

Business Case: Target SQL

Context:

Target is a globally renowned brand and a prominent retailer in the United States. Target makes itself a preferred shopping destination by offering outstanding value, inspiration, innovation and an exceptional guest experience that no other retailer can deliver.

This particular business case focuses on the operations of Target in Brazil and provides insightful information about 100,000 orders placed between 2016 and 2018. The dataset offers a comprehensive view of various dimensions including the order status, price, payment and freight performance, customer location, product attributes, and customer reviews.

By analyzing this extensive dataset, it becomes possible to gain valuable insights into Target's operations in Brazil. The information can shed light on various aspects of the business, such as order processing, pricing strategies, payment and shipping efficiency, customer demographics, product characteristics, and customer satisfaction levels.

Dataset:

<https://drive.google.com/drive/folders/1TGEc66YKbD443nslRi1bWgVd238gJCnb?usp=sharing>

The data is available in 8 different csv files:

1. customers.csv
2. geolocation.csv
3. order_items.csv
4. payments.csv
5. reviews.csv
6. orders.csv
7. products.csv
8. sellers.csv

The column description for these csv files is given below.

The customers.csv contain following features:

Features	Description
customer_id	ID of the consumer who made the purchase
customer_unique_id	Unique ID of the consumer
customer_zip_code_prefix	Zip Code of consumer's location
customer_city	Name of the City from where order is made
customer_state	State Code from where order is made (Eg. são paulo - SP)

The orders.csv contain following features:

Features	Description
order_id	A Unique ID of order made by the consumers
customer_id	ID of the consumer who made the purchase
order_status	Status of the order made i.e. delivered, shipped, etc.
order_purchase_timestamp	Timestamp of the purchase
order_delivered_carrier_date	Delivery date at which carrier made the delivery
order_delivered_customer_date	Date at which customer got the product
order_estimated_delivery_date	Estimated delivery date of the products

The order_items.csv contain following features:

Features	Description
order_id	A Unique ID of order made by the consumers
order_item_id	A Unique ID given to each item ordered in the order
product_id	A Unique ID given to each product available on the site
seller_id	Unique ID of the seller registered in Target
shipping_limit_date	The date before which the ordered product must be shipped
price	Actual price of the products ordered
freight_value	Price rate at which a product is delivered from one point to another

The payments.csv contain following features:

Features	Description
order_id	A Unique ID of order made by the consumers
payment_sequential	Sequences of the payments made in case of EMI
payment_type	Mode of payment used (Eg. Credit Card)
payment_installments	Number of installments in case of EMI purchase
payment_value	Total amount paid for the purchase order

The geolocations.csv contain following features:

Features	Description
geolocation_zip_code_prefix	First 5 digits of Zip Code
geolocation_lat	Latitude
geolocation_lng	Longitude
geolocation_city	City
geolocation_state	State

The sellers.csv contains following features:

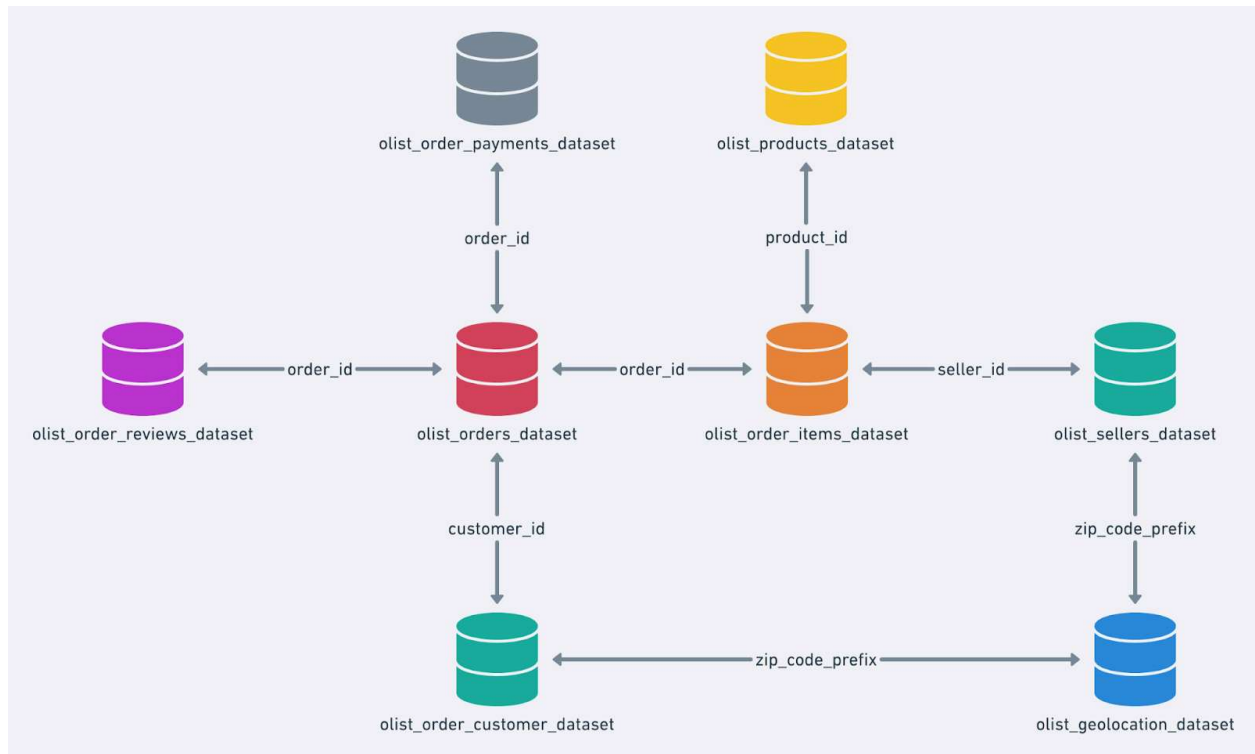
Features	Description
seller_id	Unique ID of the seller registered
seller_zip_code_prefix	Zip Code of the seller's location
seller_city	Name of the City of the seller
seller_state	State Code (Eg. são paulo - SP)

The reviews.csv contain following features:

Features	Description
review_id	ID of the review given on the product ordered by the order id
order_id	A Unique ID of order made by the consumers
review_score	Review score given by the customer for each order on a scale of 1-5
review_comment_title	Title of the review
review_comment_message	Review comments posted by the consumer for each order
review_creation_date	Timestamp of the review when it is created
review_answer_timestamp	Timestamp of the review answered

The products.csv contain following features:

Features	Description
product_id	A Unique identifier for the proposed project.
product_category_name	Name of the product category
product_name_lenght	Length of the string which specifies the name given to the products ordered
product_description_lenght	Length of the description written for each product ordered on the site
product_photos_qty	Number of photos of each product ordered available on the shopping portal
product_weight_g	Weight of the products ordered in grams
product_length_cm	Length of the products ordered in centimeters
product_height_cm	Height of the products ordered in centimeters
product_width_cm	Width of the product ordered in centimeters



```

select * from `target.customers`;
select * from `target.geolocation`;
select * from `target.order_items`;
select * from `target.order_reviews`;
select * from `target.orders`;
select * from `target.payments`;
select * from `target.products`;
select * from `target.sellers`;

```

Problem Statement:

Assuming you are a data analyst/ scientist at Target, you have been assigned the task of analyzing the given dataset to extract valuable insights and provide actionable recommendations.

What does 'good' look like?

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

1.1 Data type of all columns in the "customers" table.

```
SELECT column_name, data_type
FROM fine-rite-444304-e0.target.INFORMATION_SCHEMA.COLUMNS
WHERE table_name = 'customers';
```

The screenshot shows the Google Cloud BigQuery interface. On the left, the Explorer panel lists various datasets, including 'target' which contains tables like 'customers', 'geolocation', 'order_items', 'order_reviews', 'orders', 'payments', 'products', and 'sellers'. The 'customers' table is selected. The main panel displays the query results for the query: `select * from target.order_reviews; select * from target.orders;`. The results are shown in a table with columns 'column_name' and 'data_type'.

Row	column_name	data_type
1	customer_id	STRING
2	customer_unique_id	STRING
3	customer_zip_code_prefix	INT64
4	customer_city	STRING
5	customer_state	STRING

The interface also shows a 'SUMMARY' section at the bottom left, which is currently empty, and a 'Job history' section at the bottom right.

1.2 Get the time range between which the orders were placed.

SELECT

MIN(order_purchase_timestamp) AS earliest_order_date,

MAX(order_purchase_timestamp) AS latest_order_date

FROM `target.orders`;

The screenshot shows the Google Cloud BigQuery console interface. The left sidebar displays the Explorer view with a tree of datasets including HR_Data, cineflix, farmers_market, and target. The target dataset is expanded, showing tables like customers, geolocation, order_items, order_reviews, orders, payments, products, and sellers. The main editor area shows a query titled 'Untitled query' with the following SQL code:

```

4 select * from target.order_reviews ;
5 select * from target.orders ;
6 select * from target.payments ;
7 select * from target.products ;
8 select * from target.sellers ;
9
10 --1.1 Data type of all columns in the "customers" table.
11 SELECT column_name, data_type
12 FROM fine-rite-444304-e0.target.INFORMATION_SCHEMA.COLUMNS
13 WHERE table_name = 'customers';
14
15 --1.2 Get the time range between which the orders were placed.
16 SELECT
17     MIN(order_purchase_timestamp) AS earliest_order_date,
18     MAX(order_purchase_timestamp) AS latest_order_date
19 FROM `target.orders`;
20
21 --1.3 Count the Cities & States of customers who ordered during the given period.
22 SELECT
23

```

The query results are displayed in a table with the following columns: Row, earliest_order_date, and latest_order_date. The results show a single row with the earliest order date as 2016-09-04 21:15:19 UTC and the latest order date as 2018-10-17 17:30:18 UTC.

Row	earliest_order_date	latest_order_date
1	2016-09-04 21:15:19 UTC	2018-10-17 17:30:18 UTC

The bottom of the console shows the Job history section with a REFRESH button.

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

SELECT

```
COUNT(DISTINCT customer_city) AS cities,
COUNT(DISTINCT customer_state) AS states
FROM `target.customers`;
```

The screenshot shows the Google Cloud BigQuery console interface. The top navigation bar includes the Google Cloud logo, a search bar, and a 'Scaler-DSML-SQL' tab. The main area is divided into three sections: Explorer, Query Editor, and Query Results.

Explorer: Displays a list of datasets under the 'target' project, including customers, geolocation, order_items, order_reviews, orders, payments, products, and sellers.

Query Editor: Contains a SQL query titled 'Untitled query'. The query is as follows:

```
17 MIN(order_purchase_timestamp) AS earliest_order_date,
18 MAX(order_purchase_timestamp) AS latest_order_date
19 FROM `target.orders`;
20
21 --1.3 Count the Cities & States of customers who ordered during the given period.
22
23 SELECT
24   COUNT(DISTINCT customer_city) AS cities,
25   COUNT(DISTINCT customer_state) AS states
26 FROM `target.customers`;
27
28 --2.1 Is there a growing trend in the number of orders placed over the past years?
29 SELECT
30   EXTRACT(YEAR FROM order_purchase_timestamp) AS order_year,
31   COUNT(order_id) AS total_orders
32 FROM `target.orders`
33 GROUP BY order_year
34 ORDER BY order_year;
35
```

Query Results: Displays the results of the query. The first result is a table with two columns: 'cities' and 'states'. The values are 4119 and 27 respectively.

Row	cities	states
1	4119	27

The bottom of the console shows the 'Job history' section, which is currently empty.

2.In-depth Exploration:

2.1 Is there a growing trend in the number of orders placed over the past years?

SELECT

```
EXTRACT(YEAR FROM order_purchase_timestamp) AS order_year,
COUNT(order_id) AS total_orders
```

FROM `target.orders`

GROUP BY order_year

ORDER BY order_year;

The screenshot shows the Google Cloud BigQuery interface. On the left is the Explorer pane with a tree view of datasets including HR_Data, cinefix, farmers_market, target, customers, geolocation, order_items, order_reviews, orders, payments, products, and sellers. The 'target' dataset is expanded, showing 'orders' as a table. The main editor shows a query titled 'Untitled query' with the following SQL:

```

23 SELECT
24   COUNT(DISTINCT customer_city) AS cities,
25   COUNT(DISTINCT customer_state) AS states
26 FROM   `target.customers`;
27
28 --2.1 Is there a growing trend in the number of orders placed over the past years?
29 SELECT
30   EXTRACT(YEAR FROM order_purchase_timestamp) AS order_year,
31   COUNT(order_id) AS total_orders
32 FROM   `target.orders`
33 GROUP BY order_year
34 ORDER BY order_year;
35
36 --2.2 Can we see some kind of monthly seasonality in terms of the number of orders being placed?

```

The query results are displayed in a table with columns 'order_year' and 'total_orders'. The results show a clear upward trend from 2016 to 2018.

Row	order_year	total_orders
1	2016	329
2	2017	45101
3	2018	54011

At the bottom, the 'Job history' section is visible, showing the execution details of the query.

2.2 Can we see some kind of monthly seasonality in terms of the number of orders being placed?

SELECT

EXTRACT(MONTH FROM order_purchase_timestamp) AS order_month,

COUNT(order_id) AS total_orders

FROM `target.orders`

GROUP BY order_month

ORDER BY order_month;

The screenshot displays the Google Cloud BigQuery interface. On the left, the Explorer panel shows the 'target' dataset with various tables like 'customers', 'geolocation', 'order_items', 'order_reviews', 'orders', 'payments', 'products', and 'sellers'. The main panel shows a query titled 'Untitled query' with the following SQL code:

```

32 FROM `target.orders`
33 GROUP BY order_month
34 ORDER BY order_month;
35
36 --2.2 Can we see some kind of monthly seasonality in terms of the number of orders being placed?
37

```

The query results are displayed in a table with the following columns: Row, order_month, and total_orders. The results show a decreasing trend in the number of orders over the 12 months.

Row	order_month	total_orders
1	1	8069
2	2	8508
3	3	9893
4	4	9343
5	5	10573
6	6	9412
7	7	10318
8	8	10843
9	9	4305
10	10	4959
11	11	7544
12	12	5674

The bottom of the interface shows the 'Job history' section with a 'REFRESH' button.

2.3 During what time of the day do 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
```

```
FROM `target.orders`
```

```
GROUP BY time_of_day
```

```
ORDER BY total_orders desc;
```

The screenshot shows the Google Cloud BigQuery interface. The query editor displays the following SQL query:

```
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
FROM `target.orders`
GROUP BY time_of_day
ORDER BY total_orders DESC;
```

The query results are displayed in a table with the following data:

Row	time_of_day	total_orders
1	Afternoon	38135
2	Night	28331
3	Morning	27733
4	Dawn	5242

The interface also shows the Explorer panel on the left with a list of datasets, including 'target'. The bottom of the interface shows the Job history section.

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

3.1 Get the month-on-month number of orders placed in each state

SELECT

c.customer_state,

EXTRACT(YEAR FROM o.order_purchase_timestamp) AS order_year,

EXTRACT(MONTH FROM o.order_purchase_timestamp) AS order_month,

COUNT(o.order_id) AS total_orders

FROM `target.orders` o

JOIN `target.customers` c

ON o.customer_id = c.customer_id

GROUP BY order_year, order_month, c.customer_state

ORDER BY c.customer_state, order_year, order_month, total_orders;

The screenshot shows the Google Cloud BigQuery interface. On the left is the Explorer panel with a tree view of datasets including HR_Data, cinefix, farmers_market, target, customers, geolocation, order_items, order_reviews, orders, payments, products, and sellers. The main panel displays a query titled 'Untitled query' with the following SQL code:

```

54 when extract(hour from order_purchase_timestamp) between 0 and 6 Then "Dawn"
55 when extract(hour from order_purchase_timestamp) between 7 and 17 Then "Morning"

```

The query results are shown in a table with the following columns: Row, customer_state, order_year, order_month, and total_orders. The results are for the state of AC.

Row	customer_state	order_year	order_month	total_orders
1	AC	2017	1	2
2	AC	2017	2	3
3	AC	2017	3	2
4	AC	2017	4	5
5	AC	2017	5	8
6	AC	2017	6	4
7	AC	2017	7	5
8	AC	2017	8	4
9	AC	2017	9	5
10	AC	2017	10	6
11	AC	2017	11	5
12	AC	2017	12	5
13	AC	2018	1	6
14	AC	2018	2	3

At the bottom, there is a 'Job history' section with a 'REFRESH' button.

3.2 How are the customers distributed across all the states?

SELECT

```
customer_state,
COUNT(DISTINCT customer_id) AS total_customers
```

FROM `target.customers`

GROUP BY 1

ORDER BY 1;

The screenshot shows the Google Cloud BigQuery console interface. The top navigation bar includes the Google Cloud logo, a search bar, and user profile information. The left sidebar contains the Explorer panel with a tree view of BigQuery resources, including datasets like HR_Data, cineflix, farmers_market, and target. The target dataset is expanded, showing tables such as customers, geolocation, order_items, order_reviews, orders, payments, products, and sellers. The main panel displays an 'Untitled query' editor with the following SQL query:

```
67 c.customer_state,
68 EXTRACT(YEAR FROM o.order_purchase_timestamp) AS order_year,
```

Below the query editor, the 'Query results' tab is active, showing a table with 14 rows and 2 columns: customer_state and total_customers. The results are sorted by total_customers in descending order. The bottom of the console shows the 'Job history' section with a 'REFRESH' button.

Row	customer_state	total_customers
1	AC	81
2	AL	413
3	AM	148
4	AP	68
5	BA	3380
6	CE	1336
7	DF	2140
8	ES	2033
9	GO	2020
10	MA	747
11	MG	11635
12	MS	715
13	MT	907
14	PA	975

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

4.1 Get the % increase in the cost of orders from 2017 to 2018 (Jan to Aug only)

```
WITH time_btw AS
(SELECT EXTRACT(YEAR FROM o.order_purchase_timestamp) AS Year,
sum(p.payment_value) AS cost
FROM target.orders AS o
INNER JOIN target.payments AS 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
ORDER BY Year),
lag_btw AS
(SELECT *, LAG(cost) OVER(ORDER BY Year) AS lagg
FROM time_btw)

SELECT Year,
ROUND(ifnull(((cost-lagg)/lagg)*100,0),2) AS Percentage_increase
FROM lag_btw
ORDER BY Year;
```

The screenshot displays the Google Cloud BigQuery interface. The query editor shows the SQL code for calculating the percentage increase in order costs. The query results are displayed in a table with the following data:

Row	Year	Percentage_increase
1	2017	0.0
2	2018	136.98

The interface also includes a sidebar with a file explorer, a search bar, and various toolbars for query management and execution. The job history section at the bottom shows the execution details of the query.

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

```
select c.customer_state ,
sum(oi.price) as total_price,
SUM(oi.price)/count(distinct o.order_id) as average_price
from `target.order_items` oi
inner join `target.orders` o
on oi.order_id = o.order_id
inner join `target.customers` c
on o.customer_id = c.customer_id
group by c.customer_state
order by total_price
```

Google Cloud | Scaler-DSML-SQL | Search (/) for resources, docs, products, and more

SANDBOX Set up billing to upgrade to the full BigQuery experience. [Learn more](#)

DISMISS UPGRADE

Explorer + ADD

Search BigQuery resources

Show starred only

- HR_Data
- cineflix
- farmers_market
- target
 - customers
 - geolocation
 - order_items
 - order_reviews
 - orders
 - payments
 - products
 - sellers

SUMMARY

Nothing currently selected

Untitled query RUN SAVE DOWNLOAD SHARE SCHEDULE OPEN IN

Processing location: US

Query results

SAVE RESULTS OPEN IN

Row	customer_state	total_price	average_price
1	RR	7829.429999999...	170.2049999999...
2	AP	13474.299999999...	198.1514705882...
3	AC	15982.949999999...	197.3203703703...
4	AM	22356.840000000...	152.0873469387...
5	RO	46140.640000000...	186.8042105263...
6	TO	49621.740000000...	177.8556989247...
7	SE	58920.850000000...	170.7850724637...
8	AL	80314.81	195.4131630170...
9	RN	83034.980000000...	172.2717427385...
10	PI	86914.080000000...	176.2963083164...
11	PB	115268.0799999...	216.6693233082...
12	MS	116812.6399999...	164.7568970380...
13	MA	119648.2199999...	161.6867837837...
14	MT	156453.5299999...	173.2597231450...

Results per page: 50 1 - 27 of 27

Job history REFRESH

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

SELECT

c.customer_state,

SUM(oi.freight_value) AS total_freight_value,

SUM(oi.freight_value)/count(distinct o.order_id) AS avg_freight_value

FROM `target.order_items` oi

INNER JOIN `target.orders` o

ON oi.order_id = o.order_id

INNER JOIN `target.customers` c

ON o.customer_id = c.customer_id

GROUP BY 1

ORDER BY 1

The screenshot shows the Google Cloud BigQuery console interface. On the left is the Explorer pane with a tree view of datasets including 'target'. The main area displays the query results for an 'Untitled query'. The query is: `--4.3 Calculate the Total & Average value of order freight for each state`. The results are shown in a table with columns: 'customer_state', 'total_freight_value', and 'avg_freight_value'. The table lists 14 rows of data for different US states. The bottom of the interface shows 'Results per page: 50' and '1 - 27 of 27'.

Row	customer_state	total_freight_value	avg_freight_value
1	AC	3686.750000000...	45.51543209876...
2	AL	15914.589999999...	38.72163017031...
3	AM	5478.890000000...	37.27136054421...
4	AP	2788.500000000...	41.00735294117...
5	BA	100156.6799999...	29.82628945801...
6	CE	48351.589999999...	36.43676714393...
7	DF	50625.499999999...	23.82376470588...
8	ES	49764.599999999...	24.57511111111...
9	GO	53114.979999999...	26.46486297957...
10	MA	31523.770000000...	42.59968918918...
11	MG	270853.4600000...	23.46270443520...
12	MS	19144.030000000...	27.00145275035...
13	MT	29715.430000000...	32.90745293456...
14	PA	38699.300000000...	39.89618556701...

5 Analysis based on sales, freight and delivery time.

5.1 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. Do this in a single query.

You can calculate the delivery time and the difference between the estimated & actual delivery date using the given formula:

* $\text{time_to_deliver} = \text{order_delivered_customer_date} - \text{order_purchase_timestamp}$

* $\text{diff_estimated_delivery} = \text{order_delivered_customer_date} - \text{order_estimated_delivery_date}$

SELECT

o.order_id,

c.customer_state,

DATE_DIFF(DATE(o.order_delivered_customer_date), DATE(o.order_purchase_timestamp), DAY) AS time_to_deliver,

DATE_DIFF(DATE(o.order_delivered_customer_date),

DATE(o.order_estimated_delivery_date), DAY) AS diff_estimated_delivery

FROM `target.orders` o

JOIN `target.customers` c

ON o.customer_id = c.customer_id

WHERE o.order_delivered_customer_date IS NOT NULL;

The screenshot shows the Google Cloud BigQuery console interface. The query editor at the top contains the SQL query provided. The query results are displayed in a table with the following columns: order_id, customer_state, time_to_deliver, and diff_estimated_delivery. The results show 14 rows of data.

Row	order_id	customer_state	time_to_deliver	diff_estimated_delivery
1	1950d777989f6a877539f5379...	MG	30	12
2	2c45c33d2f9cb8ff6b1e86cc28...	SC	31	-29
3	65d1e226dfaeb8cdc42f66542...	RJ	36	-17
4	635c894d068ac37e6e03dc54e...	RS	31	-2
5	3b97562c3aee8bdedcb5c2e45...	MT	33	-1
6	68f47f50f04c4cb6774570cfe...	SE	30	-2
7	276e9ec344d3b029ff83a161c...	CE	44	4
8	54e1a3c2b97fb0809da548a59...	SC	41	4
9	fd04fa105ee8045f6a0139ca5...	PE	37	1
10	302bb8109d097a9f6e9e9efc5...	RJ	34	5
11	66057d37308e787052a32828...	AL	39	6
12	19135c945c554eebfdf7576c73...	PA	36	2
13	4493e45e7ca1084efcd38ddeb...	MA	34	0
14	70c77e51e0f179d75a6a6141...	RS	43	11

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

```
WITH RankedFreight AS (
  SELECT
    c.customer_state,
    ROUND(AVG(oi.freight_value), 2) AS avg_freight,
    DENSE_RANK() OVER(ORDER BY AVG(oi.freight_value) DESC) AS rank_highest,
    DENSE_RANK() OVER(ORDER BY AVG(oi.freight_value)) AS rank_lowest
  FROM target.order_items oi
  JOIN target.orders o ON oi.order_id = o.order_id
  JOIN target.customers c ON o.customer_id = c.customer_id
  GROUP BY c.customer_state
)
SELECT customer_state, avg_freight
FROM RankedFreight
WHERE rank_highest <= 5 OR rank_lowest <= 5
ORDER BY avg_freight DESC;
```

The screenshot shows the Google Cloud BigQuery console interface. The query results are displayed in a table with the following data:

Row	customer_state	avg_freight
1	RR	42.98
2	PB	42.72
3	RO	41.07
4	AC	40.07
5	PI	39.15
6	DF	21.04
7	RJ	20.96
8	MG	20.63
9	PR	20.53
10	SP	15.15

The interface also shows the Explorer panel on the left with a tree view of BigQuery resources, and the bottom section with job history and pagination controls.

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

```
WITH RankedStates AS (
  SELECT
    c.customer_state,
    ROUND(AVG(DATE_DIFF(DATE(order_delivered_customer_date),
DATE(order_purchase_timestamp), DAY)), 2) AS avg_del_time,
    DENSE_RANK() OVER(ORDER BY AVG(DATE_DIFF(DATE(order_delivered_customer_date),
DATE(order_purchase_timestamp), DAY)) DESC) AS rank_highest,
    DENSE_RANK() OVER(ORDER BY AVG(DATE_DIFF(DATE(order_delivered_customer_date),
DATE(order_purchase_timestamp), DAY))) AS rank_lowest
  FROM target.orders o
  JOIN target.customers c ON o.customer_id = c.customer_id
  GROUP BY c.customer_state
)
SELECT customer_state, avg_del_time
FROM RankedStates
WHERE rank_highest <= 5 OR rank_lowest <= 5
ORDER BY avg_del_time DESC;
```

The screenshot shows the Google Cloud BigQuery interface. The query editor contains the SQL code from the previous block. The query has been executed successfully, and the results are displayed in a table. The table has two columns: 'customer_state' and 'avg_del_time'. The results are ordered by 'avg_del_time' in descending order.

Row	customer_state	avg_del_time
1	RR	29.34
2	AP	27.18
3	AM	26.36
4	AL	24.5
5	PA	23.73
6	SC	14.91
7	DF	12.9
8	MG	11.95
9	PR	11.94
10	SP	8.7

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

You can use the difference between the averages of actual & estimated delivery date to figure out how fast the delivery was for each state.

SELECT

```
c.customer_state, ROUND(AVG(DATE_DIFF(DATE(order_estimated_delivery_date), DATE(order_delivered_customer_date), Day)), 2) AS avg_delivery_days
FROM `target.orders` AS o
INNER JOIN `target.customers` AS c
ON o.customer_id = c.customer_id
GROUP BY c.customer_state
ORDER BY avg_delivery_days
LIMIT 5;
```

The screenshot shows the Google Cloud BigQuery console interface. The query editor displays the following SQL query:

```
--5.4 Find Top 5 States Where Orders Were Delivered Faster Than Estimated
SELECT
c.customer_state, ROUND(AVG(DATE_DIFF(DATE(order_estimated_delivery_date), DATE(order_delivered_customer_date), Day)), 2) AS avg_delivery_days
FROM `target.orders` AS o
INNER JOIN `target.customers` AS c
ON o.customer_id = c.customer_id
GROUP BY c.customer_state
ORDER BY avg_delivery_days
LIMIT 5;
```

The query results are displayed in a table with the following data:

Row	customer_state	avg_delivery_days
1	AL	8.71
2	MA	9.57
3	SE	10.02
4	ES	10.5
5	BA	10.79

6 Analysis based on the payments:

6.1 Find the month on month no. of orders placed using different payment types.

SELECT

```
EXTRACT(YEAR FROM o.order_purchase_timestamp) AS order_year,
EXTRACT(MONTH FROM o.order_purchase_timestamp) AS order_month,
p.payment_type,
```

```

COUNT(DISTINCT o.order_id) AS total_orders
FROM `target.orders` o
JOIN `target.payments` p ON o.order_id = p.order_id
GROUP BY order_year, order_month, p.payment_type
ORDER BY order_year, order_month, total_orders DESC;

```

Google Cloud | Scaler-DSML-SQL | Search (/) for resources, docs, products, and more

SANDBOX Set up billing to upgrade to the full BigQuery experience. [Learn more](#)

DISMISS UPGRADE

Explorer + ADD

Search BigQuery resources

Show starred only

- HR_Data
- cineflix
- farmers_market
- target
 - customers
 - geolocation
 - order_items
 - order_reviews
 - orders
 - payments
 - products
 - sellers

SUMMARY

Nothing currently selected

Untitled query

RUN SAVE DOWNLOAD SHARE SCHEDULE OPEN IN

Query completed.

Processing location: US

Query results

SAVE RESULTS OPEN IN

JOB INFORMATION RESULTS CHART JSON EXECUTION DETAILS EXECUTION GRAPH

Row	order_year	order_month	payment_type	total_orders
1	2016	9	credit_card	3
2	2016	10	credit_card	253
3	2016	10	UPI	63
4	2016	10	voucher	11
5	2016	10	debit_card	2
6	2016	12	credit_card	1
7	2017	1	credit_card	582
8	2017	1	UPI	197
9	2017	1	voucher	33
10	2017	1	debit_card	9
11	2017	2	credit_card	1347
12	2017	2	UPI	398

Results per page: 50 1 - 50 of 90

Job history REFRESH

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

SELECT

```
payment_installments AS installments,
COUNT(order_id) AS num_orders,
FROM `target.payments`
WHERE payment_installments >= 1
GROUP BY payment_installments
ORDER BY num_orders DESC
```

The screenshot shows the Google Cloud BigQuery console interface. On the left is the Explorer pane with a tree view of datasets including 'target'. The main area displays a query titled 'Untitled query' with the following SQL code:

```
SELECT
COUNT(*) AS num_orders,
payment_installments AS installments
FROM `target.payments`
WHERE payment_installments >= 1
GROUP BY payment_installments
ORDER BY num_orders DESC
```

The query results are shown in a table with 14 rows. The columns are 'installments' and 'num_orders'. The results are ordered by 'num_orders' in descending order.

Row	installments	num_orders
1	1	52546
2	2	12413
3	3	10461
4	4	7098
5	10	5328
6	5	5239
7	8	4268
8	6	3920
9	7	1626
10	9	644
11	12	133
12	15	74
13	18	27
14	11	23

At the bottom, there is a 'Job history' section and a 'Results per page' dropdown set to 30, showing 1 - 23 of 23 results.