

Project: Target-SQL Business Case

- Mrudul Jain

Q1. Import the dataset and do usual exploratory analysis steps like checking the structure & characteristics of the dataset

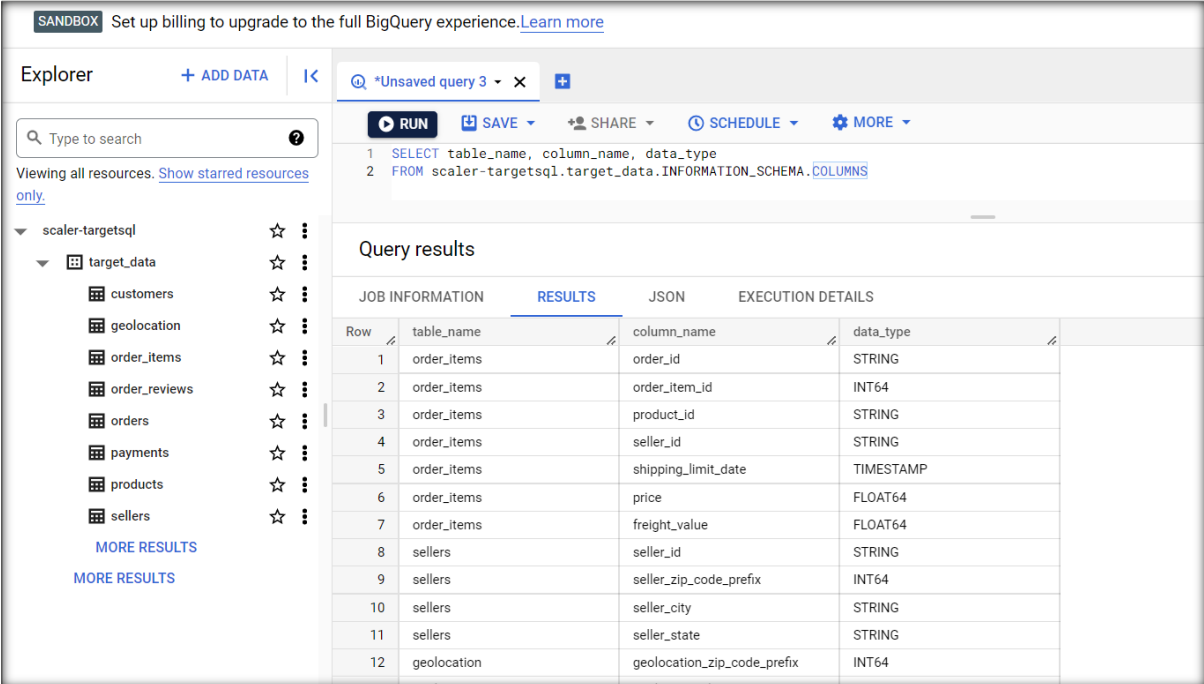
- 1) Data type of columns in a table
- 2) Time period for which the data is given
- 3) Cities and States covered in the dataset

Ans:

- 1) `SELECT table_name, column_name, data_type
FROM scaler-targetsql.target_data.INFORMATION_SCHEMA.COLUMNS`

To get specific table:

```
SELECT table_name, column_name, data_type  
FROM scaler-targetsql.target_data.INFORMATION_SCHEMA.COLUMNS  
WHERE table_name = "orders"
```



The screenshot shows the Google BigQuery interface. On the left, the Explorer pane shows the project 'scaler-targetsql' with a folder 'target_data' containing tables like 'customers', 'geolocation', 'order_items', 'order_reviews', 'orders', 'payments', 'products', and 'sellers'. The main panel shows a query titled '*Unsaved query 3' with the following SQL:

```
1 SELECT table_name, column_name, data_type  
2 FROM scaler-targetsql.target_data.INFORMATION_SCHEMA.COLUMNS
```

The query results are displayed in a table with columns: Row, table_name, column_name, and data_type. The results show the structure of the 'orders' table.

Row	table_name	column_name	data_type
1	order_items	order_id	STRING
2	order_items	order_item_id	INT64
3	order_items	product_id	STRING
4	order_items	seller_id	STRING
5	order_items	shipping_limit_date	TIMESTAMP
6	order_items	price	FLOAT64
7	order_items	freight_value	FLOAT64
8	sellers	seller_id	STRING
9	sellers	seller_zip_code_prefix	INT64
10	sellers	seller_city	STRING
11	sellers	seller_state	STRING
12	geolocation	geolocation_zip_code_prefix	INT64

- 2) `SELECT MAX(order_purchase_timestamp) AS end_date, MIN(order_purchase_timestamp) AS start_date,
DATE_DIFF(MAX(order_purchase_timestamp), MIN(order_purchase_timestamp), DAY) AS Difference
FROM scaler-targetsql.target_data.orders`

*Unsaved query 3 ✕				
orders ✕				
<div> <div>▶ RUN</div> <div>💾 SAVE ▾</div> <div>👤 SHARE ▾</div> <div>🕒 SCHEDULE ▾</div> <div>⚙️ MORE ▾</div> </div>				
<pre> 1 SELECT MAX(order_purchase_timestamp) AS end_date, MIN(order_purchase_timestamp) AS start_date, 2 DATE_DIFF(MAX(order_purchase_timestamp), MIN(order_purchase_timestamp), DAY) AS Difference 3 FROM scaler-targetsql.target_data.orders </pre>				
Query results				
<div>JOB INFORMATION RESULTS JSON EXECUTION DETAILS</div>				
Row	end_date	start_date	Difference	
1	2018-10-17 17:30:18 UTC	2016-09-04 21:15:19 UTC	772	

3) `SELECT DISTINCT geolocation_city AS city, geolocation_state AS state, FROM scaler-targetsql.target_data.geolocation`

To get count of cities within each state:

```

SELECT geolocation_state AS state, COUNT(DISTINCT(geolocation_
city)) AS city_count,
FROM scaler-targetsql.target_data.geolocation
GROUP BY geolocation_state

```

To get total count of cities and states

```

SELECT
COUNT(DISTINCT geolocation_city) AS city_count,
COUNT(DISTINCT geolocation_state) AS state_count,
FROM scaler-targetsql.target_data.geolocation

```

Explorer

+ ADD DATA

1<

Type to search

Viewing all resources. [Show starred resources only.](#)

▼ scaler-targetsqli

▼ target_data

customers

geolocation

order_items

order_reviews

orders

payments

products

sellers

MORE RESULTS

MORE RESULTS

*Unsaved query 3

geolocation

+ RUN

SAVE

SHARE

SCHEDULE

MORE

1 WITH cte1 AS (
2 SELECT
3 DISTINCT geolocation_city AS city, geolocation_state AS state,
4 FROM scaler-targetsqli.target_data.geolocation
5),
6

Query results

JOB INFORMATION

RESULTS

JSON

EXECUTION DETAILS

Row	city	state
1	aracaju	SE
2	riachuelo	SE
3	nossa senhora do socorro	SE
4	barra dos coqueiros	SE
5	itaporanga d'ajuda	SE
6	sao cristovao	SE
7	são cristóvão	SE
8	santo amaro das brotas	SE
9	pirambu	SE
10	laranjeiras	SE
11	umbaua	SE

PERSONAL HISTORY

PROJECT HISTORY

Explorer

+ ADD DATA

1<

Type to search

Viewing all resources. [Show starred resources only.](#)

▼ scaler-targetsqli

▼ target_data

customers

geolocation

order_items

order_reviews

orders

payments

products

sellers

MORE RESULTS

MORE RESULTS

*Unsaved query 3

geolocation

+ RUN

SAVE

SHARE

SCHEDULE

MORE

8 SELECT
9 geolocation_state AS state,
10 COUNT(DISTINCT(geolocation_city)) AS city_count,
11 FROM scaler-targetsqli.target_data.geolocation
12 GROUP BY geolocation_state
13

Query results

JOB INFORMATION

RESULTS

JSON

EXECUTION DETAILS

Row	state	city_count
1	SE	96
2	AL	130
3	PI	278
4	AP	17
5	AM	74
6	RR	14
7	AC	34
8	RO	83
9	TO	173
10	BA	652
11	CE	260

PERSONAL HISTORY

PROJECT HISTORY

The screenshot shows a data exploration tool interface. On the left is an 'Explorer' panel with a search bar and a tree view of resources. The tree view shows a hierarchy: 'scaler-targetsql' > 'target_data' > 'geolocation'. The 'geolocation' resource is selected. On the right is a query editor showing a SQL query. Below the query editor is a 'Query results' section with a table of results.

Explorer Panel:

- Search: Type to search
- Viewing all resources. [Show starred resources only.](#)
- Resources:
 - scaler-targetsql
 - target_data
 - customers
 - geolocation**
 - order_items
 - order_reviews
 - orders
 - payments
 - products
 - sellers
- [MORE RESULTS](#)
- [MORE RESULTS](#)

Query Editor:

```

10 COUNT(DISTINCT(geolocation_city)) AS city_count,
11 FROM scaler-targetsql.target_data.geolocation
12 GROUP BY geolocation_state
13 ),
14
15 cte3 AS (
16 SELECT
17 COUNT(DISTINCT geolocation_city) AS city_count,
18 COUNT(DISTINCT geolocation_state) AS state_count,
19 FROM scaler-targetsql.target_data.geolocation
20 )
21
22 SELECT * from cte3;

```

Query results:

JOB INFORMATION		RESULTS	JSON	EXECUTION DETAILS
Row	city_count	state_count		
1	8011	27		

Q2. In-depth Exploration:

1. Is there a growing trend on e-commerce in Brazil? How can we describe a complete scenario? Can we see some seasonality with peaks at specific months?
2. What time do Brazilian customers tend to buy (Dawn, Morning, Afternoon or Night)?

Ans:

1) **SELECT**
DISTINCT **EXTRACT**(**MONTH** **FROM** order_purchase_timestamp) **AS** month
S,
COUNT(orders.order_id) **OVER** (**PARTITION** **BY** **EXTRACT**(**MONTH** **FROM** o
 rder_purchase_timestamp)) **AS** order_count,
ROUND(**SUM**(payments.payment_value) **OVER** (**PARTITION** **BY** **EXTRACT**(**M**
 ONTH **FROM** order_purchase_timestamp)) , 2) **AS** total_purchase_va
 lue

FROM scaler-targetsql.target_data.orders **AS** orders
LEFT JOIN scaler-targetsql.target_data.payments **AS** payments
ON orders.order_id = payments.order_id
ORDER BY order_count **DESC**, total_purchase_value **DESC**

// This query checks seasonality based on order count and total purchase value of good. As we can clearly see from the output(in screenshot). In the winter months (Sept → Feb) the sales are low as compared to the summer months(March → August)

<div> <div>RUN</div> <div>SAVE</div> <div>SHARE</div> <div>SCHEDULE</div> <div>MORE</div> </div> <div>Query completed</div>				
<pre> 1 SELECT 2 DISTINCT EXTRACT(MONTH FROM order_purchase_timestamp) AS months, 3 COUNT(orders.order_id) OVER (PARTITION BY EXTRACT(MONTH FROM order_purchase_timestamp)) AS order_count, 4 ROUND(SUM(payments.payment_value) OVER (PARTITION BY EXTRACT(MONTH FROM order_purchase_timestamp)), 2) AS total_purchase_value 5 FROM scaler-targetsqli.target_data.orders AS orders 6 LEFT JOIN scaler-targetsqli.target_data.payments AS payments 7 ON orders.order_id = payments.order_id 8 ORDER BY order_count DESC, total_purchase_value DESC </pre> <div>Press Alt+F1 for accessibility options</div>				
<div>Query results</div> <div>SAVE RESULTS</div> <div>EXPLORE DATA</div>				
<div>JOB INFORMATION</div> <div>RESULTS</div> <div>JSON</div> <div>EXECUTION DETAILS</div>				
Row	months	order_count	total_purcha...	
1	8	11248	1696821.64	
2	5	11079	1746900.97	
3	7	10824	1658923.67	
4	3	10349	1609515.72	
5	6	9855	1535156.88	
6	4	9780	1578573.51	
7	2	8838	1284371.35	
8	1	8413	1253492.22	
9	11	7863	1194882.8	
10	12	5896	878421.1	
11	10	5206	839358.03	
<div>Results per page: 50 1 - 12 of 12</div> <div>PERSONAL HISTORY PROJECT HISTORY</div> <div>REFRESH</div>				

To see yearly trend:

```
SELECT
EXTRACT(YEAR FROM order_purchase_timestamp) AS years,
COUNT(orders.order_id) AS total_orders,
SUM(price) AS total_price
FROM `target_data.orders` AS orders
INNER JOIN `target_data.order_items` AS order_items
ON orders.order_id = order_items.order_id
GROUP BY years
ORDER BY years
```

<div> <div>RUN</div> <div>SAVE</div> <div>SHARE</div> <div>SCHEDULE</div> <div>MORE</div> </div>				
<pre> 1 SELECT 2 EXTRACT(YEAR FROM order_purchase_timestamp) AS years, 3 COUNT(orders.order_id) AS total_orders, 4 SUM(price) AS total_price 5 FROM `target_data.orders` AS orders 6 INNER JOIN `target_data.order_items` AS order_items 7 ON orders.order_id = order_items.order_id 8 GROUP BY years 9 ORDER BY years 10 </pre>				
Query results				
<div>JOB INFORMATION</div> <div>RESULTS</div> <div>JSON</div> <div>EXECUTION DETAILS</div>				
Row	years	total_orders	total_price	
1	2016	370	49785.9200...	
2	2017	50864	6155806.98...	
3	2018	61416	7386050.80...	

To see quarterly trend:

```
SELECT
EXTRACT(YEAR FROM order_purchase_timestamp) AS years,
EXTRACT(QUARTER FROM order_purchase_timestamp) AS quarters,
COUNT(order_id) AS total_orders
FROM `scaler-targetsql.target_data.orders`
GROUP BY years, quarters
ORDER BY years DESC, quarters DESC;
```

// As we can see, in Q3 and Q4 of 2016, there were very few sales as compared to Q3 and Q4 of 2017. And there has been an increase in sales in all the quarters from 2017 to 2018. So, the market for e-commerce increased very quickly from 2016 → 2017 but it has slowed down by Q3 of 2018

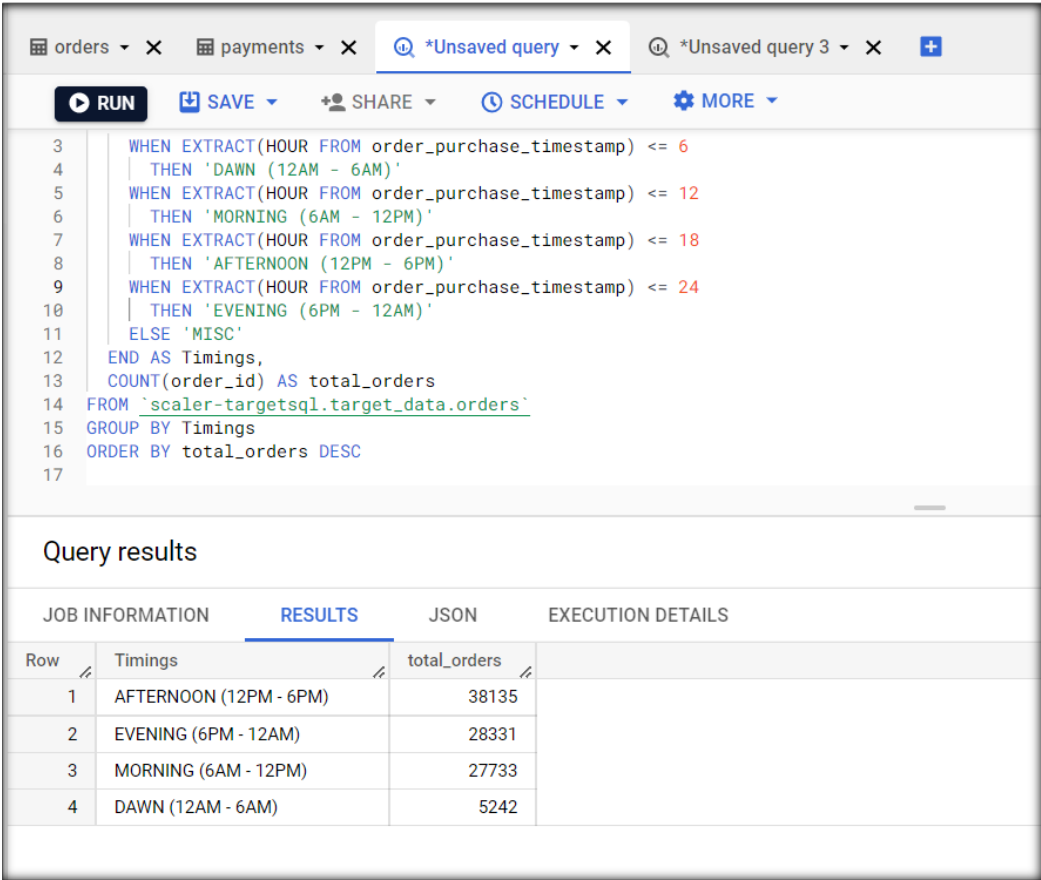
orders × payments × *Unsaved query × +				
RUN SAVE SHARE SCHEDULE MORE				
<pre>1 SELECT 2 EXTRACT(YEAR FROM order_purchase_timestamp) AS years, 3 EXTRACT(QUARTER FROM order_purchase_timestamp) AS quarters, 4 COUNT(order_id) AS total_orders 5 FROM `scaler-targetsql.target_data.orders` 6 GROUP BY years, quarters 7 ORDER BY years DESC, quarters DESC;</pre>				
Query results				
JOB INFORMATION		RESULTS	JSON	EXECUTION DETAILS
Row	years	quarters	total_orders	
1	2018	4	4	
2	2018	3	12820	
3	2018	2	19979	
4	2018	1	21208	
5	2017	4	17848	
6	2017	3	12642	
7	2017	2	9349	
8	2017	1	5262	
9	2016	4	325	
10	2016	3	4	
PERSONAL HISTORY		PROJECT HISTORY		

```

2) SELECT
CASE
    WHEN EXTRACT(HOUR FROM order_purchase_timestamp) <= 6
        THEN 'DAWN (12AM - 6AM)'
    WHEN EXTRACT(HOUR FROM order_purchase_timestamp) <= 12
        THEN 'MORNING (6AM - 12PM)'
    WHEN EXTRACT(HOUR FROM order_purchase_timestamp) <= 18
        THEN 'AFTERNOON (12PM - 6PM)'
    WHEN EXTRACT(HOUR FROM order_purchase_timestamp) <= 24
        THEN 'EVENING (6PM - 12AM)'
    ELSE 'MISC'
END AS Timings,
COUNT(order_id) AS total_orders
FROM `scaler-targetsql.target_data.orders`
GROUP BY Timings
ORDER BY total_orders DESC

```

// Most customers shop during the afternoon hours in Brazil followed by the morning and night hours (equal). The dawn/mid-night hours are the least shopped. So, in-case we need to do server maintenance we can do it during midnight time. And our servers need to be able to handle most load during the day (6am → 12am) timings



The screenshot shows a SQL query editor with a toolbar containing 'RUN', 'SAVE', 'SHARE', 'SCHEDULE', and 'MORE' buttons. The query is displayed in a text area, and below it, the 'Query results' section is visible. The results are presented in a table with four columns: 'Row', 'Timings', 'total_orders', and an empty column. The data is sorted by 'total_orders' in descending order.

Row	Timings	total_orders	
1	AFTERNOON (12PM - 6PM)	38135	
2	EVENING (6PM - 12AM)	28331	
3	MORNING (6AM - 12PM)	27733	
4	DAWN (12AM - 6AM)	5242	

Q3. Evolution of E-commerce orders in the Brazil region:

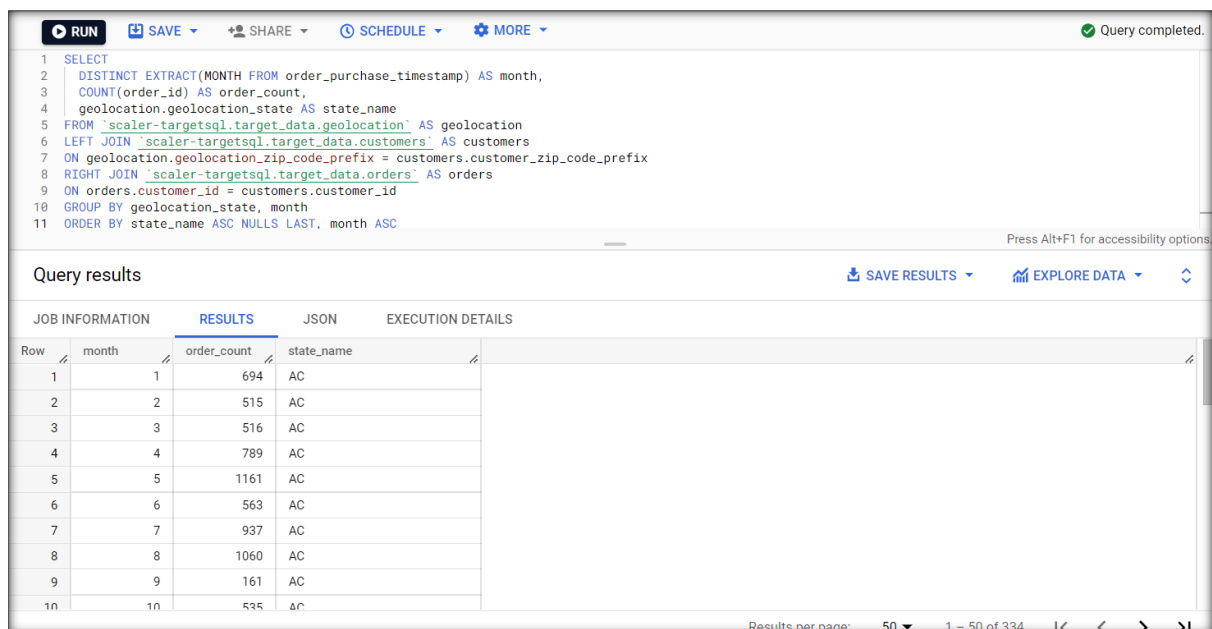
1. Get month on month orders by region, states
2. How are customers distributed in Brazil

Ans:

1) SELECT

```
DISTINCT EXTRACT(MONTH FROM order_purchase_timestamp) AS month
COUNT(order_id) AS order_count,
geolocation.geolocation_state AS state_name
FROM `scaler-targetsql.target_data.geolocation` AS geolocation
LEFT JOIN `scaler-targetsql.target_data.customers` AS customers
ON geolocation.geolocation_zip_code_prefix = customers.customer_zip_code_prefix
RIGHT JOIN `scaler-targetsql.target_data.orders` AS orders
ON orders.customer_id = customers.customer_id
GROUP BY geolocation_state, month
ORDER BY state_name ASC NULLS LAST, month ASC
```

// This query gets total orders by month, to get total orders by city we can replace geolocation_state in GROUP BY with geolocation_city. There is nothing such as “regions” within the database, so we can’t get details by region



The screenshot shows a SQL query editor with a query that has been executed successfully. The query is a SELECT statement that groups orders by month and state. The results are displayed in a table with columns for month, order_count, and state_name. The table shows data for 10 months, with order counts ranging from 161 to 1161, all for the state of AC.

Row	month	order_count	state_name
1	1	694	AC
2	2	515	AC
3	3	516	AC
4	4	789	AC
5	5	1161	AC
6	6	563	AC
7	7	937	AC
8	8	1060	AC
9	9	161	AC
10	10	535	AC

To get count of orders by a specific city/state:

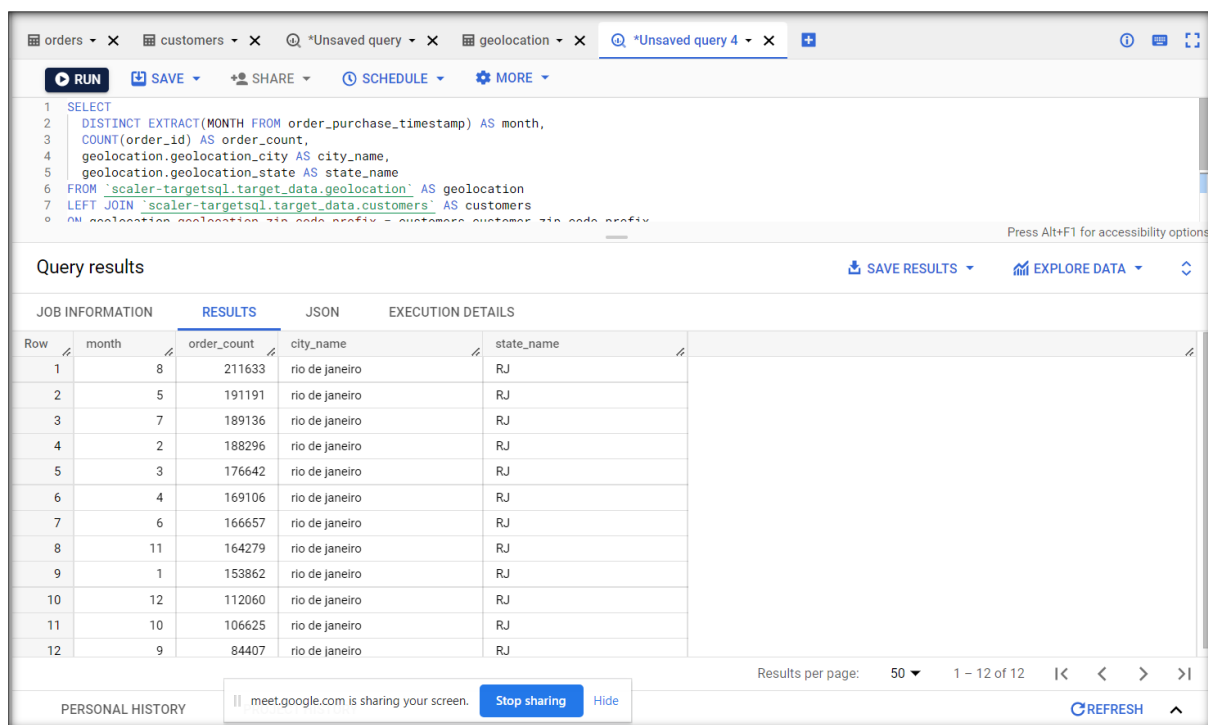
```
SELECT DISTINCT EXTRACT(MONTH FROM order_purchase_timestamp) AS month,
COUNT(order_id) AS order_count,
```



```

geolocation.geolocation_city AS city_name,
geolocation.geolocation_state AS state_name
FROM `scaler-targetsql.target_data.geolocation` AS geolocation
LEFT JOIN `scaler-targetsql.target_data.customers` AS customers
ON geolocation.geolocation_zip_code_prefix = customers.customer_zip_code_prefix
RIGHT JOIN `scaler-targetsql.target_data.orders` AS orders
ON orders.customer_id = customers.customer_id
GROUP BY geolocation_city, geolocation_state, month
HAVING geolocation_city = "rio de janeiro" AND geolocation_state = "RJ"
ORDER BY order_count DESC;

```



The screenshot shows a SQL query editor with a query and its results. The query is a SELECT statement that joins three tables: geolocation, customers, and orders. It filters for geolocation_city = "rio de janeiro" and geolocation_state = "RJ", and orders the results by order_count in descending order. The results table shows 12 rows of data, with columns for month, order_count, city_name, and state_name.

Query results

Row	month	order_count	city_name	state_name
1	8	211633	rio de janeiro	RJ
2	5	191191	rio de janeiro	RJ
3	7	189136	rio de janeiro	RJ
4	2	188296	rio de janeiro	RJ
5	3	176642	rio de janeiro	RJ
6	4	169106	rio de janeiro	RJ
7	6	166657	rio de janeiro	RJ
8	11	164279	rio de janeiro	RJ
9	1	153862	rio de janeiro	RJ
10	12	112060	rio de janeiro	RJ
11	10	106625	rio de janeiro	RJ
12	9	84407	rio de janeiro	RJ

2)

```

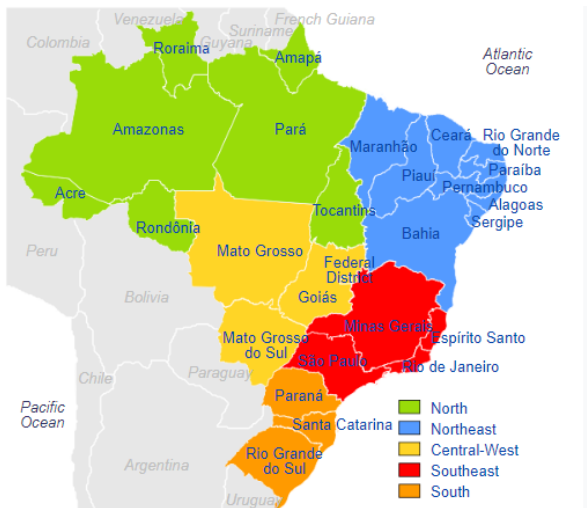
SELECT COUNT(order_id) AS order_count,
geolocation.geolocation_state AS state_name
FROM `scaler-targetsql.target_data.geolocation` AS geolocation
LEFT JOIN `scaler-targetsql.target_data.customers` AS customers
ON geolocation.geolocation_zip_code_prefix = customers.customer_zip_code_prefix
RIGHT JOIN `scaler-targetsql.target_data.orders` AS orders
ON orders.customer_id = customers.customer_id
GROUP BY geolocation_state
ORDER BY order_count DESC

```

Query results

JOB INFORMATION		RESULTS	JSON	EXECUTION DETAILS
Row	order_count	state_name		
1	5620430	SP		
2	3015690	RJ		
3	2878728	MG		
4	805370	RS		
5	626021	PR		
6	538638	SC		
7	365875	BA		
8	316654	ES		
9	133146	GO		
10	122395	MT		

// As we can see here, most of the customers are distributed in the south-east and southern regions of Brazil. There is no “region” field in the database hence I have used an official map for this purpose

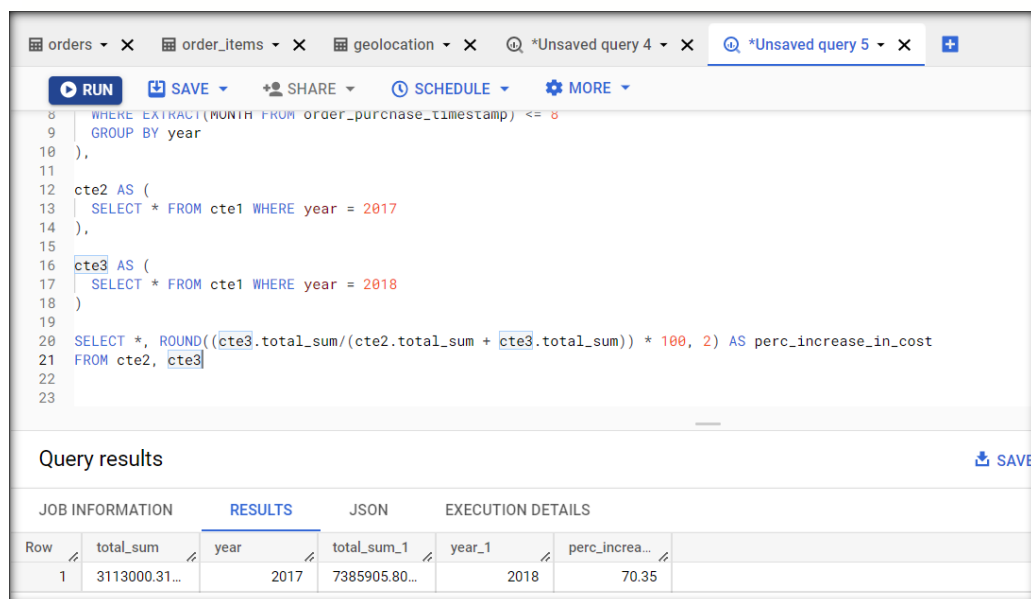


Q.4) Impact on Economy: Analyze the money movemented by e-commerce by looking at order prices, freight and others.

1. Get % increase in cost of orders from 2017 to 2018 (include months between Jan to Aug only)
2. Mean & Sum of price and freight value by customer state

Ans:

```
1) WITH cte1 AS (  
    SELECT  
    SUM(price) AS total_sum,  
    EXTRACT(YEAR FROM order_purchase_timestamp) AS year  
    FROM `target_data.order_items` AS order_items  
    INNER JOIN `target_data.orders` AS orders  
    ON orders.order_id = order_items.order_id  
    WHERE EXTRACT(MONTH FROM order_purchase_timestamp) <= 8  
    GROUP BY year  
) ,  
  
cte2 AS (  
    SELECT * FROM cte1 WHERE year = 2017  
) ,  
  
cte3 AS (  
    SELECT * FROM cte1 WHERE year = 2018  
)  
  
SELECT *, ROUND((cte3.total_sum/(cte2.total_sum + cte3.total_s  
um)) * 100, 2) AS perc_increase_in_cost  
FROM cte2, cte3
```



The screenshot shows a SQL query editor with the following query:

```
WHERE EXTRACT(MONTH FROM order_purchase_timestamp) <= 8  
GROUP BY year  
) ,  
  
cte2 AS (  
    SELECT * FROM cte1 WHERE year = 2017  
) ,  
  
cte3 AS (  
    SELECT * FROM cte1 WHERE year = 2018  
)  
  
SELECT *, ROUND((cte3.total_sum/(cte2.total_sum + cte3.total_s  
um)) * 100, 2) AS perc_increase_in_cost  
FROM cte2, cte3
```

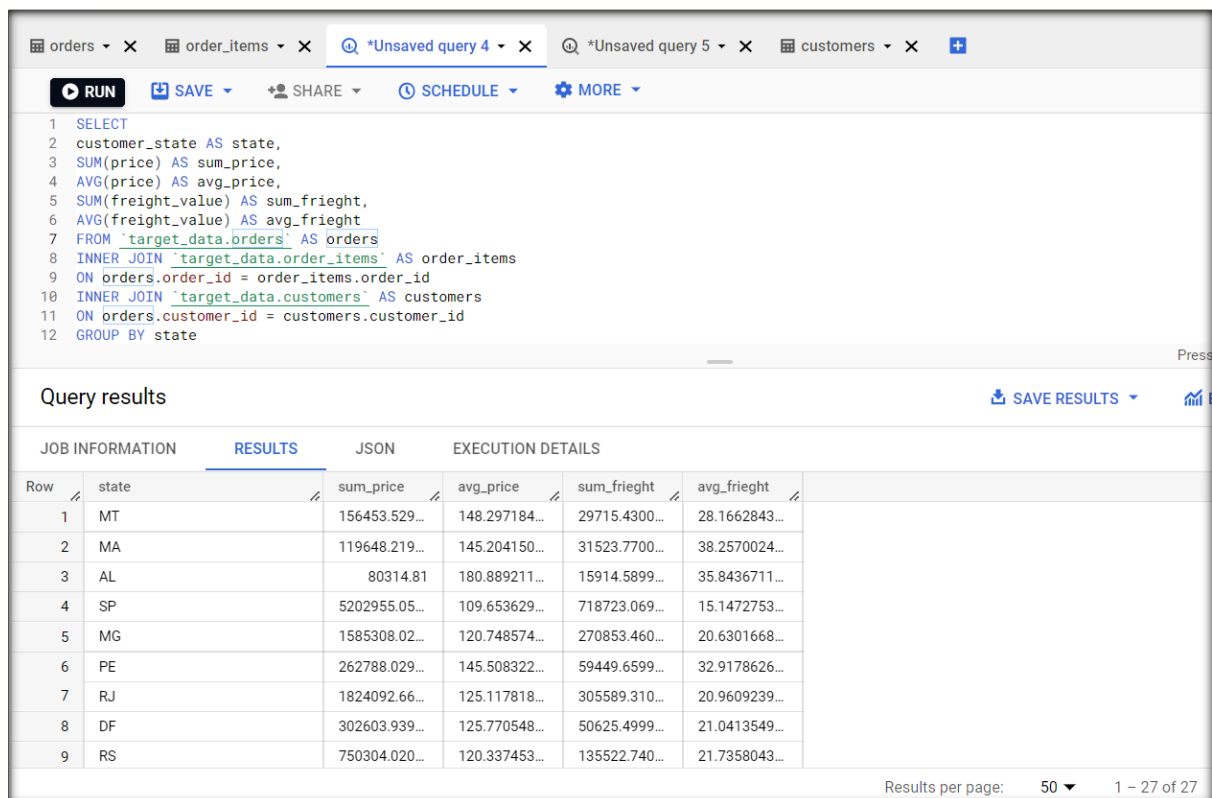
Below the query editor, the "Query results" section is displayed with a table showing the results of the query. The table has 7 columns: Row, total_sum, year, total_sum_1, year_1, perc_increa..., and an empty column. The results are as follows:

Row	total_sum	year	total_sum_1	year_1	perc_increa...	
1	3113000.31...	2017	7385905.80...	2018	70.35	

```

2) SELECT
  customer_state AS state,
  SUM(price) AS sum_price,
  AVG(price) AS avg_price,
  SUM(freight_value) AS sum_frieght,
  AVG(freight_value) AS avg_frieght
FROM `target_data.orders` AS orders
INNER JOIN `target_data.order_items` AS order_items
ON orders.order_id = order_items.order_id
INNER JOIN `target_data.customers` AS customers
ON orders.customer_id = customers.customer_id
GROUP BY state

```



The screenshot shows a SQL query editor with a query named '*Unsaved query 4'. The query is a SELECT statement that calculates aggregate values for each state. The results are displayed in a table with 9 rows, one for each state (MT, MA, AL, SP, MG, PE, RJ, DF, RS). The columns in the results table are: Row, state, sum_price, avg_price, sum_frieght, and avg_frieght. The 'sum_frieght' column has a typo in the header and data ('frieght' instead of 'freight').

Row	state	sum_price	avg_price	sum_frieght	avg_frieght
1	MT	156453.529...	148.297184...	29715.4300...	28.1662843...
2	MA	119648.219...	145.204150...	31523.7700...	38.2570024...
3	AL	80314.81	180.889211...	15914.5899...	35.8436711...
4	SP	5202955.05...	109.653629...	718723.069...	15.1472753...
5	MG	1585308.02...	120.748574...	270853.460...	20.6301668...
6	PE	262788.029...	145.508322...	59449.6599...	32.9178626...
7	RJ	1824092.66...	125.117818...	305589.310...	20.9609239...
8	DF	302603.939...	125.770548...	50625.4999...	21.0413549...
9	RS	750304.020...	120.337453...	135522.740...	21.7358043...

Q.5) Analysis on sales, freight and delivery time

- Calculate days between purchasing, delivering and estimated delivery. Create columns:
 - $\text{time_to_delivery} = \text{order_purchase_timestamp} - \text{order_delivered_customer_date}$
 - $\text{diff_estimated_delivery} = \text{order_estimated_delivery_date} - \text{order_delivered_customer_date}$
- Group data by state, take mean of freight_value, time_to_delivery, diff_estimated_delivery
- Sort the data to get the following:

- Top 5 states with highest/lowest average freight value - sort in desc/asc limit 5
- Top 5 states with highest/lowest average time to delivery
- Top 5 states where delivery is really fast/ not so fast compared to estimated date

Ans:

- // Before, diving all data into columns, I noticed that order_delivered customer date as some NULL values, so we will check them first and **remove them in main query**

NULL value check:

```
SELECT COUNT(*), 'null_tally' AS narrative
FROM `target_data.orders` AS orders
WHERE orders.order_delivered_customer_date IS NULL
```

UNION ALL

```
SELECT COUNT(*), 'not_null_tally' AS narrative
FROM `target_data.orders` AS orders
WHERE orders.order_delivered_customer_date IS NOT NULL;
```

```
1 SELECT COUNT(*), 'null_tally' AS narrative
2 FROM `target_data.orders` AS orders
3 WHERE orders.order_delivered_customer_date IS NULL
4
5 UNION ALL
6
7 SELECT COUNT(*), 'not_null_tally' AS narrative
8 FROM `target_data.orders` AS orders
9 WHERE orders.order_delivered_customer_date IS NOT NULL;
10 |
```

Query results

JOB INFORMATION		RESULTS	JSON	EXECUTION DETAILS
Row	f0_	narrative		
1	2965	null_tally		
2	96476	not_null_tally		

Main Query:

```
SELECT
  EXTRACT(DATE FROM order_purchase_timestamp) AS purchasing_time,
```

```

    EXTRACT(DATE FROM order_delivered_customer_date) AS actual_d
elivery_date,
    DATE_DIFF(order_purchase_timestamp, order_delivered_customer
_date, DAY) AS time_to_delivery,
    EXTRACT(DATE FROM order_estimated_delivery_date) AS est_deli
very_date,
    DATE_DIFF(order_estimated_delivery_date, order_purchase_time
stamp, DAY) AS diff_estimated_delivery
FROM `target_data.orders` AS orders
WHERE order_delivered_customer_date IS NOT NULL

```

<div> <div>▶ RUN</div> <div>🔒 SAVE</div> <div>👤 SHARE</div> <div>🕒 SCHEDULE</div> <div>⚙️ MORE</div> </div>						
<pre> 1 SELECT 2 EXTRACT(DATE FROM order_purchase_timestamp) AS purchasing_time, 3 EXTRACT(DATE FROM order_delivered_customer_date) AS actual_delivery_date, 4 DATE_DIFF(order_purchase_timestamp, order_delivered_customer_date, DAY) AS time_to_delivery, 5 EXTRACT(DATE FROM order_estimated_delivery_date) AS est_delivery_date, 6 DATE_DIFF(order_estimated_delivery_date, order_purchase_timestamp, DAY) AS diff_estimated_delivery 7 FROM `target_data.orders` AS orders 8 WHERE order_delivered_customer_date IS NOT NULL 9 </pre>						
Query results 📄 SAVE RESULTS 📊						
JOB INFORMATION		RESULTS	JSON	EXECUTION DETAILS		
Row	purchasing_time	actual_delivery_date	time_to_delivery	est_delivery_date	diff_estimated_delivery	
1	2016-10-07	2016-10-14	-7	2016-11-29	52	
2	2016-10-09	2016-11-09	-30	2016-12-08	59	
3	2016-10-09	2016-10-16	-7	2016-11-30	51	
4	2016-10-08	2016-10-19	-10	2016-11-30	52	
5	2016-10-03	2016-11-08	-35	2016-11-25	52	
6	2017-03-17	2017-04-07	-20	2017-05-18	61	
7	2017-03-20	2017-03-30	-10	2017-05-18	58	
8	2017-03-21	2017-04-18	-28	2017-05-18	57	
9	2018-08-20	2018-08-29	-9	2018-10-04	44	
10	2018-08-12	2018-08-23	-10	2018-10-04	52	
11	2018-08-16	2018-08-23	-6	2018-10-04	48	
Results per page: 50 1 – 50 of 96476						

```

2) SELECT
    AVG(freight_value) AS avg_frieght,
    AVG(DATE_DIFF(order_purchase_timestamp, order_delivered_cust
omer_date, DAY)) AS avg_time_to_delivery,
    AVG(DATE_DIFF(order_estimated_delivery_date, order_purchase_
timestamp, DAY)) AS avg_diff_estimated_delivery,
    customer_state AS state
FROM `target_data.orders` AS orders
INNER JOIN `target_data.order_items` AS order_items
ON orders.order_id = order_items.order_id
INNER JOIN `target_data.customers` AS customers
ON orders.customer_id = customers.customer_id
WHERE order_delivered_customer_date IS NOT NULL
GROUP BY state

```

<div> <div></div> <div>SAVE</div> <div>SHARE</div> <div>SCHEDULE</div> <div>MORE</div> </div>																																																						
<pre> 1 SELECT 2 AVG(freight_value) AS avg_frieght, 3 AVG(DATE_DIFF(order_purchase_timestamp, order_delivered_customer_date, DAY)) AS avg_time_to_delivery, 4 AVG(DATE_DIFF(order_estimated_delivery_date, order_purchase_timestamp, DAY)) AS avg_diff_estimated_delivery, 5 customer_state AS state 6 FROM `target_data.orders` AS orders 7 INNER JOIN `target_data.order_items` AS order_items 8 ON orders.order_id = order_items.order_id 9 INNER JOIN `target_data.customers` AS customers 10 ON orders.customer_id = customers.customer_id 11 WHERE order_delivered_customer_date IS NOT NULL 12 GROUP BY state </pre>																																																						
Query results <div>SAVE RESULTS</div>																																																						
<div>JOB INFORMATION RESULTS JSON EXECUTION DETAILS</div> <table> <tr> <th>Row</th><th>avg_frieght</th><th>avg_time_to...</th><th>avg_diff_est...</th><th>state</th></tr> <tr><td>1</td><td>20.9097843...</td><td>-14.689382...</td><td>26.0895659...</td><td>RJ</td></tr> <tr><td>2</td><td>20.6258372...</td><td>-11.515522...</td><td>24.2633738...</td><td>MG</td></tr> <tr><td>3</td><td>21.5066276...</td><td>-14.520985...</td><td>25.5229380...</td><td>SC</td></tr> <tr><td>4</td><td>15.1149940...</td><td>-8.2596085...</td><td>18.8701634...</td><td>SP</td></tr> <tr><td>5</td><td>22.5628678...</td><td>-14.948177...</td><td>26.6297760...</td><td>GO</td></tr> <tr><td>6</td><td>21.6142703...</td><td>-14.708299...</td><td>28.2687102...</td><td>RS</td></tr> <tr><td>7</td><td>26.4875563...</td><td>-18.774640...</td><td>29.1751289...</td><td>BA</td></tr> <tr><td>8</td><td>27.9969141...</td><td>-17.508196...</td><td>31.4792671...</td><td>MT</td></tr> <tr><td>9</td><td>36.5731733...</td><td>-20.978666...</td><td>30.4213333...</td><td>SE</td></tr> </table>					Row	avg_frieght	avg_time_to...	avg_diff_est...	state	1	20.9097843...	-14.689382...	26.0895659...	RJ	2	20.6258372...	-11.515522...	24.2633738...	MG	3	21.5066276...	-14.520985...	25.5229380...	SC	4	15.1149940...	-8.2596085...	18.8701634...	SP	5	22.5628678...	-14.948177...	26.6297760...	GO	6	21.6142703...	-14.708299...	28.2687102...	RS	7	26.4875563...	-18.774640...	29.1751289...	BA	8	27.9969141...	-17.508196...	31.4792671...	MT	9	36.5731733...	-20.978666...	30.4213333...	SE
Row	avg_frieght	avg_time_to...	avg_diff_est...	state																																																		
1	20.9097843...	-14.689382...	26.0895659...	RJ																																																		
2	20.6258372...	-11.515522...	24.2633738...	MG																																																		
3	21.5066276...	-14.520985...	25.5229380...	SC																																																		
4	15.1149940...	-8.2596085...	18.8701634...	SP																																																		
5	22.5628678...	-14.948177...	26.6297760...	GO																																																		
6	21.6142703...	-14.708299...	28.2687102...	RS																																																		
7	26.4875563...	-18.774640...	29.1751289...	BA																																																		
8	27.9969141...	-17.508196...	31.4792671...	MT																																																		
9	36.5731733...	-20.978666...	30.4213333...	SE																																																		
Results per page: 50 1 – 27 of 27																																																						

3)
Top 5 avg frieght ASC :

ORDER BY avg_frieght ASC
LIMIT 5

<div> <div></div> <div>SAVE</div> <div>SHARE</div> <div>SCHEDULE</div> <div>MORE</div> </div>																																		
<pre> 6 FROM `target_data.orders` AS orders 7 INNER JOIN `target_data.order_items` AS order_items 8 ON orders.order_id = order_items.order_id 9 INNER JOIN `target_data.customers` AS customers 10 ON orders.customer_id = customers.customer_id 11 WHERE order_delivered_customer_date IS NOT NULL 12 GROUP BY state 13 ORDER BY avg_frieght ASC 14 LIMIT 5 </pre>																																		
Query results																																		
<div>JOB INFORMATION RESULTS JSON EXECUTION DETAILS</div> <table> <tr> <th>Row</th><th>avg_frieght</th><th>avg_time_to...</th><th>avg_diff_est...</th><th>state</th></tr> <tr><td>1</td><td>15.1149940...</td><td>-8.2596085...</td><td>18.8701634...</td><td>SP</td></tr> <tr><td>2</td><td>20.4718162...</td><td>-11.480793...</td><td>24.3793591...</td><td>PR</td></tr> <tr><td>3</td><td>20.6258372...</td><td>-11.515522...</td><td>24.2633738...</td><td>MG</td></tr> <tr><td>4</td><td>20.9097843...</td><td>-14.689382...</td><td>26.0895659...</td><td>RJ</td></tr> <tr><td>5</td><td>21.0721613...</td><td>-12.501486...</td><td>24.0942675...</td><td>DF</td></tr> </table>					Row	avg_frieght	avg_time_to...	avg_diff_est...	state	1	15.1149940...	-8.2596085...	18.8701634...	SP	2	20.4718162...	-11.480793...	24.3793591...	PR	3	20.6258372...	-11.515522...	24.2633738...	MG	4	20.9097843...	-14.689382...	26.0895659...	RJ	5	21.0721613...	-12.501486...	24.0942675...	DF
Row	avg_frieght	avg_time_to...	avg_diff_est...	state																														
1	15.1149940...	-8.2596085...	18.8701634...	SP																														
2	20.4718162...	-11.480793...	24.3793591...	PR																														
3	20.6258372...	-11.515522...	24.2633738...	MG																														
4	20.9097843...	-14.689382...	26.0895659...	RJ																														
5	21.0721613...	-12.501486...	24.0942675...	DF																														

Top 5 avg delivery time ASC:

```
ORDER BY avg_time_to_delivery DESC
LIMIT 5
```

// Descending order because values are negative, higher negative value (more towards 0) = less delivery time


7	INNER JOIN `target_data.order_items` AS order_items
8	ON orders.order_id = order_items.order_id
9	INNER JOIN `target_data.customers` AS customers
10	ON orders.customer_id = customers.customer_id
11	WHERE order_delivered_customer_date IS NOT NULL
12	GROUP BY state
13	ORDER BY avg_time_to_delivery DESC
14	LIMIT 5
15	
16	
17	
18	


Query results


JOB INFORMATION		RESULTS	JSON	EXECUTION DETAILS	
Row	avg_frieght	avg_time_to...	avg_diff_est...	state	
1	15.1149940...	-8.2596085...	18.8701634...	SP	
2	20.4718162...	-11.480793...	24.3793591...	PR	
3	20.6258372...	-11.515522...	24.2633738...	MG	
4	21.0721613...	-12.501486...	24.0942675...	DF	
5	21.5066276...	-14.520985...	25.5229380...	SC	


Top 5 states where Avg. Difference between expected and actual time is least:


```
ORDER BY avg_diff_estimated_delivery ASC
LIMIT 5
```

 RUN

 SAVE

 SHARE

 SCHEDULE

 MORE

```
7 INNER JOIN `target_data.order_items` AS order_items
8 ON orders.order_id = order_items.order_id
9 INNER JOIN `target_data.customers` AS customers
10 ON orders.customer_id = customers.customer_id
11 WHERE order_delivered_customer_date IS NOT NULL
12 GROUP BY state
13 ORDER BY avg_diff_estimated_delivery ASC
14 LIMIT 5
15
16 |
17
18
```

Query results

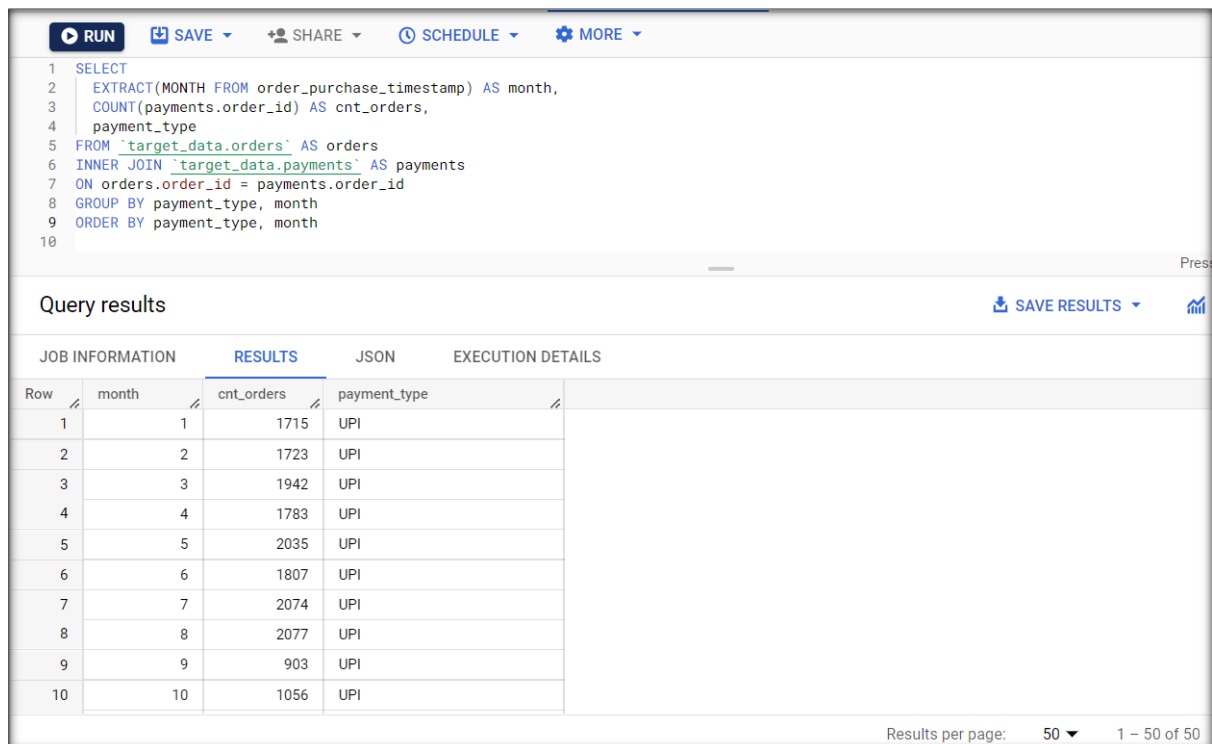
JOB INFORMATION		RESULTS	JSON	EXECUTION DETAILS	
Row	avg_frieght	avg_time_to	avg_diff_est...	state	
1	15.1149940...	-8.2596085...	18.8701634...	SP	
2	21.0721613...	-12.501486...	24.0942675...	DF	
3	20.6258372...	-11.515522...	24.2633738...	MG	
4	20.4718162...	-11.480793...	24.3793591...	PR	
5	22.0289797...	-15.192808...	25.2337078...	ES	

Q.6) Payment type analysis:

1. Month over Month count of orders for different payment types
2. Distribution of payment instalments and count of orders

Ans:

```
1) SELECT
    EXTRACT(MONTH FROM order_purchase_timestamp) AS month,
    COUNT(payments.order_id) AS cnt_orders,
    payment_type
FROM `target_data.orders` AS orders
INNER JOIN `target_data.payments` AS payments
ON orders.order_id = payments.order_id
GROUP BY payment_type, month
ORDER BY payment_type, month
```



The screenshot shows a SQL query execution interface. At the top, there are buttons for RUN, SAVE, SHARE, SCHEDULE, and MORE. Below these is the SQL query text, which is the same as the one provided in the previous block. The query results are displayed in a table with the following columns: Row, month, cnt_orders, and payment_type. The results show 10 rows of data, all with payment_type 'UPI'. The cnt_orders values are 1715, 1723, 1942, 1783, 2035, 1807, 2074, 2077, 903, and 1056. The interface also includes a 'Query results' section with a 'SAVE RESULTS' button and a 'RESULTS' tab. At the bottom right, it shows 'Results per page: 50' and '1 - 50 of 50'.

Row	month	cnt_orders	payment_type
1	1	1715	UPI
2	2	1723	UPI
3	3	1942	UPI
4	4	1783	UPI
5	5	2035	UPI
6	6	1807	UPI
7	7	2074	UPI
8	8	2077	UPI
9	9	903	UPI
10	10	1056	UPI

```
2) SELECT
    COUNT(payments.order_id) AS cnt_orders,
    payment_installments
FROM `target_data.payments` AS payments
GROUP BY payment_installments
ORDER BY cnt_orders DESC
```

RUN

SAVE

SHARE

SCHEDULE

MORE

1SELECT

2COUNT(payments.order_id) AS cnt_orders,

3payment_installments

4FROM `target_data.payments` AS payments

5GROUP BY payment_installments

6ORDER BY cnt_orders DESC

7

8

9

Query results

JOB INFORMATIONRESULTSJSONEXECUTION DETAILS

Row	cnt_orders	payment_in...
1	52546	1
2	12413	2
3	10461	3
4	7098	4
5	5328	10
6	5239	5
7	4268	8
8	3920	6
9	1626	7
10	644	9