SQL CODING CHALLENGE

HARSHINI V

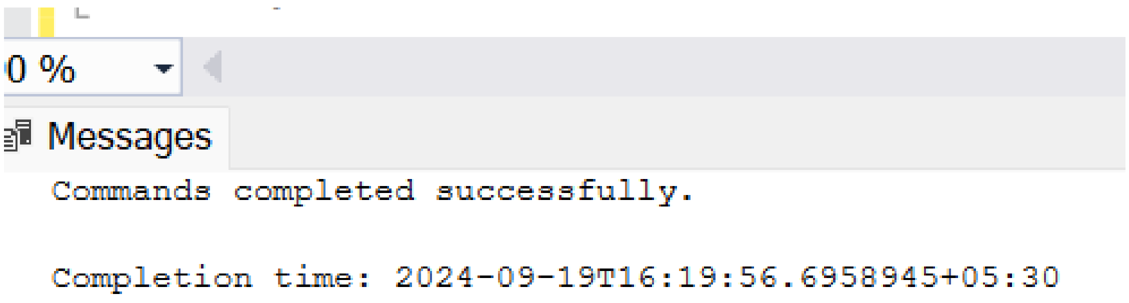
Database Insights: -

1. Banglore , gurgaon & delhi were popular in the bookings, whereas Kolkata is less popular in bookings
2. Nature of Bookings:

- Nearly 50 % of the bookings were made on the day of check in only.
- Nearly 85 % of the bookings were made with less than 4 days prior to the date of check in.
- Very few no.of bookings were made in advance(i.e over a 1 month or 2 months).
- Most of the bookings involved only a single room.
- Nearly 80% of the bookings involved a stay of 1 night only.

3. Oyo should acquire more hotels in the cities of Pune, Kolkata & Mumbai. Because their average room rates are comparetively higher so more revenue will come.
4. The % cancellation Rate is high on all 9 cities except pune , so Oyo should focus on finding reasons about cancellation.

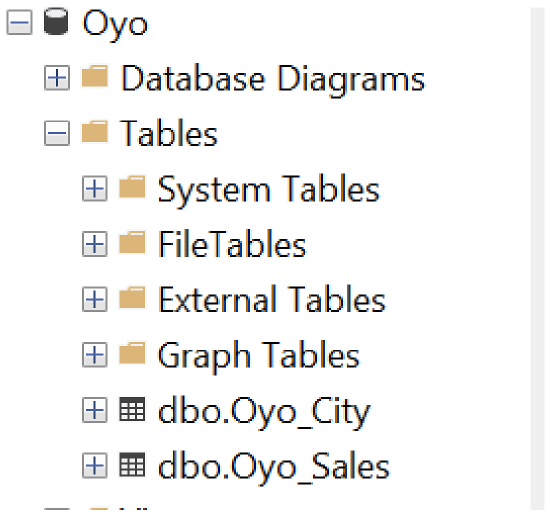
-- create a database Oyo



```
Commands completed successfully.

Completion time: 2024-09-19T16:19:56.6958945+05:30
```

-- import the excel files into MS SQL



```
Oyo
  Database Diagrams
  Tables
    System Tables
    FileTables
    External Tables
    Graph Tables
    dbo.Oyo_City
    dbo.Oyo_Sales
```

1. Querying Data by Using Joins, Subqueries & Subtotals:

Joins:

Joins are SQL operations that combine rows from two or more tables based on a shared column between them. They're essential when data is stored across multiple tables but needs to be analysed together. There are different types of joins, each serving a unique purpose:

- **INNER JOIN**: Only returns rows with matching values in both tables.
- **LEFT JOIN** (or LEFT OUTER JOIN): Returns all rows from the left table and matching rows from the right table, filling unmatched rows with NULLs.
- **RIGHT JOIN** (or RIGHT OUTER JOIN): Returns all rows from the right table and matching rows from the left, with NULLs for unmatched rows in the left table.
- **FULL JOIN** (or FULL OUTER JOIN): Returns all rows with matches from either table, filling unmatched columns with NULLs.
- **CROSS JOIN**: Produces every possible combination of rows from both tables, creating a Cartesian product.

Subqueries:

A subquery (or inner query) is a query embedded within another SQL statement. Subqueries allow you to execute an initial query that provides a result to be used by the outer query. They're versatile and can be applied in clauses like SELECT, FROM, WHERE, and HAVING to create complex filtering and calculations.

There are two main types of subqueries:

- **Scalar Subqueries**: Return a single value, suitable where one value is needed.
- **Table Subqueries**: Return a set of rows, commonly used with IN or EXISTS.

Subtotals:

Subtotals are intermediate or partial totals within a dataset, usually within specific groups. To calculate subtotals, grouping functions like SUM or COUNT are combined with the GROUP BY clause, allowing totals to be calculated within each group.

SQL offers extensions like **ROLLUP** and **CUBE** to generate subtotals automatically:

- **GROUP BY ROLLUP**: Calculates subtotals for each specified group plus an overall total.
- **GROUP BY CUBE**: Calculates subtotals for all combinations of grouped columns.

QUERIES:

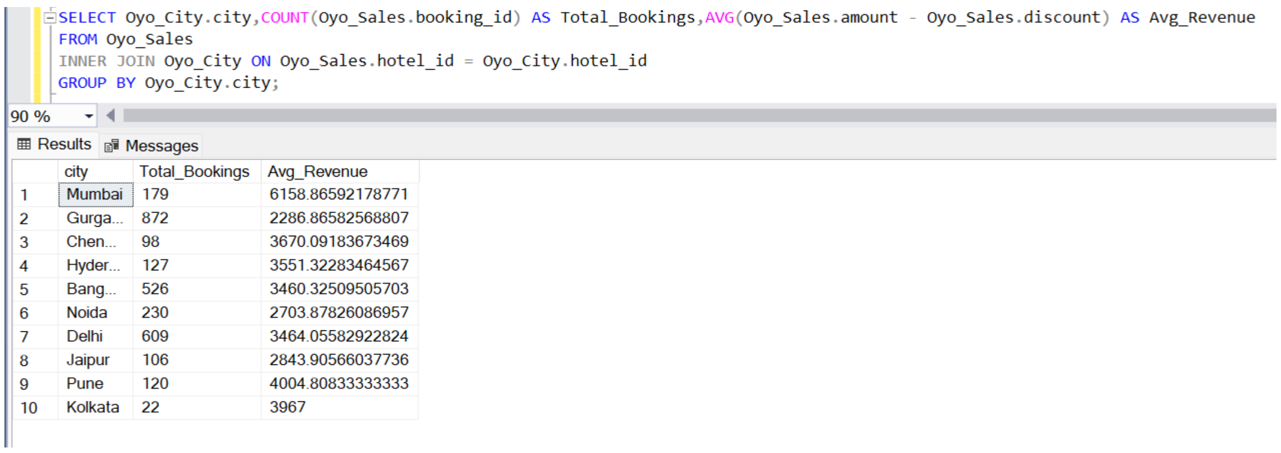
1. Average Revenue and Total Bookings per Each City

```
SELECT Oyo_City.city,COUNT(Oyo_Sales.booking_id) AS Total_Bookings,AVG(Oyo_Sales.amount
- Oyo_Sales.discount) AS Avg_Revenue

FROM Oyo_Sales

INNER JOIN Oyo_City ON Oyo_Sales.hotel_id = Oyo_City.hotel_id

GROUP BY Oyo_City.city;
```



```
SELECT Oyo_City.city,COUNT(Oyo_Sales.booking_id) AS Total_Bookings,AVG(Oyo_Sales.amount - Oyo_Sales.discount) AS Avg_Revenue
FROM Oyo_Sales
INNER JOIN Oyo_City ON Oyo_Sales.hotel_id = Oyo_City.hotel_id
GROUP BY Oyo_City.city;
```

90 %

Results | Messages

	city	Total_Bookings	Avg_Revenue
1	Mumbai	179	6158.86592178771
2	Gurga...	872	2286.86582568807
3	Chen...	98	3670.09183673469
4	Hyder...	127	3551.32283464567
5	Bang...	526	3460.32509505703
6	Noida	230	2703.87826086957
7	Delhi	609	3464.05582922824
8	Jaipur	106	2843.90566037736
9	Pune	120	4004.80833333333
10	Kolkata	22	3967

2. Highest Room Rate per City

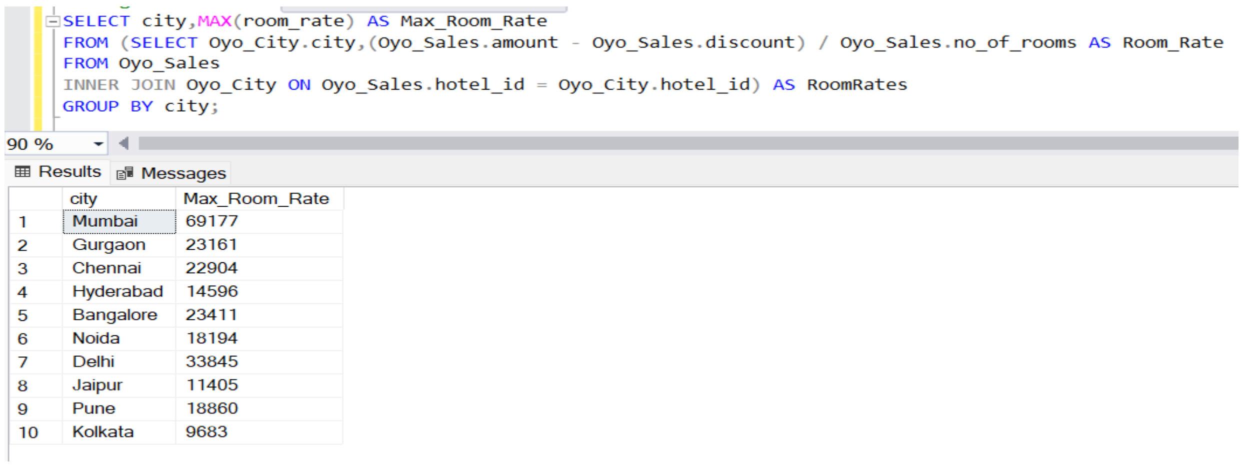
SELECT city,MAX(room_rate) AS Max_Room_Rate

FROM (SELECT Oyo_City.city,(Oyo_Sales.amount - Oyo_Sales.discount) / Oyo_Sales.no_of_rooms AS Room_Rate

FROM Oyo_Sales

INNER JOIN Oyo_City ON Oyo_Sales.hotel_id = Oyo_City.hotel_id) AS RoomRates

GROUP BY city;



```
SELECT city,MAX(room_rate) AS Max_Room_Rate
FROM (SELECT Oyo_City.city,(Oyo_Sales.amount - Oyo_Sales.discount) / Oyo_Sales.no_of_rooms AS Room_Rate
FROM Oyo_Sales
INNER JOIN Oyo_City ON Oyo_Sales.hotel_id = Oyo_City.hotel_id) AS RoomRates
GROUP BY city;
```

90 %

Results | Messages

	city	Max_Room_Rate
1	Mumbai	69177
2	Gurgaon	23161
3	Chennai	22904
4	Hyderabad	14596
5	Bangalore	23411
6	Noida	18194
7	Delhi	33845
8	Jaipur	11405
9	Pune	18860
10	Kolkata	9683

3. Bookings Made on the Same Day as Check-in by City

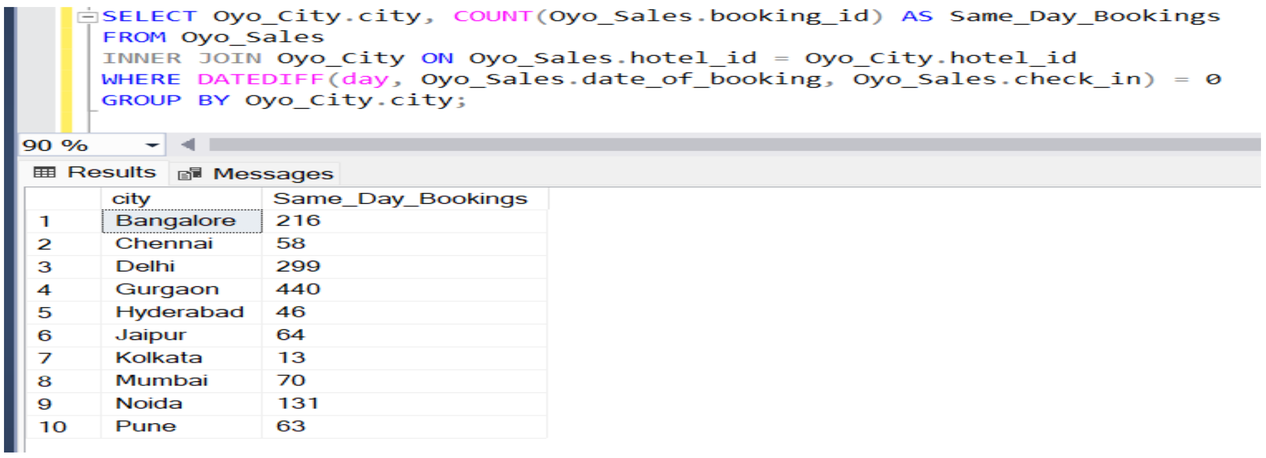
SELECT Oyo_City.city, COUNT(Oyo_Sales.booking_id) AS Same_Day_Bookings

FROM Oyo_Sales

INNER JOIN Oyo_City ON Oyo_Sales.hotel_id = Oyo_City.hotel_id

WHERE DATEDIFF(day, Oyo_Sales.date_of_booking, Oyo_Sales.check_in) = 0

GROUP BY Oyo_City.city;



```
SELECT Oyo_City.city, COUNT(Oyo_Sales.booking_id) AS Same_Day_Bookings
FROM Oyo_Sales
INNER JOIN Oyo_City ON Oyo_Sales.hotel_id = Oyo_City.hotel_id
WHERE DATEDIFF(day, Oyo_Sales.date_of_booking, Oyo_Sales.check_in) = 0
GROUP BY Oyo_City.city;
```

	city	Same_Day_Bookings
1	Bangalore	216
2	Chennai	58
3	Delhi	299
4	Gurgaon	440
5	Hyderabad	46
6	Jaipur	64
7	Kolkata	13
8	Mumbai	70
9	Noida	131
10	Pune	63

2.Manipulate data by using SQL commands using Groupby and Having clause:

GROUP BY Clause

The GROUP BY clause in SQL is used to arrange identical data into groups. When combined with aggregate functions like SUM, COUNT, AVG, MIN, and MAX, GROUP BY enables you to summarize data for each group of values in a selected column or columns. This is especially useful when calculating totals, averages, or other summary statistics across categories, such as sales by region or total revenue per customer. The GROUP BY clause essentially condenses rows with shared attributes, letting you analyze data based on these grouped results rather than individual rows.

HAVING Clause

The HAVING clause is used alongside GROUP BY to filter grouped records based on a specified condition. While the WHERE clause filters rows before grouping, HAVING filters groups after they have been created, making it useful for applying conditions to aggregate data. This means you can filter groups based on aggregate function results, such as only showing groups where the total revenue exceeds a certain amount or where the count of items meets a threshold. By using HAVING, you gain precise control over the output of grouped data, allowing for more focused analysis based on aggregate conditions.

QUERIES:

1. List Of Cities with Over 100 Bookings

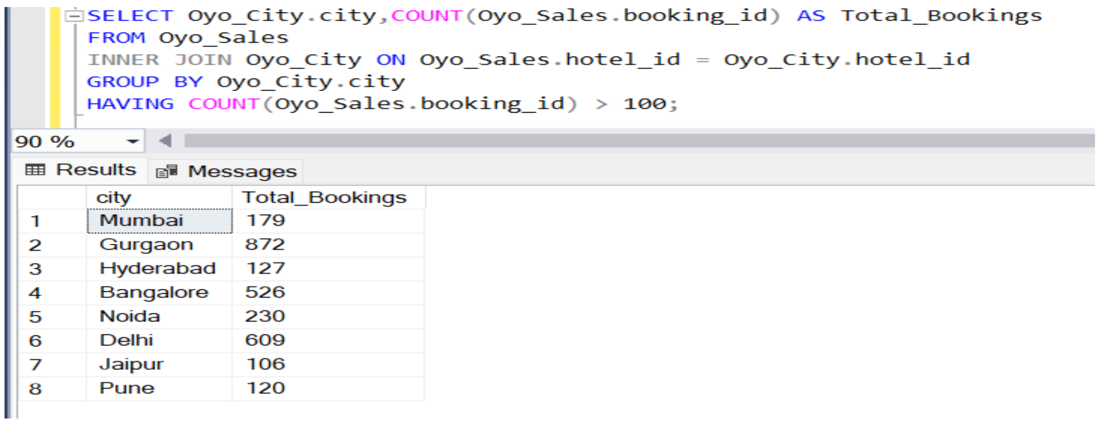
SELECT Oyo_City.city,COUNT(Oyo_Sales.booking_id) AS Total_Bookings

FROM Oyo_Sales

INNER JOIN Oyo_City ON Oyo_Sales.hotel_id = Oyo_City.hotel_id

GROUP BY Oyo_City.city

HAVING COUNT(Oyo_Sales.booking_id) > 100;



```sql
SELECT Oyo_City.city,COUNT(Oyo_Sales.booking_id) AS Total_Bookings
FROM Oyo_Sales
INNER JOIN Oyo_City ON Oyo_Sales.hotel_id = Oyo_City.hotel_id
GROUP BY Oyo_City.city
HAVING COUNT(Oyo_Sales.booking_id) > 100;
```

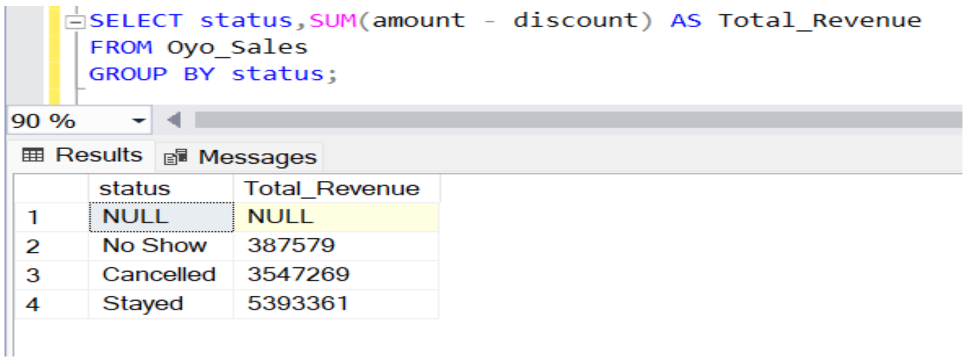
	city	Total_Bookings
1	Mumbai	179
2	Gurgaon	872
3	Hyderabad	127
4	Bangalore	526
5	Noida	230
6	Delhi	609
7	Jaipur	106
8	Pune	120

2. Total Revenue Generated for Each Status of Bookings

SELECT status,SUM(amount - discount) AS Total_Revenue

FROM Oyo_Sales

GROUP BY status;



```sql
SELECT status,SUM(amount - discount) AS Total_Revenue
FROM Oyo_Sales
GROUP BY status;
```

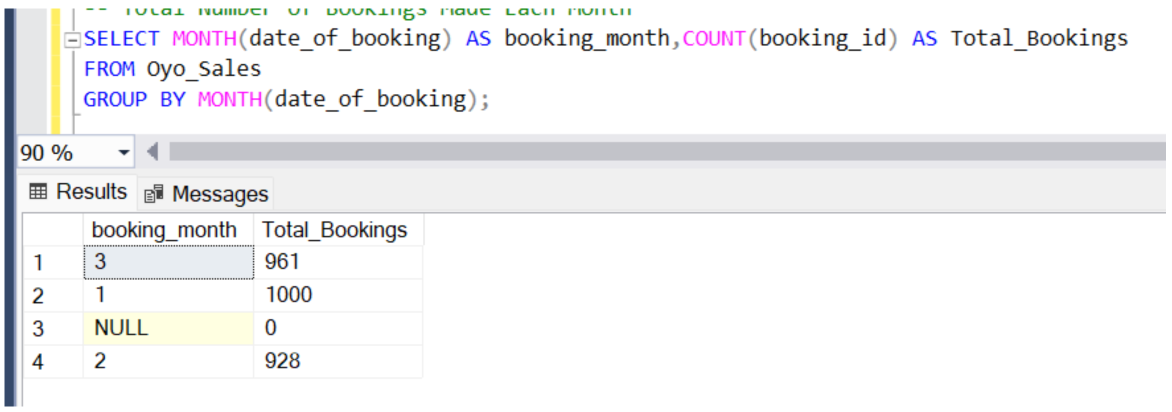
	status	Total_Revenue
1	NULL	NULL
2	No Show	387579
3	Cancelled	3547269
4	Stayed	5393361

3. Total Number of Bookings Made Each Month

SELECT MONTH(date_of_booking) AS booking_month,COUNT(booking_id) AS Total_Bookings

FROM Oyo_Sales

GROUP BY MONTH(date_of_booking);



```sql
-- Total Number of Bookings Made Each Month
SELECT MONTH(date_of_booking) AS booking_month,COUNT(booking_id) AS Total_Bookings
FROM Oyo_Sales
GROUP BY MONTH(date_of_booking);
```

	booking_month	Total_Bookings
1	3	961
2	1	1000
3	NULL	0
4	2	928

4. Average Discount Given for Bookings Exceeding $600 in Amount

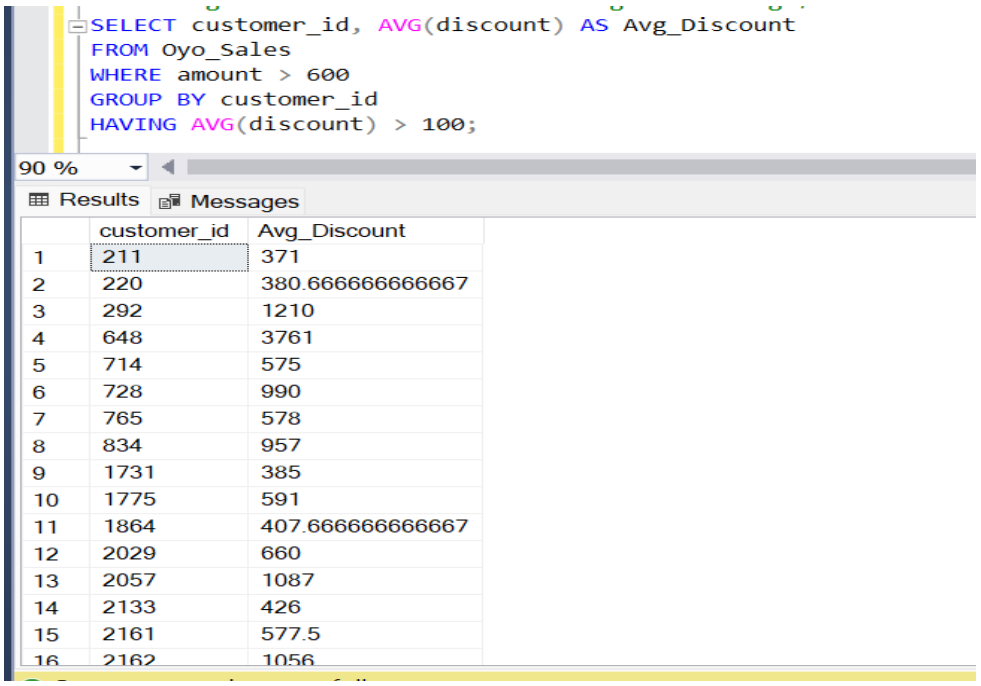
SELECT customer_id, AVG(discount) AS Avg_Discount

FROM Oyo_Sales

WHERE amount > 600

GROUP BY customer_id

HAVING AVG(discount) > 100;



```
SELECT customer_id, AVG(discount) AS Avg_Discount
FROM Oyo_Sales
WHERE amount > 600
GROUP BY customer_id
HAVING AVG(discount) > 100;
```

	customer_id	Avg_Discount
1	211	371
2	220	380.666666666667
3	292	1210
4	648	3761
5	714	575
6	728	990
7	765	578
8	834	957
9	1731	385
10	1775	591
11	1864	407.666666666667
12	2029	660
13	2057	1087
14	2133	426
15	2161	577.5
16	2162	1056

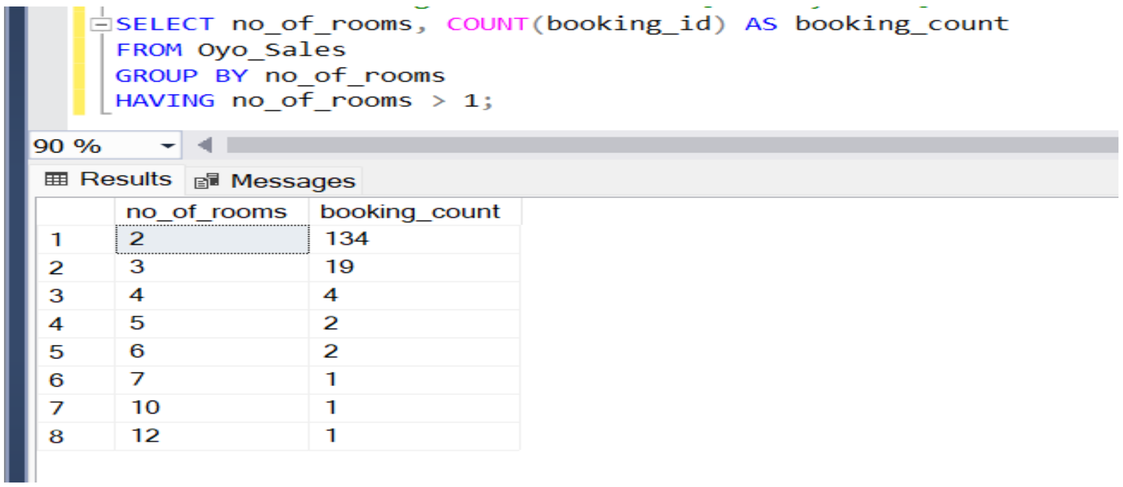
5. Count of Bookings for Each Room Quantity for Quantities Greater Than 1

SELECT no_of_rooms, COUNT(booking_id) AS booking_count

FROM Oyo_Sales

GROUP BY no_of_rooms

HAVING no_of_rooms > 1;



```
SELECT no_of_rooms, COUNT(booking_id) AS booking_count
FROM Oyo_Sales
GROUP BY no_of_rooms
HAVING no_of_rooms > 1;
```

	no_of_rooms	booking_count
1	2	134
2	3	19
3	4	4
4	5	2
5	6	2
6	7	1
7	10	1
8	12	1