| Sports    |      10000 |
| Concert   |        300 |
| Movie     |          0 |
| Sports    |      20000 |
| Movie     |        100 |
| Concert   |        500 |
| Sports    |      12000 |
| Movie     |        150 |
| Concert   |       2000 |
| Sports    |      25000 |
| Movie     |        100 |
| Concert   |        500 |
| Sports    |      18000 |
| Movie     |        100 |
| Concert   |        200 |
| Sports    |      22000 |
| Movie     |        100 |
| Concert   |        500 |
| Sports    |      28000 |
| Concert   |      20000 |
+-----------+------------+
20 rows in set (0.00 sec)
```

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



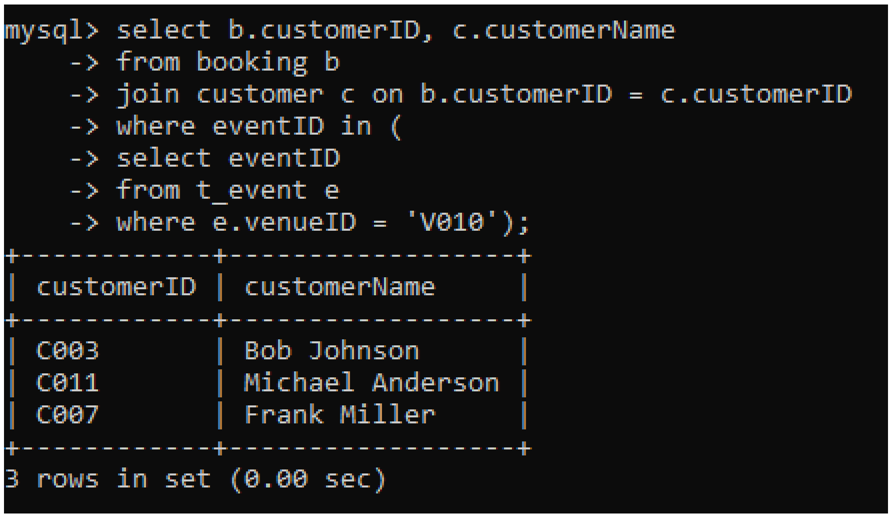
```
mysql> select eventID, eventName, ticketPrice
    -> from t_event
    -> where ticketPrice > (select avg(ticketPrice)
    -> from t_event);
+---------+-----------------------------+-------------+
| eventID | eventName                   | ticketPrice |
+---------+-----------------------------+-------------+
| E002    | Concert Spectacular         |     2500.00 |
| E006    | Concert in the Park         |     2800.00 |
| E007    | Basketball Showdown         |     2000.00 |
| E010    | Soccer Showpiece            |     2200.00 |
| E012    | Rock Concert Blast          |     2700.00 |
| E015    | Symphony Orchestra Showcase |     2600.00 |
| E016    | Baseball Championship       |     2400.00 |
| E018    | Jazz Night: Smooth Sounds   |     2900.00 |
| E019    | Tennis Championship         |     2700.00 |
+---------+-----------------------------+-------------+
9 rows in set (0.00 sec)
```

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



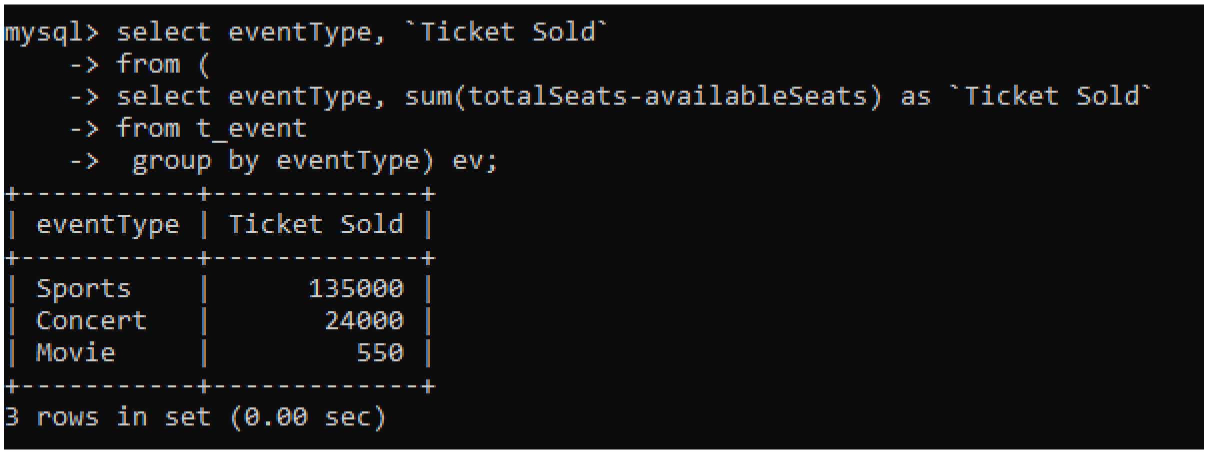
```
mysql> SELECT C.customerID,C.customerName,(
    -> SELECT SUM(e.ticketprice)
    -> FROM T_Event e
    -> WHERE e.eventID IN (
    -> SELECT b.eventID
    -> FROM Booking b
    -> WHERE b.customerID = C.customerID)
    -> ) AS TotalRevenue
    -> FROM
    -> Customer C;
+------------+------------------+--------------+
| customerID | customerName     | TotalRevenue |
+------------+------------------+--------------+
| C001       | John Doe         |      2800.00 |
| C002       | Jane Smith       |      2800.00 |
| C003       | Bob Johnson      |      2200.00 |
| C004       | Alice Williams   |      1500.00 |
| C005       | Charlie Brown    |      2800.00 |
| C006       | Eva Davis        |      2900.00 |
| C007       | Frank Miller     |      4000.00 |
| C008       | Grace Wilson     |      2100.00 |
| C009       | David Lee        |      2700.00 |
| C010       | Sophie Taylor    |      5100.00 |
| C011       | Michael Anderson |      1500.00 |
| C012       | Emma Martinez    |      1200.00 |
| C013       | James Wright     |      1100.00 |
| C014       | Olivia Brown     |      1100.00 |
| C015       | Daniel White     |      1300.00 |
+------------+------------------+--------------+
15 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.



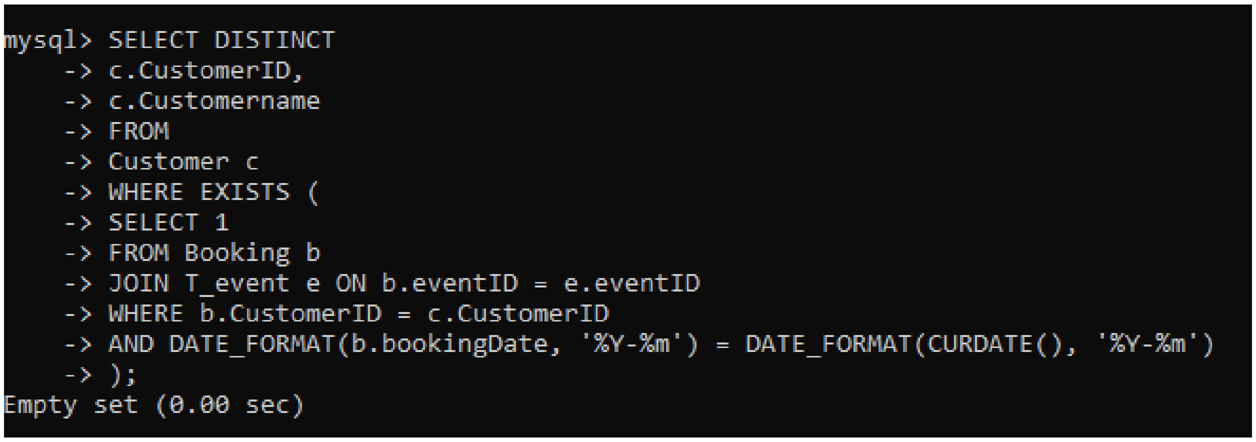
```
mysql> select b.customerID, c.customerName
    -> from booking b
    -> join customer c on b.customerID = c.customerID
    -> where eventID in (
    -> select eventID
    -> from t_event e
    -> where e.venueID = 'V010');
+------------+------------------+
| customerID | customerName     |
+------------+------------------+
| C003       | Bob Johnson      |
| C011       | Michael Anderson |
| C007       | Frank Miller     |
+------------+------------------+
3 rows in set (0.00 sec)
```

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



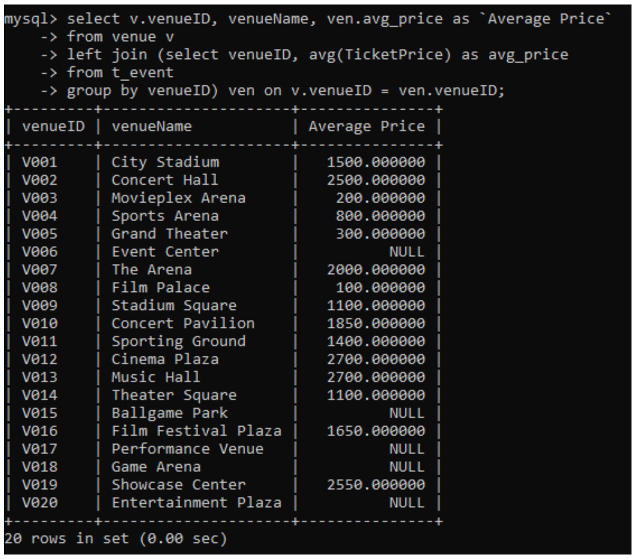
```
mysql> select eventType, `Ticket Sold`
    -> from (
    -> select eventType, sum(totalSeats-availableSeats) as `Ticket Sold`
    -> from t_event
    ->  group by eventType) ev;
+-----------+-------------+
| eventType | Ticket Sold |
+-----------+-------------+
| Sports    |      135000 |
| Concert   |       24000 |
| Movie     |         550 |
+-----------+-------------+
3 rows in set (0.00 sec)
```

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



```
mysql> SELECT DISTINCT
    -> c.CustomerID,
    -> c.Customername
    -> FROM
    -> Customer c
    -> WHERE EXISTS (
    -> SELECT 1
    -> FROM Booking b
    -> JOIN T_event e ON b.eventID = e.eventID
    -> WHERE b.CustomerID = c.CustomerID
    -> AND DATE_FORMAT(b.bookingDate, '%Y-%m') = DATE_FORMAT(CURDATE(), '%Y-%m')
    -> );
Empty set (0.00 sec)
```

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



```
mysql> select v.venueID, venueName, ven.avg_price as `Average Price`
    -> from venue v
    -> left join (select venueID, avg(TicketPrice) as avg_price
    -> from t_event
    -> group by venueID) ven on v.venueID = ven.venueID;
+---------+--------------------+---------------+
| venueID | venueName          | Average Price |
+---------+--------------------+---------------+
| V001    | City Stadium       |   1500.000000 |
| V002    | Concert Hall       |   2500.000000 |
| V003    | Movieplex Arena    |    200.000000 |
| V004    | Sports Arena       |    800.000000 |
| V005    | Grand Theater      |    300.000000 |
| V006    | Event Center       |          NULL |
| V007    | The Arena          |   2000.000000 |
| V008    | Film Palace        |    100.000000 |
| V009    | Stadium Square     |   1100.000000 |
| V010    | Concert Pavilion   |   1850.000000 |
| V011    | Sporting Ground    |   1400.000000 |
| V012    | Cinema Plaza       |   2700.000000 |
| V013    | Music Hall         |   2700.000000 |
| V014    | Theater Square     |   1100.000000 |
| V015    | Ballgame Park      |          NULL |
| V016    | Film Festival Plaza|   1650.000000 |
| V017    | Performance Venue  |          NULL |
| V018    | Game Arena         |          NULL |
| V019    | Showcase Center    |   2550.000000 |
| V020    | Entertainment Plaza|          NULL |
+---------+--------------------+---------------+
20 rows in set (0.00 sec)
```