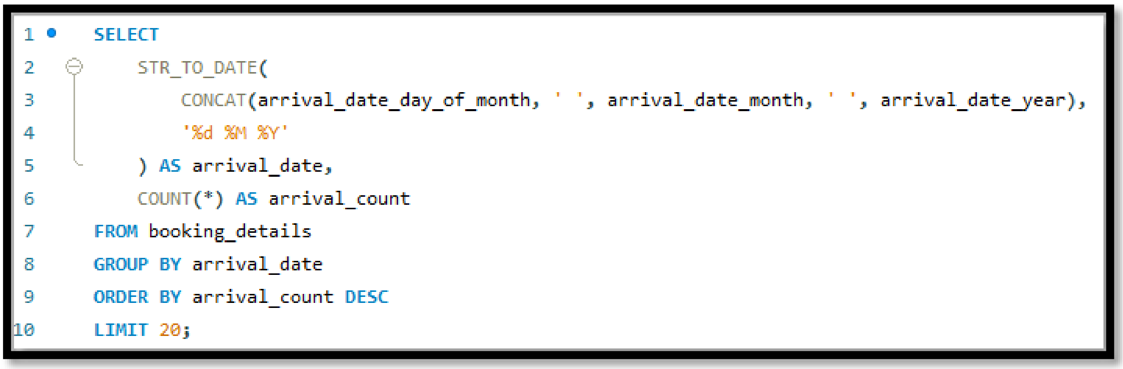
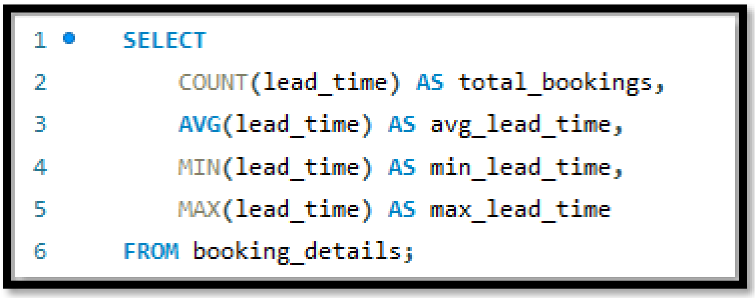
EDA Question’s Conclusion

1. Understand the distribution of arrival dates, including the most common arrival days and summary statistics for lead times.



```sql
SELECT
    STR_TO_DATE(
        CONCAT(arrival_date_day_of_month, ' ', arrival_date_month, ' ', arrival_date_year),
        '%d %M %Y'
    ) AS arrival_date,
    COUNT(*) AS arrival_count
FROM booking_details
GROUP BY arrival_date
ORDER BY arrival_count DESC
LIMIT 20;
```



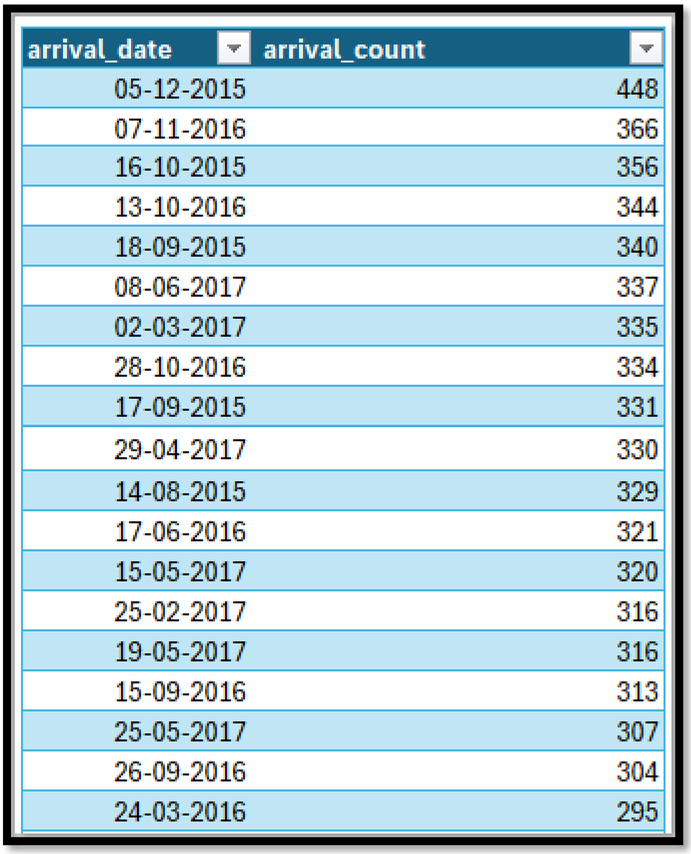
```sql
SELECT
    COUNT(lead_time) AS total_bookings,
    AVG(lead_time) AS avg_lead_time,
    MIN(lead_time) AS min_lead_time,
    MAX(lead_time) AS max_lead_time
FROM booking_details;
```

Query Explanation 1:

1. STR_TO_DATE Conversion:
   - The **arrival_date_day_of_month**, **arrival_date_month**, and **arrival_date_year** columns are combined into a single string using CONCAT().
   - The resulting string (e.g., "15 March 2023") is converted into a proper DATE format using STR_TO_DATE('%d %M %Y').
2. Column Selection:
   - **arrival_date**: The properly formatted date of arrival.
   - **arrival_count**: The number of bookings for each arrival date **(COUNT(*)).**
3. Data Source:
   - The query retrieves data from the **booking_details** table.
4. Grouping & Counting:
   - **GROUP BY arrival_date**: Groups records by arrival date to count the number of bookings per date.
5. Sorting:
   - **ORDER BY arrival_count DESC**: Orders results in descending order of booking count.
6. Limiting Results:
   - **LIMIT 20**: Retrieves only the top 20 most frequent arrival dates.

Purpose:
This query is useful for analyzing booking trends by identifying peak arrival dates based on past booking data.

Query Result1 :



arrival_date	arrival_count
05-12-2015	448
07-11-2016	366
16-10-2015	356
13-10-2016	344
18-09-2015	340
08-06-2017	337
02-03-2017	335
28-10-2016	334
17-09-2015	331
29-04-2017	330
14-08-2015	329
17-06-2016	321
15-05-2017	320
25-02-2017	316
19-05-2017	316
15-09-2016	313
25-05-2017	307
26-09-2016	304
24-03-2016	295

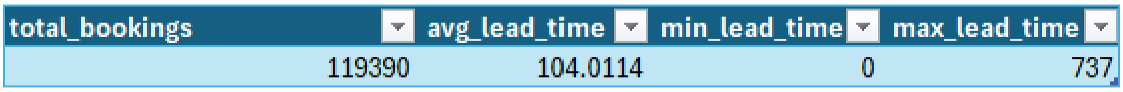
Query Explanation 2:

This SQL query retrieves statistical insights from the booking_details table regarding the lead_time column. Here's what each part does:

- **COUNT(lead_time) AS total_bookings:** Counts the total number of bookings.
- **AVG(lead_time) AS avg_lead_time:** Calculates the average lead time.
- **MIN(lead_time) AS min_lead_time:** Finds the minimum lead time.
- **MAX(lead_time) AS max_lead_time:** Finds the maximum lead time.

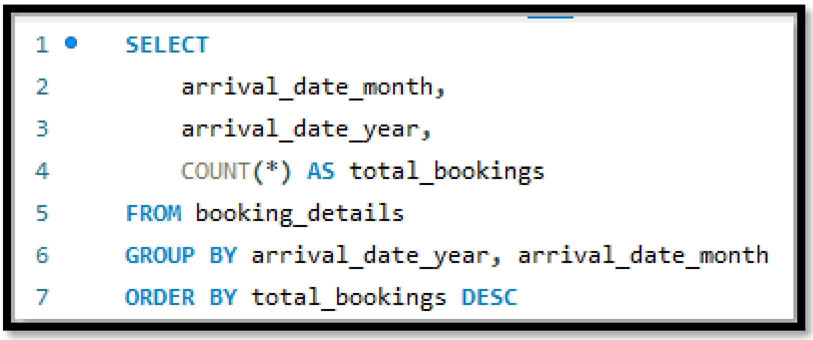
Purpose:
This query helps in understanding booking trends by analyzing how much time in advance customers book their reservations.

Query Result 2 :



total_bookings	avg_lead_time	min_lead_time	max_lead_time
119390	104.0114	0	737

2. Identify peak booking months and analyze reasons for spikes in bookings, including holidays or events.



```sql
SELECT
    arrival_date_month,
    arrival_date_year,
    COUNT(*) AS total_bookings
FROM booking_details
GROUP BY arrival_date_year, arrival_date_month
ORDER BY total_bookings DESC
```

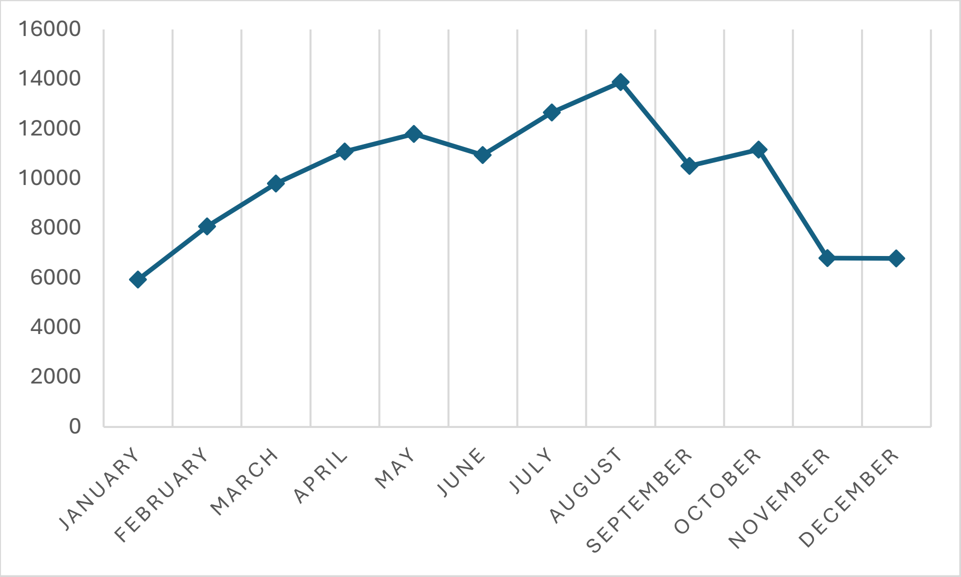
Query Explanation :

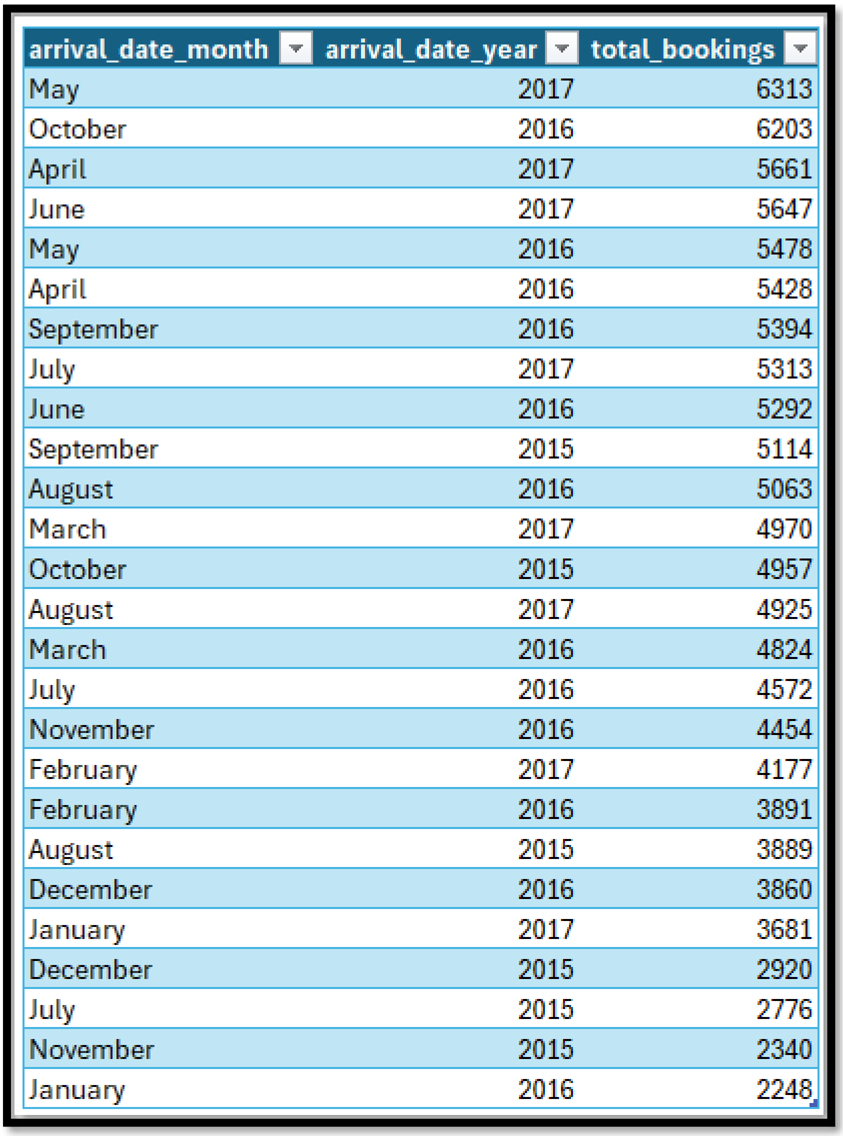
This SQL query retrieves the number of bookings per month and year from the booking_details table. Here's what each part does:

- **arrival_date_month, arrival_date_year:** Selects the month and year of arrival.
- **COUNT(*) AS total_bookings:** Counts the total number of bookings for each month and year.
- **FROM booking_details:** Specifies the table from which data is retrieved.
- **GROUP BY arrival_date_year, arrival_date_month:** Groups the results by year and month to aggregate booking counts.
- **ORDER BY total_bookings DESC:** Sorts the results in descending order based on the number of bookings.

Purpose:
This query helps in analyzing seasonal trends by identifying which months and years had the highest number of bookings.

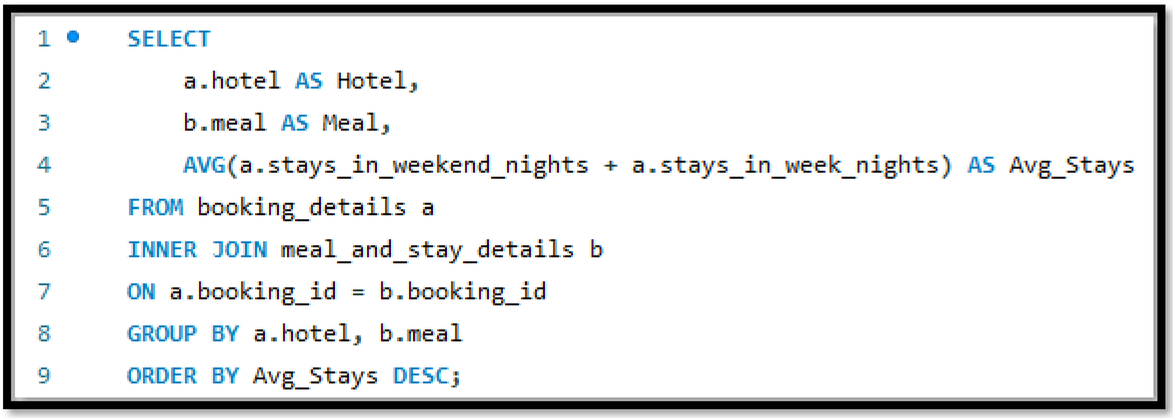
Query Result :





arrival_date_month	arrival_date_year	total_bookings
May	2017	6313
October	2016	6203
April	2017	5661
June	2017	5647
May	2016	5478
April	2016	5428
September	2016	5394
July	2017	5313
June	2016	5292
September	2015	5114
August	2016	5063
March	2017	4970
October	2015	4957
August	2017	4925
March	2016	4824
July	2016	4572
November	2016	4454
February	2017	4177
February	2016	3891
August	2015	3889
December	2016	3860
January	2017	3681
December	2015	2920
July	2015	2776
November	2015	2340
January	2016	2248

3. **Calculate the average length of stays for different hotel types and explore variations by meal plans.**



```sql
SELECT
    a.hotel AS Hotel,
    b.meal AS Meal,
    AVG(a.stays_in_weekend_nights + a.stays_in_week_nights) AS Avg_Stays
FROM booking_details a
INNER JOIN meal_and_stay_details b
ON a.booking_id = b.booking_id
GROUP BY a.hotel, b.meal
ORDER BY Avg_Stays DESC;
```

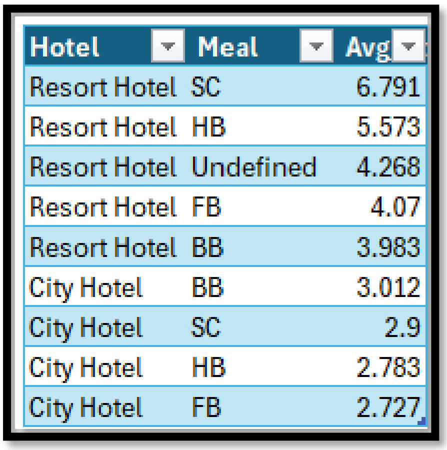
Query Explanation :

This SQL query calculates the average length of stay (in nights) for different meal types across hotels. Here's what each part does:

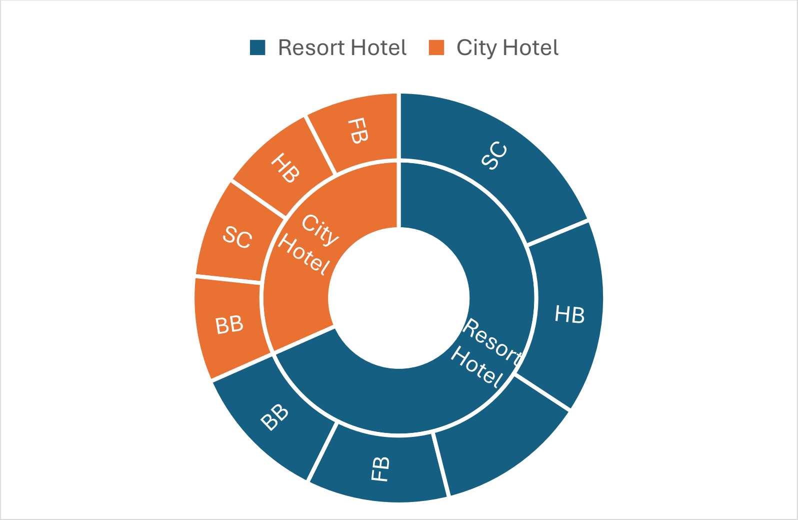
- **a.hotel AS Hotel:** Selects the hotel name from the `booking_details` table (aliased as `a`).
- **b.meal AS Meal:** Selects the meal type from the `meal_and_stay_details` table (aliased as `b`).
- **AVG(a.stays_in_weekend_nights + a.stays_in_week_nights) AS Avg_Stays**: Computes the average total nights stayed (weekend + weekday) for each hotel-meal combination.
- **FROM booking_details a INNER JOIN meal_and_stay_details b ON a.booking_id = b.booking_id:** Joins the two tables using the `booking_id` to correlate bookings with meal types.
- **GROUP BY a.hotel, b.meal**: Groups results by hotel and meal type to aggregate average stays.
- **ORDER BY Avg_Stays DESC:** Sorts results by average stay duration (longest to shortest).

Purpose:
This query helps identify which meal types are associated with longer guest stays at different hotels. The insights can guide hoteliers in optimizing meal offerings to potentially increase occupancy duration (e.g., promoting meal packages tied to extended stays).

Query Result :



Hotel	Meal	Avg
Resort Hotel	SC	6.791
Resort Hotel	HB	5.573
Resort Hotel	Undefined	4.268
Resort Hotel	FB	4.07
Resort Hotel	BB	3.983
City Hotel	BB	3.012
City Hotel	SC	2.9
City Hotel	HB	2.783
City Hotel	FB	2.727



4. Analyze how booking patterns have evolved over the years, including yearoveryear changes in bookings and cancellations.

Query Explanation :

This SQL query analyzes yearly booking trends and cancellation rates, with a focus on year-over-year (YOY) comparisons. Here's what each part does:

Common Table Expression (CTE) `test`:

- **arrival_date_year "Year":** Extracts the year from the arrival date and labels it as "Year".
- **count(Booking_id) "Total_Booking"**: Counts the total number of bookings per year.
- **sum(is_canceled) "cancel":** Sums cancellations (assumes `is_canceled` is a binary flag where 1 = canceled).
- **FROM booking_details GROUP BY arrival_date_year:** Aggregates data by year.

Main Query:

- **Year, Total_Booking:** Displays the year and total bookings from the CTE.
- **Total_Booking - LAG(Total_Booking) OVER (ORDER BY year) AS YOY_Booking_Difference:**

- Uses the LAG() window function to compare the current year's bookings with the previous year's.
- Calculates the absolute change in bookings YOY.

- **cancel:** Shows total cancellations per year.
- **cancel - LAG(cancel) OVER (ORDER BY Year) AS YOY_Cancellation_Difference:**
    - Computes the absolute change in cancellations YOY (similar to bookings).

Execution Flow:

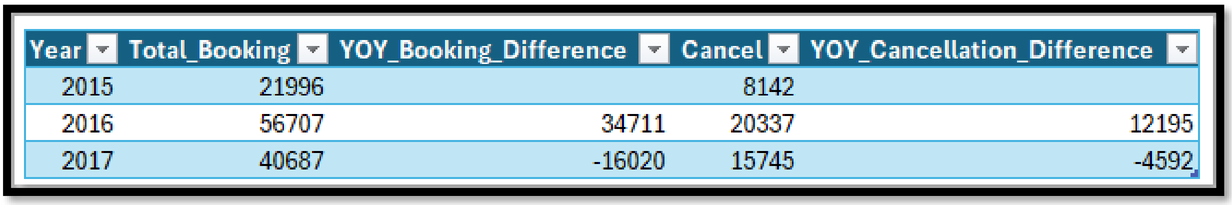
1. The CTE (` test `) aggregates raw data by year.
2. The main query adds YOY comparisons using ` LAG() ` to track trends.

Purpose:

This query helps identify:

- Growth/decline in booking volume over time.
- Whether cancellation rates are improving or worsening YOY.
- Correlations between bookings and cancellations (e.g., higher bookings may lead to higher absolute cancellations).

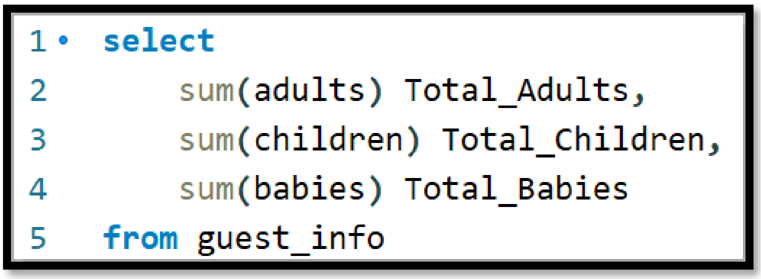
Query Result :



Year	Total_Booking	YOY_Booking_Difference	Cancel	YOY_Cancellation_Difference
2015	21996		8142	
2016	56707	34711	20337	12195
2017	40687	-16020	15745	-4592

---

5. Understand the distribution of the number of adults, children, and babies and identify any outliers.



```sql
select
	sum(adults) Total_Adults,
	sum(children) Total_Children,
	sum(babies) Total_Babies
from guest_info
```

Query Explanation 1 :

This SQL query calculates the total number of adults, children, and babies from the guest_info table. Here's what each part does:

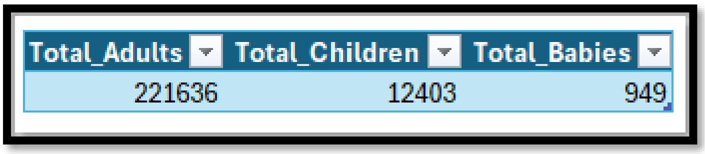
- **sum(adults) Total_Adults:** Adds up all values in the adults column and names the result Total_Adults
- **sum(children) Total_Children:** Adds up all values in the children column and names the result Total_Children
- **sum(babies) Total_Babies:** Adds up all values in the babies column and names the result Total_Babies
- **from guest_info**: Specifies the table containing the guest information

Purpose:
This query provides a simple count of different guest types (adults, children, babies) across all records in the database. It can be used to:

- Understand overall guest demographics
- Plan for facility needs (like cribs or child-friendly amenities)
- Track changes in guest composition over time when run periodically

The results show the absolute numbers of each guest category, which is useful for high-level resource planning and trend analysis. For more detailed insights, you might want to add grouping by date, hotel, or other categories.

Query Result 1 :



Total_Adults	Total_Children	Total_Babies
221636	12403	949

```
WITH Ranked AS (
  SELECT
    adults,
    children,
    babies,
    ROW_NUMBER() OVER (ORDER BY adults) AS adult_rank,
    ROW_NUMBER() OVER (ORDER BY children) AS children_rank,
    ROW_NUMBER() OVER (ORDER BY babies) AS babies_rank,
    COUNT(*) OVER () AS total_count
  FROM guest_info
), Quartiles AS (
  SELECT
    adults AS q1_adults,
    LEAD(adults) OVER () AS q3_adults,
    children AS q1_children,
    LEAD(children) OVER () AS q3_children,
    babies AS q1_babies,
    LEAD(babies) OVER () AS q3_babies
  FROM Ranked
  WHERE adult_rank = FLOOR(total_count * 0.25)
   OR adult_rank = FLOOR(total_count * 0.75)
)
SELECT
  g.*,
  CASE
    WHEN g.adults < (q.q1_adults - 1.5 * (q.q3_adults - q.q1_adults))
     OR g.adults > (q.q3_adults + 1.5 * (q.q3_adults - q.q1_adults))
    THEN 'Outlier' ELSE 'Normal'
  END AS adults_outlier,
  CASE
    WHEN g.babies < (q.q1_babies - 1.5 * (q.q3_babies - q.q1_babies))
     OR g.babies > (q.q3_babies + 1.5 * (q.q3_babies - q.q1_babies))
    THEN 'Outlier' ELSE 'Normal'
  END AS babies_outlier
FROM guest_info g
CROSS JOIN Quartiles q
WHERE
  (g.adults < (q.q1_adults - 1.5 * (q.q3_adults - q.q1_adults))
  OR g.adults > (q.q3_adults + 1.5 * (q.q3_adults - q.q1_adults)))
                                    OR
            (g.children < (q.q1_children - 1.5 * (q.q3_children - q.q1_children))
          OR g.children > (q.q3_children + 1.5 * (q.q3_children - q.q1_children)))
                                    OR
               (g.babies < (q.q1_babies - 1.5 * (q.q3_babies - q.q1_babies))
             OR g.babies > (q.q3_babies + 1.5 * (q.q3_babies - q.q1_babies)));
```

Query Explanation:

This query identifies outlier values in guest demographics (adults, children, babies) using statistical quartile analysis. Here's how it works:

1. **First CTE (Ranked):**

- Creates row numbers for each guest category (adults, children, babies) when sorted

- Calculates total record count for quartile position reference

2. **Second CTE (Quartiles):**

- Extracts the 1st quartile (25th percentile) and 3rd quartile (75th percentile) values:
 * q1_adults/q3_adults for adult counts
 * q1_children/q3_children for child counts
 * q1_babies/q3_babies for baby counts
- Uses LEAD() to pair the quartile values together

3. **Main Query:**

- Joins the original data with the quartile calculations
- Flags records as outliers using the 1.5×IQR rule:
 -> Any value below Q1 - 1.5×IQR OR above Q3 + 1.5×IQR is an outlier
 -> IQR = Q3 - Q1 (interquartile range)
- Returns only records with at least one outlier value
- Includes classification columns showing which categories contain outliers

Purpose:

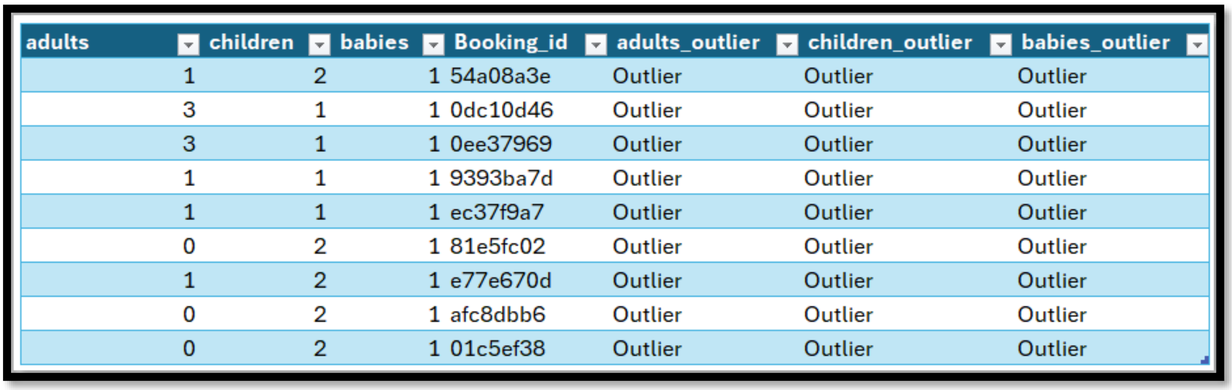
This analysis helps:

- Identify unusual booking patterns (extremely large/small groups)
- Detect potential data quality issues
- Find exceptional cases that might need special handling
- Understand the normal distribution of group sizes

The 1.5×IQR rule is a common statistical method for outlier detection that works well for many distributions. The query could be enhanced by:

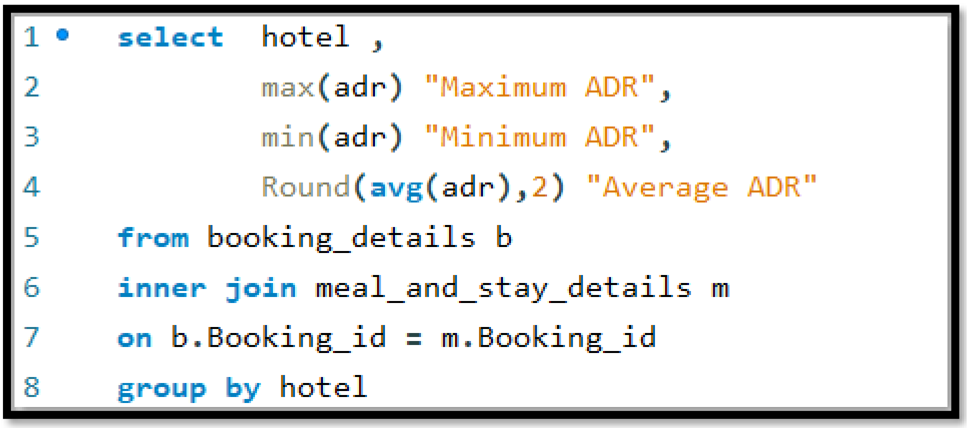
1. Adding date filters to analyze trends
2. Including hotel/category information
3. Calculating percentage of outliers
4. Adding visualization-friendly output formats

Query Result 2 :



adults	children	babies	Booking_id	adults_outlier	children_outlier	babies_outlier
1	2	1	54a08a3e	Outlier	Outlier	Outlier
3	1	1	0dc10d46	Outlier	Outlier	Outlier
3	1	1	0ee37969	Outlier	Outlier	Outlier
1	1	1	9393ba7d	Outlier	Outlier	Outlier
1	1	1	ec37f9a7	Outlier	Outlier	Outlier
0	2	1	81e5fc02	Outlier	Outlier	Outlier
1	2	1	e77e670d	Outlier	Outlier	Outlier
0	2	1	afc8dbb6	Outlier	Outlier	Outlier
0	2	1	01c5ef38	Outlier	Outlier	Outlier

6. **Calculate summary statistics for ADR and explore differences between Resort Hotel and City Hotel bookings.**



```
1 •  select  hotel ,
2            max(adr) "Maximum ADR",
3            min(adr) "Minimum ADR",
4            Round(avg(adr),2) "Average ADR"
5    from booking_details b
6    inner join meal_and_stay_details m
7    on b.Booking_id = m.Booking_id
8    group by hotel
```

Query Explanation:

This query analyzes hotel pricing data by calculating key metrics for the Average Daily Rate (ADR) across different hotels. Here's the breakdown:

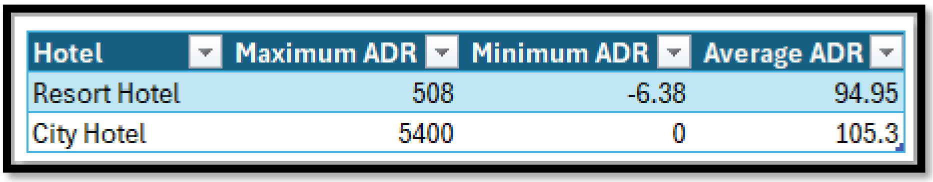
1. Line 1: Selects the hotel column to group results by property
2. Line 2: Calculates the maximum ADR (highest daily rate) for each hotel
3. Line 3: Calculates the minimum ADR (lowest daily rate) for each hotel
4. Line 4: Computes the average ADR rounded to 2 decimal places
5. Lines 5-7: Joins the booking_details table (aliased as b) with meal_and_stay_details (aliased as m) using Booking_id as the common key
6. Line 8: Groups all calculations by hotel to show separate metrics for each property

Purpose:

This query provides valuable insights into hotel pricing strategies by showing:

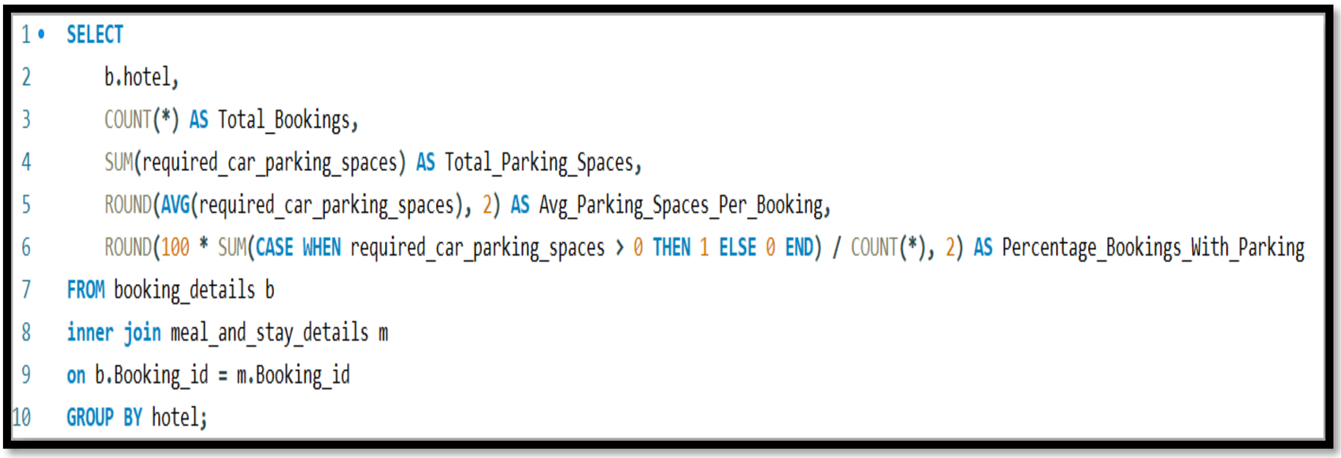
- The full price range (minimum to maximum) offered by each hotel
- The typical price point (average ADR) for each property
- Potential pricing consistency or variability across properties

Query Result :



Hotel	Maximum ADR	Minimum ADR	Average ADR
Resort Hotel	508	-6.38	94.95
City Hotel	5400	0	105.3

7. **Analyze the distribution of required car parking spaces for each hotel type and determine if one type attracts more guests with cars.**



```sql
SELECT
    b.hotel,
    COUNT(*) AS Total_Bookings,
    SUM(required_car_parking_spaces) AS Total_Parking_Spaces,
    ROUND(AVG(required_car_parking_spaces), 2) AS Avg_Parking_Spaces_Per_Booking,
    ROUND(100 * SUM(CASE WHEN required_car_parking_spaces > 0 THEN 1 ELSE 0 END) / COUNT(*), 2) AS Percentage_Bookings_With_Parking
FROM booking_details b
inner join meal_and_stay_details m
on b.Booking_id = m.Booking_id
GROUP BY hotel;
```

Query Explanation:

This query analyzes parking space demand patterns across different hotels by examining booking data. Here's what each part does:

1. Column Selection:

- **`b.hotel`:** Groups results by hotel property
- **`COUNT(*) AS Total_Bookings`:** Counts all bookings for each hotel
- **`SUM(required_car_parking_spaces)`:** Calculates total parking spaces requested
- **`ROUND(AVG(required_car_parking_spaces), 2**)`: Shows average parking spaces per booking (rounded to 2 decimals)
- **Complex percentage calculation:** `ROUND(100 * SUM(CASE WHEN required_car_parking_spaces > 0 THEN 1 ELSE 0 END) / COUNT(*), 2)`

This computes the percentage of bookings that requested at least one parking space

2. Data Sources:

- Joins `booking_details` (aliased as b) with `meal_and_stay_details` (aliased as m)
- Uses `Booking_id` as the join key to connect booking records with stay details

3. Aggregation:

- Groups all metrics by hotel property
- All calculations are performed separately for each hotel

Purpose:

This query provides valuable insights for:

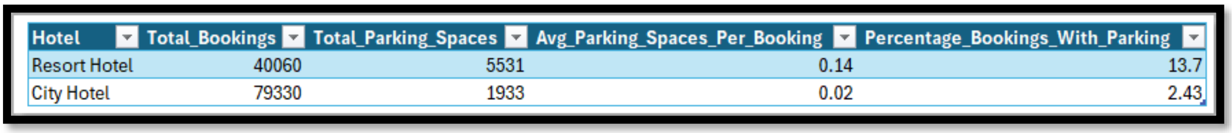
1. **Parking Facility Planning:**
   - Shows total parking demand at each property
   - Reveals what percentage of guests require parking
   - Indicates average parking needs per booking

2. **Operational Decisions:**
   - Helps determine if current parking capacity is adequate
   - Identifies properties with higher parking demands
   - Can inform pricing strategies for parking spaces

3. **Guest Behavior Analysis:**
   - Shows how frequently guests arrive by car
   - May indicate differences in guest demographics or transportation patterns between properties

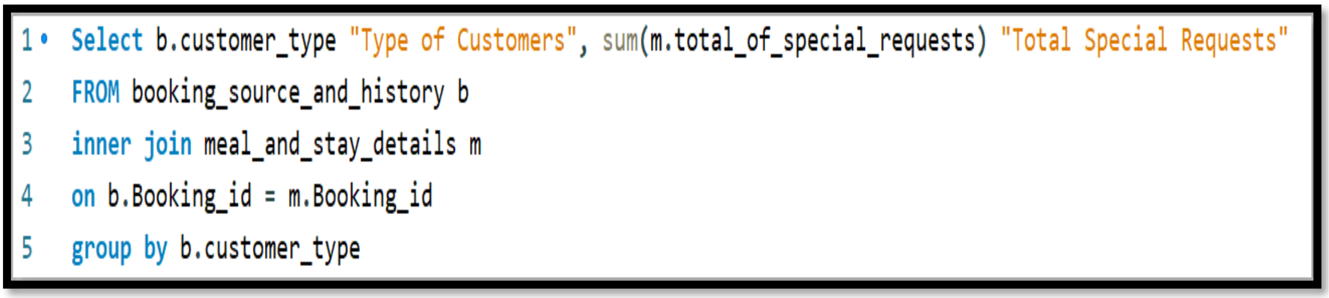
The percentage calculation is particularly useful as it shows parking space utilization rates independent of the total number of bookings, allowing for fair comparison between large and small properties.

Query Result :



Hotel	Total_Bookings	Total_Parking_Spaces	Avg_Parking_Spaces_Per_Booking	Percentage_Bookings_With_Parking
Resort Hotel	40060	5531	0.14	13.7
City Hotel	79330	1933	0.02	2.43

8. **Compare the total number of special requests made by different customer types (e.g., Transient, Group) and identify which customer type makes more requests.**



```sql
Select b.customer_type "Type of Customers", sum(m.total_of_special_requests) "Total Special Requests"
FROM booking_source_and_history b
inner join meal_and_stay_details m
on b.Booking_id = m.Booking_id
group by b.customer_type
```

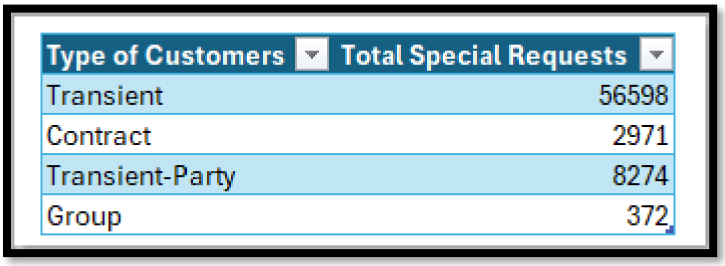
Query-Explanation:
This query analyzes the relationship between customer types and special requests made during bookings. Here's the breakdown:

1. **Line 1 (SELECT):**
    - **b.customer_type "Type of Customers":** Retrieves the customer type and labels it clearly
    - **sum(m.total_of_special_requests) "Total Special Requests":** Calculates the sum of all special requests made by each customer type
2. **Lines 2-4 (FROM and JOIN):**
    - Combines data from two tables:
        - **booking_source_and_history (aliased as 'b')** - contains customer and booking information
        - **meal_and_stay_details (aliased as 'm')** - contains details about special requests
    - Joins them using Booking_id as the common identifier
3. **Line 5 (GROUP BY):**
    - Groups the results by customer type to show aggregate special request totals for each category

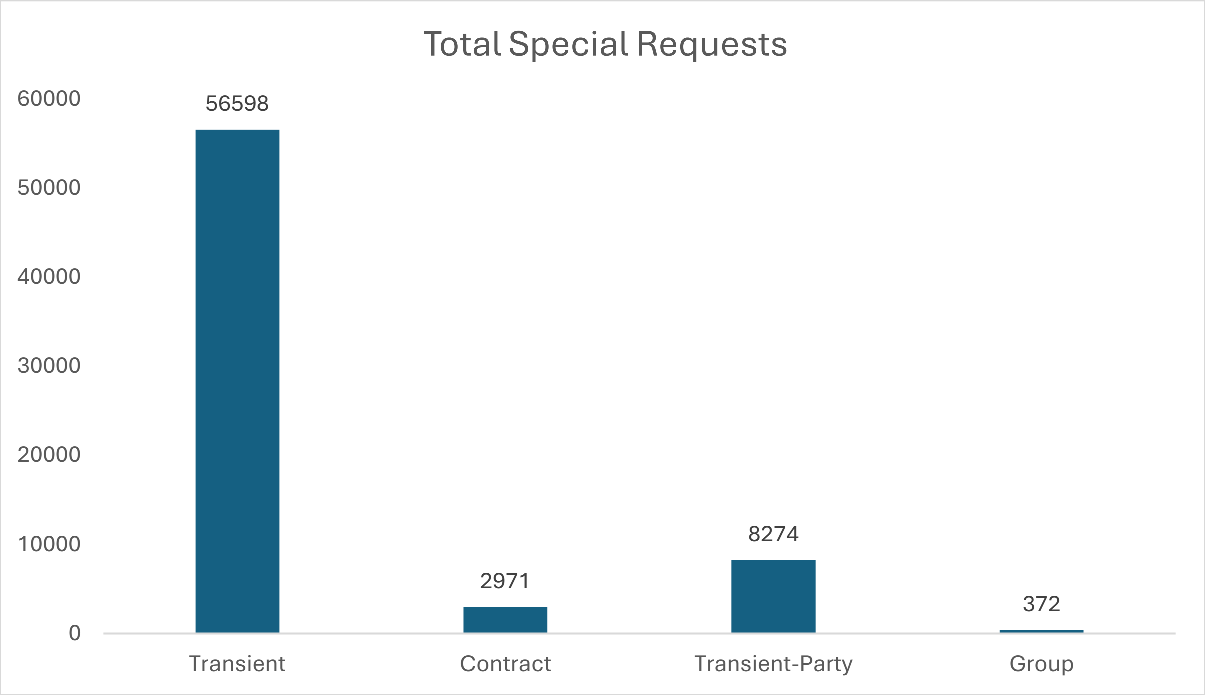
Purpose:
This query helps understand:

- Which customer types make the most special requests
- How customer preferences vary by category
- Potential patterns in customer behavior

Query Result :

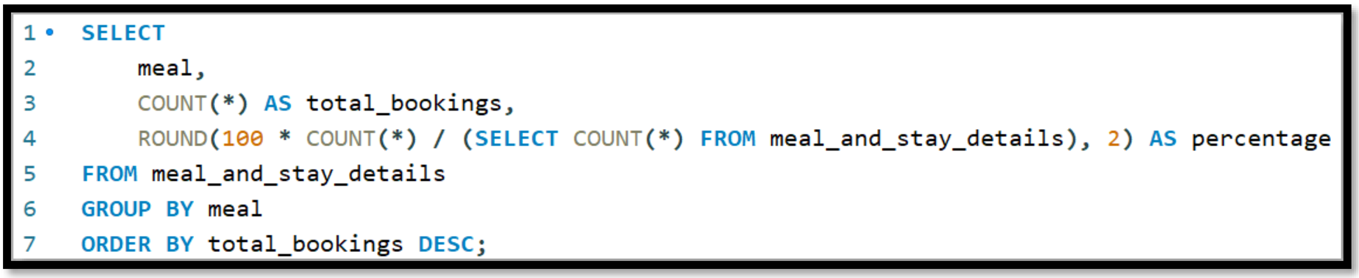


Type of Customers	Total Special Requests
Transient	56598
Contract	2971
Transient-Party	8274
Group	372



---

9. Understand the distribution of meal plans (e.g., BB, HB, FB, SC) and identify any patterns or preferences.



```
1    SELECT
2        meal,
3        COUNT(*) AS total_bookings,
4        ROUND(100 * COUNT(*) / (SELECT COUNT(*) FROM meal_and_stay_details), 2) AS percentage
5    FROM meal_and_stay_details
6    GROUP BY meal
7    ORDER BY total_bookings DESC;
```

Query-Explanation:
This query analyzes the distribution of meal preferences across all bookings. Here's what each part does:

1. **Line 1-2 (SELECT meal):**
   - Selects the meal type from the meal_and_stay_details table
2. **Line 3 (COUNT(*) AS total_bookings):**
   - Counts the total number of bookings for each meal type
3. **Line 4 (Percentage calculation):**
   - Calculates the percentage share of each meal type
   - Uses a subquery (SELECT COUNT(*) FROM meal_and_stay_details) to get the total bookings count

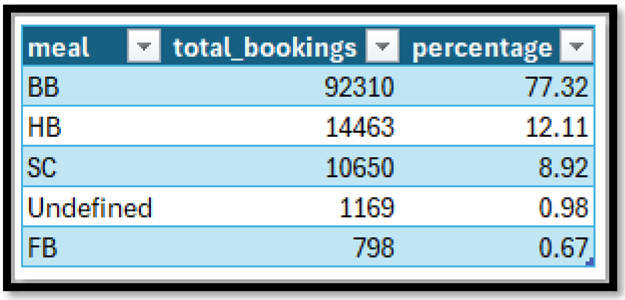
    - Multiplies by 100 and rounds to 2 decimal places for readability

4. **Line 5 (FROM clause):**
    - Specifies the source table (meal_and_stay_details)
5. **Line 6 (GROUP BY meal):**
    - Groups results by meal type to aggregate the counts
6. **Line 7 (ORDER BY):**
    - Sorts results by total bookings in descending order (most popular meals first)

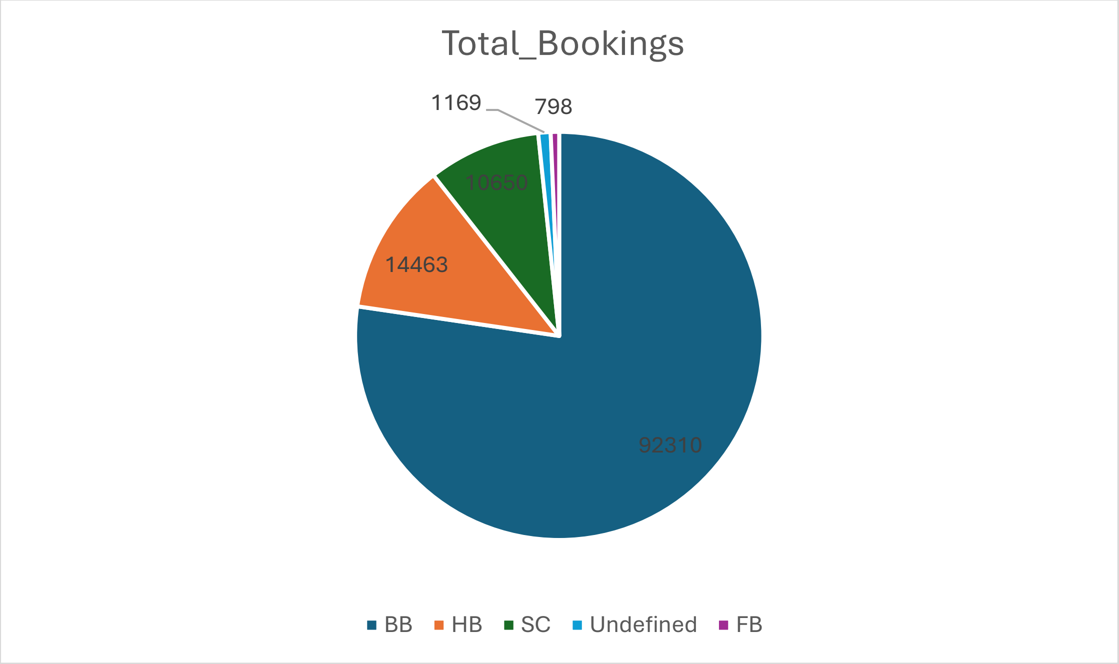
Purpose:
This query helps understand:

- The popularity distribution of different meal options
- Which meal types are most/least preferred by guests
- The relative market share of each meal category

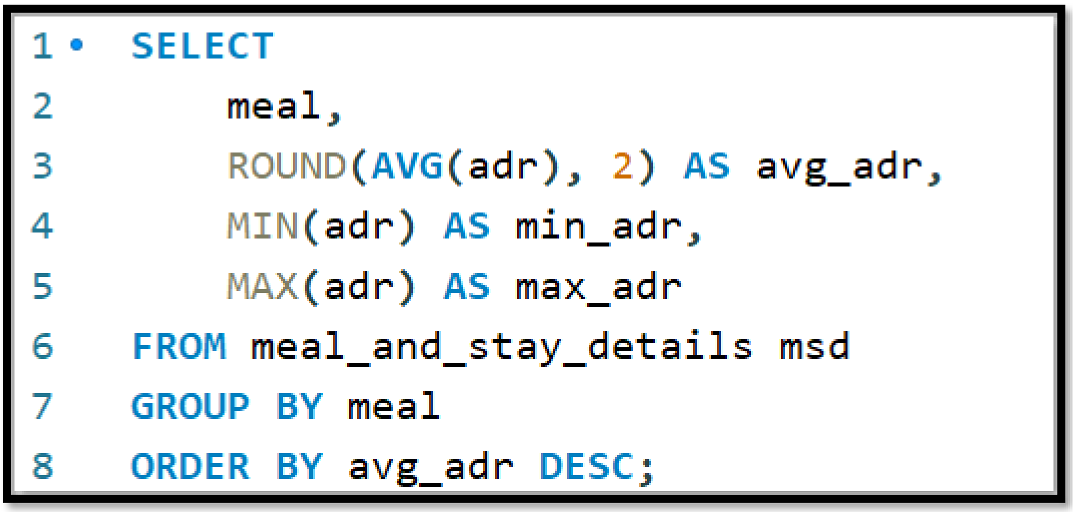
Query Result :



meal	total_bookings	percentage
BB	92310	77.32
HB	14463	12.11
SC	10650	8.92
Undefined	1169	0.98
FB	798	0.67



10. Analyze Average Daily Rates (ADR) by meal plan type to identify variations in pricing.



```
1 •  SELECT
2        meal,
3        ROUND(AVG(adr), 2) AS avg_adr,
4        MIN(adr) AS min_adr,
5        MAX(adr) AS max_adr
6    FROM meal_and_stay_details msd
7    GROUP BY meal
8    ORDER BY avg_adr DESC;
```

Query Explanation:
This query calculates key metrics for Average Daily Rates (ADR) by meal type. Here's the breakdown:

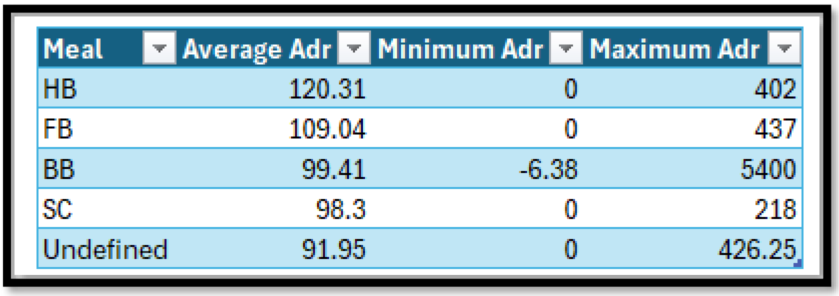
1. **Line 1-2 (SELECT meal):**
   - Selects the meal type from the dataset
2. **Line 3 (ROUND(AVG(adr), 2) AS avg_adr):**
   - Calculates the average ADR for each meal type
   - Rounds the result to 2 decimal places for cleaner presentation
3. **Line 4 (MIN(adr) AS min_adr):**
   - Finds the lowest ADR for each meal type
4. **Line 5 (MAX(adr) AS max_adr):**
   - Identifies the highest ADR for each meal type
5. **Line 6 (FROM meal_and_stay_details msd):**
   - Uses the meal_and_stay_details table (aliased as 'msd')
6. **Line 7 (GROUP BY meal):**
   - Groups all calculations by meal type
7. **Line 8 (ORDER BY avg_adr DESC):**
   - Sorts results by average ADR in descending order (highest-priced meals first)

Purpose:

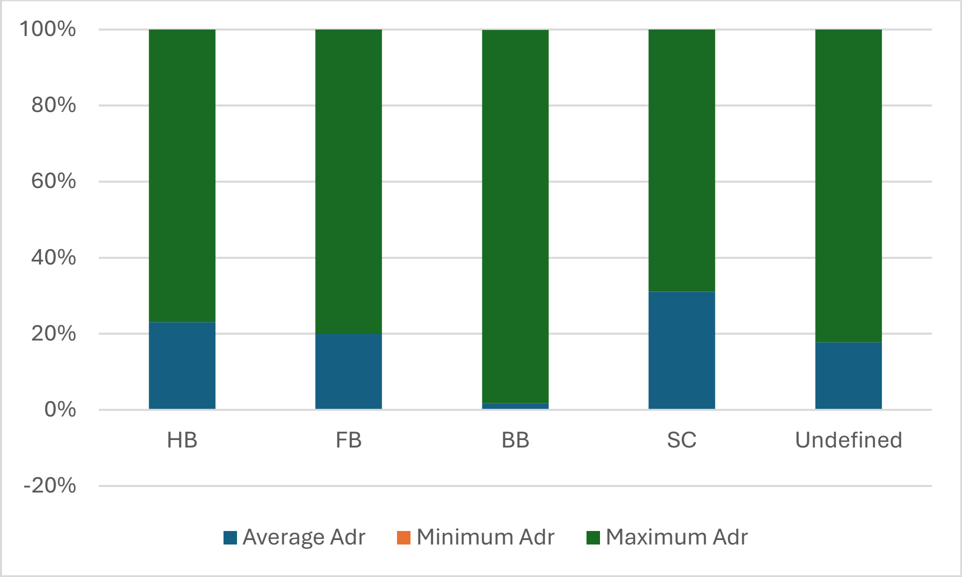
This analysis reveals:

- Which meal types command premium pricing
- The price range for each meal category
- How meal options are positioned in the pricing structure

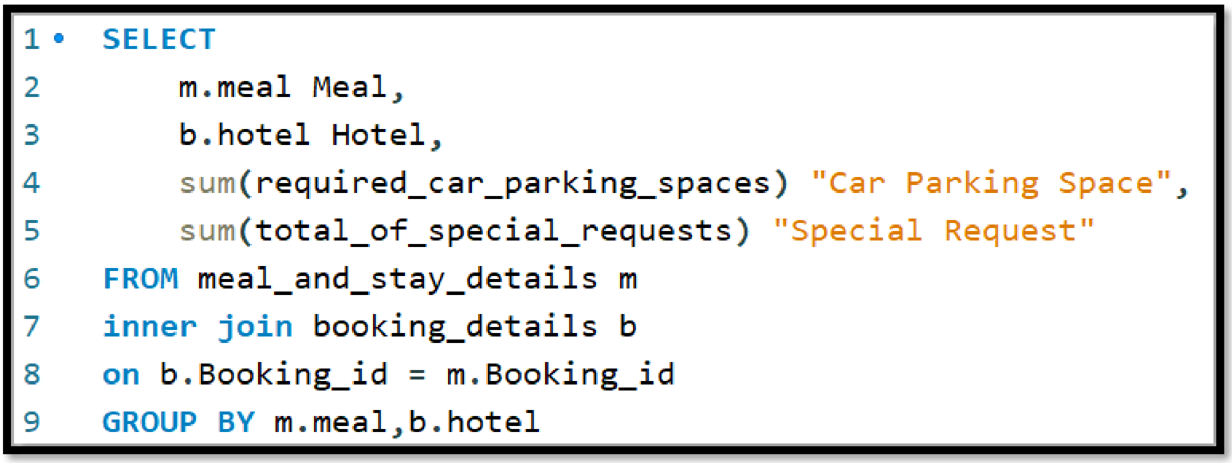
Query Result :



Meal	Average Adr	Minimum Adr	Maximum Adr
HB	120.31	0	402
FB	109.04	0	437
BB	99.41	-6.38	5400
SC	98.3	0	218
Undefined	91.95	0	426.25



11. **Investigate the distribution of required car parking spaces and special requests by hotel type and meal plan. Compare the distribution of meal plans among different customer types (e.g., Transient, Group) to identify preferences.**



```sql
SELECT
    m.meal Meal,
    b.hotel Hotel,
    sum(required_car_parking_spaces) "Car Parking Space",
    sum(total_of_special_requests) "Special Request"
FROM meal_and_stay_details m
inner join booking_details b
on b.Booking_id = m.Booking_id
GROUP BY m.meal,b.hotel
```

Query Explanation:

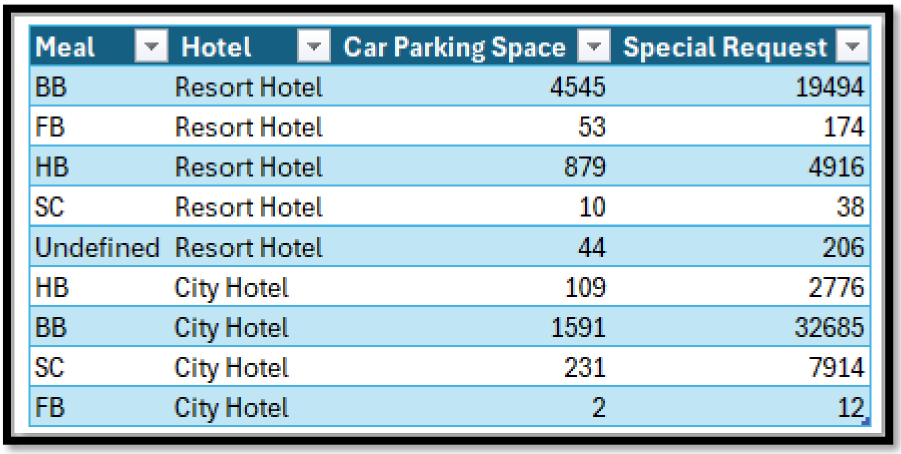
This SQL query analyzes the relationship between meal types, hotels, and their associated parking and special request needs. Here's the breakdown:

1. **Columns Selected:**
    - **m.meal_Meal**: The type of meal from the meal_and_stay_details table
    - **b.hotel_Hotel:** The hotel name from the booking_details table
    - **sum(required_car_parking_spaces):** Total parking spaces requested
    - **sum(total_of_special_requests):** Total special requests made
2. **Tables Used:**
    - meal_and_stay_details (aliased as m)
    - booking_details (aliased as b)
3. **Join Operation:**
    - Inner join on Booking_id to connect meal details with booking records
4. **Grouping:**
    - Results are grouped by both meal type and hotel

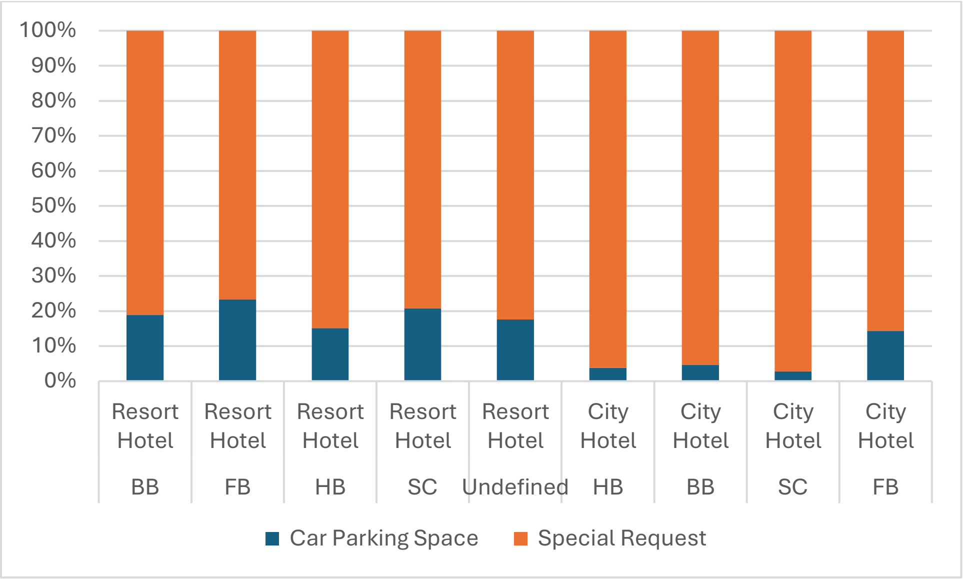
Purpose:
This query provides a breakdown of car parking space requirements and special requests by meal type and hotel. It helps analyze:

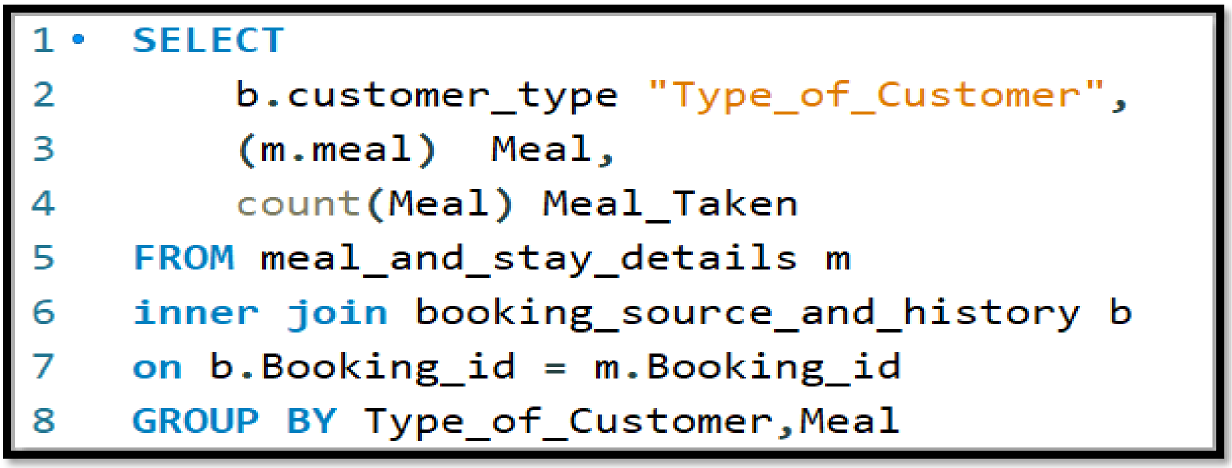
- Parking demand patterns across different meal options and hotel properties
- Special request frequency relative to meal types and hotels

Query Result :



Meal	Hotel	Car Parking Space	Special Request
BB	Resort Hotel	4545	19494
FB	Resort Hotel	53	174
HB	Resort Hotel	879	4916
SC	Resort Hotel	10	38
Undefined	Resort Hotel	44	206
HB	City Hotel	109	2776
BB	City Hotel	1591	32685
SC	City Hotel	231	7914
FB	City Hotel	2	12





```
SELECT
        b.customer_type "Type_of_Customer",
        (m.meal)  Meal,
        count(Meal) Meal_Taken
FROM meal_and_stay_details m
inner join booking_source_and_history b
on b.Booking_id = m.Booking_id
GROUP BY Type_of_Customer,Meal
```

Second Query Explanation:

This SQL query examines customer meal preferences by customer type. Here's the breakdown:

1. **Columns Selected:**
   - **b.customer_type:** Customer category from booking_source_and_history

   - **m.meal:** Meal type from meal_and_stay_details
   - **count(Meal):** Count of each meal type taken

2. **Tables Used:**
   - meal_and_stay_details (aliased as m)
   - booking_source_and_history (aliased as b)

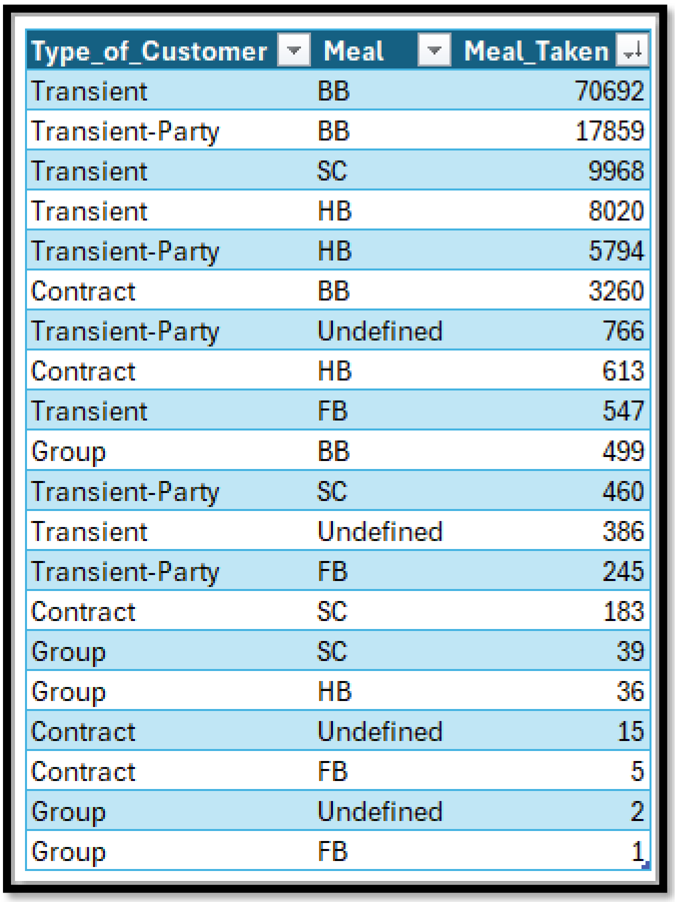
3. **Join Operation:**
   - Inner join on Booking_id to connect customer data with meal choices

4. **Grouping:**
   - Results are grouped by both customer type and meal type

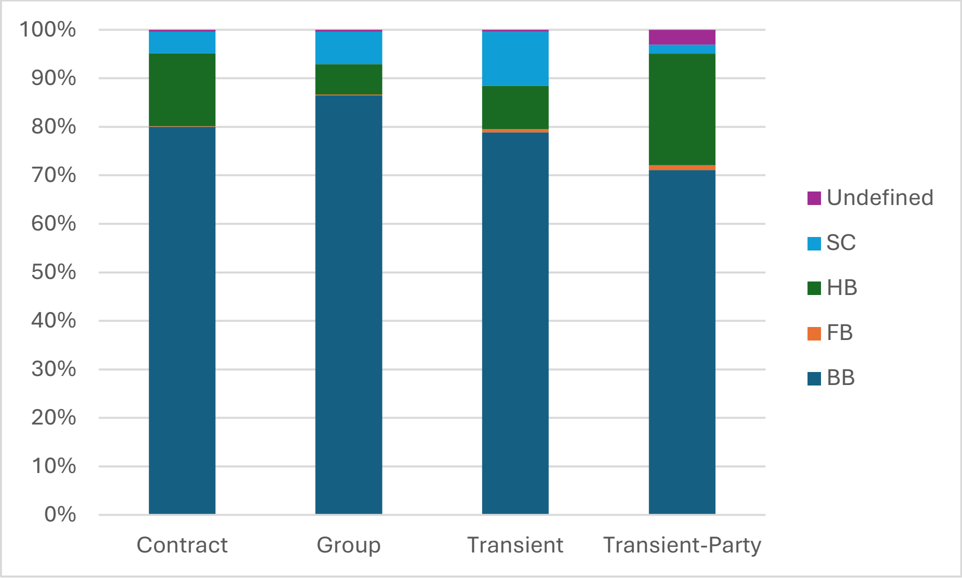
Purpose:
This query shows the distribution of meal preferences across different customer types. It helps understand:

- Which customer segments prefer which meal options
- Popularity of meals among different customer categories

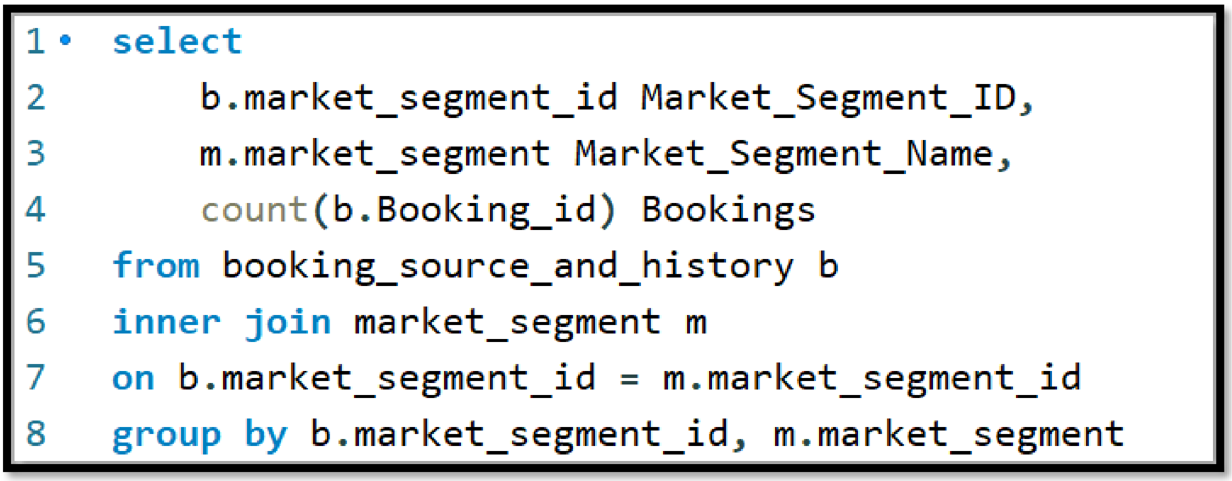
Query Result :



Type_of_Customer	Meal	Meal_Taken
Transient	BB	70692
Transient-Party	BB	17859
Transient	SC	9968
Transient	HB	8020
Transient-Party	HB	5794
Contract	BB	3260
Transient-Party	Undefined	766
Contract	HB	613
Transient	FB	547
Group	BB	499
Transient-Party	SC	460
Transient	Undefined	386
Transient-Party	FB	245
Contract	SC	183
Group	SC	39
Group	HB	36
Contract	Undefined	15
Contract	FB	5
Group	Undefined	2
Group	FB	1



12. **Understand the distribution of bookings across different market segments and calculate summary statistics for lead times within each segment.**



```sql
select
    b.market_segment_id Market_Segment_ID,
    m.market_segment Market_Segment_Name,
    count(b.Booking_id) Bookings
from booking_source_and_history b
inner join market_segment m
on b.market_segment_id = m.market_segment_id
group by b.market_segment_id, m.market_segment
```

Query Explanation:

This SQL query analyzes booking distribution across different market segments. Here's what each part does:

1. **Columns Selected:**
   - **`b.market_segment_id` (aliased as Market_Segment_ID):** The unique identifier for each market segment
   - **`m.market_segment` (aliased as Market_Segment_Name):** The descriptive name of each market segment
   - **`count(b.Booking_id)` (aliased as Bookings):** Counts the number of bookings per segment

2. **Tables Used:**
   - **`booking_source_and_history` (aliased as b):** Contains booking records with segment IDs
   - **`market_segment` (aliased as m):** Contains segment ID-name mappings

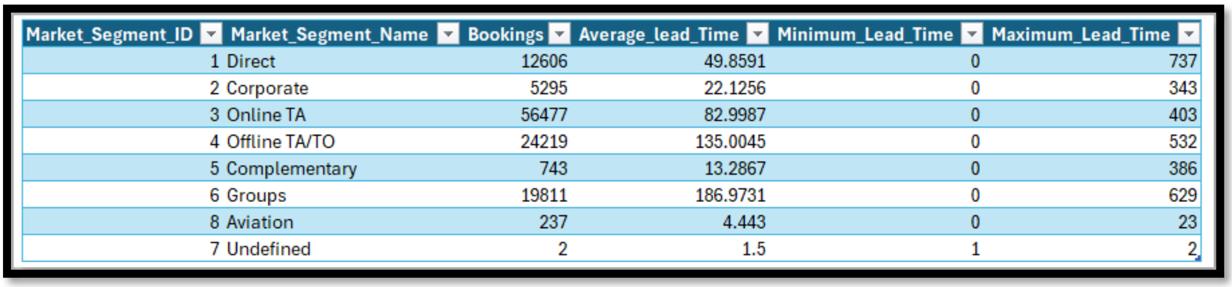
3. **Join Operation:**
   - Inner join on `market_segment_id` connects booking records with segment names

4. **Grouping:**
   - Results are grouped by both segment ID and name to ensure accurate counting

Purpose:

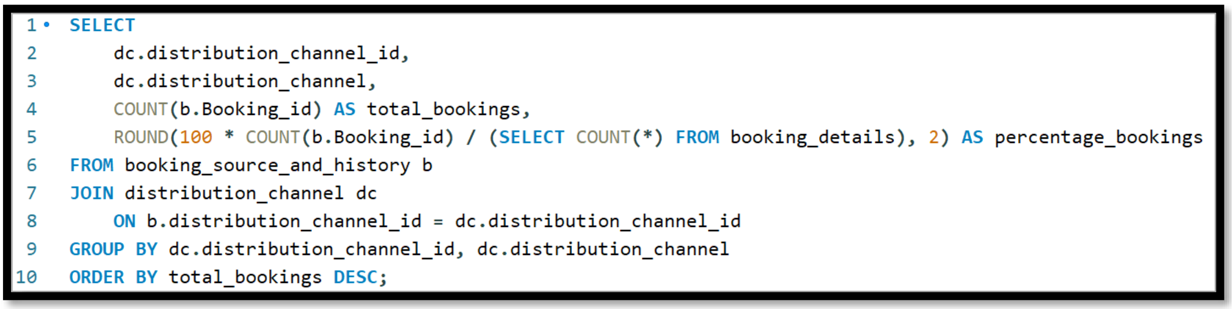
This query provides a clear view of booking volume distribution across different market segments, enabling analysis of which customer segments generate the most business. The results show both the technical ID and human-readable name for each segment along with their respective booking counts.

Query Result :



Market_Segment_ID	Market_Segment_Name	Bookings	Average_lead_Time	Minimum_Lead_Time	Maximum_Lead_Time
1	Direct	12606	49.8591	0	737
2	Corporate	5295	22.1256	0	343
3	Online TA	56477	82.9987	0	403
4	Offline TA/TO	24219	135.0045	0	532
5	Complementary	743	13.2867	0	386
6	Groups	19811	186.9731	0	629
8	Aviation	237	4.443	0	23
7	Undefined	2	1.5	1	2

13. Analyze the distribution of bookings through different booking channels (e.g., online travel agents, direct bookings) and calculate the percentage of bookings through each channel.



```sql
SELECT
    dc.distribution_channel_id,
    dc.distribution_channel,
    COUNT(b.Booking_id) AS total_bookings,
    ROUND(100 * COUNT(b.Booking_id) / (SELECT COUNT(*) FROM booking_details), 2) AS percentage_bookings
FROM booking_source_and_history b
JOIN distribution_channel dc
    ON b.distribution_channel_id = dc.distribution_channel_id
GROUP BY dc.distribution_channel_id, dc.distribution_channel
ORDER BY total_bookings DESC;
```

Query Explanation:

This SQL query analyzes booking distribution across different sales channels. Here's the detailed breakdown:

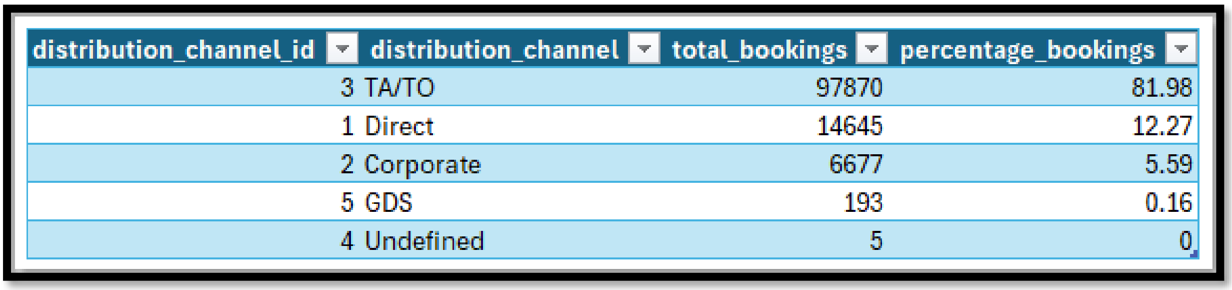
1. **Columns Selected:**
   - **dc.distribution_channel_id:** Unique identifier for each distribution channel
   - **dc.distribution_channel:** Descriptive name of each sales channel
   - **COUNT(b.Booking_id):** Total bookings per channel
   - **Percentage calculation:** Shows each channel's share of total bookings
2. **Tables Used:**
   - booking_source_and_history (source of booking records)
   - distribution_channel (channel reference data)
3. **Key Operations:**
   - **Join:** Links bookings to channel descriptions via distribution_channel_id
   - **Subquery:** (SELECT COUNT(*) FROM booking_details) calculates total bookings for percentage

- **Rounding:** Percentages shown to 2 decimal places
- **Sorting:** Results ordered by booking volume (highest first)

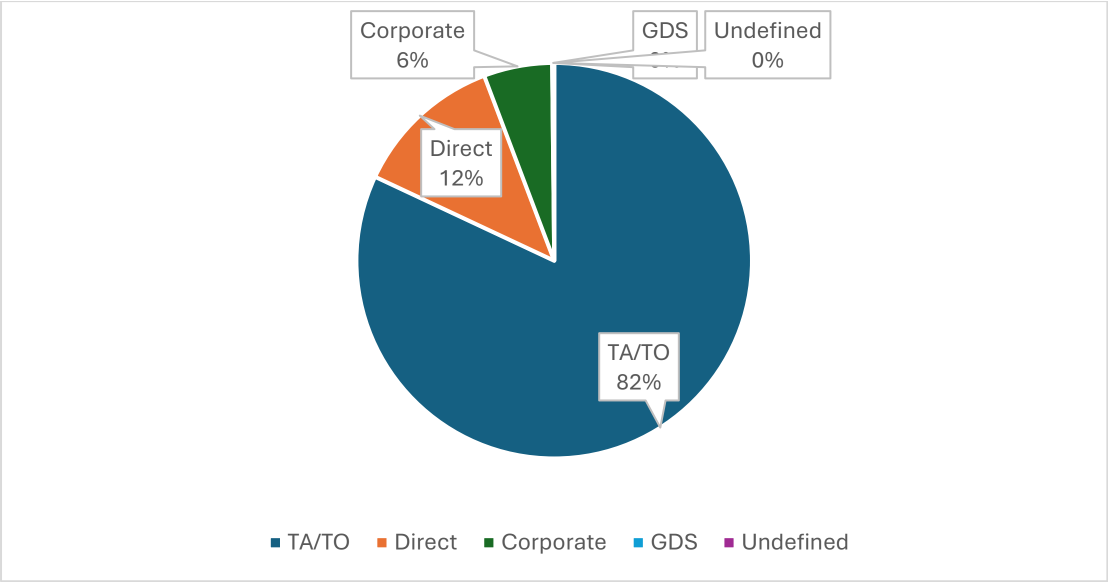
Purpose:
This query provides a comprehensive view of:

1. Which sales channels generate the most bookings
2. The relative market share of each channel
3. The distribution of bookings across all available channels

Query Result :

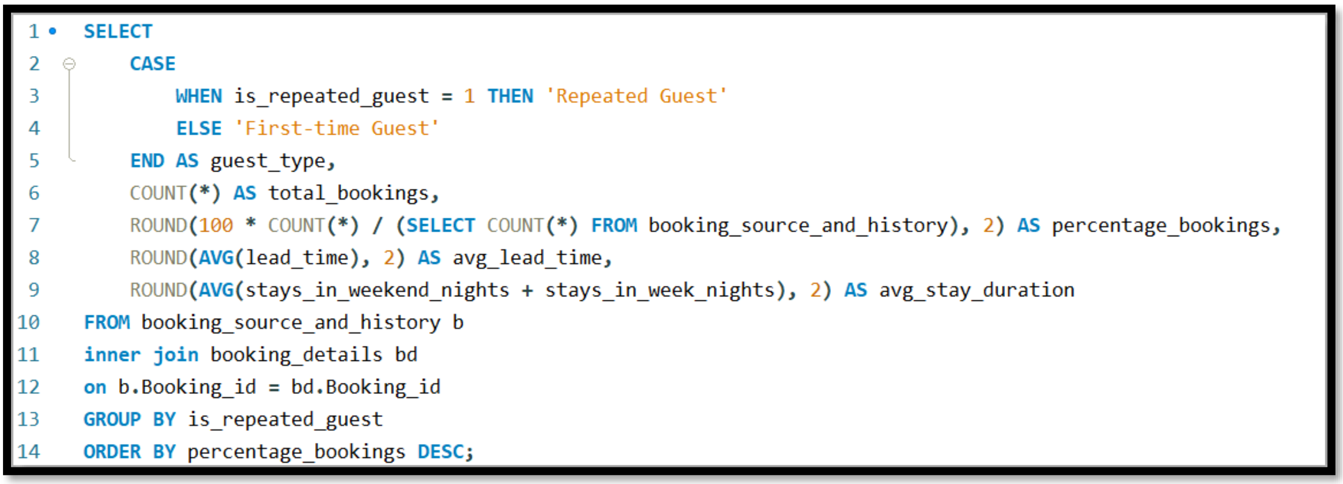


distribution_channel_id	distribution_channel	total_bookings	percentage_bookings
3	TA/TO	97870	81.98
1	Direct	14645	12.27
2	Corporate	6677	5.59
5	GDS	193	0.16
4	Undefined	5	0



---

14. Calculate the proportion of repeated guests and investigate their booking behavior. Identify any patterns or differences in preferences compared to firsttime guests.



```sql
SELECT
    CASE
        WHEN is_repeated_guest = 1 THEN 'Repeated Guest'
        ELSE 'First-time Guest'
    END AS guest_type,
    COUNT(*) AS total_bookings,
    ROUND(100 * COUNT(*) / (SELECT COUNT(*) FROM booking_source_and_history), 2) AS percentage_bookings,
    ROUND(AVG(lead_time), 2) AS avg_lead_time,
    ROUND(AVG(stays_in_weekend_nights + stays_in_week_nights), 2) AS avg_stay_duration
FROM booking_source_and_history b
inner join booking_details bd
on b.Booking_id = bd.Booking_id
GROUP BY is_repeated_guest
ORDER BY percentage_bookings DESC;
```

Query Explanation:
This query compares booking patterns between first-time and repeat guests. The analysis includes:

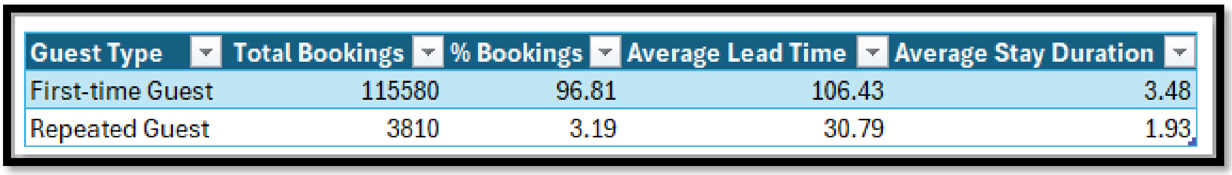
1. **Guest Classification:**
    - Creates a "guest_type" column that labels each booking as either:
        - 'Repeated Guest' (is_repeated_guest = 1)
        - 'First-time Guest' (all others)
2. **Key Metrics Calculated:**
    - **total_bookings:** Count of bookings per guest type
    - **percentage_bookings**: Percentage share of total bookings (using subquery for total count)
    - **avg_lead_time:** Average days between booking and arrival
    - **avg_stay_duration:** Average total nights stayed (weekend + weekday)
3. **Data Sources:**
    - Joins booking_source_and_history (guest information) with booking_details (stay information)
    - Uses Booking_id as the join key
4. **Organization:**
    - Groups results by guest type (is_repeated_guest)
    - Orders by booking percentage (highest first)

Purpose:
The query helps understand behavioral differences between first-time and returning guests across several dimensions:

- Booking volume distribution

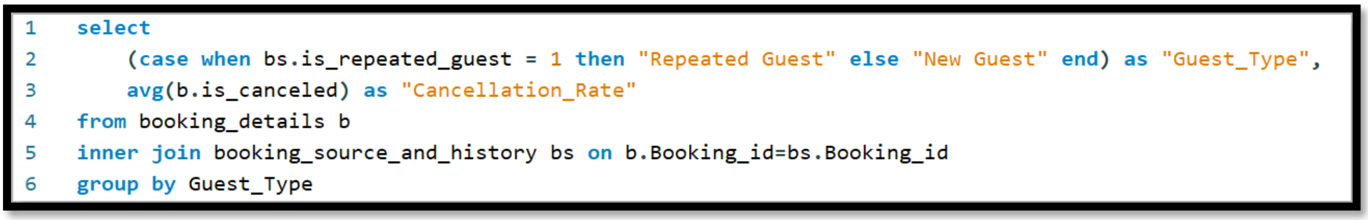
- Advance planning habits (lead time)
- Length of stay patterns

Query Result :



Guest Type	Total Bookings	% Bookings	Average Lead Time	Average Stay Duration
First-time Guest	115580	96.81	106.43	3.48
Repeated Guest	3810	3.19	30.79	1.93

15. Explore the impact of a guest's booking history on their likelihood of canceling a current booking. Calculate cancellation rates based on previous cancellations and noncanceled bookings.



```
select
    (case when bs.is_repeated_guest = 1 then "Repeated Guest" else "New Guest" end) as "Guest_Type",
    avg(b.is_canceled) as "Cancellation_Rate"
from booking_details b
inner join booking_source_and_history bs on b.Booking_id=bs.Booking_id
group by Guest_Type
```

Query Explanation:

This SQL query analyzes cancellation rates by guest type (new vs. returning). Here's the detailed breakdown:

1. **Guest Classification:**
   - Uses a CASE statement to create a "Guest_Type" column:
     - "Repeated Guest" when is_repeated_guest = 1
     - "New Guest" for all other cases
2. **Key Metric:**
   - Calculates avg(b.is_canceled) as "Cancellation_Rate":
     - Since is_canceled is typically binary (1=canceled, 0=not canceled), the average gives the cancellation rate
     - For example: 0.25 would indicate 25% of bookings were canceled
3. **Data Sources:**
   - Joins booking_details (contains cancellation status) with booking_source_and_history (contains guest repeat status)
   - Uses Booking_id as the join key

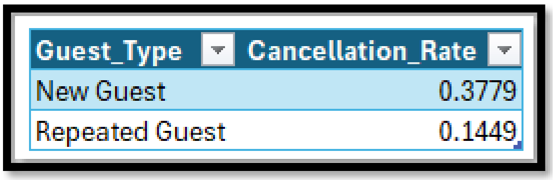
4. **Organization:**
   - Groups results by the created Guest_Type
   - Returns exactly two rows: one for new guests, one for repeat guests

Purpose:
This query specifically compares cancellation behavior between:

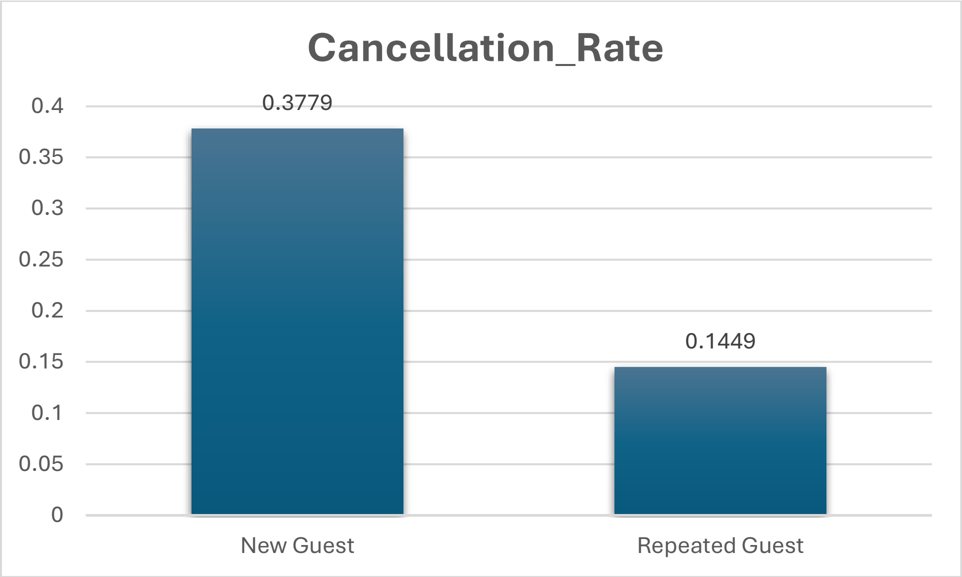
- First-time guests (New Guest)
- Returning guests (Repeated Guest)

The cancellation rate metric helps identify whether repeat guests are more or less likely to cancel bookings compared to new guests, which can inform customer retention strategies and booking policies.

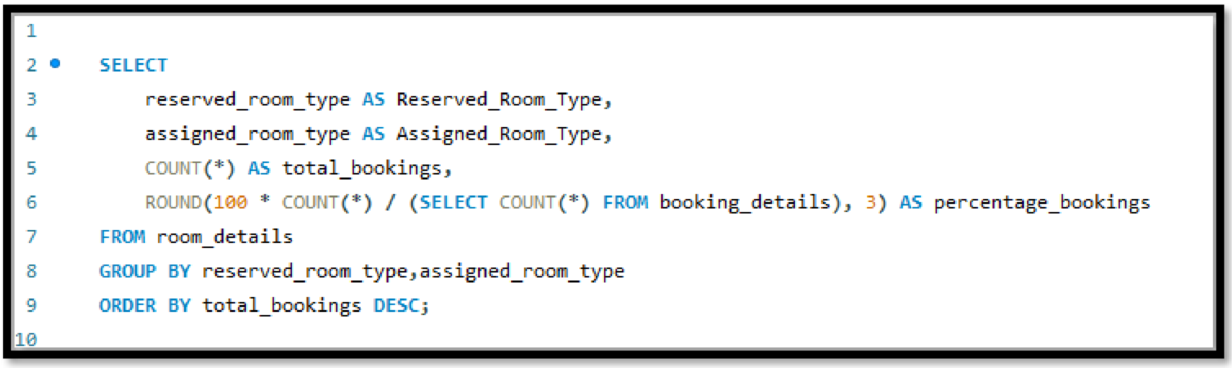
Query Result :



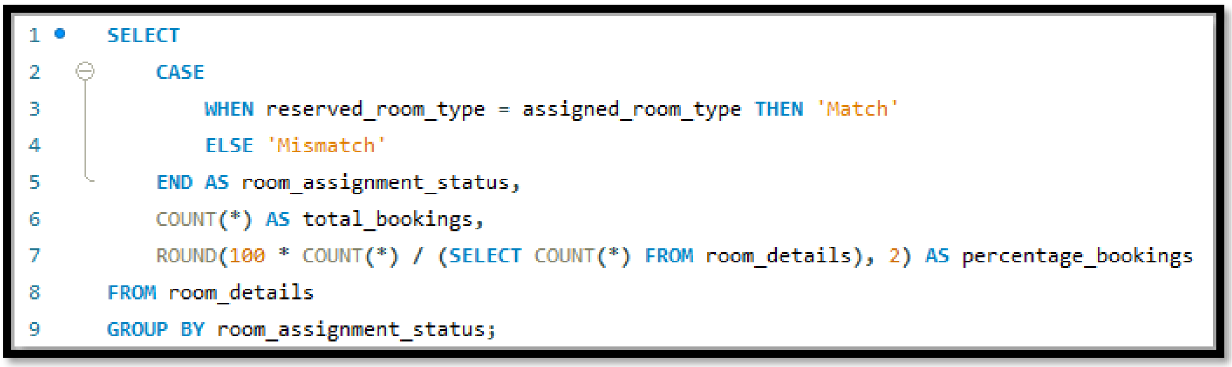
Guest_Type	Cancellation_Rate
New Guest	0.3779
Repeated Guest	0.1449



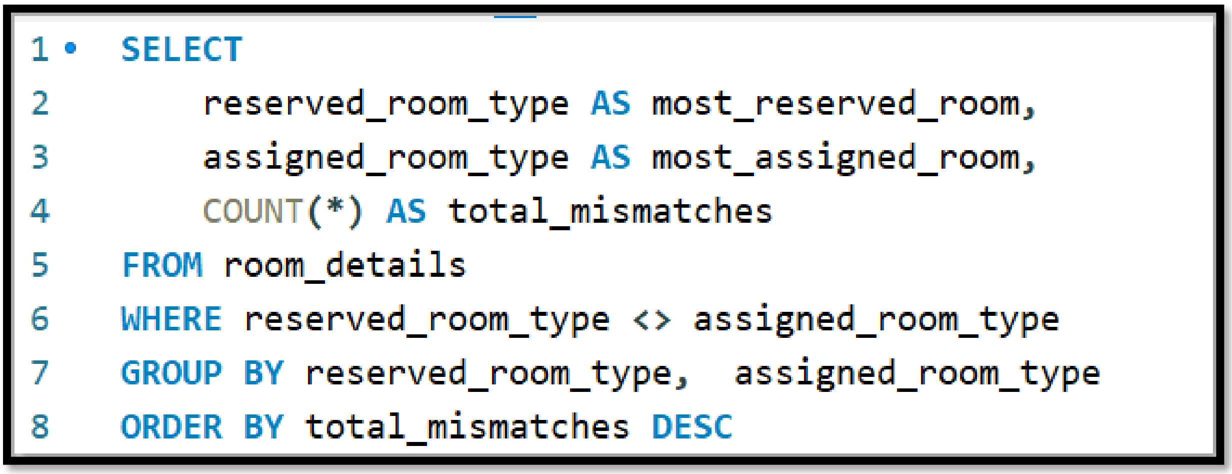
16. Understand the distribution of reserved and assigned room types. Calculate summary statistics for the consistency between reserved and assigned room types.



```
SELECT
    reserved_room_type AS Reserved_Room_Type,
    assigned_room_type AS Assigned_Room_Type,
    COUNT(*) AS total_bookings,
    ROUND(100 * COUNT(*) / (SELECT COUNT(*) FROM booking_details), 3) AS percentage_bookings
FROM room_details
GROUP BY reserved_room_type,assigned_room_type
ORDER BY total_bookings DESC;
```



```
SELECT
    CASE
        WHEN reserved_room_type = assigned_room_type THEN 'Match'
        ELSE 'Mismatch'
    END AS room_assignment_status,
    COUNT(*) AS total_bookings,
    ROUND(100 * COUNT(*) / (SELECT COUNT(*) FROM room_details), 2) AS percentage_bookings
FROM room_details
GROUP BY room_assignment_status;
```



```
SELECT
        reserved_room_type AS most_reserved_room,
        assigned_room_type AS most_assigned_room,
        COUNT(*) AS total_mismatches
FROM room_details
WHERE reserved_room_type <> assigned_room_type
GROUP BY reserved_room_type,  assigned_room_type
ORDER BY total_mismatches DESC
```

Combined Purpose:
These queries work together to provide comprehensive insights into room assignment practices:

1. Query 1 establishes the baseline distribution of all room type combinations (both matches and mismatches)
2. Query 2 calculates the overall match/mismatch rate (aggregate view)
3. Query 3 drills down into specific mismatch patterns (detailed view)

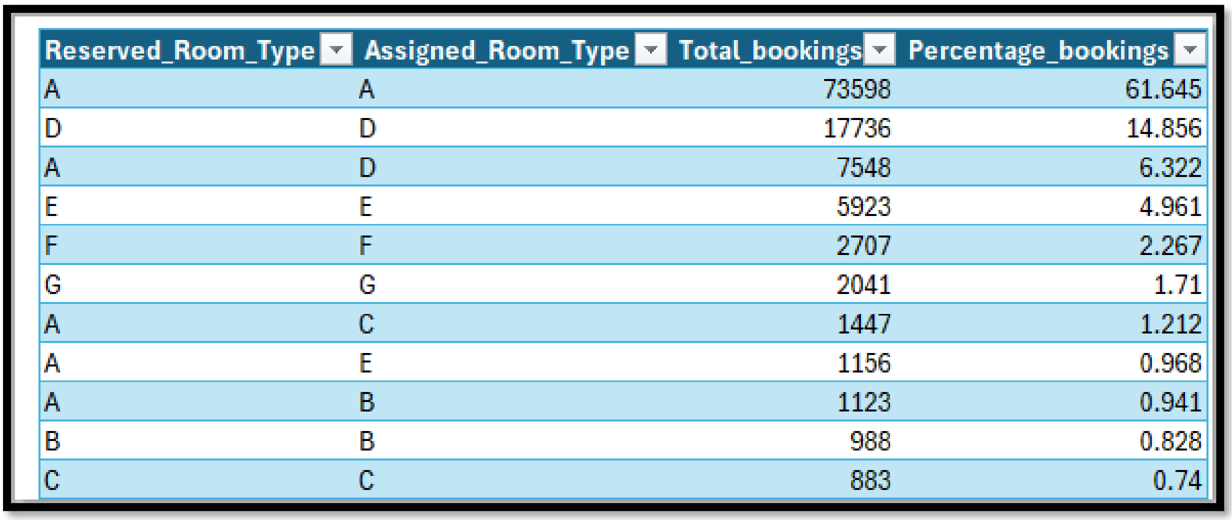
Key Analysis Capabilities:

- Measures operational efficiency in room assignments
- Identifies most common room type substitutions

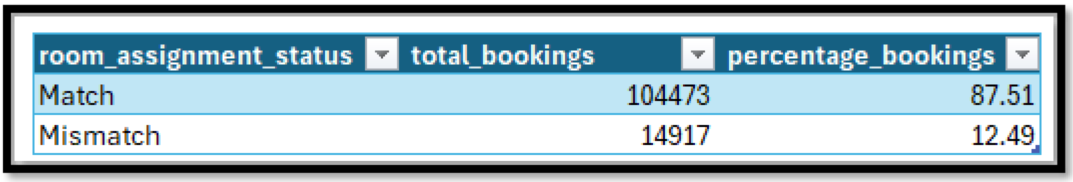
- Quantifies the frequency of mismatches
- Reveals potential inventory management issues
- Provides metrics for service quality evaluation

The percentage calculations in Queries 1 and 2 provide normalized comparisons, while Query 3's count-based ranking highlights the most prevalent substitution patterns. Together they form a complete picture of room assignment accuracy and patterns.

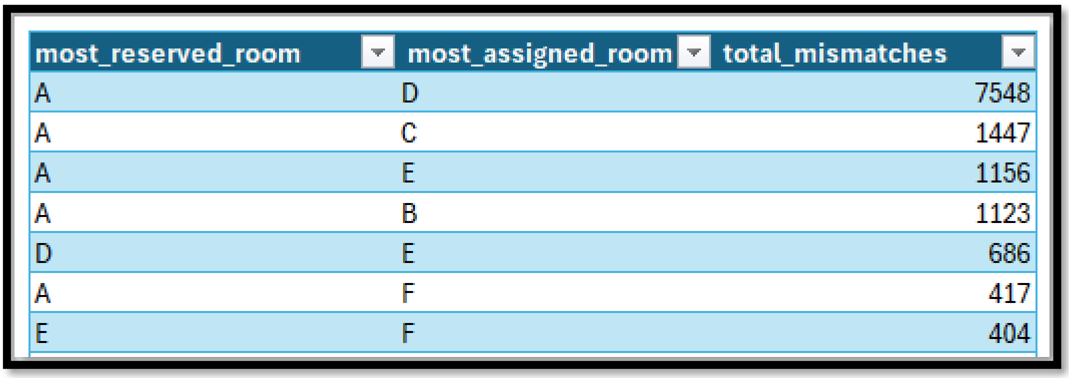
Query Result :



Reserved_Room_Type	Assigned_Room_Type	Total_bookings	Percentage_bookings
A	A	73598	61.645
D	D	17736	14.856
A	D	7548	6.322
E	E	5923	4.961
F	F	2707	2.267
G	G	2041	1.71
A	C	1447	1.212
A	E	1156	0.968
A	B	1123	0.941
B	B	988	0.828
C	C	883	0.74



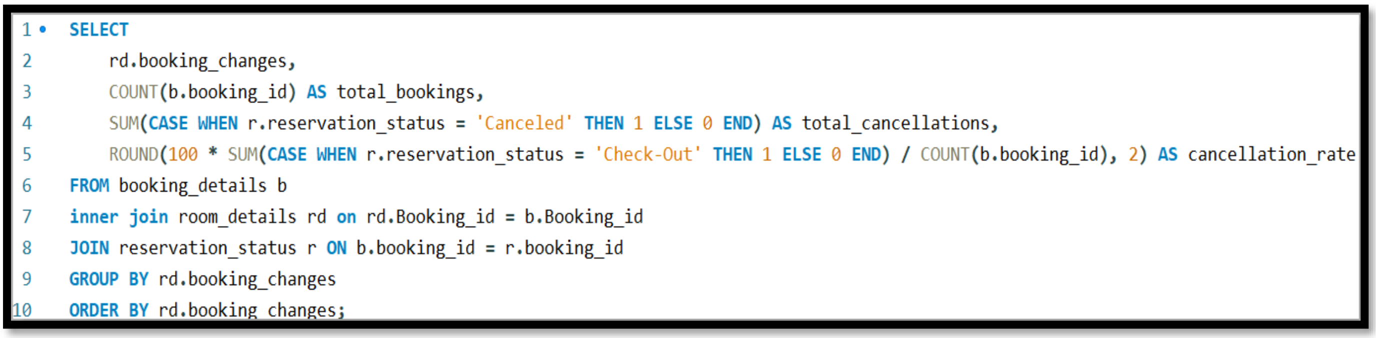
room_assignment_status	total_bookings	percentage_bookings
Match	104473	87.51
Mismatch	14917	12.49



most_reserved_room	most_assigned_room	total_mismatches
A	D	7548
A	C	1447
A	E	1156
A	B	1123
D	E	686
A	F	417
E	F	404

---

17. Analyze the impact of booking changes on cancellation rates. Calculate cancellation rates for bookings with different numbers of changes.



```
SELECT
    rd.booking_changes,
    COUNT(b.booking_id) AS total_bookings,
    SUM(CASE WHEN r.reservation_status = 'Canceled' THEN 1 ELSE 0 END) AS total_cancellations,
    ROUND(100 * SUM(CASE WHEN r.reservation_status = 'Check-Out' THEN 1 ELSE 0 END) / COUNT(b.booking_id), 2) AS cancellation_rate
FROM booking_details b
inner join room_details rd on rd.Booking_id = b.Booking_id
JOIN reservation_status r ON b.booking_id = r.booking_id
GROUP BY rd.booking_changes
ORDER BY rd.booking_changes;
```

Query Explanation:

This query analyzes the relationship between booking modifications and cancellation rates. Here's the breakdown:

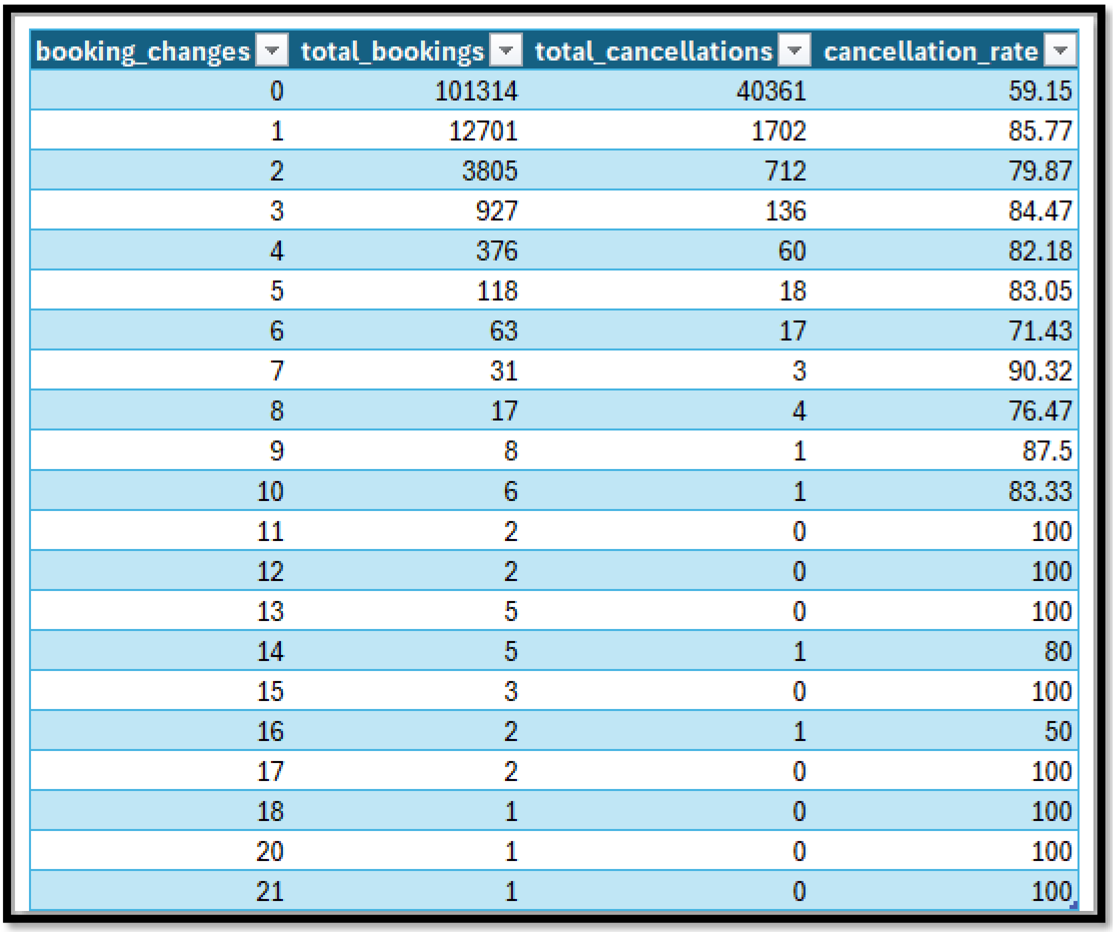
1. **Columns Selected:**
   - **rd.booking_changes:** Number of modifications made to each booking
   - **COUNT(b.booking_id):** Total number of bookings for each modification count
   - **SUM(CASE WHEN...Canceled...):** Count of canceled bookings per modification count
   - **Cancellation rate calculation**: Percentage of bookings canceled per modification count
2. **Tables Joined:**
   - booking_details (core booking information)
   - room_details (contains booking_changes data)
   - reservation_status (contains cancellation status)
3. **Key Corrections:**
   - Changed SUN to SUM (function name correction)
   - Fixed the cancellation rate calculation to use canceled status
   - Standardized JOIN syntax

Purpose:
This query helps understand:

- How frequently bookings are modified
- Whether modification frequency correlates with cancellation likelihood
- The cancellation risk associated with different numbers of booking changes

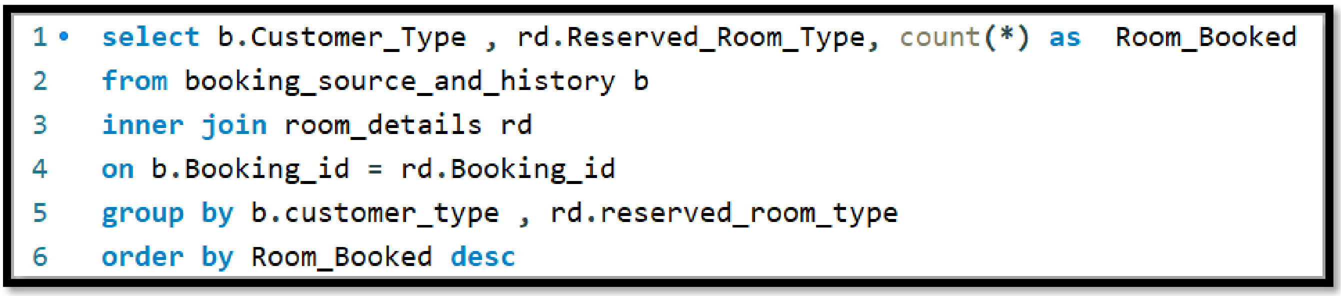
The results can help identify if multiple booking changes serve as an early warning indicator for potential cancellations.

Query Result :



booking_changes	total_bookings	total_cancellations	cancellation_rate
0	101314	40361	59.15
1	12701	1702	85.77
2	3805	712	79.87
3	927	136	84.47
4	376	60	82.18
5	118	18	83.05
6	63	17	71.43
7	31	3	90.32
8	17	4	76.47
9	8	1	87.5
10	6	1	83.33
11	2	0	100
12	2	0	100
13	5	0	100
14	5	1	80
15	3	0	100
16	2	1	50
17	2	0	100
18	1	0	100
20	1	0	100
21	1	0	100

18. **Explore how room type preferences vary across different customer types (e.g., Transient, Group). Identify if certain customer types have specific room preferences.**



```
select b.Customer_Type , rd.Reserved_Room_Type, count(*) as  Room_Booked
from booking_source_and_history b
inner join room_details rd
on b.Booking_id = rd.Booking_id
group by b.customer_type , rd.reserved_room_type
order by Room_Booked desc
```

Query Explanation:

This query analyzes room booking patterns by customer type and reserved room type. Here's the breakdown:

1. **Columns Selected:**

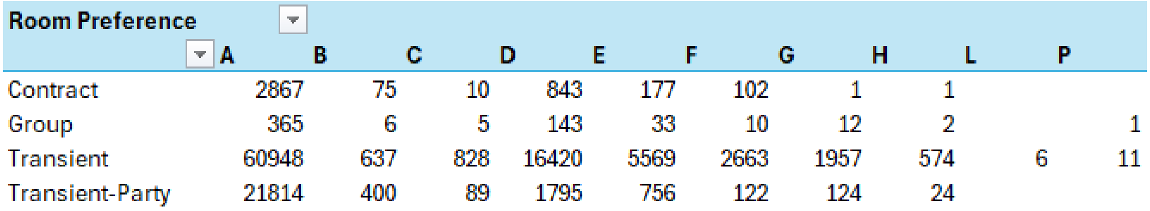
- **b.Customer_Type:** Categorizes customers by their type (e.g., business, leisure)
  - **rd.Reserved_Room_Type:** Shows which room types customers reserved
  - **count(*) as Room_Booked**: Counts how many times each room type was booked by each customer type

2. **Tables Joined:**
   - booking_source_and_history (contains customer information)
   - room_details (contains room reservation details)
3. **Join Condition:**
   - Connected via Booking_id to match customers with their room reservations
4. **Grouping:**
   - Results are grouped by both customer type and reserved room type
   - This creates unique combinations of customer segments and room preferences
5. **Sorting:**
   - Orders results by booking count in descending order (most popular combinations first)

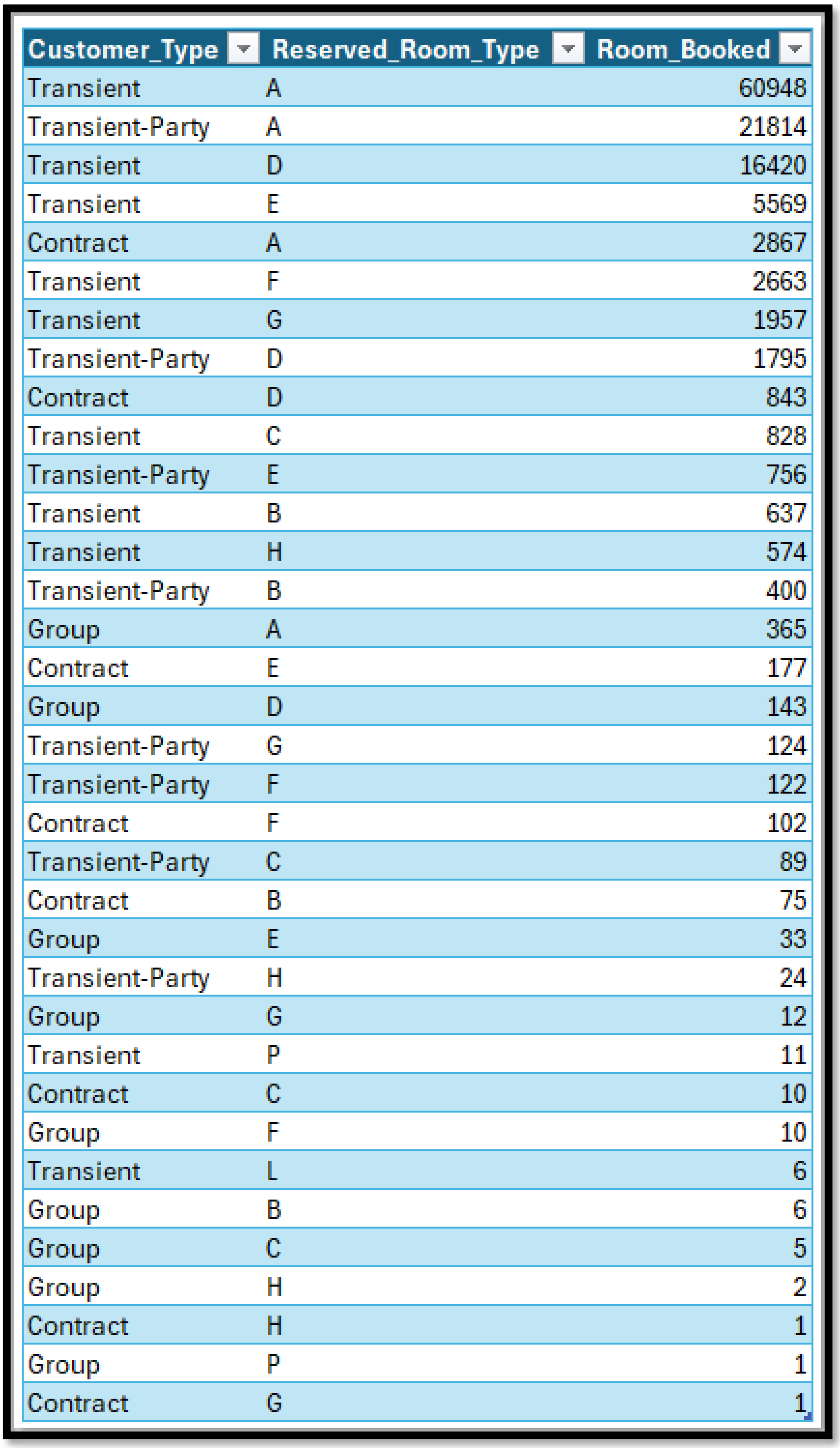
Purpose:
This analysis helps understand:

- Which customer segments prefer which room types
- The popularity distribution of room types across different customer categories
- Potential relationships between customer profiles and room preferences

Query Result :



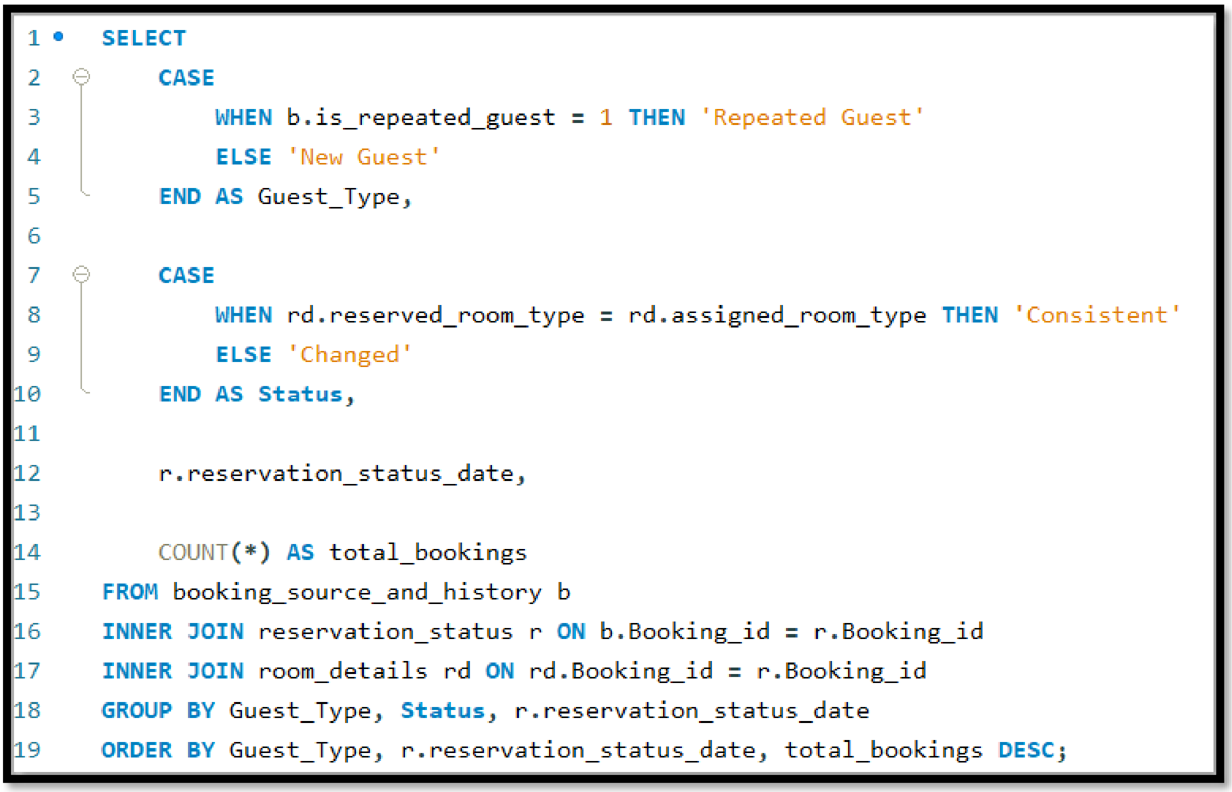
Room Preference										
	A	B	C	D	E	F	G	H	L	P
Contract	2867	75	10	843	177	102	1	1		
Group	365	6	5	143	33	10	12	2		1
Transient	60948	637	828	16420	5569	2663	1957	574	6	11
Transient-Party	21814	400	89	1795	756	122	124	24		



Customer_Type	Reserved_Room_Type	Room_Booked
Transient	A	60948
Transient-Party	A	21814
Transient	D	16420
Transient	E	5569
Contract	A	2867
Transient	F	2663
Transient	G	1957
Transient-Party	D	1795
Contract	D	843
Transient	C	828
Transient-Party	E	756
Transient	B	637
Transient	H	574
Transient-Party	B	400
Group	A	365
Contract	E	177
Group	D	143
Transient-Party	G	124
Transient-Party	F	122
Contract	F	102
Transient-Party	C	89
Contract	B	75
Group	E	33
Transient-Party	H	24
Group	G	12
Transient	P	11
Contract	C	10
Group	F	10
Transient	L	6
Group	B	6
Group	C	5
Group	H	2
Contract	H	1
Group	P	1
Contract	G	1

---

19. Examine whether guests who make multiple bookings have consistent room type preferences or if their preferences change over time.



```
1    SELECT
2        CASE
3            WHEN b.is_repeated_guest = 1 THEN 'Repeated Guest'
4            ELSE 'New Guest'
5        END AS Guest_Type,
6
7        CASE
8            WHEN rd.reserved_room_type = rd.assigned_room_type THEN 'Consistent'
9            ELSE 'Changed'
10       END AS Status,
11
12       r.reservation_status_date,
13
14       COUNT(*) AS total_bookings
15   FROM booking_source_and_history b
16   INNER JOIN reservation_status r ON b.Booking_id = r.Booking_id
17   INNER JOIN room_details rd ON rd.Booking_id = r.Booking_id
18   GROUP BY Guest_Type, Status, r.reservation_status_date
19   ORDER BY Guest_Type, r.reservation_status_date, total_bookings DESC;
```

Query Explanation:
This query analyzes booking patterns by guest type (new vs. returning) and room assignment consistency. Here's the breakdown:

1. **Columns Selected:**
    - **Guest classification (lines 2-5):**
        - 'Repeated Guest' for is_repeated_guest = 1
        - 'New Guest' for all others
    - **Room assignment status (lines 6-9):**
        - 'Consistent' when reserved and assigned room types match
        - 'Changed' when they differ
    - Reservation status date (line 12)
    - Booking count (line 14)
2. **Tables Joined:**
    - booking_source_and_history (guest information)
    - reservation_status (booking status data)
    - room_details (room assignment information)
3. b
    - All tables joined via Booking_id

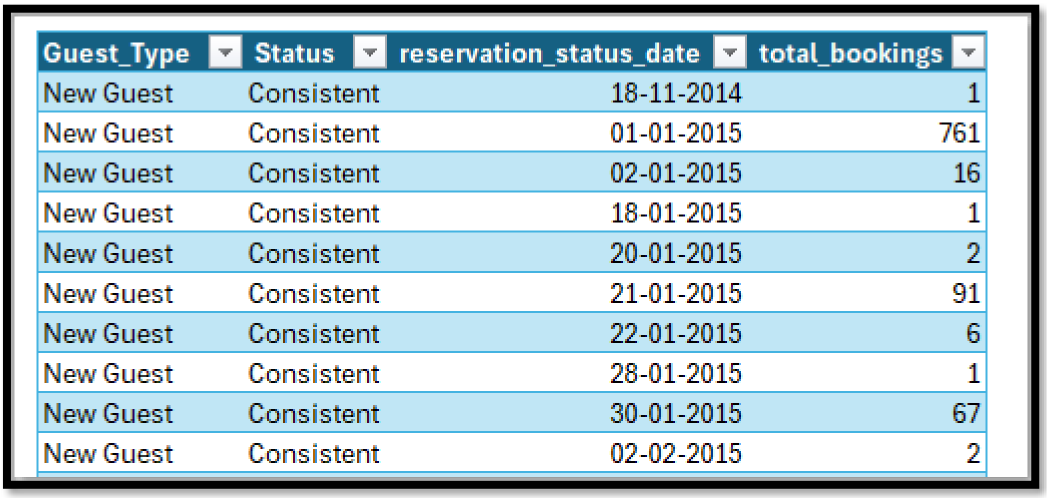
4. **Grouping:**
    - Results grouped by:
        1. Guest type (new/repeat)
        2. Room assignment status (consistent/changed)
        3. Reservation status date
5. **Sorting:**
    - Primary sort: Guest type
    - Secondary sort: Date
    - Tertiary sort: Booking count (high to low)

Purpose:
This query helps track:

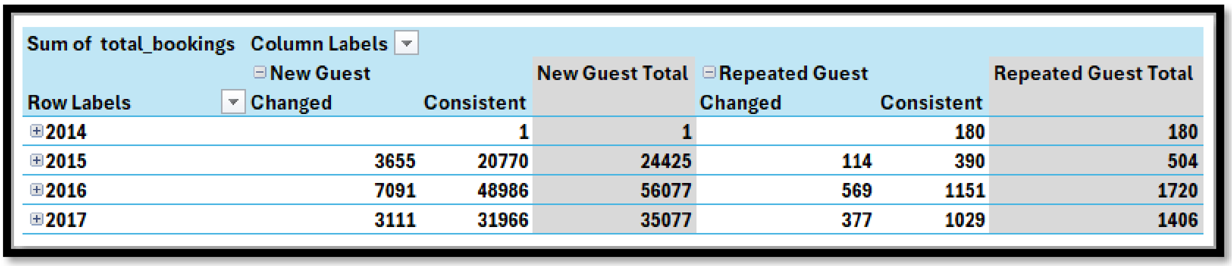
- How often room assignments match reservations
- Whether repeat guests experience more/fewer room changes
- How these patterns evolve over time

The date-based grouping allows for temporal analysis of these metrics, showing whether room assignment practices or guest experiences are improving or worsening over time.

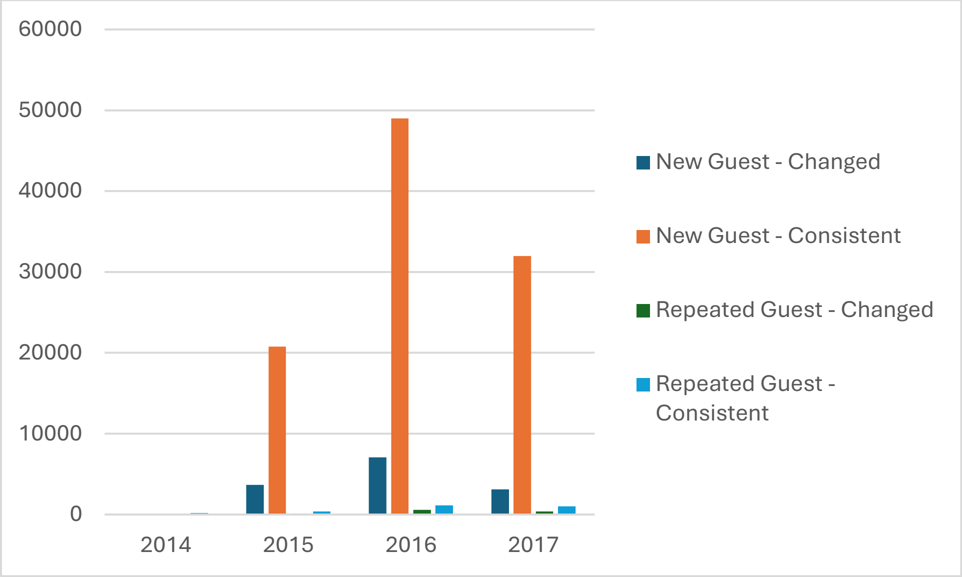
Query Result :



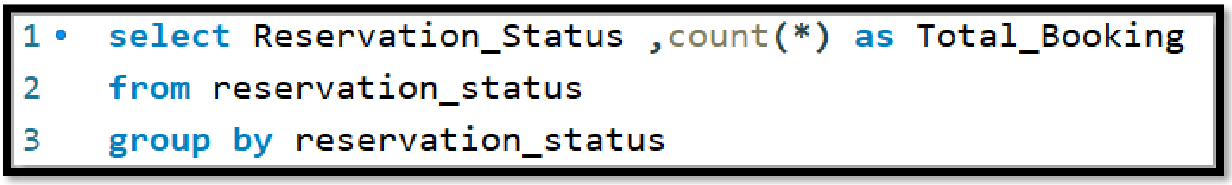
Guest_Type	Status	reservation_status_date	total_bookings
New Guest	Consistent	18-11-2014	1
New Guest	Consistent	01-01-2015	761
New Guest	Consistent	02-01-2015	16
New Guest	Consistent	18-01-2015	1
New Guest	Consistent	20-01-2015	2
New Guest	Consistent	21-01-2015	91
New Guest	Consistent	22-01-2015	6
New Guest	Consistent	28-01-2015	1
New Guest	Consistent	30-01-2015	67
New Guest	Consistent	02-02-2015	2



Sum of total_bookings	Column Labels					
	New Guest		New Guest Total	Repeated Guest		Repeated Guest Total
Row Labels	Changed	Consistent		Changed	Consistent	
2014		1	1		180	180
2015	3655	20770	24425	114	390	504
2016	7091	48986	56077	569	1151	1720
2017	3111	31966	35077	377	1029	1406



20. Understand the distribution of reservation statuses and calculate summary statistics for reservation status dates.



```sql
select Reservation_Status ,count(*) as Total_Booking
from reservation_status
group by reservation_status
```

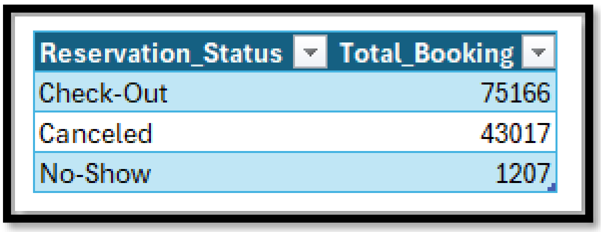
Query Explanation:

1. Selects reservation status and counts occurrences
2. Groups results by each unique status value
3. Returns the total number of bookings for each status category

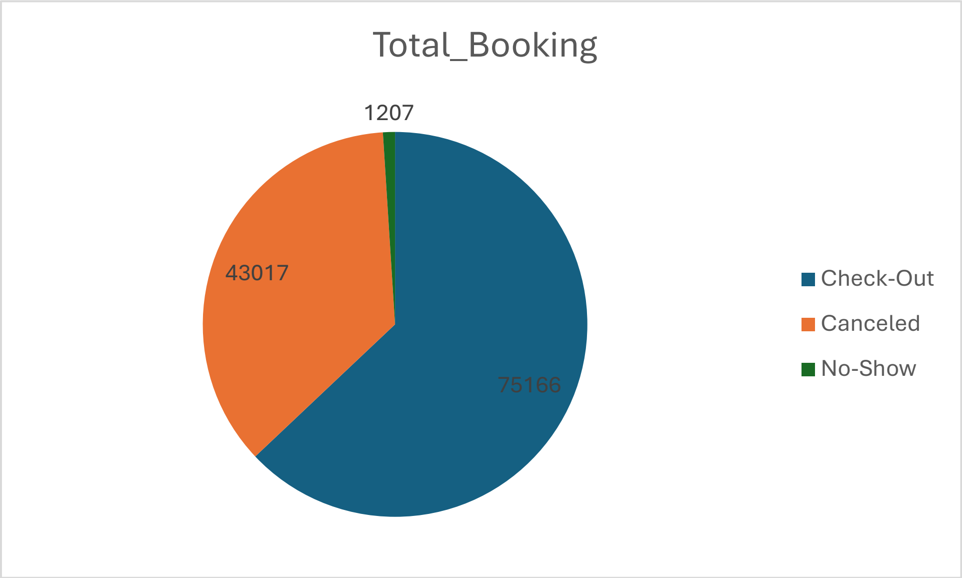
Purpose:

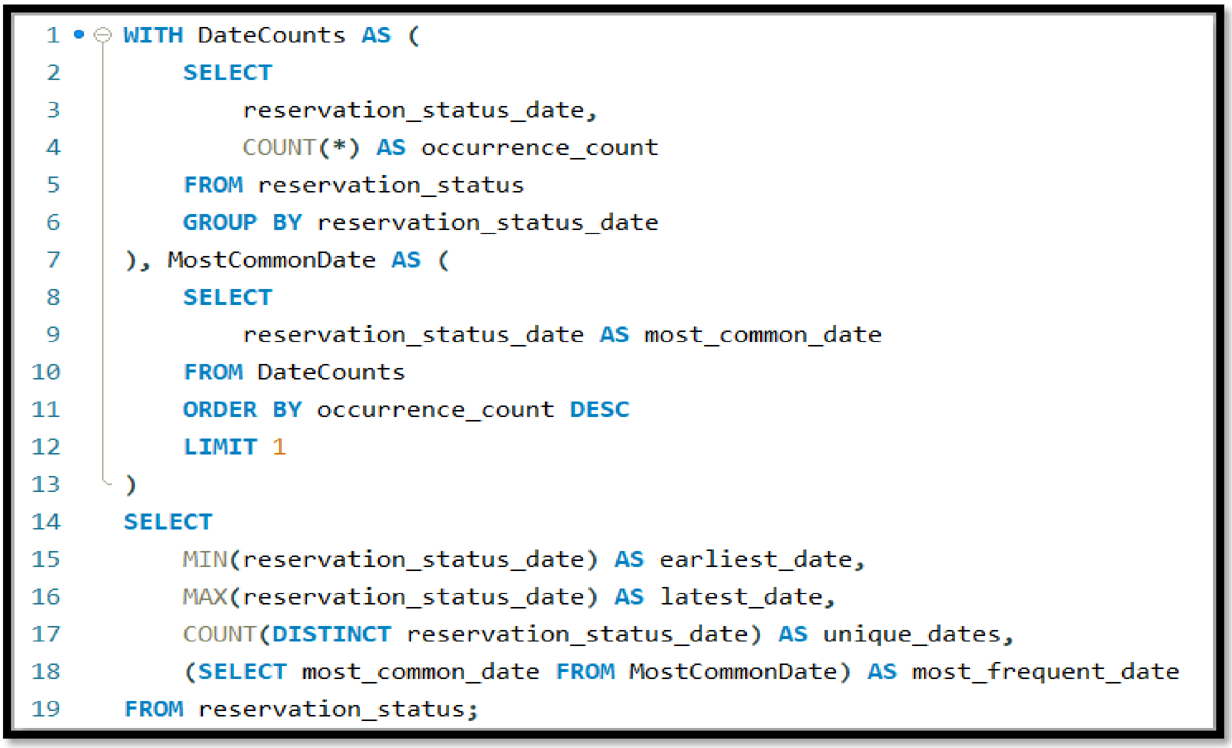
- Provides a basic distribution of booking statuses
- Shows how many bookings are in each state (e.g., Checked-Out, Canceled, No-Show)
- Helps understand the overall composition of reservations

Query Result :



Reservation_Status	Total_Booking
Check-Out	75166
Canceled	43017
No-Show	1207





```
WITH DateCounts AS (
    SELECT
        reservation_status_date,
        COUNT(*) AS occurrence_count
    FROM reservation_status
    GROUP BY reservation_status_date
), MostCommonDate AS (
    SELECT
        reservation_status_date AS most_common_date
    FROM DateCounts
    ORDER BY occurrence_count DESC
    LIMIT 1
)
SELECT
    MIN(reservation_status_date) AS earliest_date,
    MAX(reservation_status_date) AS latest_date,
    COUNT(DISTINCT reservation_status_date) AS unique_dates,
    (SELECT most_common_date FROM MostCommonDate) AS most_frequent_date
FROM reservation_status;
```

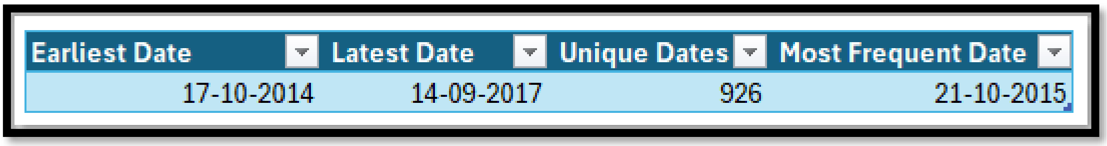
Query Explanation:

1. Creates a CTE (DateCounts) that counts bookings per date
2. Creates a second CTE (MostCommonDate) that identifies the date with most bookings
3. Main query returns:
   - Earliest and latest dates in the data
   - Count of unique dates
   - The single most frequent date

Purpose:

- Provides temporal boundaries of the dataset
- Identifies date distribution patterns
- Highlights peak activity dates
- Helps assess data quality and coverage period

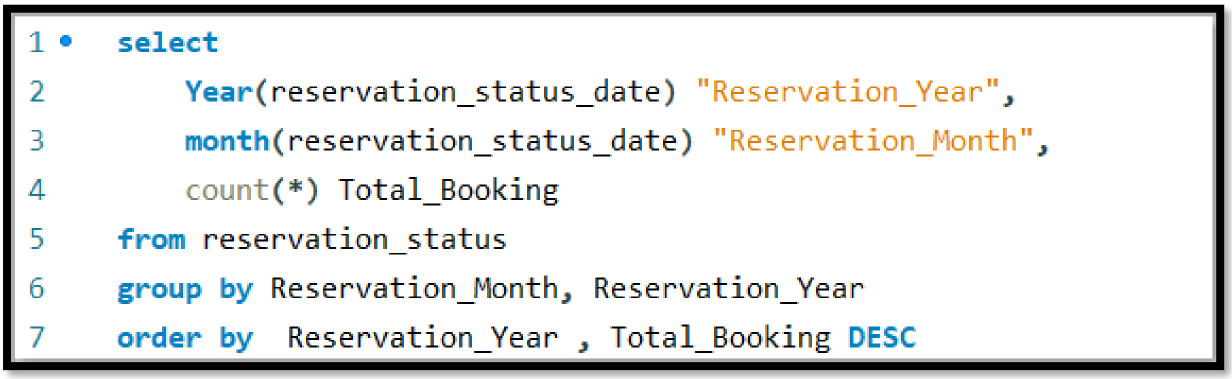
Query Result :



Earliest Date	Latest Date	Unique Dates	Most Frequent Date
17-10-2014	14-09-2017	926	21-10-2015

---

21. Analyze trends in reservation status dates, including the most common checkout dates and any seasonality patterns.



```
select
    Year(reservation_status_date) "Reservation_Year",
    month(reservation_status_date) "Reservation_Month",
    count(*) Total_Booking
from reservation_status
group by Reservation_Month, Reservation_Year
order by  Reservation_Year , Total_Booking DESC
```

Query Explanation:

1. **Columns Selected:**
   - **YEAR(reservation_status_date)** → Extracts the year from the reservation date.

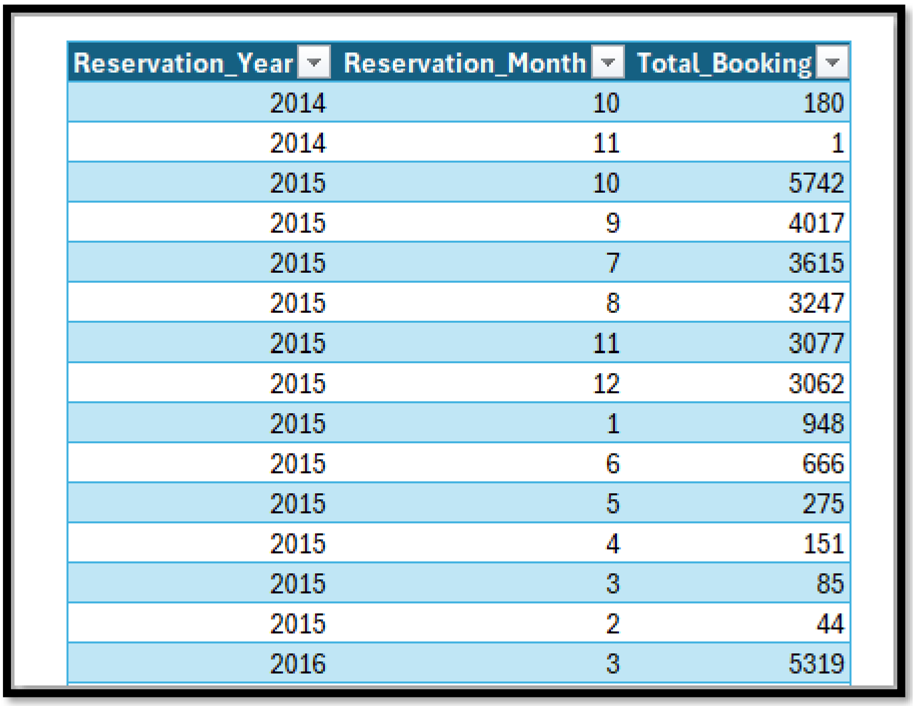
   - **MONTH(reservation_status_date)** → Extracts the month (1-12).
   - **COUNT(*) AS Total_Booking** → Counts the number of bookings per month-year.

2. **Grouping & Sorting:**
   - **GROUP BY Reservation_Month, Reservation_Year** → Aggregates bookings by month and year.
   - **ORDER BY Reservation_Year, Total_Booking DESC** →
     - First sorts chronologically by year.
     - Then sorts months within each year by booking volume (highest first).

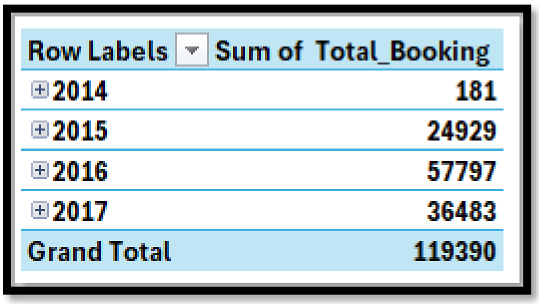
Purpose:

- Identifies seasonal trends (peak vs. low-demand months).
- Tracks year-over-year growth (compare 2023 vs. 2024 bookings).
- Supports demand forecasting (staffing, pricing, promotions).

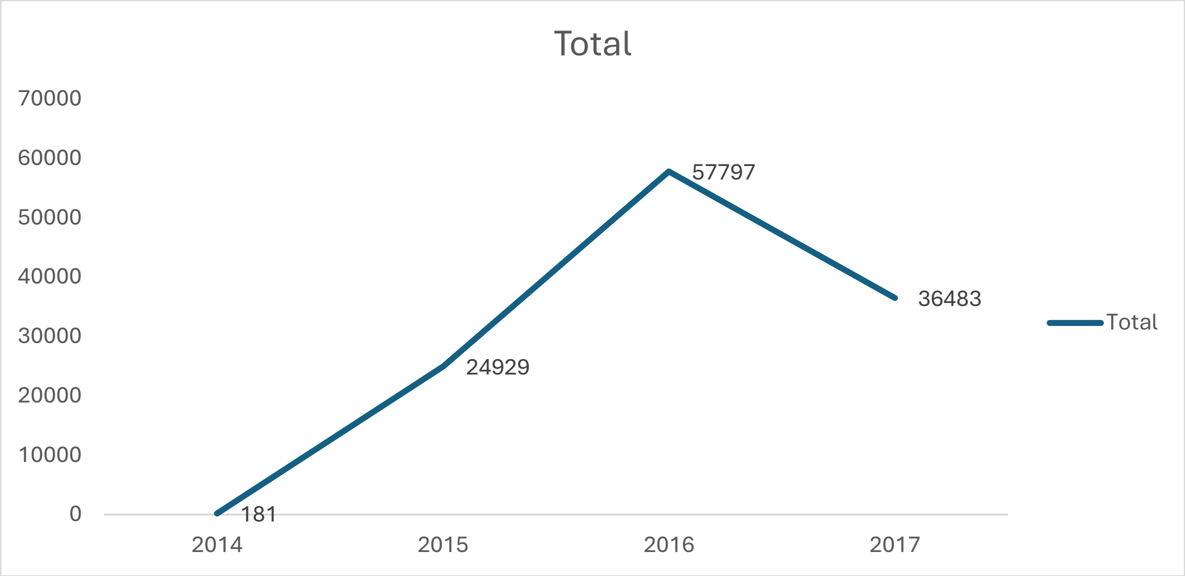
Query Result :

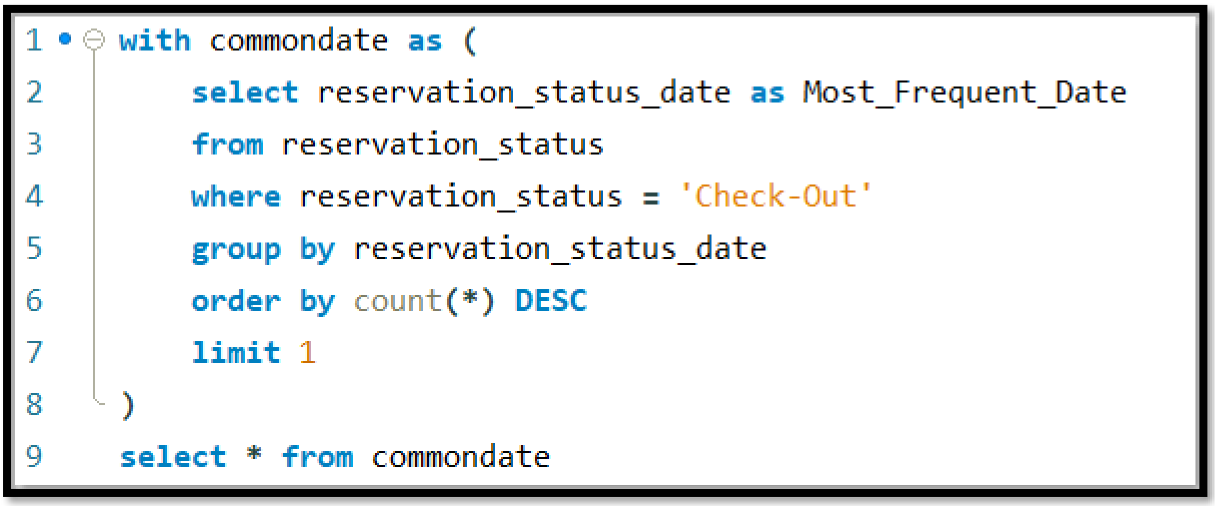


Reservation_Year	Reservation_Month	Total_Booking
2014	10	180
2014	11	1
2015	10	5742
2015	9	4017
2015	7	3615
2015	8	3247
2015	11	3077
2015	12	3062
2015	1	948
2015	6	666
2015	5	275
2015	4	151
2015	3	85
2015	2	44
2016	3	5319



Row Labels	Sum of Total_Booking
⊞2014	181
⊞2015	24929
⊞2016	57797
⊞2017	36483
Grand Total	119390





```sql
with commondate as (
	select reservation_status_date as Most_Frequent_Date
	from reservation_status
	where reservation_status = 'Check-Out'
	group by reservation_status_date
	order by count(*) DESC
	limit 1
)
select * from commondate
```

Query Explanation:

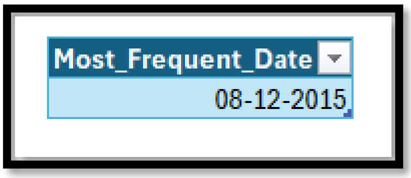
1. CTE (commondate):
   - Filters for completed stays (reservation_status = 'Check-Out').
   - Groups by date and counts check-outs per day.
   - Orders by frequency (highest first) and picks the top 1 date (LIMIT 1).

2. Final Output:
   - Returns the single date with the most check-outs.

Purpose:

- Identifies peak operational days (when max guests check out).
- Helps optimize staffing (housekeeping, front desk).
- Highlights potential bottlenecks (e.g., long wait times).

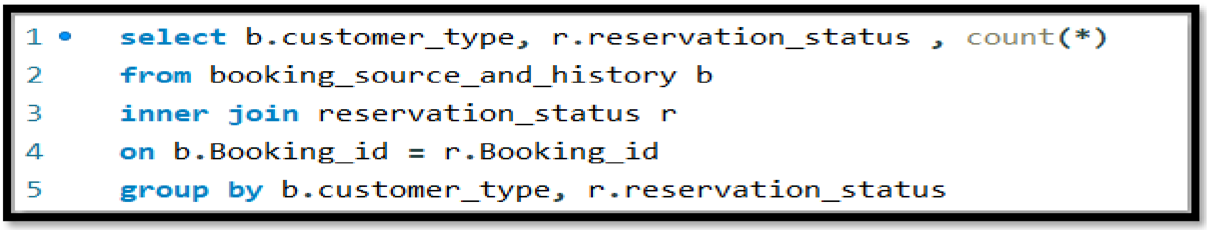
Query Result :



Most_Frequent_Date
08-12-2015

---

22. Explore how reservation statuses vary across different customer types (e.g., Transient, Group) using Excel or SQL. Calculate cancellation rates by customer type.



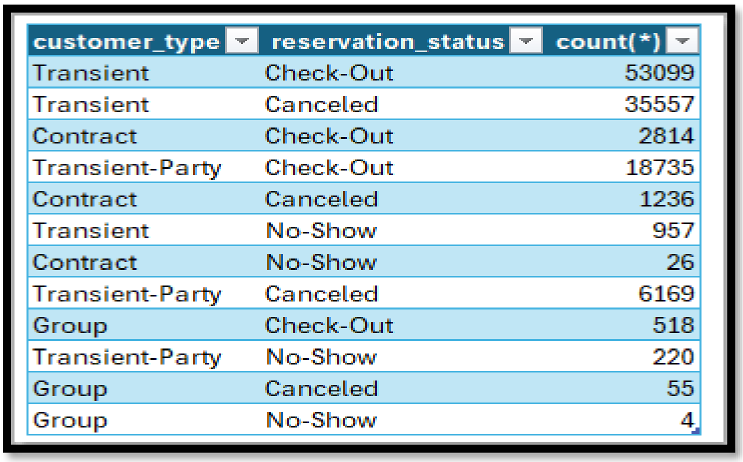
```
select b.customer_type, r.reservation_status , count(*)
from booking_source_and_history b
inner join reservation_status r
on b.Booking_id = r.Booking_id
group by b.customer_type, r.reservation_status
```

Purpose:
This query provides a breakdown of booking statuses (e.g., Checked-Out, Canceled, No-Show) across different customer types.

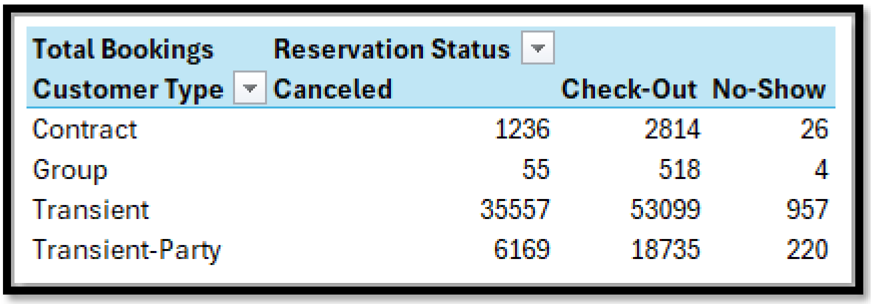
Key Features:

1. Joins customer data with reservation status data
2. Groups results by both customer type and reservation status
3. Counts occurrences of each status per customer type

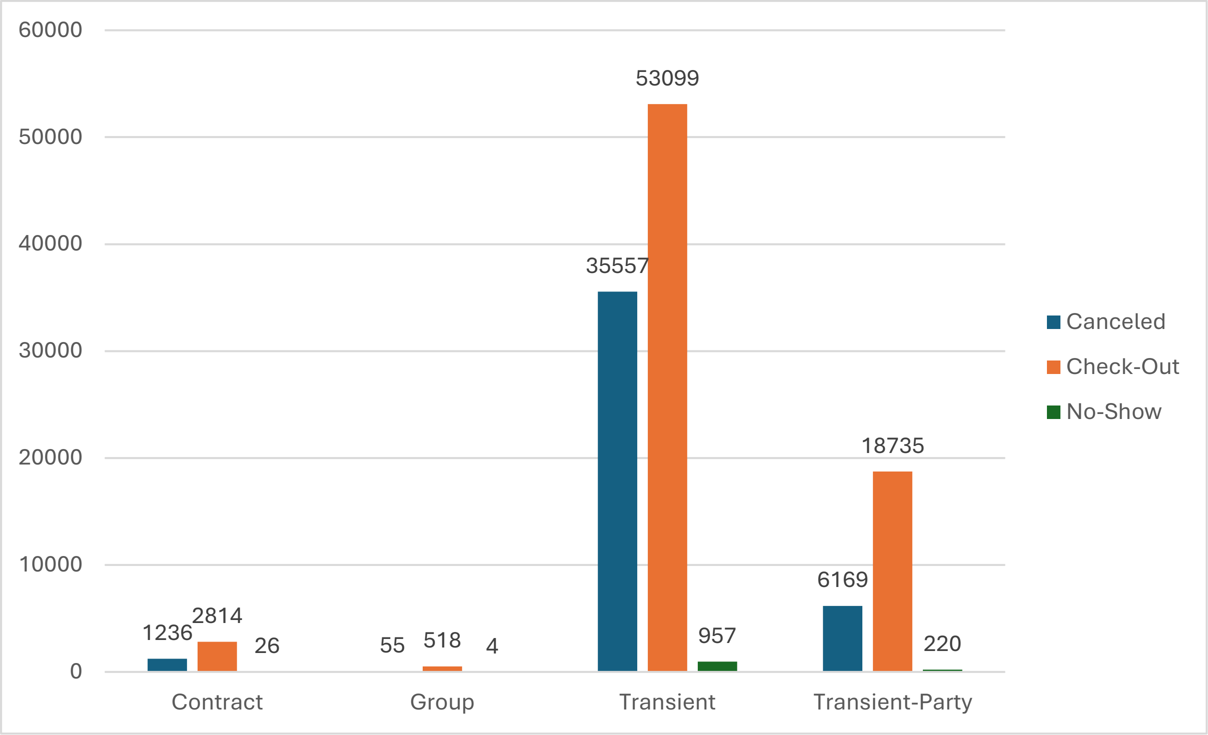
Query Result :

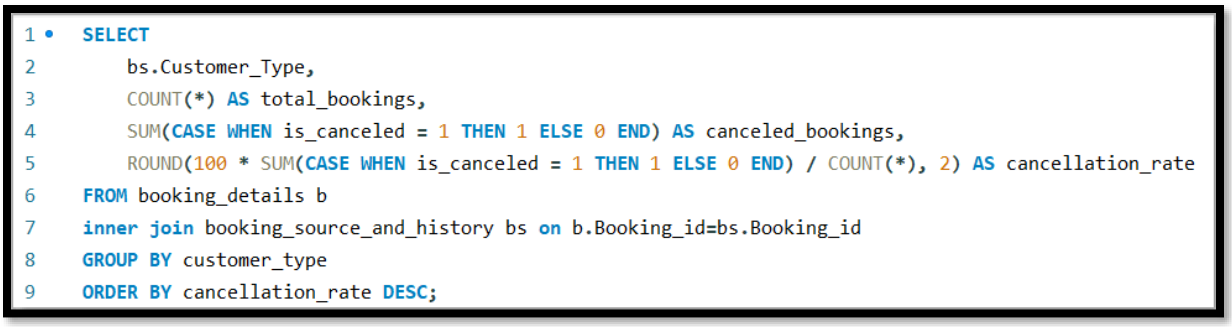


customer_type	reservation_status	count(*)
Transient	Check-Out	53099
Transient	Canceled	35557
Contract	Check-Out	2814
Transient-Party	Check-Out	18735
Contract	Canceled	1236
Transient	No-Show	957
Contract	No-Show	26
Transient-Party	Canceled	6169
Group	Check-Out	518
Transient-Party	No-Show	220
Group	Canceled	55
Group	No-Show	4



Total Bookings	Reservation Status		
Customer Type	Canceled	Check-Out	No-Show
Contract	1236	2814	26
Group	55	518	4
Transient	35557	53099	957
Transient-Party	6169	18735	220





```
SELECT
    bs.Customer_Type,
    COUNT(*) AS total_bookings,
    SUM(CASE WHEN is_canceled = 1 THEN 1 ELSE 0 END) AS canceled_bookings,
    ROUND(100 * SUM(CASE WHEN is_canceled = 1 THEN 1 ELSE 0 END) / COUNT(*), 2) AS cancellation_rate
FROM booking_details b
inner join booking_source_and_history bs on b.Booking_id=bs.Booking_id
GROUP BY customer_type
ORDER BY cancellation_rate DESC;
```

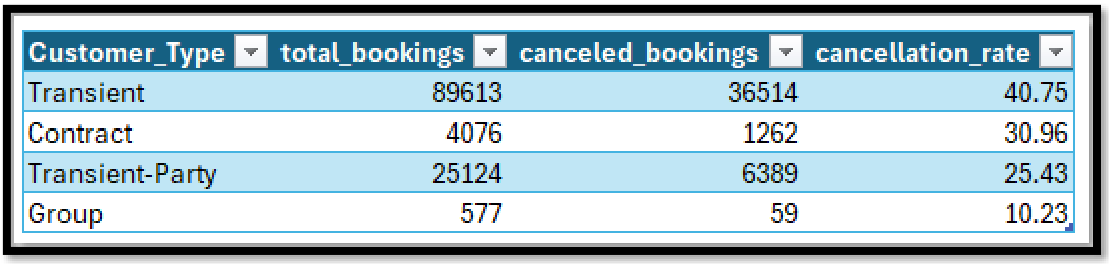
Purpose:

This query calculates and compares cancellation rates across different customer types.

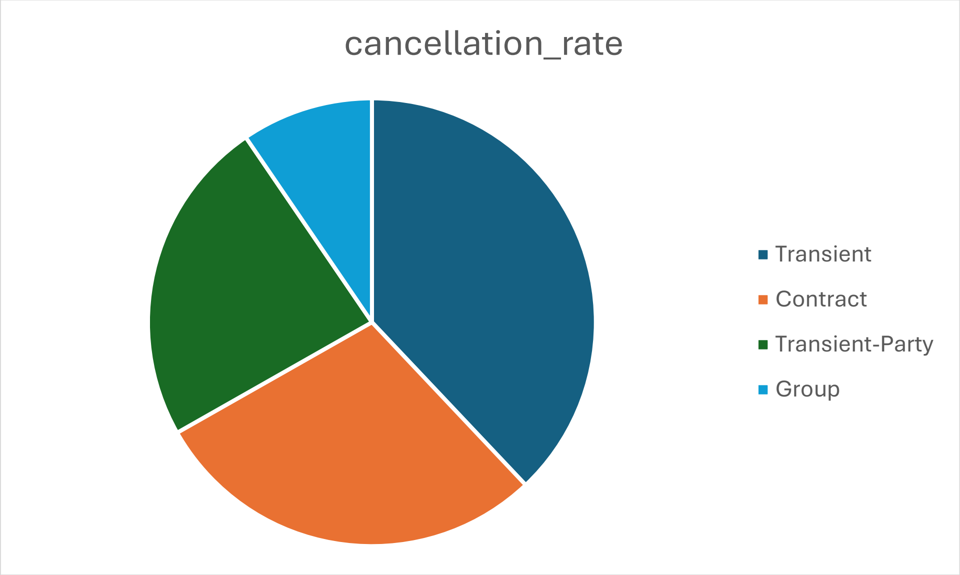
Key Features:

1. Uses conditional aggregation to count cancellations
2. Calculates cancellation percentage for each customer type
3. Orders results by cancellation rate (highest first)

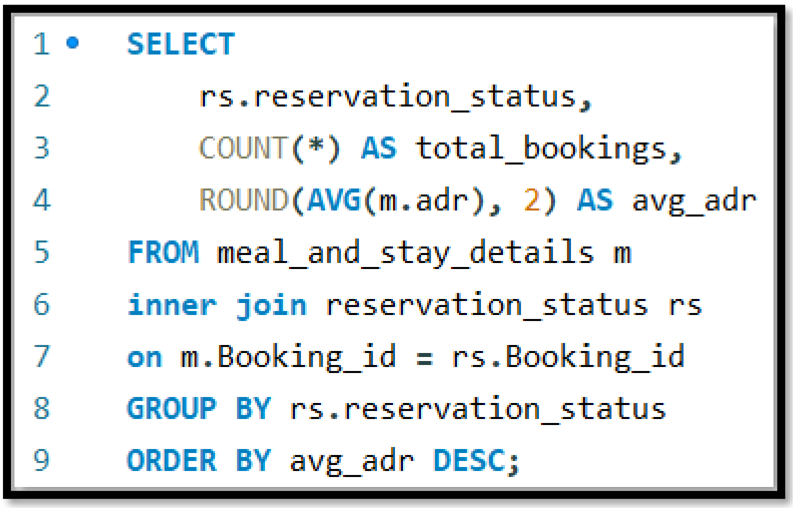
Query Result :



Customer_Type	total_bookings	canceled_bookings	cancellation_rate
Transient	89613	36514	40.75
Contract	4076	1262	30.96
Transient-Party	25124	6389	25.43
Group	577	59	10.23



23. Investigate whether there are differences in Average Daily Rates (ADR) based on reservation status (e.g., canceled vs. checkedout).



```sql
SELECT
    rs.reservation_status,
    COUNT(*) AS total_bookings,
    ROUND(AVG(m.adr), 2) AS avg_adr
FROM meal_and_stay_details m
inner join reservation_status rs
on m.Booking_id = rs.Booking_id
GROUP BY rs.reservation_status
ORDER BY avg_adr DESC;
```

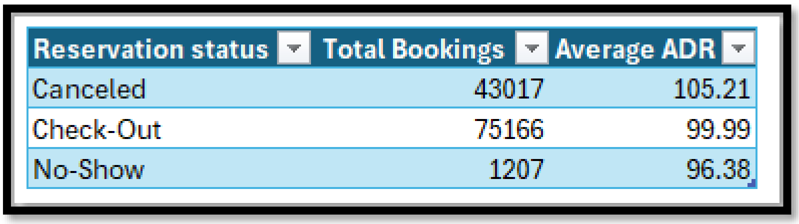
Query Explanation:
This query analyzes the relationship between booking statuses and average daily rates (ADR). Here's the breakdown:

1. **Columns Selected:**
    - **rs.reservation_status:** The current status of each reservation (e.g., Checked-Out, Canceled, No-Show)
    - **COUNT(*) AS total_bookings:** Counts how many bookings exist for each status
    - **ROUND(AVG(m.adr), 2) AS avg_adr:** Calculates the average daily rate for each status, rounded to 2 decimal places
2. **Tables Joined:**
    - **meal_and_stay_details (aliased as 'm'**): Contains ADR (Average Daily Rate) information
    - **reservation_status (aliased as 'rs'):** Contains booking status information
3. **Join Condition:**
    - Connected via Booking_id to match rate information with booking status
4. **Grouping:**
    - Results are grouped by reservation status to aggregate the metrics
5. **Sorting:**
    - Orders results by average ADR in descending order (highest-priced bookings first)

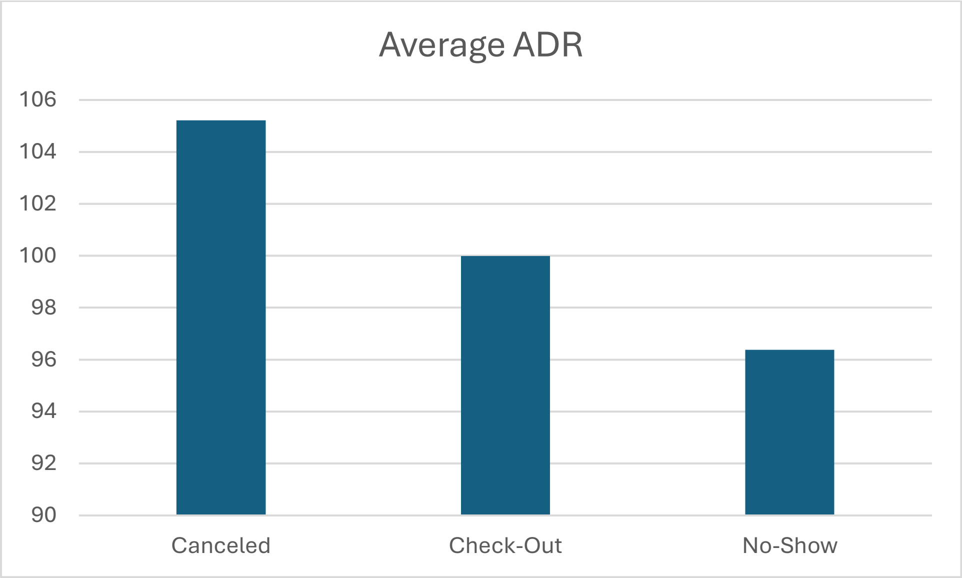
Purpose:
This analysis helps understand:

- Which booking statuses are associated with higher/lower room rates
- Whether canceled bookings tend to be higher or lower value than completed stays
- The revenue impact of different booking outcomes

Query Result :



Reservation status	Total Bookings	Average ADR
Canceled	43017	105.21
Check-Out	75166	99.99
No-Show	1207	96.38


Average ADR
Canceled
Check-Out
No-Show