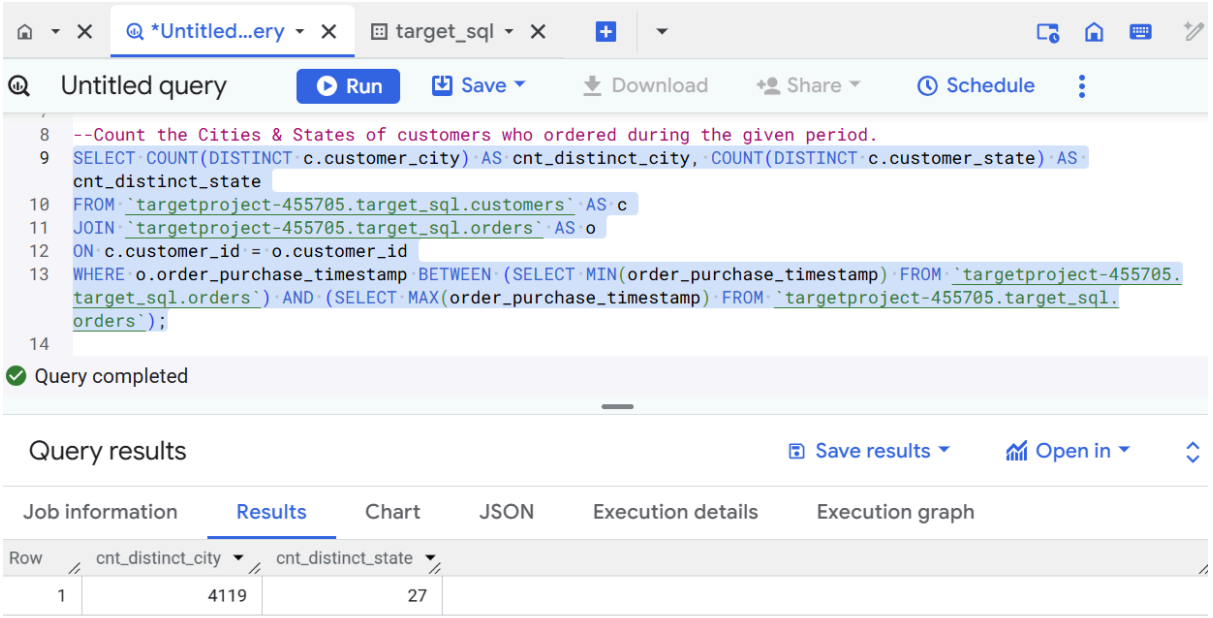
Business Case: Target_SQL:

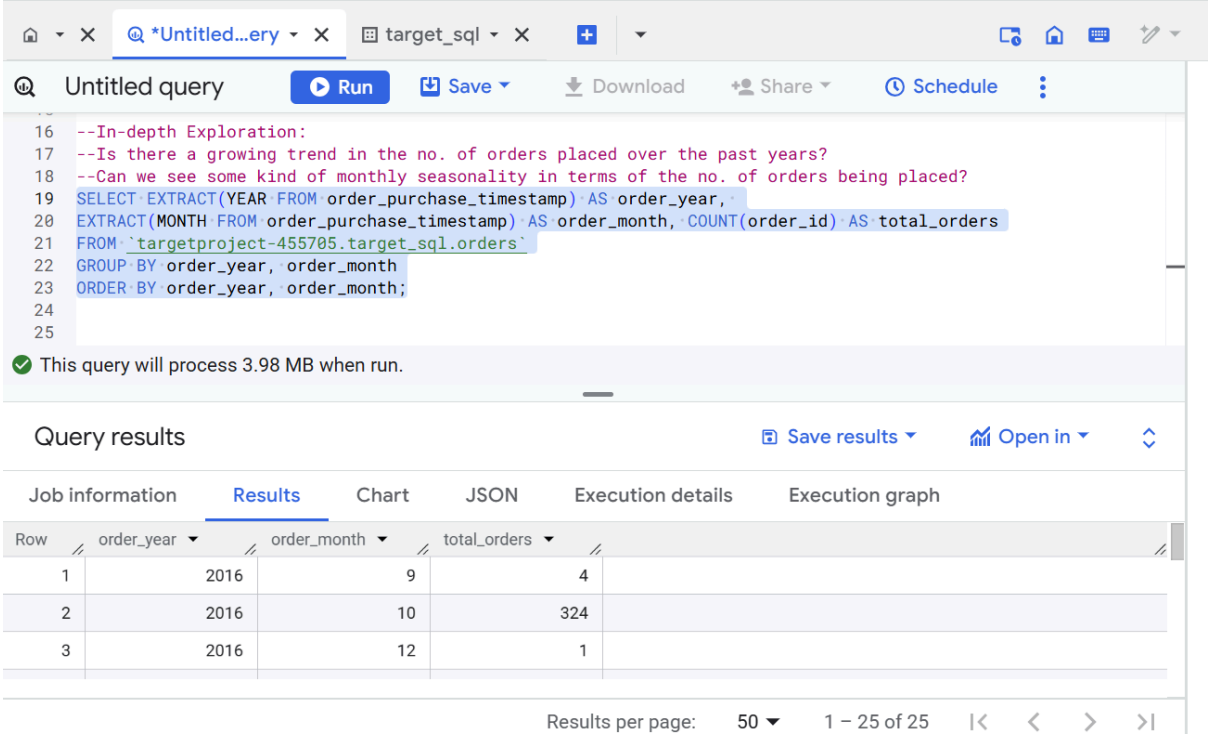
Import the dataset and do usual exploratory analysis steps like checking the structure & characteristics of the dataset a. Data type of columns in a table b. Time period for which the data is given c. Cities and States of customers ordered during the given period

```
SELECT *
FROM `targetproject-455705.target_sql.customers`;

SELECT *
FROM `targetproject-455705.target_sql.orders`;

--Count the Cities & States of customers who ordered during the given period.
SELECT COUNT(DISTINCT c.customer_city) AS cnt_distinct_city, COUNT(DISTINCT
c.customer_state) AS cnt_distinct_state
FROM `targetproject-455705.target_sql.customers` AS c
JOIN `targetproject-455705.target_sql.orders` AS o
ON c.customer_id = o.customer_id
WHERE o.order_purchase_timestamp BETWEEN (SELECT MIN(order_purchase_timestamp) FROM
`targetproject-455705.target_sql.orders`) AND (SELECT MAX(order_purchase_timestamp)
FROM `targetproject-455705.target_sql.orders`);
```




```
--In-depth Exploration:
--Is there a growing trend in the no. of orders placed over the past years?
--Can we see some kind of monthly seasonality in terms of the no. of orders being
placed?
SELECT EXTRACT(YEAR FROM order_purchase_timestamp) AS order_year,
EXTRACT(MONTH FROM order_purchase_timestamp) AS order_month, COUNT(order_id) AS
total_orders
```

```
FROM `targetproject-455705.target_sql.orders`
GROUP BY order_year, order_month
ORDER BY order_year, order_month;
```



Observations:

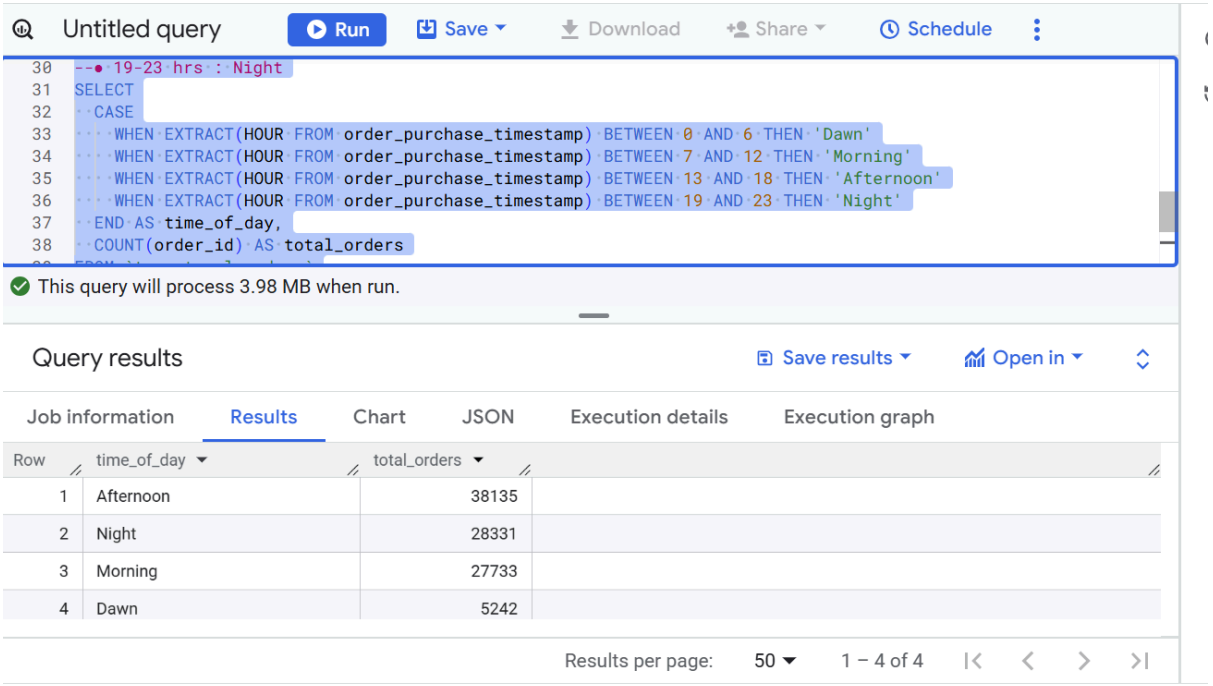
1. **Initial phase** (2016):
    - Very few orders. (only 4, 324, and 1 in Sep, Oct, Dec 2016).
    - Probably the system/platform was just launched or it was in early testing phase.
2. **2017 - Clear Growth**:
    - From Jan 2017 onwards, **orders start increasing sharply**.
    - Month-over-month, there is a steady rise.
    - **Peak months:**
        - November 2017 (7544 orders) – 🔥 highest in 2017.
        - December 2017 also strong (5673 orders).

    - So you can **already sense seasonality – more orders towards year-end (Nov-Dec)**.

3. **2018 - Sustained High Activity**:
    - Jan 2018 starts strong (7269 orders) – even higher than Dec 2017.
    - High orders continue in Feb, Mar, Apr, May, Aug.
    - Again **dips slightly after mid-year** (Jun-Jul-Aug moderate).
    - **Sept and Oct 2018** show **very low numbers (16, 4) – this looks suspicious**.
        - Maybe the data collection stopped or something went wrong?
4. **Seasonality Insight**:
    - **Orders peak between October - January** (festive seasons, holiday shopping).
    - **Spring (Feb, Mar, Apr)** is also strong.
    - **Summer months (Jun, Jul)** slightly lower but still healthy.
    - **Sept-Oct of 2018** specifically looks like missing/broken data.

```sql
--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
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
'Morning'
    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_day,
  COUNT(order_id) AS total_orders
```

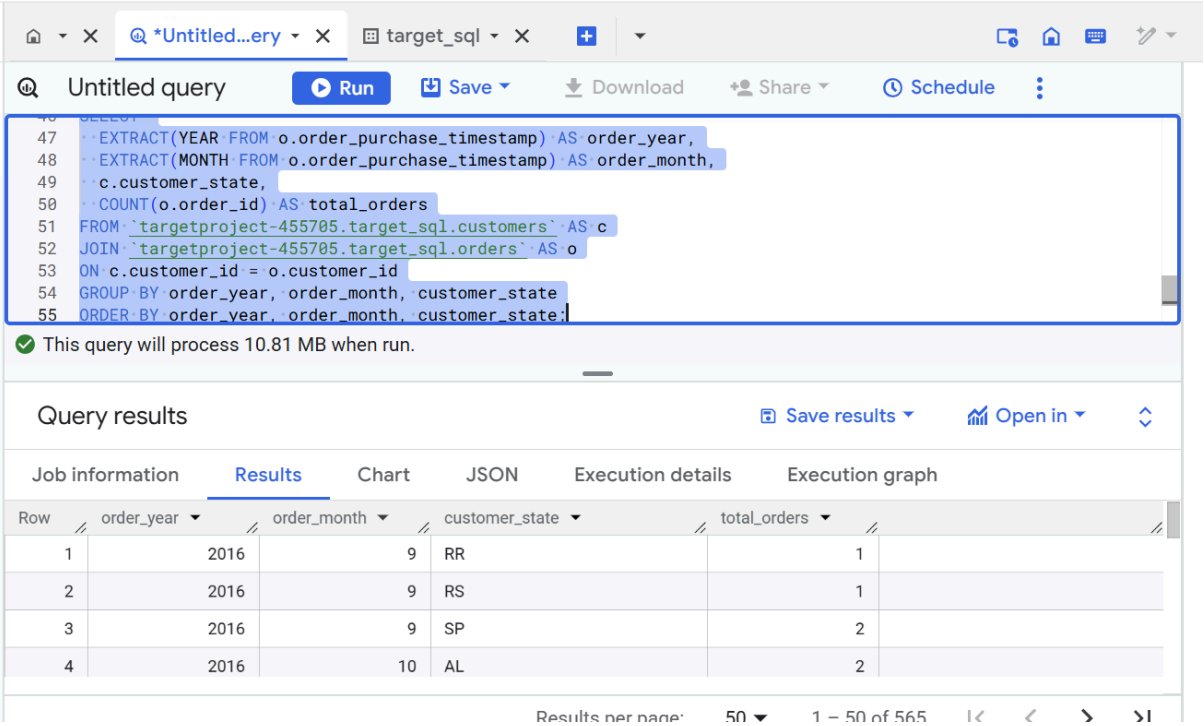
```sql
FROM `target_sql.orders`
GROUP BY time_of_day
ORDER BY total_orders DESC;
```



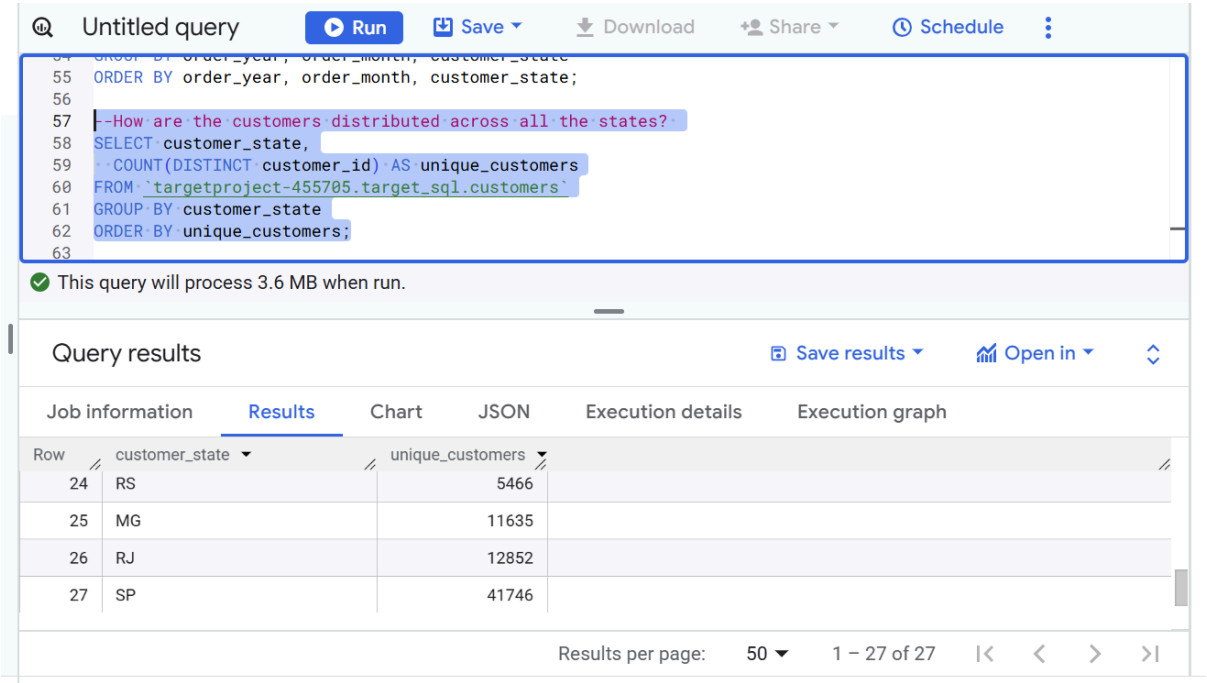
Observations:

1. **Afternoon (13:00–18:00) is the peak time** for Brazilian customers:
    - **Highest** number of orders (38,135).
    - Brazilians seem most active in the afternoon for shopping or placing orders.
2. **Night (19:00–23:00)** and **Morning (7:00–12:00)** are quite close:
    - Night: 28,331 orders.
    - Morning: 27,733 orders.
    - Both are strong periods, but Night has a slight edge over Morning.
3. **Dawn (0:00–6:00) is the least active**:
    - Only 5,242 orders.
    - Expected – this is when most people are sleeping.

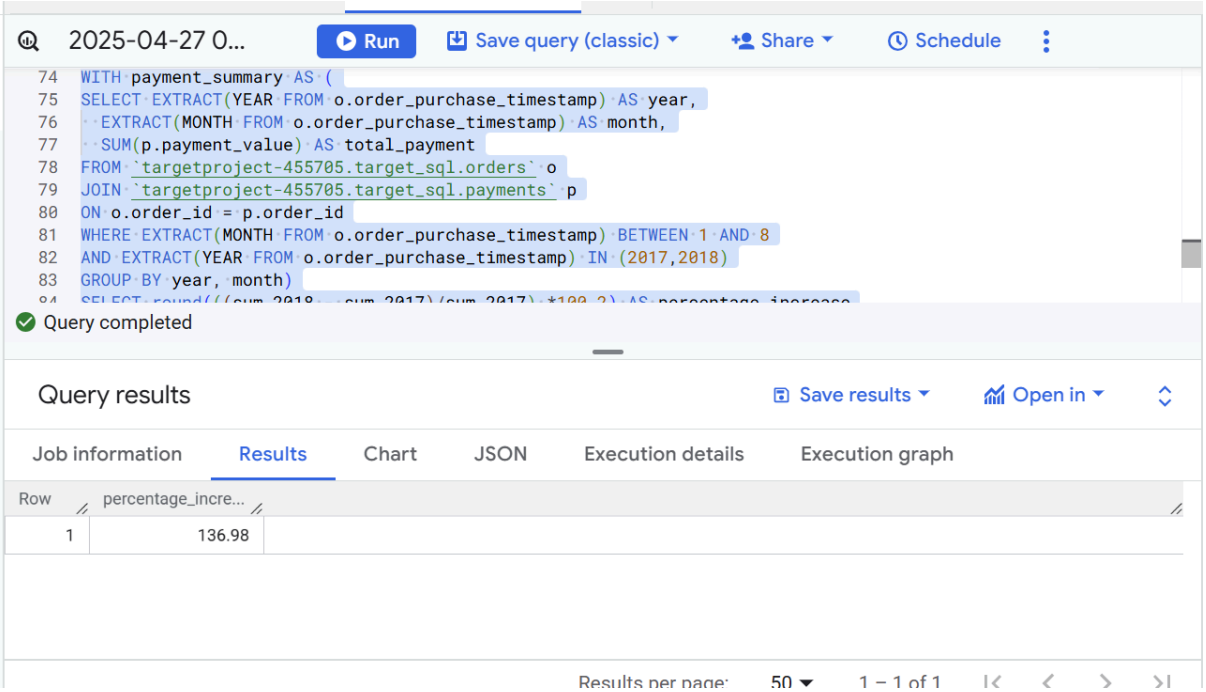
```sql
--Evolution of E-commerce orders in the Brazil region:
--Get the month on month no. of orders placed in each state.
SELECT
  EXTRACT(YEAR FROM o.order_purchase_timestamp) AS order_year,
  EXTRACT(MONTH FROM o.order_purchase_timestamp) AS order_month,
  c.customer_state,
  COUNT(o.order_id) AS total_orders
FROM `targetproject-455705.target_sql.customers` AS c
JOIN `targetproject-455705.target_sql.orders` AS o
ON c.customer_id = o.customer_id
GROUP BY order_year, order_month, customer_state
ORDER BY order_year, order_month, customer_state;
```



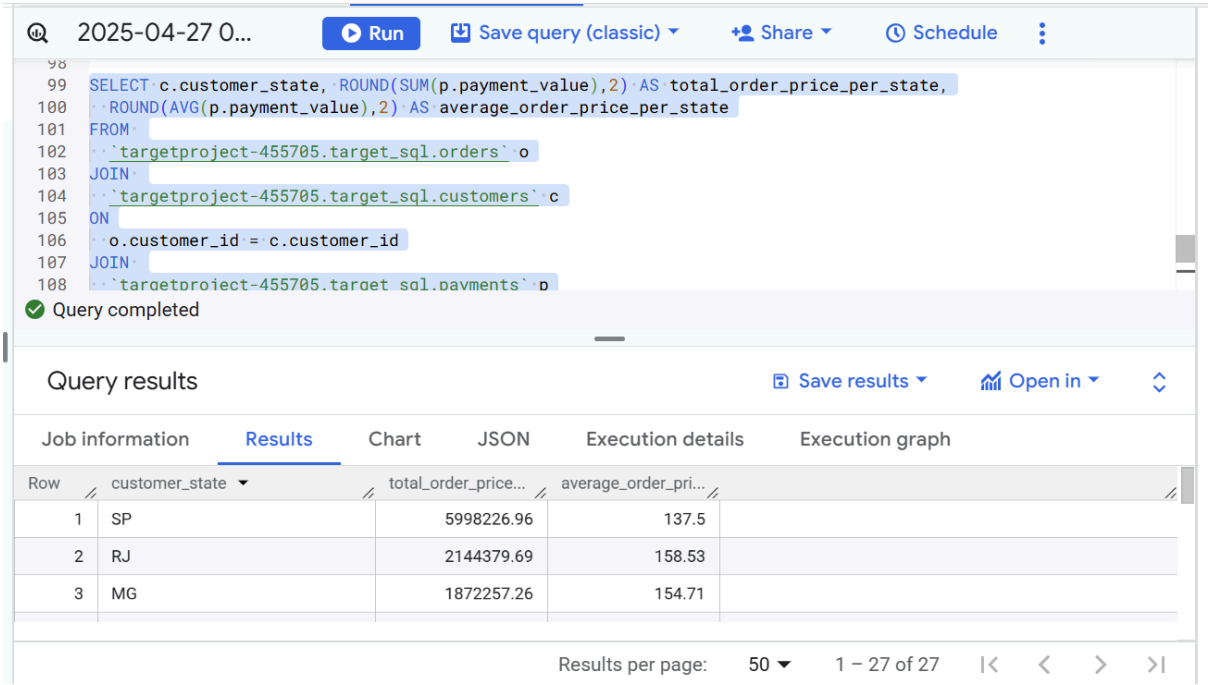

```sql
--How are the customers distributed across all the states?
SELECT customer_state,
  COUNT(DISTINCT customer_id) AS unique_customers
FROM `targetproject-455705.target_sql.customers`
GROUP BY customer_state
ORDER BY unique_customers;
```



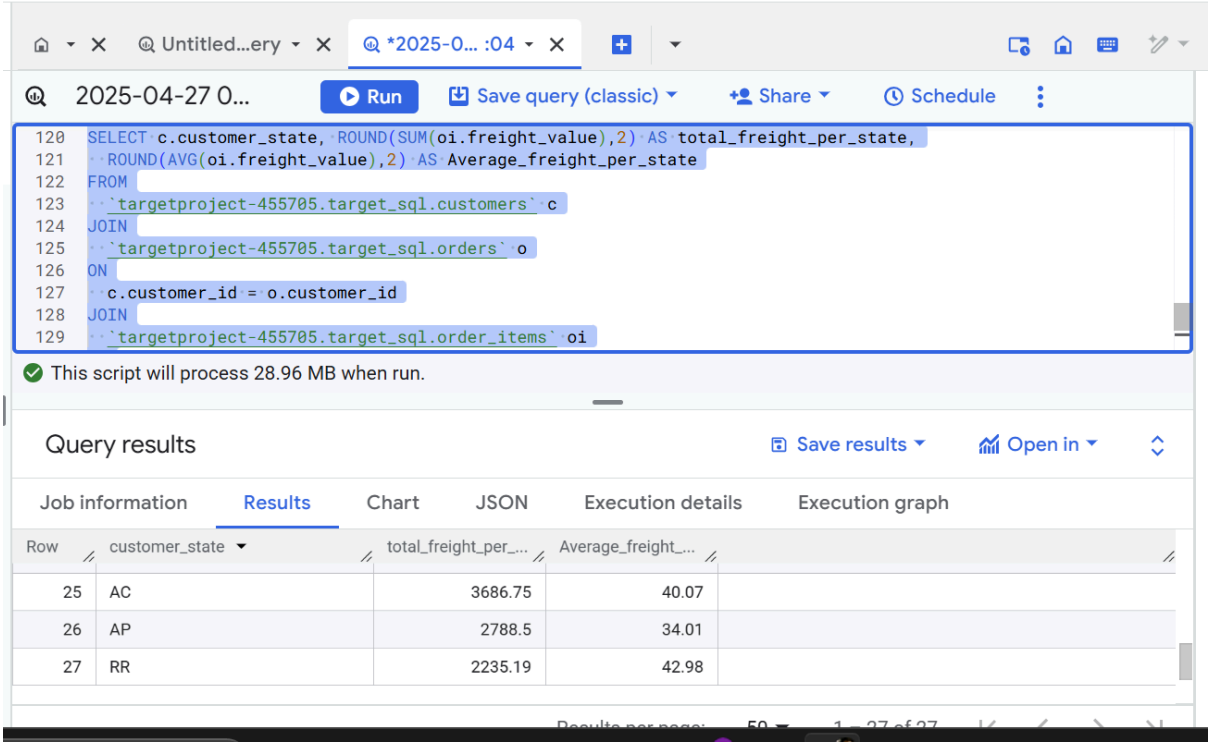

```
--Impact on Economy: Analyze the money movement by e-commerce by looking at
--order prices, freight and others.
--Get the % increase in the cost of orders from year 2017 to 2018 (include
--months between Jan to Aug only).
SELECT *
FROM `targetproject-455705.target_sql.payments`;

SELECT *
FROM `targetproject-455705.target_sql.orders`;

WITH payment_summary AS (
SELECT EXTRACT(YEAR FROM o.order_purchase_timestamp) AS year,
  EXTRACT(MONTH FROM o.order_purchase_timestamp) AS month,
  SUM(p.payment_value) AS total_payment
FROM `targetproject-455705.target_sql.orders` o
JOIN `targetproject-455705.target_sql.payments` p
ON o.order_id = p.order_id
WHERE EXTRACT(MONTH FROM o.order_purchase_timestamp) BETWEEN 1 AND 8
AND EXTRACT(YEAR FROM o.order_purchase_timestamp) IN (2017,2018)
GROUP BY year, month)
SELECT round(((sum_2018 - sum_2017)/sum_2017) *100,2) AS percentage_increase
FROM (
SELECT
  SUM(CASE WHEN year=2017 THEN total_payment ELSE 0 END ) AS sum_2017,
  SUM(CASE WHEN year=2018 THEN total_payment ELSE 0 END ) AS sum_2018
FROM payment_summary);
```



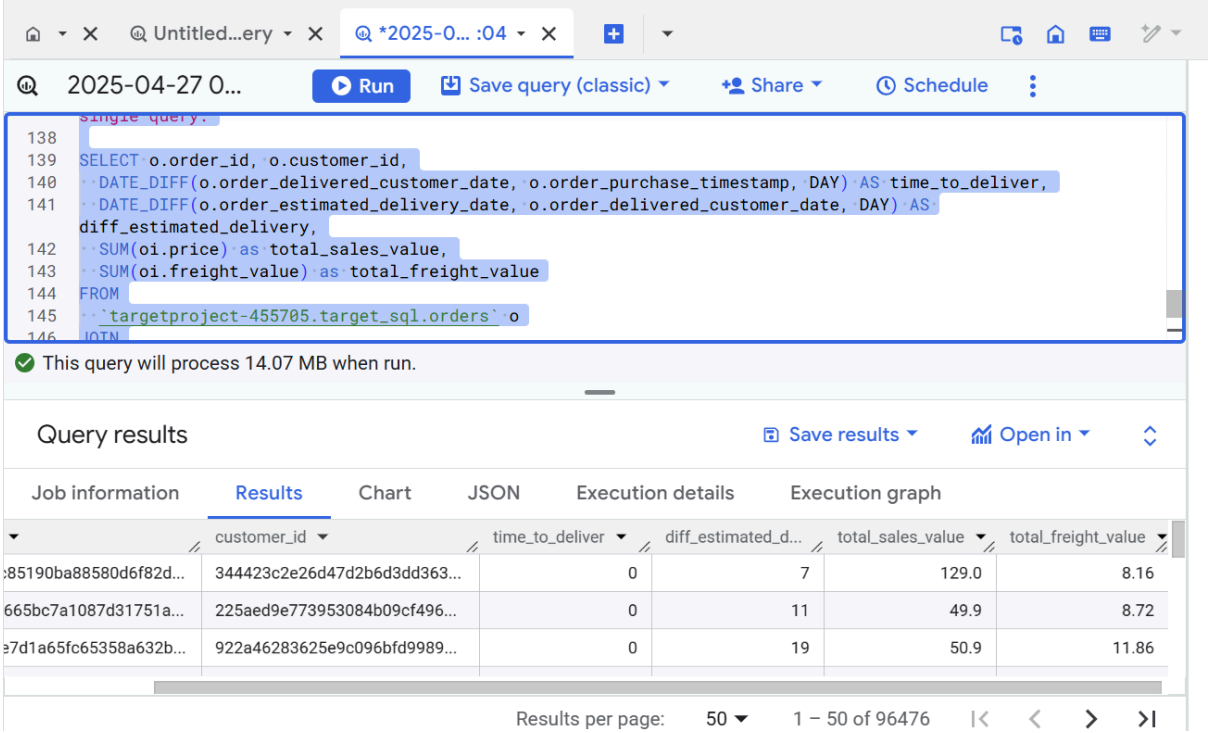

```
--Calculate the Total & Average value of order price for each state.

SELECT *
FROM `targetproject-455705.target_sql.orders`;

SELECT *
FROM `targetproject-455705.target_sql.customers`;

SELECT c.customer_state, ROUND(SUM(p.payment_value),2) AS
total_order_price_per_state,
  ROUND(AVG(p.payment_value),2) AS average_order_price_per_state
FROM
  `targetproject-455705.target_sql.orders` o
JOIN
  `targetproject-455705.target_sql.customers` c
ON
  o.customer_id = c.customer_id
JOIN
  `targetproject-455705.target_sql.payments` p
ON
  o.order_id = p.order_id
GROUP BY
  c.customer_state
ORDER BY total_order_price_per_state DESC;
```




```sql
--Calculate the Total & Average value of order freight for each state.

SELECT *
FROM `targetproject-455705.target_sql.order_items`;

SELECT c.customer_state, ROUND(SUM(oi.freight_value),2) AS total_freight_per_state,
  ROUND(AVG(oi.freight_value),2) AS Average_freight_per_state
FROM
  `targetproject-455705.target_sql.customers` c
JOIN
  `targetproject-455705.target_sql.orders` o
ON
  c.customer_id = o.customer_id
JOIN
  `targetproject-455705.target_sql.order_items` oi
ON
  o.order_id = oi.order_id
GROUP BY c.customer_state
ORDER BY total_freight_per_state DESC;
```

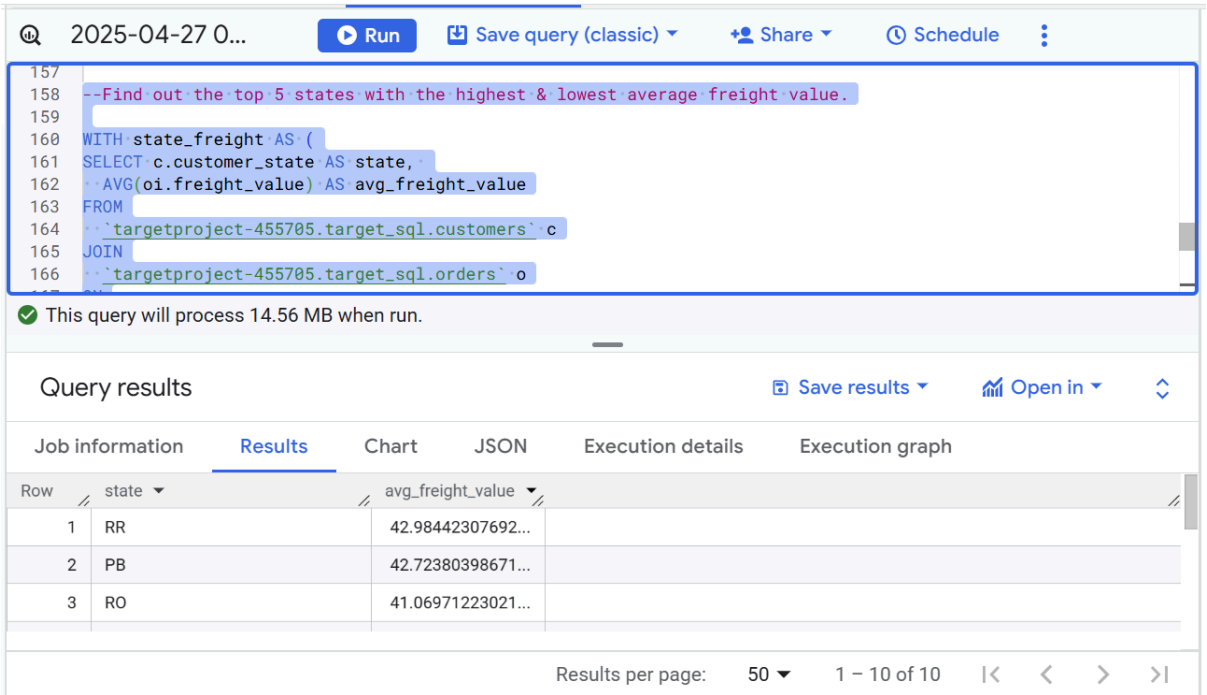


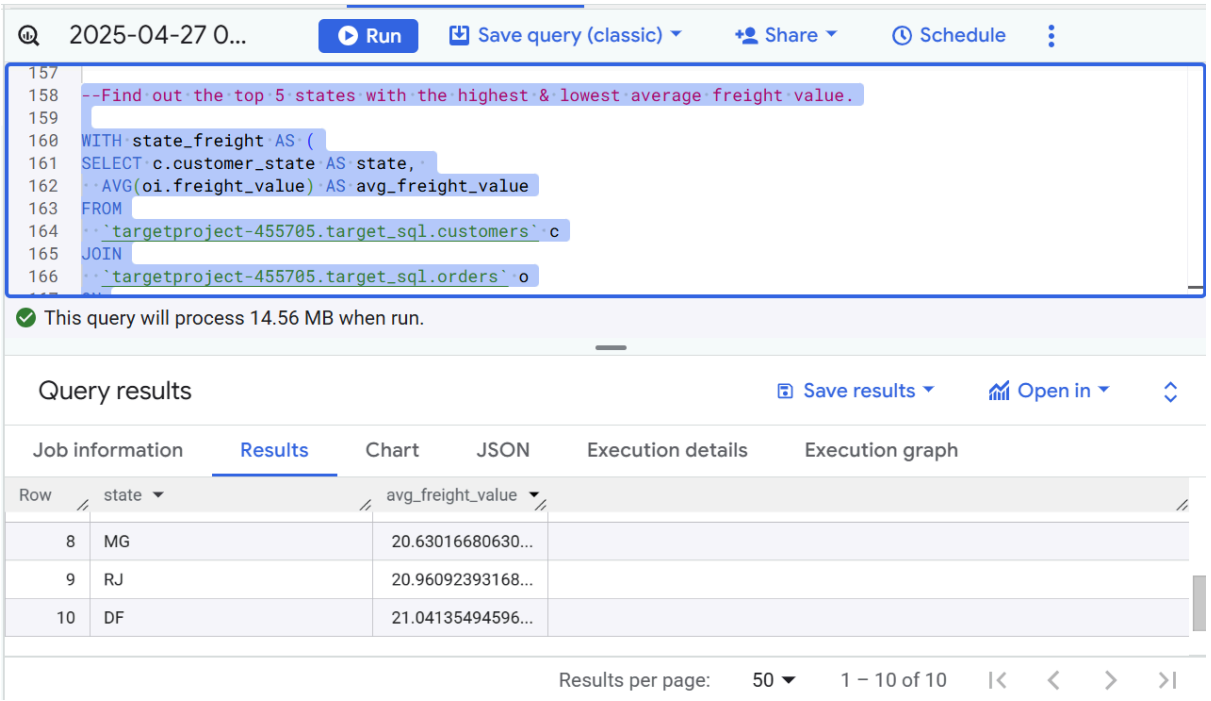

```
--Analysis based on sales, freight and delivery time.
--Find the no. of days taken to deliver each order from the order's purchase date
as delivery time.
--Also, calculate the difference (in days) between the estimated & actual delivery
date of an order in a single query.

SELECT o.order_id, o.customer_id,
  DATE_DIFF(o.order_delivered_customer_date, o.order_purchase_timestamp, DAY) AS
time_to_deliver,
  DATE_DIFF(o.order_estimated_delivery_date, o.order_delivered_customer_date, DAY)
AS diff_estimated_delivery,
  SUM(oi.price) as total_sales_value,
  SUM(oi.freight_value) as total_freight_value
FROM
  `targetproject-455705.target_sql.orders` o
JOIN
  `targetproject-455705.target_sql.order_items` oi
ON
  o.order_id = oi.order_id
WHERE
  o.order_delivered_customer_date IS NOT NULL
GROUP BY
  o.order_id, o.customer_id, o.order_purchase_timestamp,
o.order_delivered_customer_date, o.order_estimated_delivery_date
ORDER BY
  time_to_deliver;
```




```
--Find out the top 5 states with the highest & lowest average freight value.

WITH state_freight AS (
SELECT c.customer_state AS state,
  AVG(oi.freight_value) AS avg_freight_value
FROM
  `targetproject-455705.target_sql.customers` c
JOIN
  `targetproject-455705.target_sql.orders` o
ON
  c.customer_id = o.customer_id
JOIN
  `targetproject-455705.target_sql.order_items` oi
ON
  o.order_id = oi.order_id
GROUP BY
  c.customer_state)

SELECT state,
  avg_freight_value
FROM (

SELECT
  state,
  avg_freight_value
```

```sql
FROM
  state_freight
ORDER BY
  avg_freight_value ASC
LIMIT 5
)
UNION ALL

SELECT state,
  avg_freight_value
FROM (
SELECT
  state,
  avg_freight_value
FROM
  state_freight
ORDER BY
  avg_freight_value DESC
LIMIT 5)
;
```






```
--Find out the top 5 states with the highest & lowest average delivery time.

SELECT *
FROM `targetproject-455705.target_sql.orders`;


WITH state_delivery_time AS (
  SELECT
    c.customer_state AS state,
    AVG(DATE_DIFF(o.order_delivered_customer_date, o.order_purchase_timestamp,
DAY)) AS avg_delivery_time
  FROM
    `targetproject-455705.target_sql.orders` o
  JOIN
    `targetproject-455705.target_sql.customers` c
  ON
    o.customer_id = c.customer_id
  WHERE
    o.order_delivered_customer_date IS NOT NULL
  GROUP BY
    state
)

SELECT
  state,
  avg_delivery_time
FROM (
```

```sql
  SELECT
    state,
    avg_delivery_time
  FROM
    state_delivery_time
  ORDER BY
    avg_delivery_time ASC
  LIMIT 5
)

UNION ALL

SELECT
  state,
  avg_delivery_time
FROM (
  SELECT
    state,
    avg_delivery_time
  FROM
    state_delivery_time
  ORDER BY
    avg_delivery_time DESC
  LIMIT 5
)
ORDER BY
  avg_delivery_time ASC
;
```

2025-04-27 0... Run Save query (classic) Share Schedule

```
WITH state_delivery_time AS (
  SELECT
    c.customer_state AS state,
    AVG(DATE_DIFF(o.order_delivered_customer_date, o.order_purchase_timestamp, DAY)) AS avg_delivery_time
  FROM
    `targetproject-455705.target_sql.orders` o
  JOIN
    `targetproject-455705.target_sql.customers` c
  ON
    o.customer_id = c.customer_id
```

This script will process 188.78 MB when run.

Query results Save results Open in

Job information | Results | Chart | JSON | Execution details | Execution graph

Row	state	avg_delivery_time
1	SP	8.298061489072...
2	PR	11.52671135486...
3	MG	11.54381329810...

Results per page: 50 1 – 10 of 10

2025-04-27 0... Run Save query (classic) Share Schedule

```
WITH state_delivery_time AS (
  SELECT
    c.customer_state AS state,
    AVG(DATE_DIFF(o.order_delivered_customer_date, o.order_purchase_timestamp, DAY)) AS avg_delivery_time
  FROM
    `targetproject-455705.target_sql.orders` o
  JOIN
    `targetproject-455705.target_sql.customers` c
  ON
    o.customer_id = c.customer_id
```

This script will process 188.78 MB when run.

Query results Save results Open in

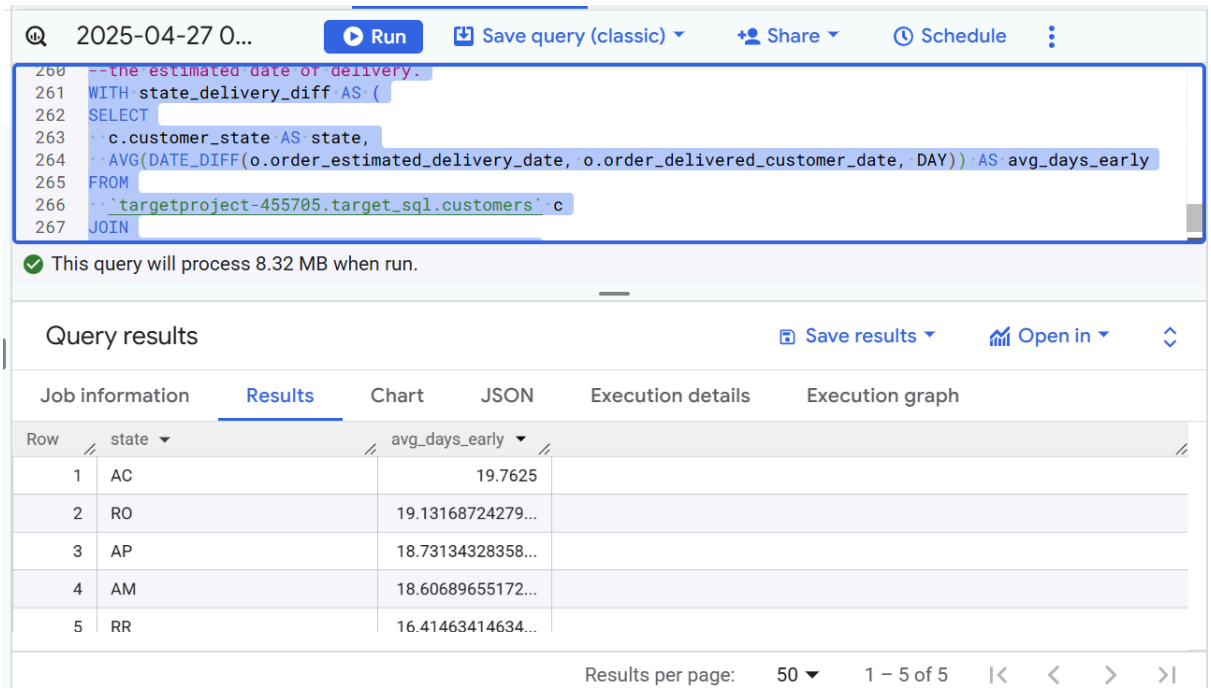
Job information | Results | Chart | JSON | Execution details | Execution graph

Row	state	avg_delivery_time
8	AM	25.98620689655...
9	AP	26.73134328358...
10	RR	28.97560975609...

Results per page: 50 1 – 10 of 10

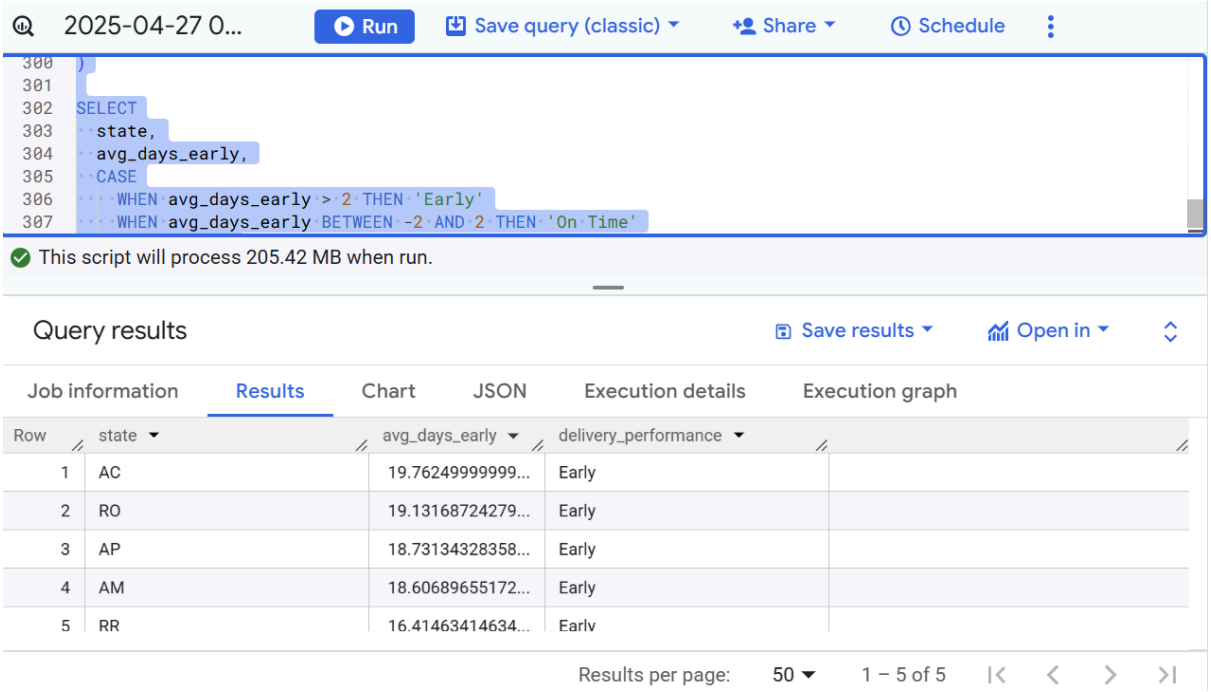
```
--Find out the top 5 states where the order delivery is really fast as compared to
--the estimated date of delivery.
WITH state_delivery_diff AS (
SELECT
  c.customer_state AS state,
  AVG(DATE_DIFF(o.order_estimated_delivery_date, o.order_delivered_customer_date,
DAY)) AS avg_days_early
FROM
  `targetproject-455705.target_sql.customers` c
JOIN
```

```
  `targetproject-455705.target_sql.orders` o
ON
  c.customer_id = o.customer_id
WHERE
  o.order_delivered_customer_date IS NOT NULL
GROUP BY
  state)

SELECT
  state,
  avg_days_early
FROM
  state_delivery_diff
ORDER BY
  avg_days_early DESC
LIMIT 5;
```




```
--Based on how much earlier or later a product is usually delivered each state as
"Early", "On Time", or "Late"
WITH state_delivery_diff AS (
  SELECT
    c.customer_state AS state,
    AVG(DATE_DIFF(o.order_estimated_delivery_date, o.order_delivered_customer_date,
DAY)) AS avg_days_early
  FROM
    `targetproject-455705.target_sql.orders` o
  JOIN
    `targetproject-455705.target_sql.customers` c
```

```sql
  ON
    o.customer_id = c.customer_id
  WHERE
    o.order_delivered_customer_date IS NOT NULL
  GROUP BY
    state
)

SELECT
  state,
  avg_days_early,
  CASE
    WHEN avg_days_early > 2 THEN 'Early'
    WHEN avg_days_early BETWEEN -2 AND 2 THEN 'On Time'
    ELSE 'Late'
  END AS delivery_performance
FROM
  state_delivery_diff
ORDER BY
  avg_days_early DESC
LIMIT 5
;
```



Row	state	avg_days_early	delivery_performance
1	AC	19.76249999999...	Early
2	RO	19.13168724279...	Early
3	AP	18.73134328358...	Early
4	AM	18.60689655172...	Early
5	RR	16.41463414634...	Early

```sql
--Analysis based on the payments
--Find the month on month no. of orders placed using different payment types.

SELECT *
FROM `targetproject-455705.target_sql.payments`;
```

```sql
SELECT *
FROM `targetproject-455705.target_sql.orders`;

SELECT
  EXTRACT(YEAR FROM o.order_purchase_timestamp) AS year,
  EXTRACT(MONTH FROM o.order_purchase_timestamp) AS month,
  p.payment_type,
  COUNT(DISTINCT o.order_id) AS number_of_orders
FROM
  `targetproject-455705.target_sql.orders` o
JOIN
  `targetproject-455705.target_sql.payments` p
ON
  o.order_id = p.order_id
WHERE
  o.order_status != 'canceled'
GROUP BY
  year,
  month,
  payment_type
ORDER BY
  year,
  month,
  payment_type;
```

2025-04-27 0... Run Save query (classic) Share Schedule

```sql
SELECT
  EXTRACT(YEAR FROM o.order_purchase_timestamp) AS year,
  EXTRACT(MONTH FROM o.order_purchase_timestamp) AS month,
  p.payment_type,
  COUNT(DISTINCT o.order_id) AS number_of_orders
FROM
```

This script will process 27.61 MB when run.

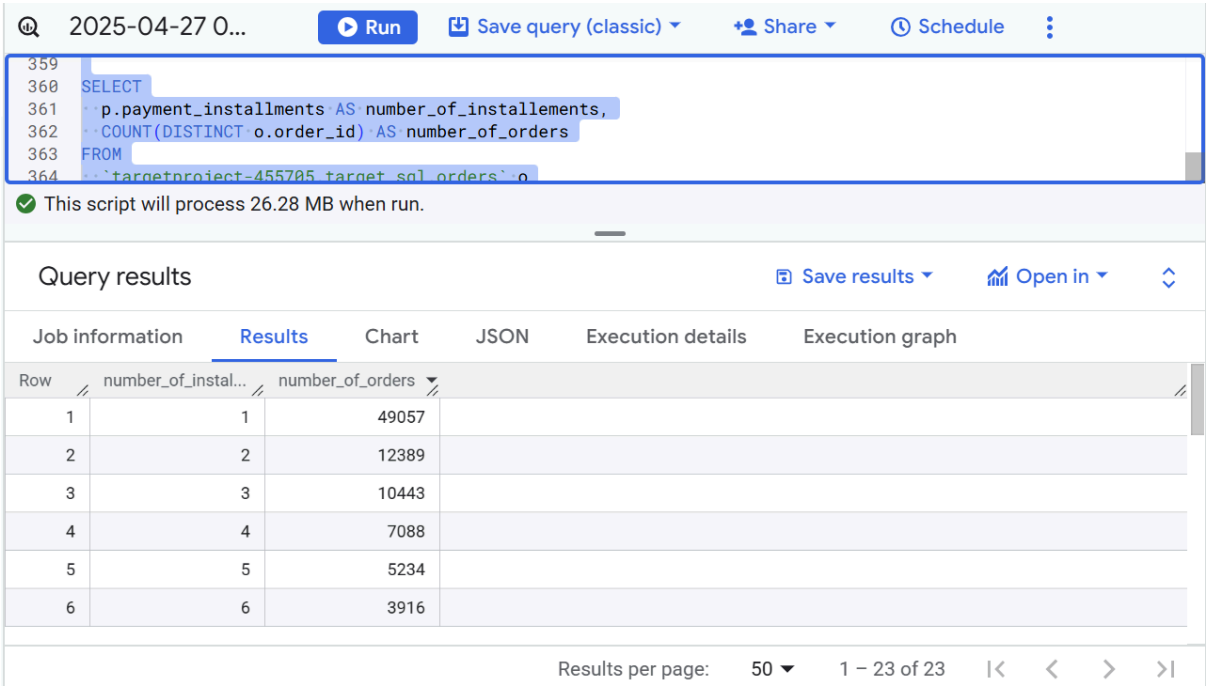
Query results Save results Open in

Job information | Results | Chart | JSON | Execution details | Execution graph

Row	year	month	payment_type	number_of_orders
1	2016	9	credit_card	1
2	2016	10	UPI	60
3	2016	10	credit_card	233
4	2016	10	debit_card	2
5	2016	10	voucher	10
6	2016	12	credit_card	1

Results per page: 50 1 – 50 of 87

```sql
--Find the no. of orders placed on the basis of the payment installments that have been paid.

SELECT *
FROM `targetproject-455705.target_sql.payments`;


SELECT *
FROM `targetproject-455705.target_sql.orders`;

SELECT
  p.payment_installments AS number_of_installements,
  COUNT(DISTINCT o.order_id) AS number_of_orders
FROM
  `targetproject-455705.target_sql.orders` o
JOIN
  `targetproject-455705.target_sql.payments` p
ON
  o.order_id = p.order_id
WHERE
  p.payment_installments > 0 and p.payment_value > 0
GROUP BY
  p.payment_installments
ORDER BY
  p.payment_installments ASC;
```



Row	number_of_instal...	number_of_orders
1	1	49057
2	2	12389
3	3	10443
4	4	7088
5	5	5234
6	6	3916