**Target- SQL**

**Topic: SQL**

1. **Import the dataset and do usual exploratory analysis steps like checking the structure & characteristics of the dataset:**
   1. A. Data type of all columns in the “customers” table.

**QUERY**

select

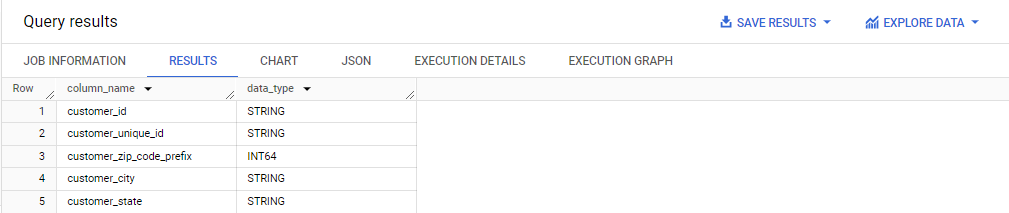
column\_name,data\_type

from TargetSql.INFORMATION\_SCHEMA.COLUMNS

WHERE TABLE\_NAME='customers'

**OUTPUT**

**OUTPUT**



**INFERENCE**

This query is expected to return information about the columns in the customers table, including their names (column\_name) and data types (data\_type).

The INFORMATION\_SCHEMA.COLUMNS view in SQL databases provides metadata about columns in database tables. By filtering on TABLE\_NAME='customers', this query specifically targets the customers table.

* 1. Get the time range between which the orders were placed.

**QUERY**

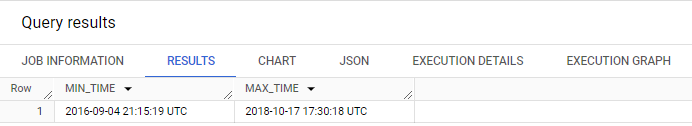
SELECT

MIN (order\_purchase\_timestamp) AS MIN\_TIME,

MAX (order\_purchase\_timestamp) AS MAX\_TIME

FROM `TargetSql.orders`

**OUTPUT**



**INFERENCE:**

**INFERENCE:**

This SQL query retrieves the earliest (MIN\_TIME) and latest (MAX\_TIME) order purchase timestamps from the orders table.

It selects the minimum order purchase timestamp (MIN(order\_purchase\_timestamp)) and labels it as MIN\_TIME.

It selects the maximum order purchase timestamp (MAX(order\_purchase\_timestamp)) and labels it as MAX\_TIME.

Both values are retrieved from the orders table

1. Count the Cities & States of customers who ordered during the given period.

**QUERY:**

SELECT

    COUNT(DISTINCT c.customer\_city) AS CustomerCity,

    COUNT(DISTINCT c.customer\_state) AS CustomerState

FROM

    TargetSql.customers c

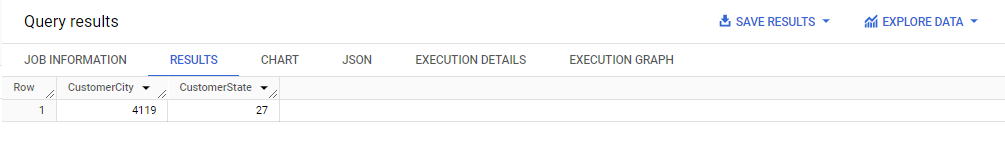
JOIN

    TargetSql.orders o ON c.customer\_id = o.customer\_id

WHERE

    o.order\_purchase\_timestamp BETWEEN '2016-01-01' AND '2018-12-31';

**OUTPUT:**

****

**INFERENCE:**

This SQL query calculates the count of distinct customer cities and distinct customer states for orders placed between January 1, 2016, and December 31, 2018.

It selects the count of distinct customer cities (CustomerCity) and the count of distinct customer states (CustomerState).

It joins the customers table (c) with the orders table (o) based on the customer ID (customer\_id).

It filters the orders based on the purchase timestamp, selecting only those orders placed between January 1, 2016, and December 31, 2018.

1. **In-depth Exploration:**
   1. Is there a growing trend in the no. of orders placed over the past years?

**QUERY**

WITH OrderDetails AS (

    SELECT

        EXTRACT(YEAR FROM order\_purchase\_timestamp) AS Year,

        EXTRACT(MONTH FROM order\_purchase\_timestamp) AS Month,

        COUNT(\*) AS no\_of\_Order

    FROM

        `TargetSql.orders`

    WHERE

        order\_status != 'canceled'

    GROUP BY

        1, 2

)

SELECT

    Year,

    Month,

    no\_of\_Order,

    CONCAT(

        ROUND(((no\_of\_Order - AVG(no\_of\_Order) OVER (ORDER BY Year, Month ROWS BETWEEN 2 PRECEDING AND CURRENT ROW)) \* 100.0 / AVG(no\_of\_Order) OVER (ORDER BY Year, Month ROWS BETWEEN 2 PRECEDING AND CURRENT ROW)), 2),

        '%'

    ) AS GrowthPercentage

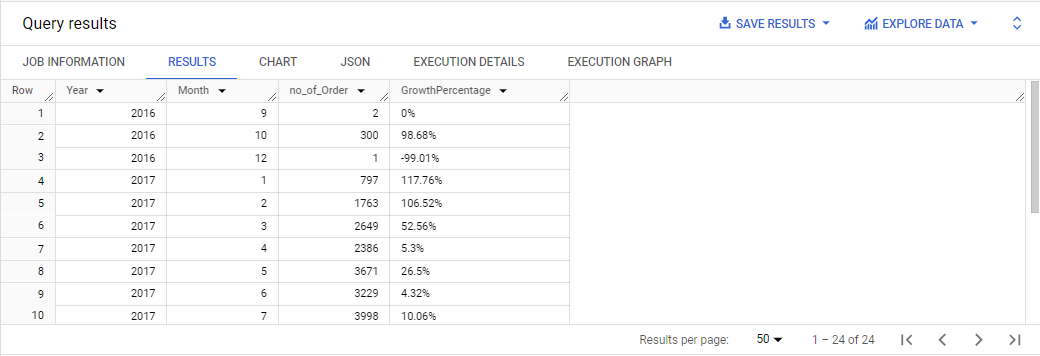
FROM

    OrderDetails

ORDER BY

    Year, Month;

**OUTPUT**

****

**INFERENCE:**

This SQL query computes the growth percentage of monthly orders compared to the three previous months, using a moving average window function.

OrderDetails :

It extracts the year and month from the order purchase timestamp and counts the number of orders for each month.

Orders with a status of "canceled" are excluded.

Results are grouped by year and month.

Main Query:

It selects the year, month, and number of orders from the OrderDetails CTE.

It calculates the growth percentage using a window function.

AVG(no\_of\_Order) OVER (ORDER BY Year, Month ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) computes the moving average of orders for the current month and the two preceding months.

((no\_of\_Order - AVG(no\_of\_Order)) \* 100.0 / AVG(no\_of\_Order))

computes the growth percentage by comparing the current month's order count with the moving average.

The result is rounded to two decimal places and concatenated with a percentage sign.

Results are ordered by year and month.

* 1. Can we see some kind of monthly seasonality in terms of the no. of orders being placed?

**QUERY:**

WITH OrderDetails AS (

    SELECT

        EXTRACT(YEAR FROM order\_purchase\_timestamp) AS Year,

        EXTRACT(MONTH FROM order\_purchase\_timestamp) AS Month,

        COUNT(\*) AS no\_of\_Order

    FROM

        `TargetSql.orders`

    WHERE

        order\_status != 'canceled'

    GROUP BY

        Year, Month

    ORDER BY

        Year, Month

)

SELECT

    CONCAT(PeriodOfTime, '----', MNTH) AS Monthly\_Seasonality,

    OrderMax

FROM (

    SELECT

        CONCAT(Year, '----', Month) AS PeriodOfTime,

        SUM(no\_of\_Order) AS OrderMax,

        CASE

            WHEN Month = 1 THEN 'JAN'

            WHEN Month = 2 THEN 'FEB'

            WHEN Month = 3 THEN 'MAR'

            WHEN Month = 4 THEN 'APR'

            WHEN Month = 5 THEN 'MAY'

            WHEN Month = 6 THEN 'JUN'

            WHEN Month = 7 THEN 'JUL'

            WHEN Month = 8 THEN 'AUG'

            WHEN Month = 9 THEN 'SEP'

            WHEN Month = 10 THEN 'OCT'

            WHEN Month = 11 THEN 'NOV'

            WHEN Month = 12 THEN 'DEC'

        END AS MNTH

    FROM

        OrderDetails

    GROUP BY

        Year, Month

)

ORDER BY

    OrderMax desc;

**OUTPUT**

**INFERENCE:**

This SQL query analyzes the monthly seasonality of orders by finding the maximum number of orders for each month over the years.

OrderDetails :

It extracts the year and month from the order purchase timestamp and counts the number of orders for each month.

Orders with a status of "canceled" are excluded.

Results are grouped by year and month and ordered accordingly.

Main Query:

It selects the concatenated year and month as "PeriodOfTime" and the sum of the number of orders as "OrderMax" for each month.

It also assigns month names based on the month number.

Results are grouped by year and month, and month names are assigned.

The final result presents the monthly seasonality with the concatenated year and month and the corresponding maximum number of orders, ordered by the maximum number of orders in descending order.

* 1. During what time of the day, do the Brazilian customers mostly place their orders? (Dawn, Morning, Afternoon or Night)
* 0-6 hrs : Dawn
* 7-12 hrs : Mornings
* 13-18 hrs : Afternoon
* 19-23 hrs : Night

**QUERY:**

WITH TimeZone as

(

SELECT

    CASE

        WHEN EXTRACT(HOUR FROM order\_purchase\_timestamp) BETWEEN 0 AND 6 THEN

'Dawn'

        WHEN EXTRACT(HOUR FROM order\_purchase\_timestamp) BETWEEN 7 AND 12 THEN 'Mornings'

        WHEN EXTRACT(HOUR FROM order\_purchase\_timestamp) BETWEEN 13 AND 18 THEN 'Afternoon'

        WHEN EXTRACT(HOUR FROM order\_purchase\_timestamp) BETWEEN 19 AND 23 THEN 'Night'

    END AS Time\_of\_the\_day,

    COUNT(\*) AS no\_of\_Order

FROM

    `TargetSql.orders`

GROUP BY

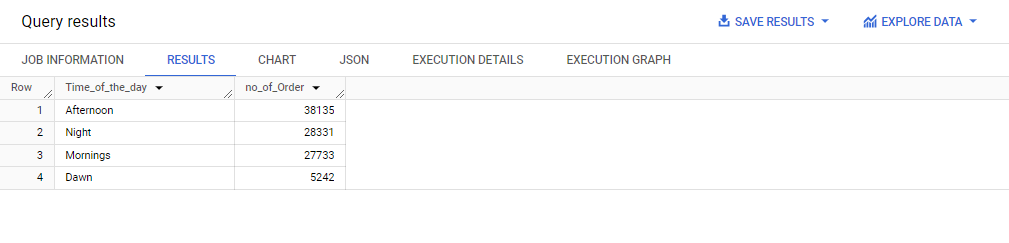
    Time\_of\_the\_day

)

SELECT \* FROM TIMEZONE

ORDER BY no\_of\_Order DESC;

**OUTPUT:**



**INFERENCE:**

This SQL query categorizes the orders based on the time of the day they were made and counts the number of orders in each category.

TimeZone:

It uses a CASE statement to categorize orders into four time periods: Dawn (00:00 - 06:00), Mornings (07:00 - 12:00), Afternoon (13:00 - 18:00), and Night (19:00 - 23:00).

The COUNT(\*) function counts the number of orders in each time period.

Results are grouped by the time of the day.

Main Query:

It selects all columns from the TimeZone CTE.

Results are ordered by the number of orders in descending order.

1. **Evolution of E-commerce orders in the Brazil region:** 
   1. Get the month on month no. of orders placed in each state.

**QUERY**:

with MonthlyOrders as

(

select

c.customer\_state as State,

format\_timestamp("%Y-%m", o.order\_purchase\_timestamp)as Year\_Month,

count(\*) as No\_of\_Order\_monthly

from TargetSql.customers c

join

TargetSql.orders o

on

c.customer\_id=o.customer\_id

where

  o.order\_status!="canceled"

group by

  customer\_state,Year\_Month

order by

  customer\_state,Year\_Month

)

select

Year\_Month,

State,

max(No\_of\_Order\_monthly)over(partition by Year\_Month) as Order\_Max,

min(No\_of\_Order\_monthly)over(partition by Year\_Month) as Order\_Min,

round(avg(No\_of\_Order\_monthly)over(partition by state order by Year\_Month),2) as Avg\_monthly\_order,

sum(No\_of\_Order\_monthly)over(partition by State)as Total\_Monthly\_order,

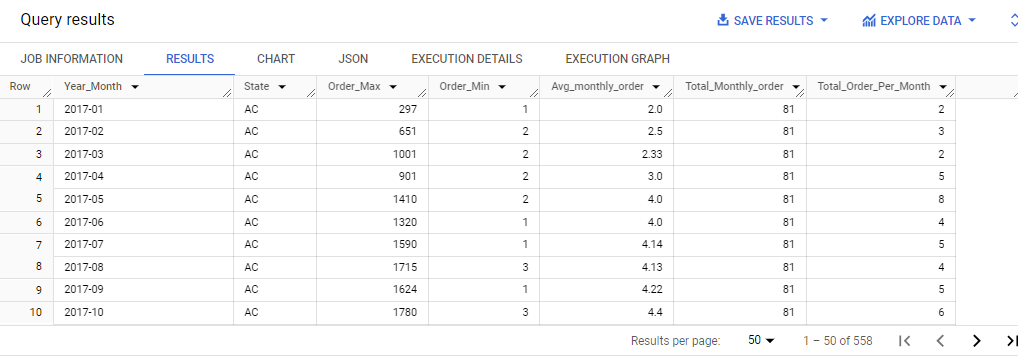
No\_of\_Order\_monthly as Total\_Order\_Per\_Month

from MonthlyOrders

group by State,Year\_Month,No\_of\_Order\_monthly

order by

  State

**OUTPUT**:

**INFERENCE**:

This SQL query seems to be analyzing monthly order statistics for each state.

MonthlyOrders: It selects the customer state (State), formats the order purchase timestamp to extract the year and month (Year\_Month), and counts the number of orders placed in each month (No\_of\_Order\_monthly).

Data is filtered to exclude orders with a status of "canceled."

Results are grouped by customer state and year-month, then ordered by customer state and year-month.

Main Query:

It selects data from the MonthlyOrders CTE.

It uses window functions to calculate various statistics:

Order\_Max: Maximum number of orders in a month for each year-month partition.

Order\_Min: Minimum number of orders in a month for each year-month partition.

Avg\_monthly\_order: Average number of orders per month for each state, ordered by year-month.

Total\_Monthly\_order: Total number of orders for each state across all months.

Total\_Order\_Per\_Month: Total number of orders for each state in each specific month.

Results are grouped by state, year-month, and the total number of orders per month, then ordered by state.

* 1. How are the customers distributed across all the states?

**QUERY**:

select

  customer\_state,

  count(customer\_id)as no\_of\_customer

from TargetSql.customers

group by

  customer\_state

order by

  customer\_state

**OUTPUT**:



\

**INFERENCE**:

This SQL query calculates the number of customers for each state.

It selects the customer state and counts the number of unique customer IDs (customer\_id) for each state.

The data is grouped by customer state.

The results are ordered by customer state in ascending order.

1. **Impact on Economy: Analyze the money movement by e-commerce by looking at order prices, freight and others.**
2. Get the % increase in the cost of orders from year 2017 to 2018 *(include months between Jan to Aug only).*

**QUERY**:

WITH Percent\_increase AS (

  SELECT

    ROUND(SUM(IF(year = 2018, Amount, 0)), 2) AS cost\_of\_2018,

    ROUND(SUM(IF(year = 2017, Amount, 0)), 2) AS cost\_of\_2017

  FROM (

    SELECT

      EXTRACT(YEAR FROM order\_purchase\_timestamp) AS Year,

      EXTRACT(MONTH FROM order\_purchase\_timestamp) AS Month,

     sum( p.payment\_value) AS Amount

    FROM

      `TargetSql.payments` p

    JOIN

      `TargetSql.orders` o

    ON

      p.order\_id = o.order\_id

    WHERE

      order\_status != "canceled" AND EXTRACT(YEAR FROM order\_purchase\_timestamp)!= 2016

    GROUP BY

      Year, Month

  ) AS t

  WHERE

    t.Month < 9  AND Year != 2016

)

SELECT

  cost\_Of\_2017 AS Amount\_Of\_2017,

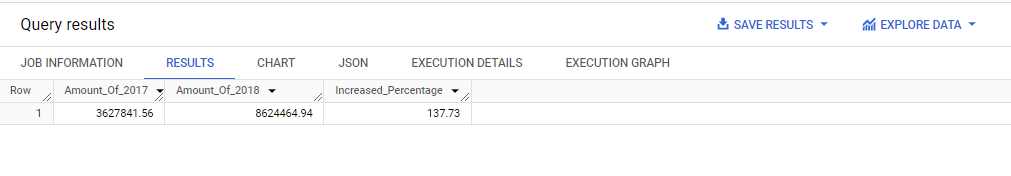
  cost\_Of\_2018 AS Amount\_Of\_2018,

  ROUND(((cost\_Of\_2018 - cost\_Of\_2017) / cost\_Of\_2017) \* 100, 2) AS Increased\_Percentage

FROM

  Percent\_increase;

**OUTPUT**



**INFERENCE:**

This SQL query seems to be calculating the percentage increase in the total payment amount between 2017 and 2018.

Percent\_increase : It aggregates the payment values for each year (2017 and 2018) and rounds the sums to 2 decimal places.

The inner query aggregates the payment values for each month, extracts the year and month from the order purchase timestamp, and sums the payment values for each year.

It filters out canceled orders and data from the year 2016.

The results are grouped by year and month.

The outer query selects only data from months before September and excludes the year 2016.

Main Query:

It selects the aggregated payment values for 2017 and 2018 from the Percent\_increase CTE.

It calculates the percentage increase in payment amount from 2017 to 2018 using the formula:

Increased Percentage=(( Amount of 2018- Amount of 2017)/ Amount of 2017)/100 .The results display the amount for 2017, the amount for 2018, and the percentage increase.

1. Calculate the Total & Average value of order price for each state.

**QUERY**:

SELECT

    c.customer\_state,

    ROUND(SUM(p.payment\_value), 2) AS Total\_amount,

    ROUND(AVG(p.payment\_value), 2) AS Avg\_amount

FROM

    TargetSql.customers c

JOIN

    TargetSql.orders o ON c.customer\_id = o.customer\_id

JOIN

    TargetSql.payments p ON o.order\_id = p.order\_id

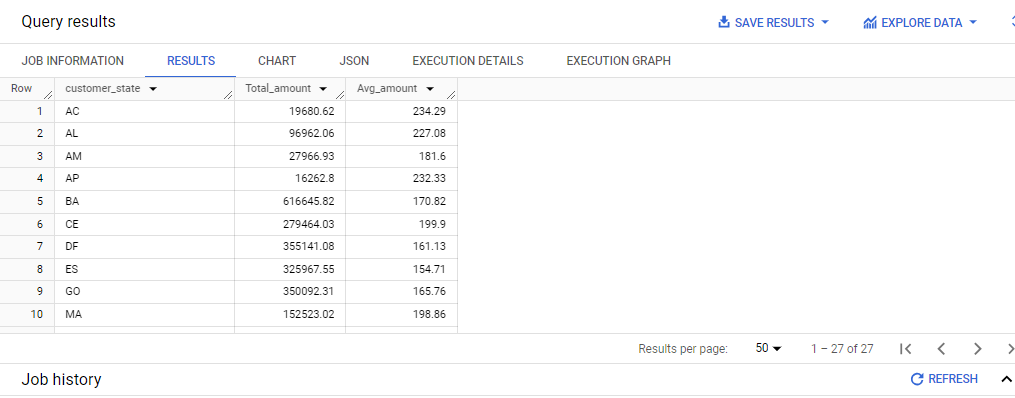
GROUP BY

    c.customer\_state

ORDER BY

    c.customer\_state;

**OUTPUT:**

****

**INFERENCE**:

This SQL query calculates the total and average payment amounts for each customer state.

It selects the customer state from the customers table.

It calculates the total payment amount for each state by summing up the payment values from the payments table, rounded to 2 decimal places.

It calculates the average payment amount for each state by averaging the payment values from the payments table, rounded to 2 decimal places.

The data is grouped by customer state.

The results are ordered by customer state.

This query provides insights into the total and average payment amounts across different customer states

1. Calculate the Total & Average value of order freight for each state.

**QUERY**:

SELECT

    c.customer\_state,

    ROUND(SUM(oi.freight\_value), 2) AS Total\_freight\_amount,

    ROUND(AVG(oi.freight\_value), 2) AS Avg\_freight\_amount

FROM

    TargetSql.customers c

JOIN

    TargetSql.orders o ON c.customer\_id = o.customer\_id

JOIN

    TargetSql.order\_items oi  ON o.order\_id=oi.order\_id

GROUP BY

    c.customer\_state

ORDER BY

    c.customer\_state

**OUTPUT**:



**INFERENCE:**

This SQL query calculates the total and average freight amounts for each customer state.

It selects the customer state from the customers table.

It calculates the total freight amount for each state by summing up the freight values from the order\_items table, rounded to 2 decimal places.

It calculates the average freight amount for each state by averaging the freight values from the order\_items table, rounded to 2 decimal places.

The data is grouped by customer state.

The results are ordered by customer state.

1. **Analysis based on sales, freight and delivery time.**
   1. Find the no. of days taken to deliver each order from the order’s purchase date as delivery time.

Also, calculate the difference (in days) between the estimated & actual delivery date of an order.

**QUERY:**

select

order\_id,

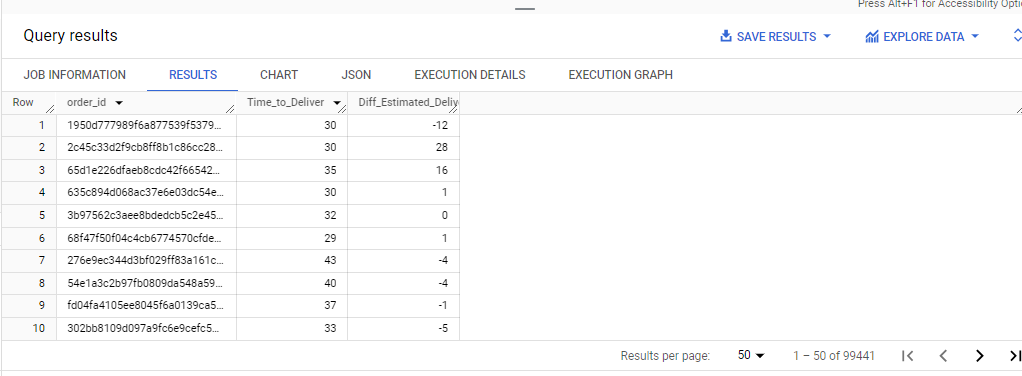
datetime\_diff(order\_delivered\_customer\_date,order\_purchase\_timestamp,day)as Time\_to\_Deliver,

datetime\_diff(order\_estimated\_delivery\_date,order\_delivered\_customer\_date,day)as Diff\_Estimated\_Delivery

from

`TargetSql.orders`

**OUTPUT:**

****

**INFERENCE:**

This SQL QUERY retrieves the order ID, the time taken to deliver the order (from purchase to delivery), and the difference between the estimated delivery date and the actual delivery date.

It selects the order ID from the orders table.

It calculates the time taken to deliver each order by finding the difference in days between the order delivered customer date and the order purchase timestamp.

It calculates the difference between the estimated delivery date and the actual delivery date for each order by finding the difference in days between the order delivered customer date and the order estimated delivery date.

* 1. Find out the top 5 states with the highest & lowest average freight value.

**QUERY:**

select State, Avg\_freight\_value

from(

(select

customer\_state as State,

max(freight\_value) as highest\_freight\_value,

round(avg(freight\_value),2) as Avg\_freight\_value

from `TargetSql.customers` c

join

  `TargetSql.orders` o

on

  c.customer\_id=o.customer\_id

join

  `TargetSql.order\_items` oi

on

  o.order\_id=oi.order\_id

group by

  customer\_state

order by

  Avg\_freight\_value desc

limit 5)

union all

(select

customer\_state as State,

min(freight\_value) as lowest\_freight\_value,

round(avg(freight\_value),2) as Avg\_freight\_value

from `TargetSql.customers` c

join

  `TargetSql.orders` o

on

  c.customer\_id=o.customer\_id

join

  `TargetSql.order\_items` oi

on

  o.order\_id=oi.order\_id

group by

  customer\_state

order by

  Avg\_freight\_value asc

limit 5)

)as t;

@or

WITH AvgFreightValues AS (

    SELECT

        customer\_state,

        ROUND(AVG(freight\_value), 2) AS avg\_freight\_value,

        ROW\_NUMBER() OVER (ORDER BY ROUND(AVG(freight\_value), 2) DESC) AS desc\_avg\_freight\_value,

        ROW\_NUMBER() OVER (ORDER BY ROUND(AVG(freight\_value), 2)) AS asc\_avg\_freight\_value

     FROM

        `TargetSql.customers` c

    JOIN

        `TargetSql.orders` o ON c.customer\_id = o.customer\_id

    JOIN

        `TargetSql.order\_items` oi ON o.order\_id = oi.order\_id

    GROUP BY

        1

)

SELECT

    customer\_state,

    CASE WHEN desc\_avg\_freight\_value BETWEEN 1 AND 5 THEN avg\_freight\_value ELSE 0 END AS highest\_freight\_value,

    CASE WHEN asc\_avg\_freight\_value BETWEEN 1 AND 5 THEN avg\_freight\_value ELSE 0 END AS lowest\_freight\_value

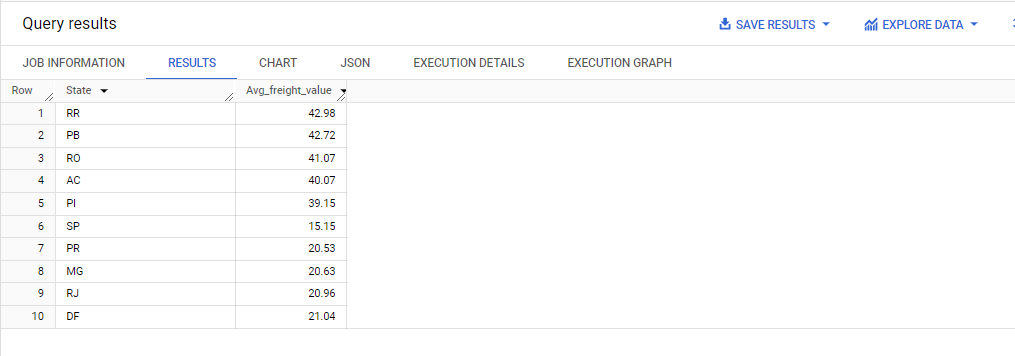
FROM

    AvgFreightValues

WHERE

    desc\_avg\_freight\_value <= 5 OR asc\_avg\_freight\_value <= 5;

**OUTPUT:**

****

****

**INFERENCE:**

This SQL QUERY retrieves the average freight value for the top 5 states with the highest average freight values and the bottom 5 states with the lowest average freight values.

The inner **QUERY** selects the customer state, calculates the highest or lowest freight value for each state, and computes the average freight value for each state. It does this by joining the customers, orders, and order\_items tables and grouping the results by customer state.

The results are split into two parts using UNION ALL the first part selects the top 5 states with the highest average freight values, ordering them by descending average freight value and limiting the result to 5 rows.

The second part selects the bottom 5 states with the lowest average freight values, ordering them by ascending average freight value and limiting the result to 5 rows.

The outer QUERY selects the state and average freight value from the combined results of the two parts.

* 1. Find out the top 5 states with the highest & lowest average delivery time.

**QUERY:**

WITH AVGdEL AS (

    SELECT

        State,

        "FAST" AS value,

        AVG(Delivery\_Time) AS Avg\_Delivery\_Time,

        DENSE\_RANK() OVER (ORDER BY AVG(Delivery\_Time) DESC) AS Rnk

    FROM

        (

            SELECT

                customer\_state AS State,

                datetime\_diff(order\_delivered\_customer\_date, order\_purchase\_timestamp, DAY) AS Delivery\_Time

            FROM

                `TargetSql.customers` c

            JOIN

                `TargetSql.orders` o ON c.customer\_id = o.customer\_id

        )

    GROUP BY

        State

    UNION ALL

    SELECT

        State,

        "SLOW" AS value,

        AVG(Delivery\_Time) AS Avg\_Delivery\_Time,

        DENSE\_RANK() OVER (ORDER BY AVG(Delivery\_Time) ASC) AS Rnk

    FROM

        (

            SELECT

                customer\_state AS State,

                datetime\_diff(order\_delivered\_customer\_date, order\_purchase\_timestamp, DAY) AS Delivery\_Time

            FROM

                `TargetSql.customers` c

            JOIN

                `TargetSql.orders` o ON c.customer\_id = o.customer\_id

        )

    GROUP BY

        State

)

SELECT

    State,

    CONCAT(value, " - ", Rnk) AS OrderList,

    Avg\_Delivery\_Time

FROM

    AVGdEL

WHERE

    Rnk <= 5

ORDER BY

    OrderList;

**OUTPUT:**



**INFERENCE**:

This SQL QUERY appears to be calculating the average delivery time for orders grouped by state, then categorizing them as either "FAST" or "SLOW" based on whether the delivery time is above or below the overall average delivery time. Additionally, it ranks these states based on their average delivery times.

The inner **QUERY** calculates the delivery time for each order by subtracting the order purchase timestamp from the order delivered customer date.

The outer QUERY groups the results by state, calculates the average delivery time for each state, and assigns a rank to each state based on its average delivery time.

The results are split into two parts using UNION ALL:

The first part calculates the average delivery time for each state where the delivery time is considered "FAST" (below the overall average delivery time).

The second part calculates the average delivery time for each state where the delivery time is considered "SLOW" (above the overall average delivery time).

* 1. Find out the top 5 states where the order delivery is really fast as compared to the estimated date of delivery.

**QUERY:**

WITH Avg\_Delivery AS (

    SELECT

        customer\_state AS State,

ROUND(AVG(DATE\_DIFF(DATE(order\_estimated\_delivery\_date), DATE(order\_delivered\_customer\_date), DAY)), 0)

AS Avg\_DiffEstimated\_delivery,

ROW\_NUMBER() OVER (ORDER BY AVG(DATE\_DIFF(DATE(order\_estimated\_delivery\_date),

DATE(order\_delivered\_customer\_date), DAY)) DESC) AS Fast\_Delivery

    FROM

        `TargetSql.customers` c

    JOIN

        `TargetSql.orders` o ON c.customer\_id = o.customer\_id

    JOIN

        `TargetSql.order\_items` oi ON o.order\_id = oi.order\_id

    WHERE

        order\_delivered\_customer\_date IS NOT NULL

    GROUP BY

        customer\_state

)

SELECT

    State,

    Avg\_DiffEstimated\_delivery

FROM

    Avg\_Delivery

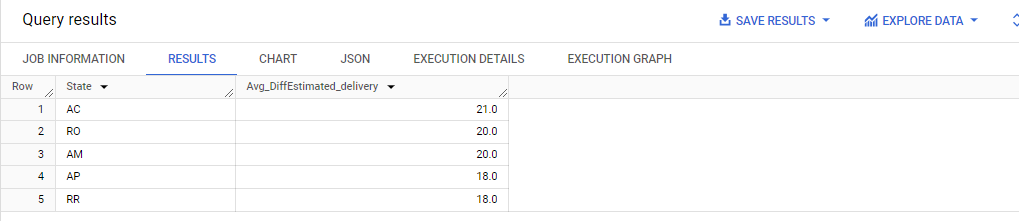
WHERE

    Fast\_Delivery <= 5

ORDER BY

    Avg\_DiffEstimated\_delivery DESC;

**OUTPUT**

****

**INFERENCE**:

This SQL code calculates the average delivery time for each state and identifies the states with the fastest delivery times.

Average Delivery Time Calculation:

The code calculates the average delivery time for each state.

It subtracts the delivery date from the estimated delivery date to find the difference in days.

Fastest Delivery Identification:

It assigns a ranking to each state based on the average delivery time, with 1 being the fastest.

Only the top 5 states with the fastest delivery times are selected.

The OUTPUT displays the state names and their corresponding average delivery times.

States are sorted based on their average delivery times in descending order.

This analysis helps identify states with efficient delivery services, which can be beneficial for customer satisfaction and retention. Businesses can focus on optimizing logistics and delivery operations in states with longer delivery times to improve overall service quality.

1. Analysis based on the payments:
2. **Find the month on month no. of orders placed using different payment types.**

**QUERY:**

WITH Order\_Diff\_Payment AS (

    SELECT

        EXTRACT(YEAR FROM order\_purchase\_timestamp) AS Year,

        EXTRACT(MONTH FROM order\_purchase\_timestamp) AS Month,

        p.payment\_type,

        COUNT(DISTINCT o.order\_id) AS No\_of\_Payments

    FROM

        `TargetSql.orders` o

    JOIN

        `TargetSql.payments` p ON p.order\_id = o.order\_id

    GROUP BY

        Year, Month, p.payment\_type

    ORDER BY

        Year, Month, p.payment\_type

)

SELECT

    Year,

    Month,

    payment\_type,

    No\_of\_Payments

FROM

    Order\_Diff\_Payment

ORDER BY

    Year, Month;

@or

WITH Order\_Diff\_Payment AS (

    SELECT

        EXTRACT(YEAR FROM order\_purchase\_timestamp) AS Year,

        EXTRACT(MONTH FROM order\_purchase\_timestamp) AS Month,

        p.payment\_type,

        COUNT(DISTINCT o.order\_id) AS No\_of\_Payments

    FROM

        `TargetSql.orders` o

    JOIN

        `TargetSql.payments` p ON p.order\_id = o.order\_id

    GROUP BY

        Year, Month, p.payment\_type

    ORDER BY

        Year, Month, p.payment\_type

),

Order\_Diff\_Payment\_With\_Lag AS (

    SELECT

        Year,

        Month,

        payment\_type,

        No\_of\_Payments,

        LAG(No\_of\_Payments, 1) OVER (PARTITION BY payment\_type ORDER BY Year, Month) AS lag\_No\_of\_Payments

    FROM

        Order\_Diff\_Payment

)

SELECT

    Year,

    Month,

    payment\_type,

    No\_of\_Payments,

    lag\_No\_of\_Payments

FROM

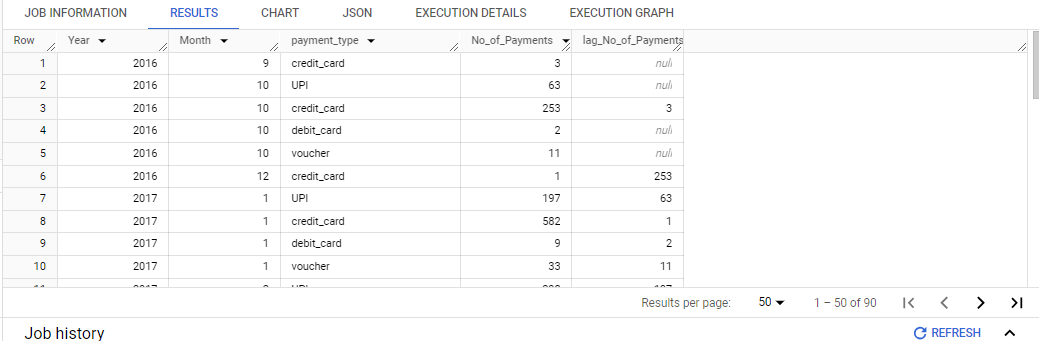
    Order\_Diff\_Payment\_With\_Lag

ORDER BY

    Year, Month;

**OUTPUT**:





**INFERENCE**:

This SQL code analyzes the distribution of orders based on payment types over different months and years.

**Data Extraction:**

The code retrieves data from the orders and payments tables.

**Payment Type Analysis:**

It examines which payment methods are used for orders.

The count of distinct order IDs is calculated for each combination of year, month, and payment type.

**Monthly and Yearly Analysis:**

Orders are grouped by year and month to understand the distribution over time.

1. **Find the no. of orders placed on the basis of the payment installments that have been paid.**

**QUERY:**

SELECT

    payment\_type,

    payment\_installments,

    COUNT(DISTINCT order\_id) AS NO\_OF\_orders

FROM

    `TargetSql.payments`

WHERE

    payment\_installments != 0

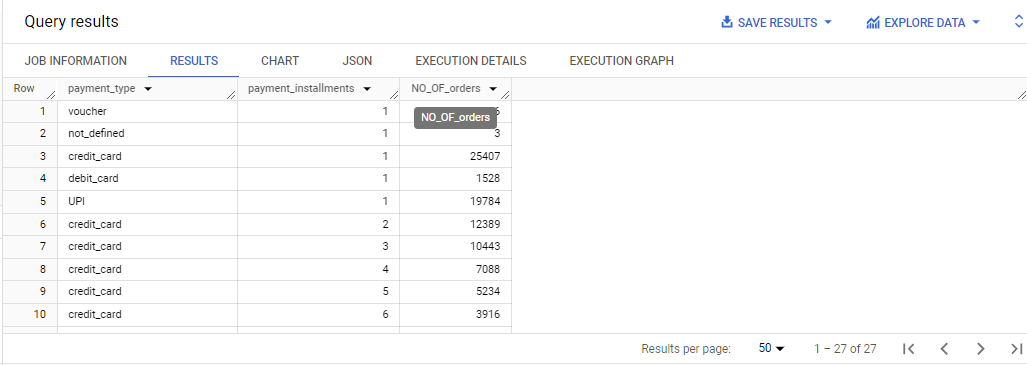
GROUP BY

    payment\_installments, payment\_type

ORDER BY

    payment\_installments;

**OUTPUT**:



**INFERENCE**:

The SQL code helps to analyze the distribution of orders based on two main factors: payment types and the number of payment installments.

**Payment Type Analysis:**

It shows which payment methods are commonly used for orders.

**Payment Installments Analysis:**

It reveals how many payment installments are associated with each order, excluding orders with zero installments.

**Order Distribution:**

It counts the number of orders for each combination of payment type and installment configuration