Business Case: A Big Retailer SQL

Submitted By – Vibha Sharma

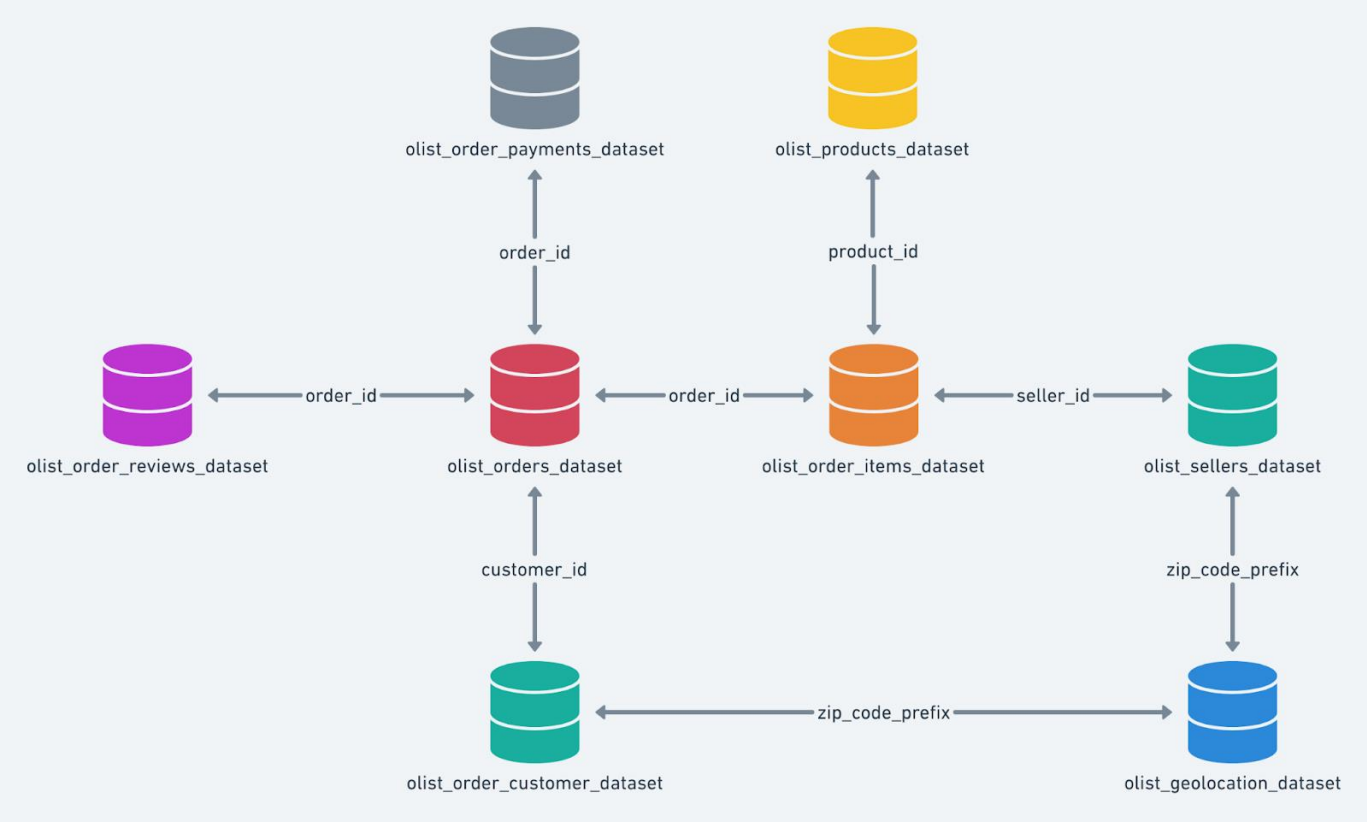
Email – vibha.ajm13@gmail.com

CONTEXT –

Big Retailer is one of the world's most recognized brands and one of America's leading retailers. It makes itself a preferred shopping destination by offering outstanding value, inspiration, innovation, and an exceptional guest experience that no other retailer can deliver.

This business case has information of 100k orders from 2016 to 2018 made at the company in Brazil. Its features allow viewing an order from multiple dimensions: from order status, price, payment and freight performance to customer location, product attributes and finally reviews written by customers.

HIGH LEVEL OVERVIEW OF RELATIONSHIP BETWEEN DATASETS –



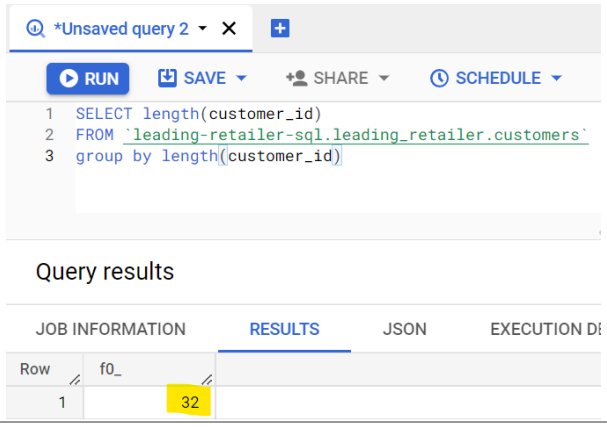
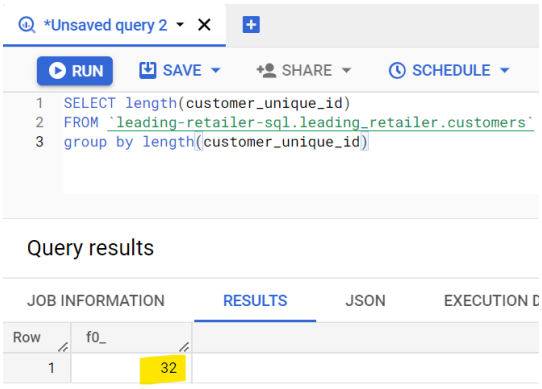
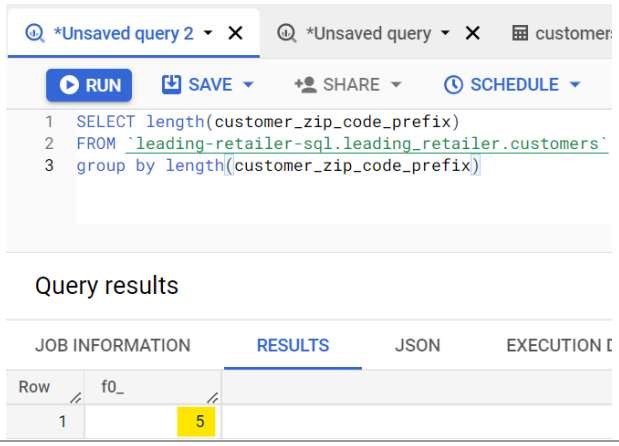
ASSUMPTIONS –

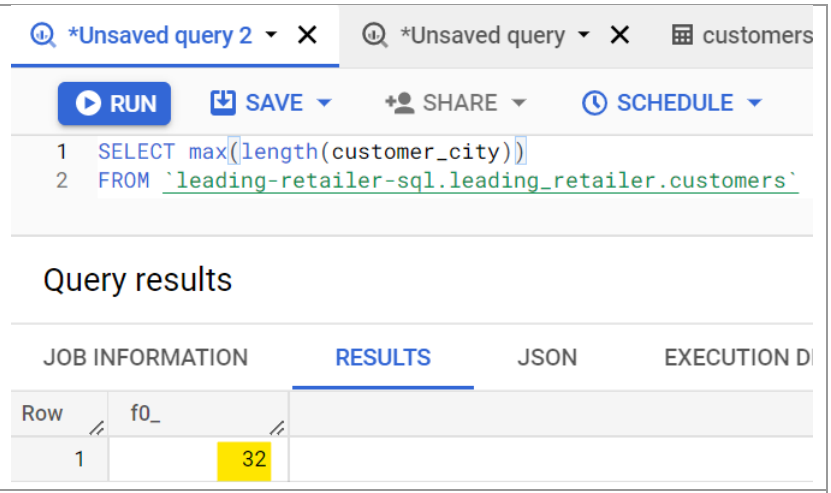
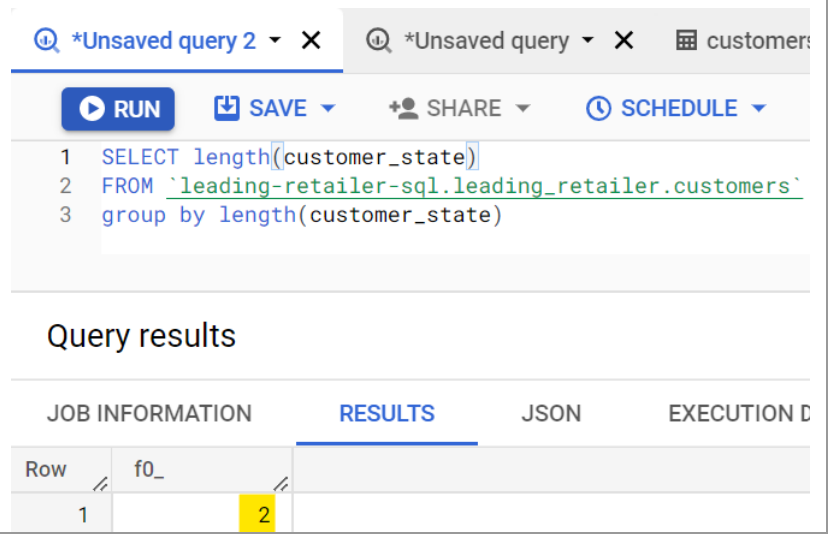
A. Since there is no information about the timezone in the data, the bellow analysis will assume the time is given in Brasilia Standard Time (even though BigQuery assumes it to be in UTC).
B. The analysis below will assume all orders to be E-commerce orders, as there is no distinction given to assume otherwise (i.e., there is no field that distinguishes manual purchases with digital or e-commerce purchases). According to the *payment_type* field we'll assume all data to be e-commerce data.
C. ORDER BY works differently with EXTRACT keyword in BigQuery, to overcome this, in below analysis I have used common table expressions or sub-queries.
D. Actionable Insights are highlighted in green.
E. Recommendations are highlighted in cyan.

1. **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**

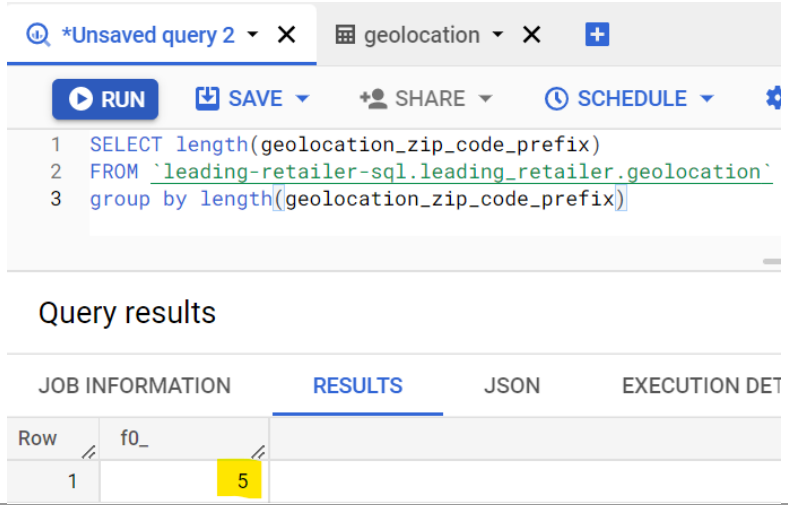
The data types below are the ideal data type that should be assigned to the fields of the tables according to their description and values. They are not the datatypes auto interpreted by Google BigQuery.

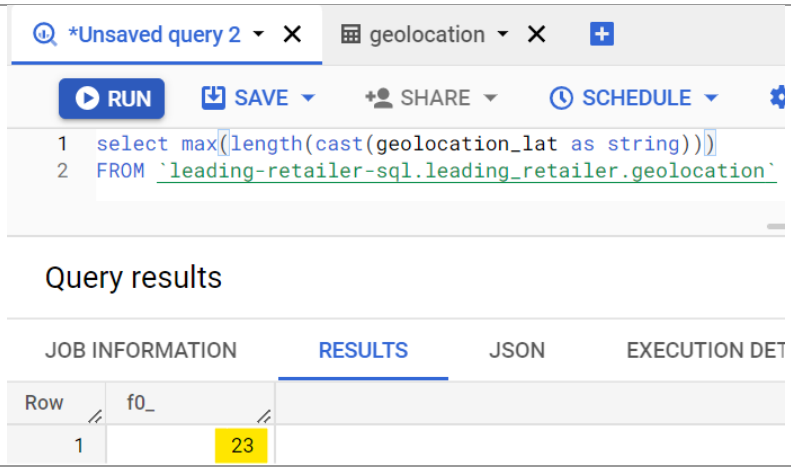
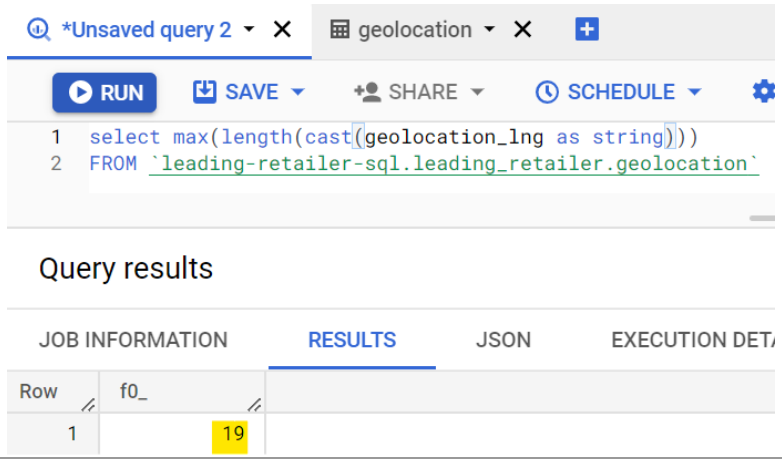
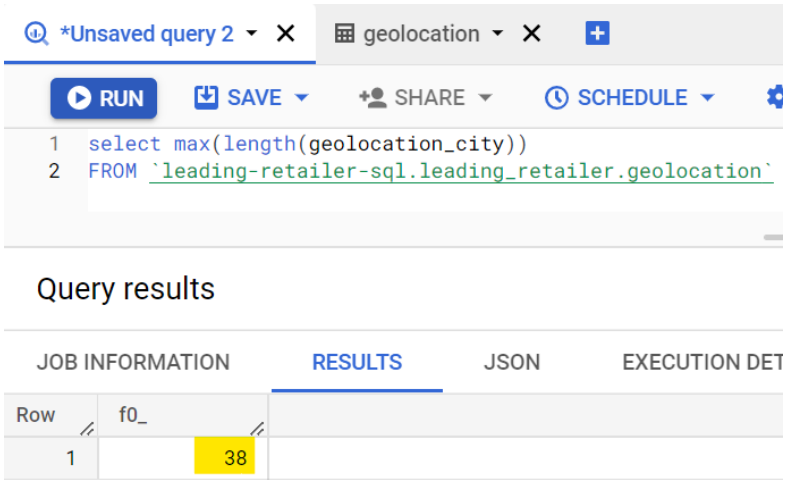
customers

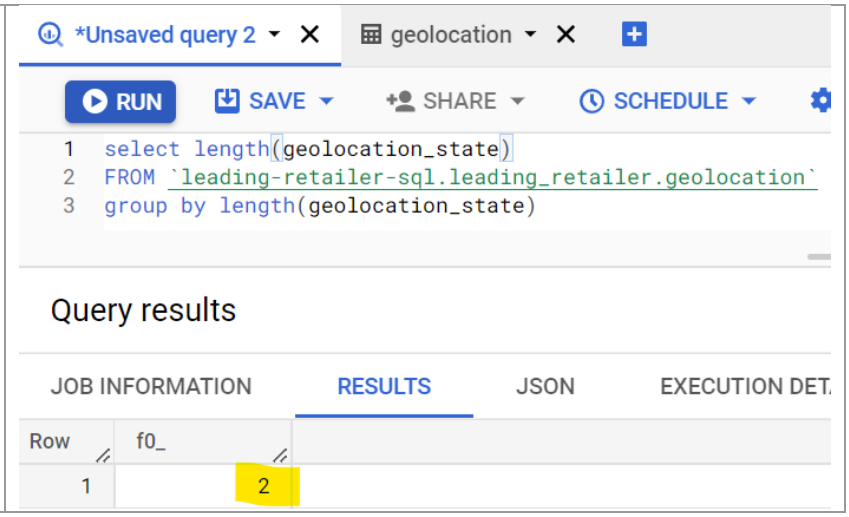
Columns	Data Type
customer_id	char(32) 
customer_unique_id	char(32) 
customer_zip_code_prefix	char(5) 
customer_city	varchar(32)

	
customer_state	char(2) 

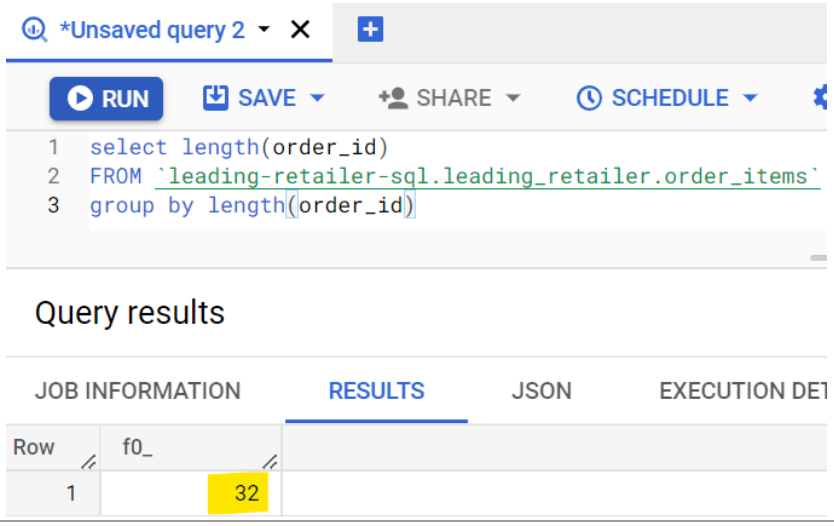
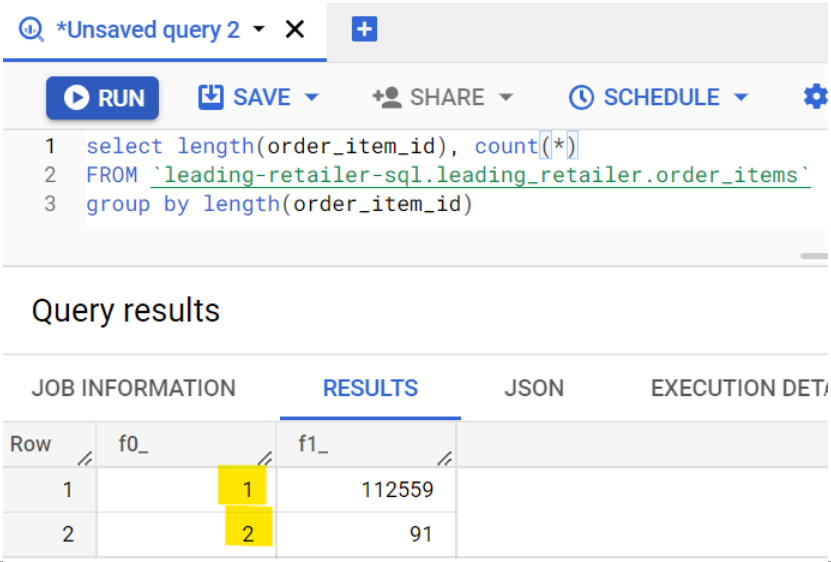
geolocation

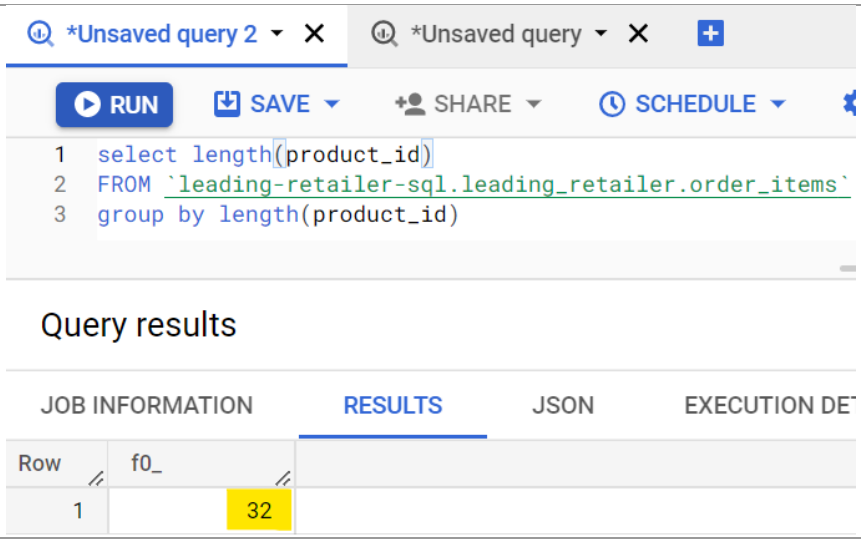
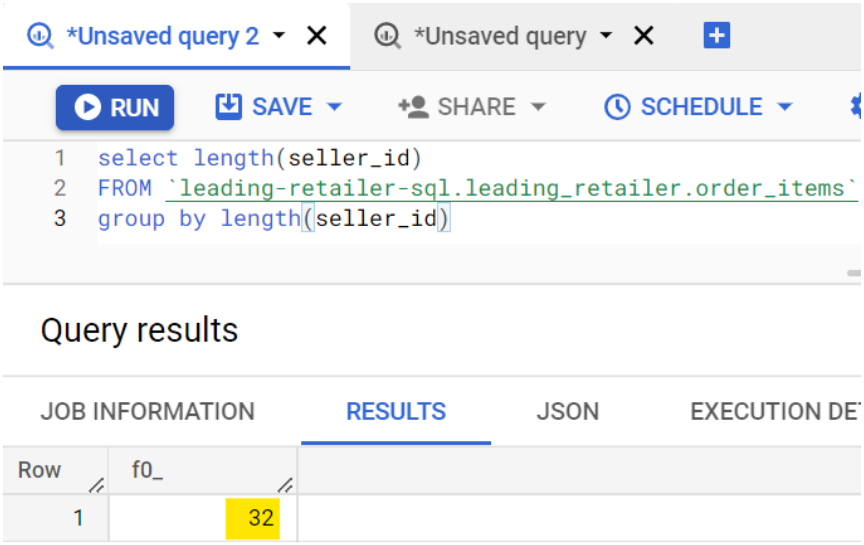
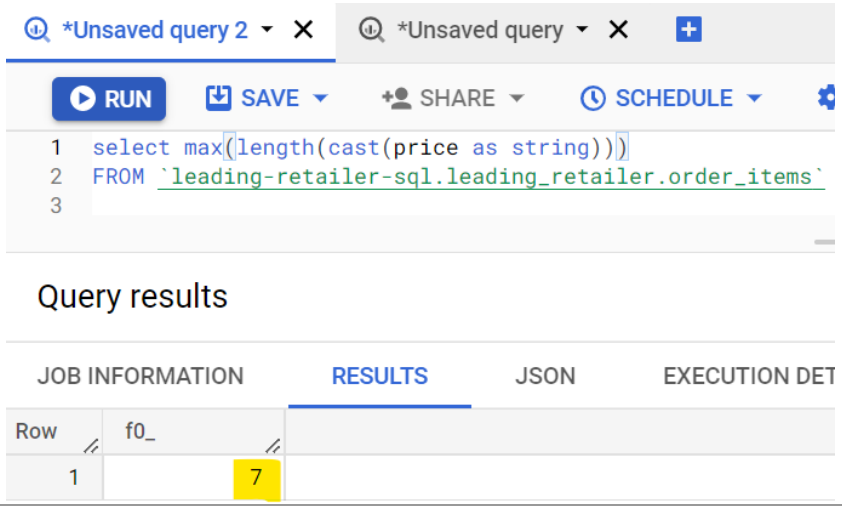
Columns	Data Type
geolocation_zip_code_prefix	char(5) 
geolocation_lat	float(23)

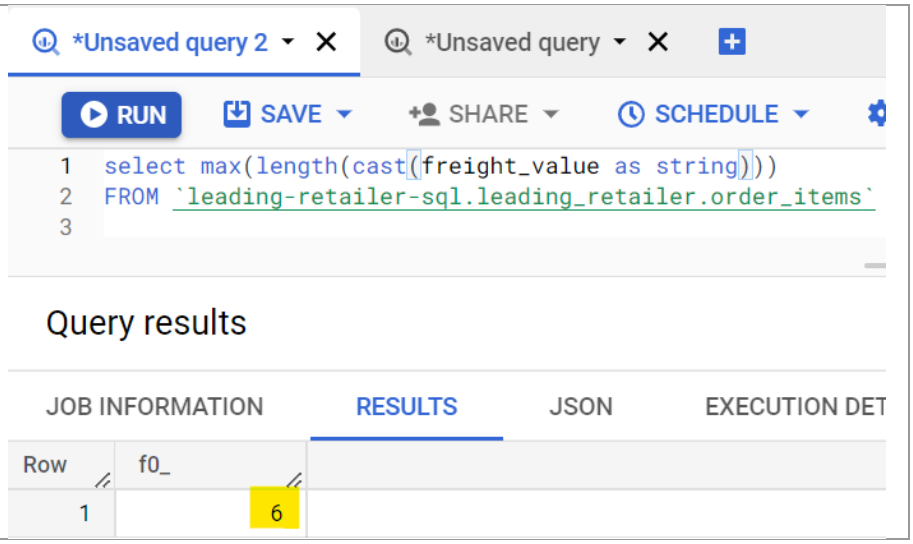
	
geolocation_lng	float(19) according to the data given or float(23) keeping consist with geolocation_lat (no data loss since max digits (i.e., 19) < 23 
geolocation_city	varchar(38) 
geolocation_state	char(2)

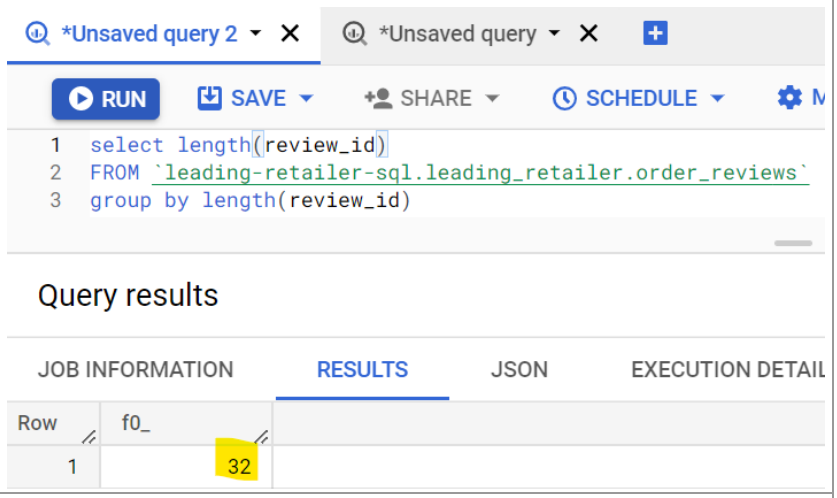
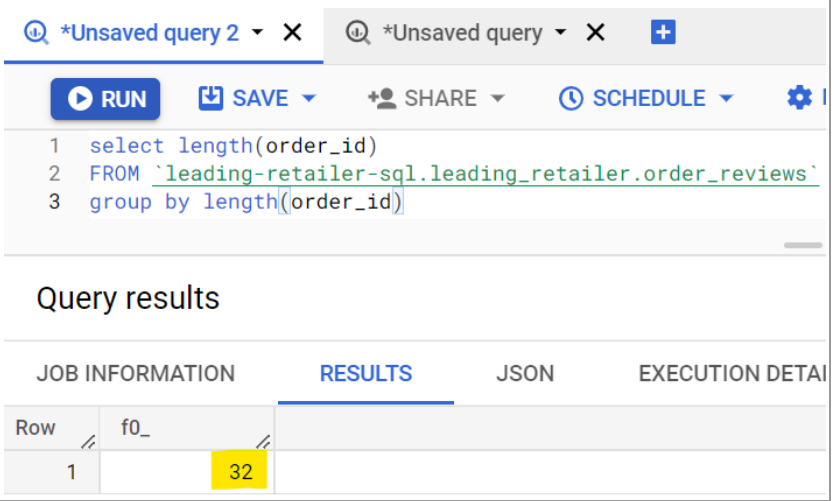
order_items

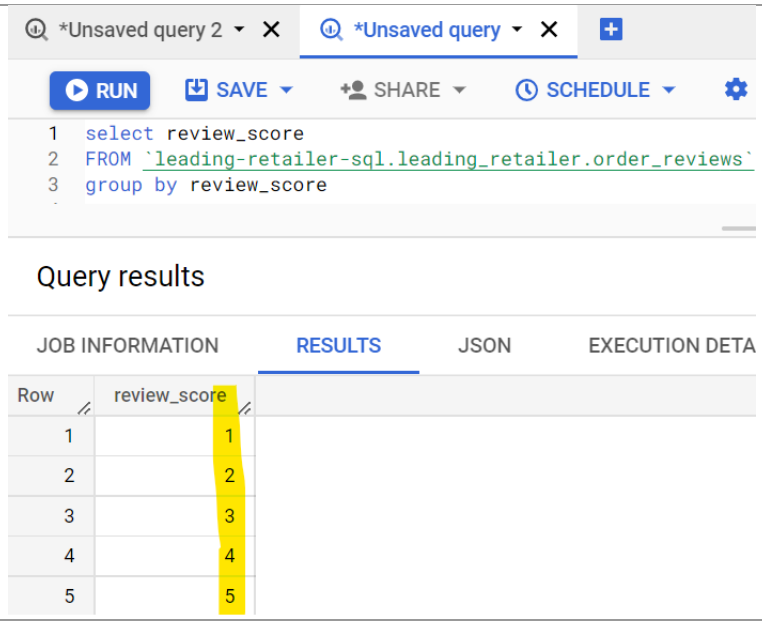
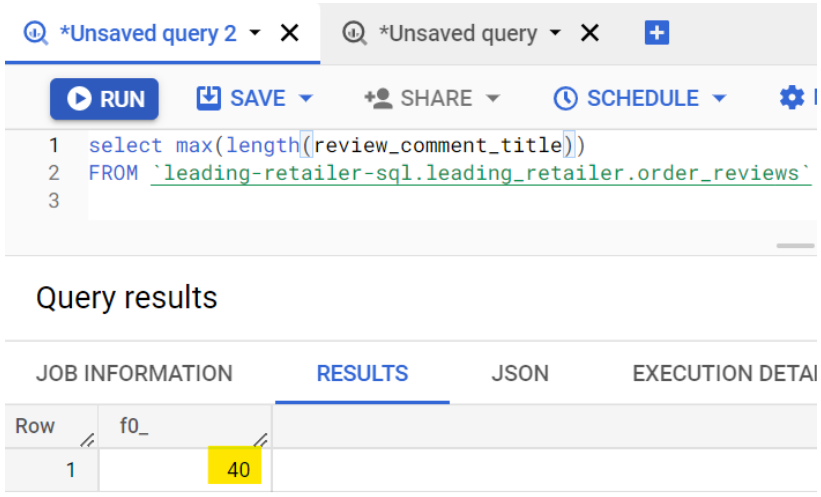
Columns	Data Type
order_id	char(32) 
order_item_id	varchar(2) 
product_id	char(32)

	
seller_id	char(32) 
shipping_limit_date	datetime
price	float(7) 
freight_value	float(6)

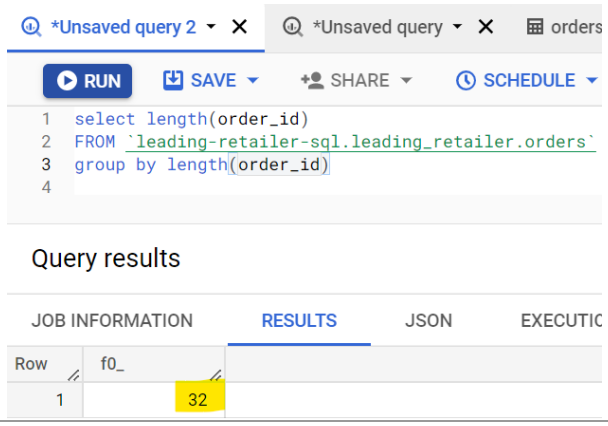
	

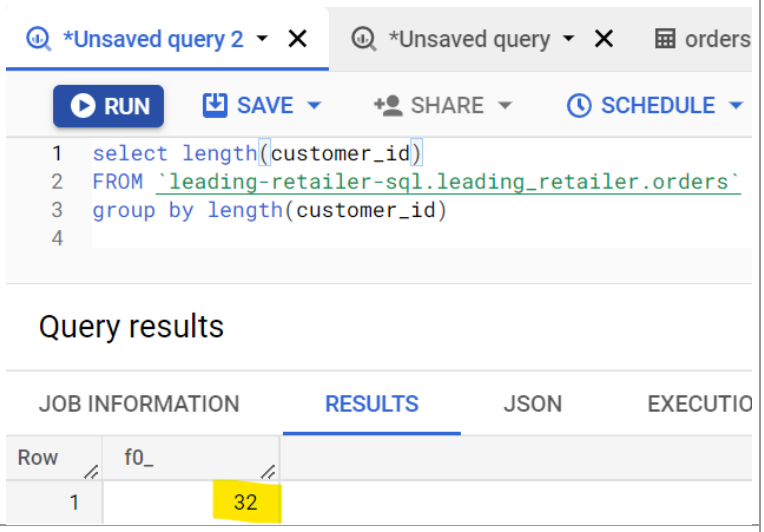
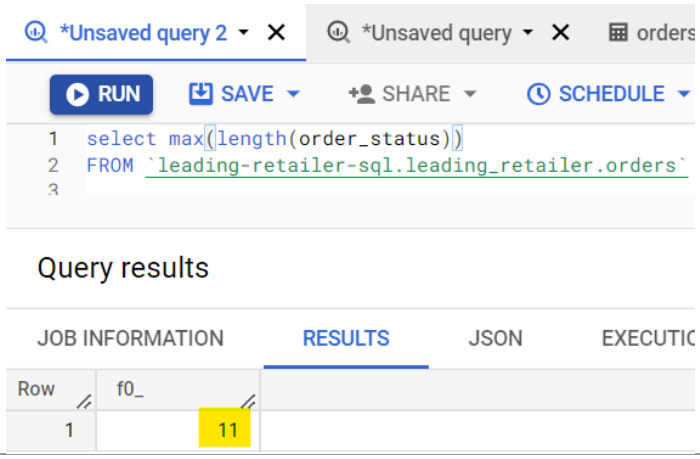
order_reviews

Columns	Data Type
review_id	char(32) 
order_id	char(32) 
review_score	tinyint

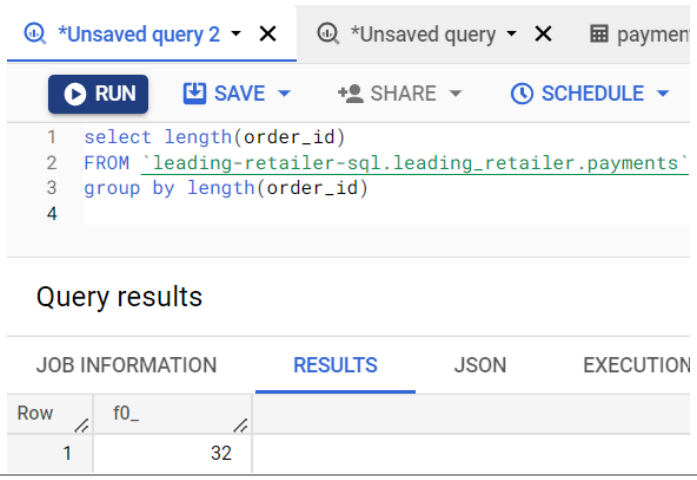
	
review_comment_title	varchar(40) or >=40 
review_comment_message	Present in the question description but not in actual dataset
review_creation_date	date or datetime
review_answer_timestamp	datetime

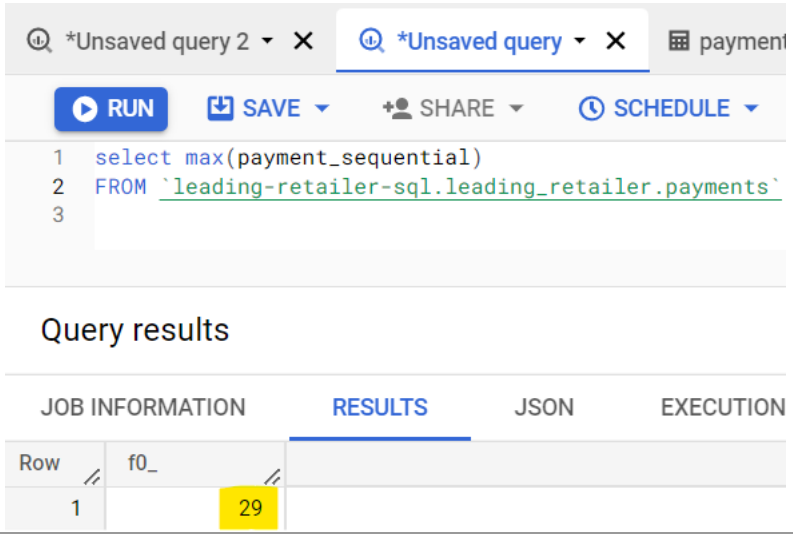
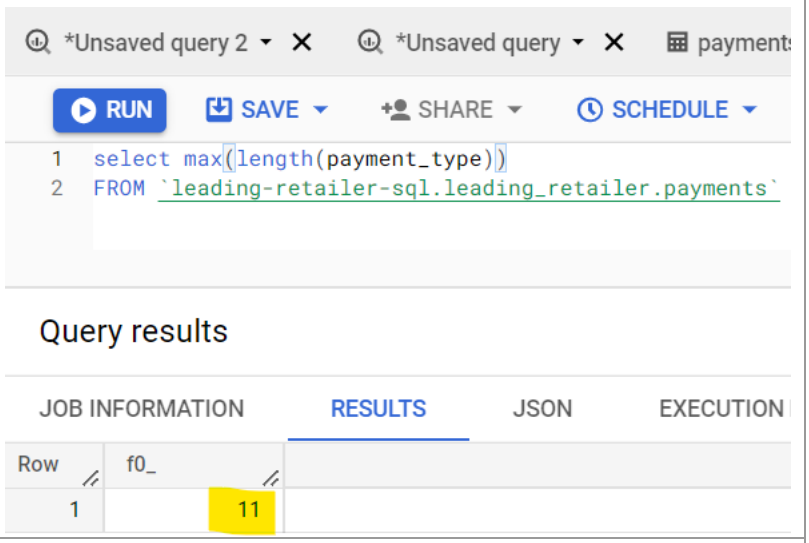
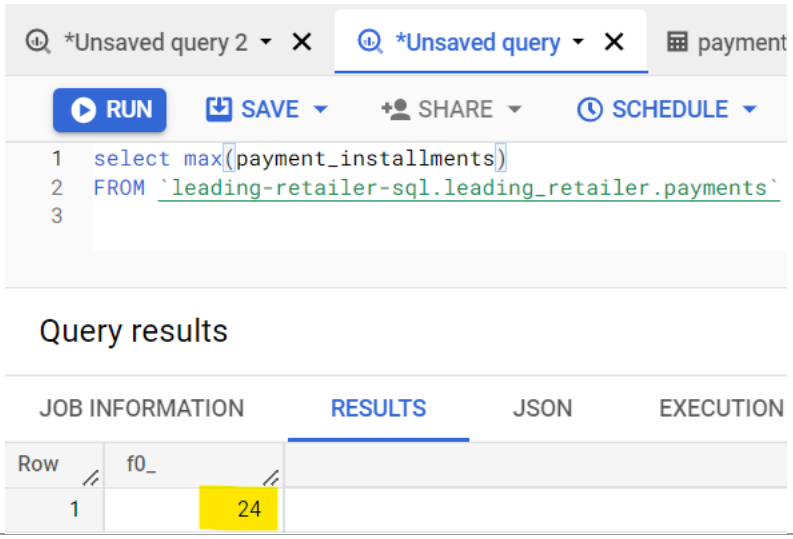
orders

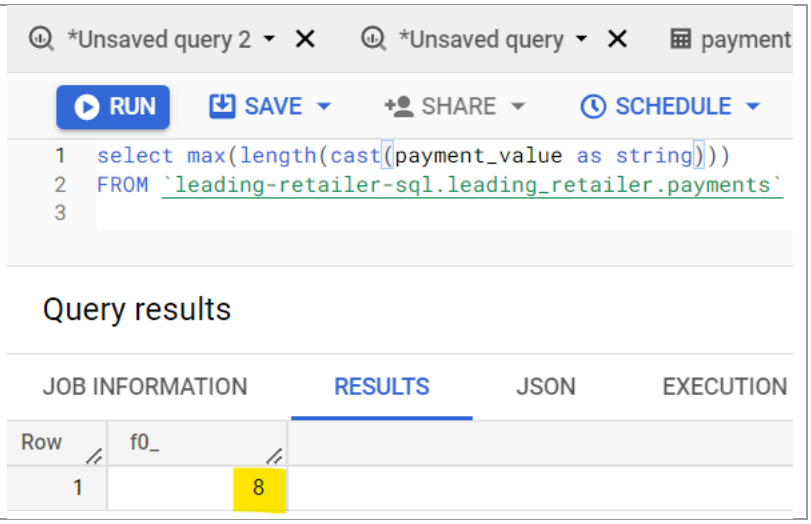
Columns	Data Type
order_id	char(32) 
customer_id	char(32)

	
order_status	varchar(11) 
order_purchase_timestamp	datetime
order_delivered_carrier_date	datetime
order_delivered_customer_date	datetime
order_estimated_delivery_date	datetime

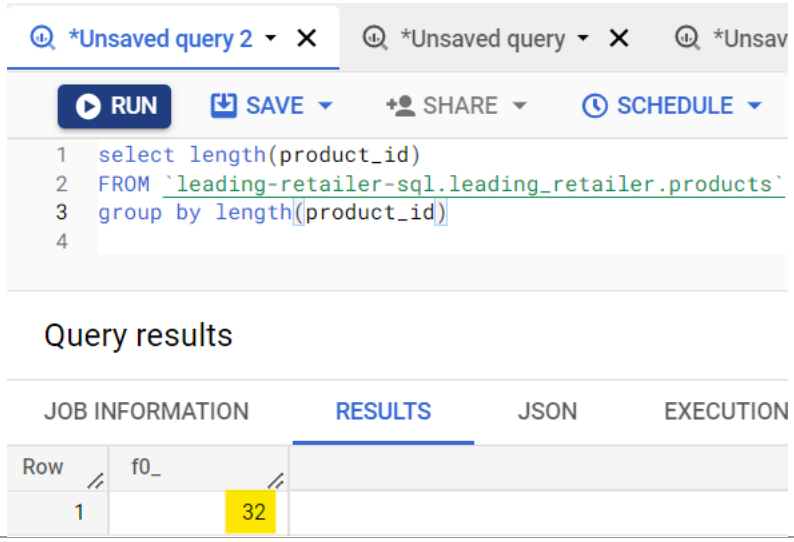
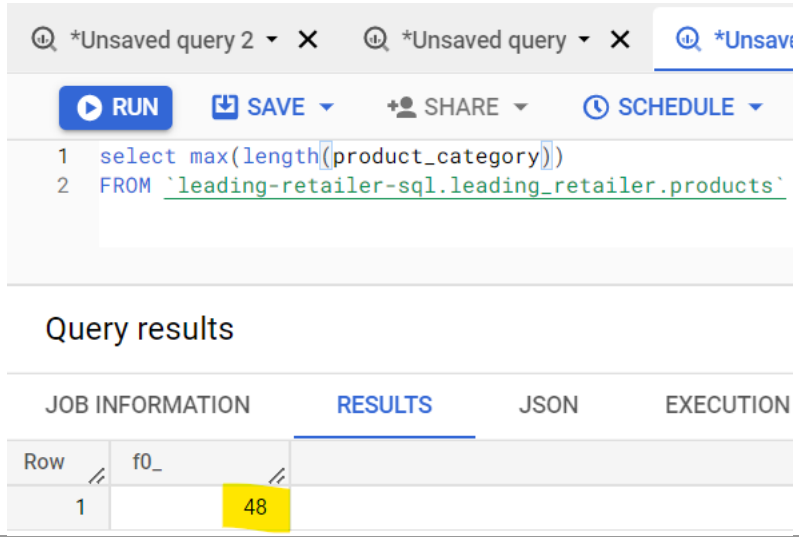
payments

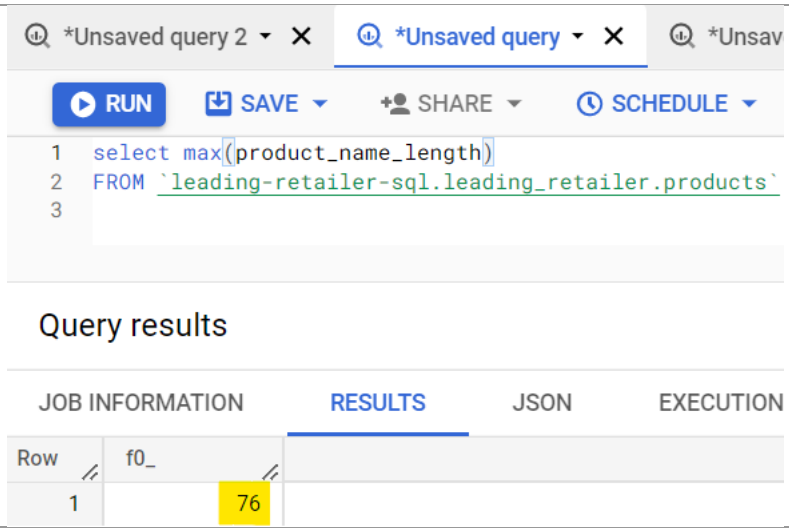
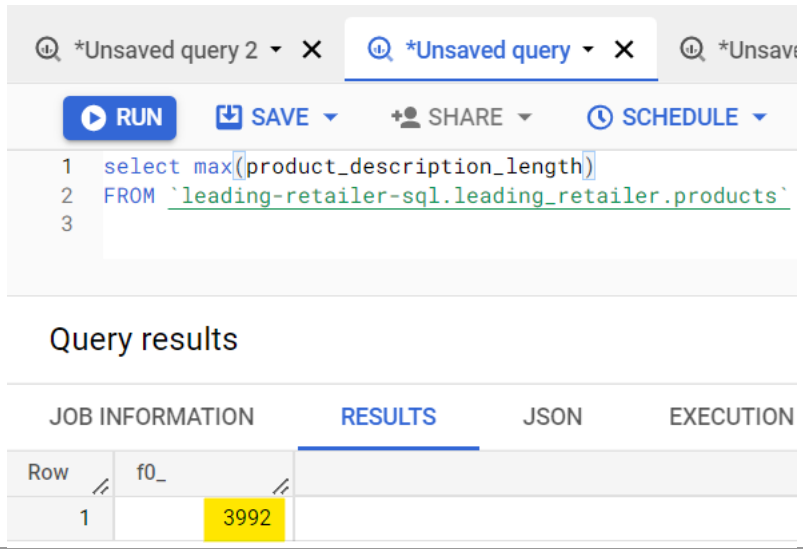
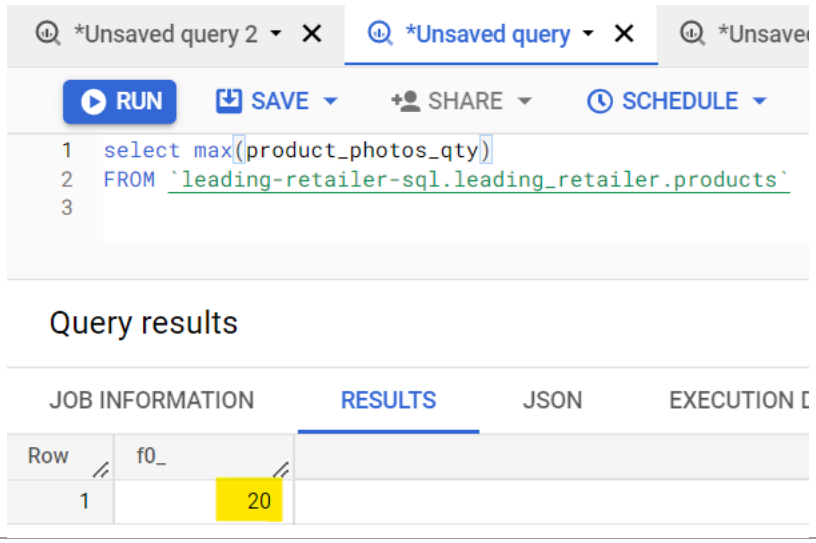
Columns	Data Type
order_id	char(32) 
payment_sequential	tinyint

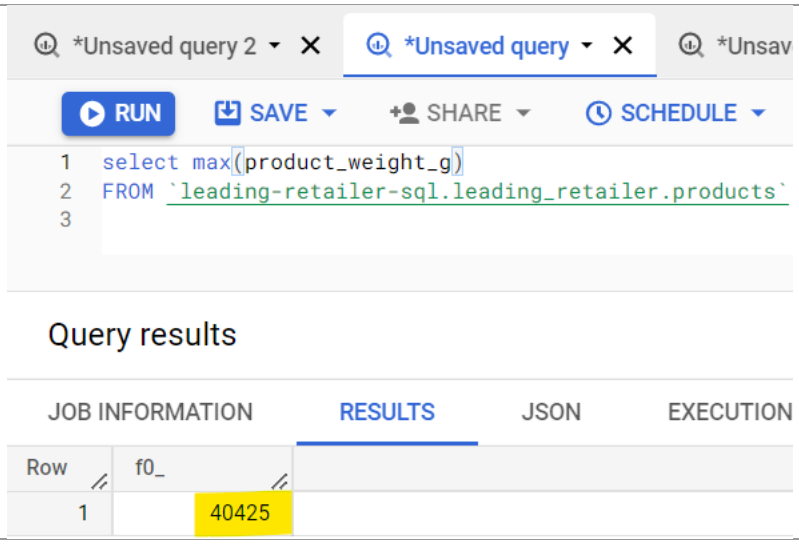
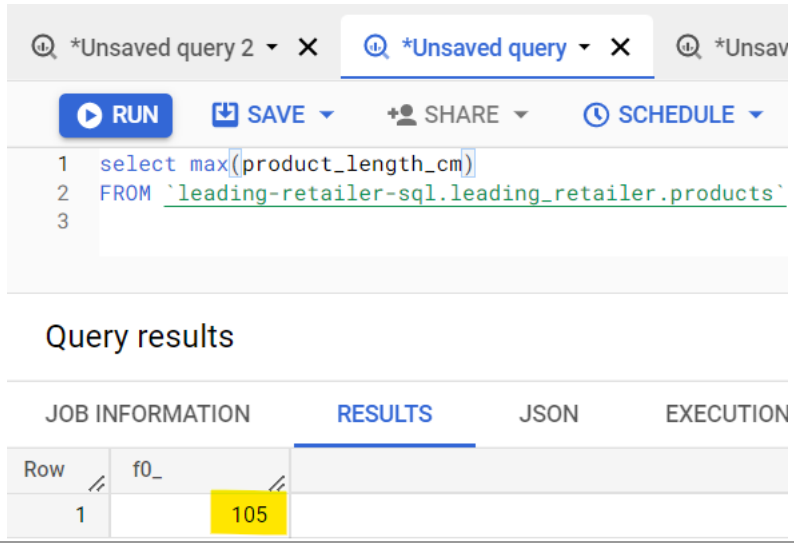
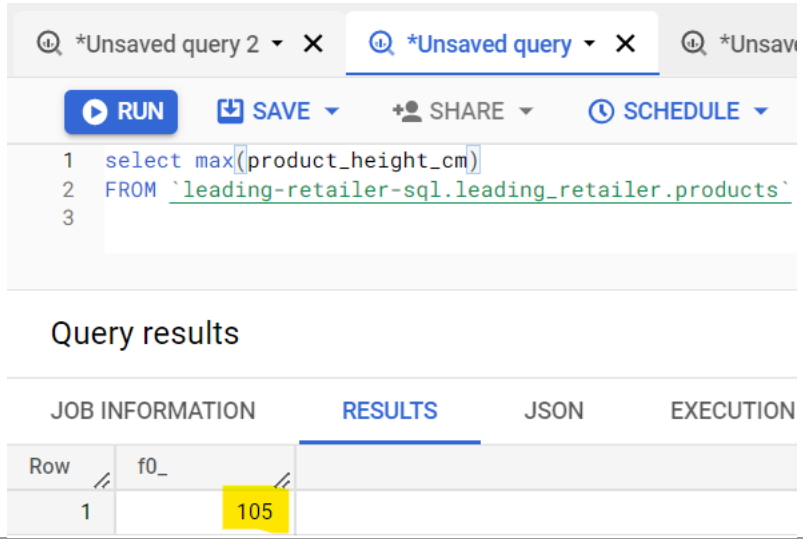
	
payment_type	varchar(11) 
payment_installments	tinyint 
payment_value	float(8)

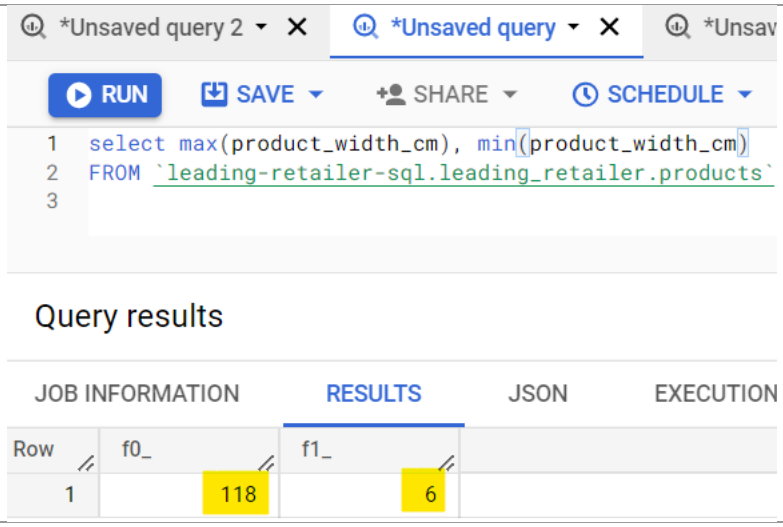
	

products

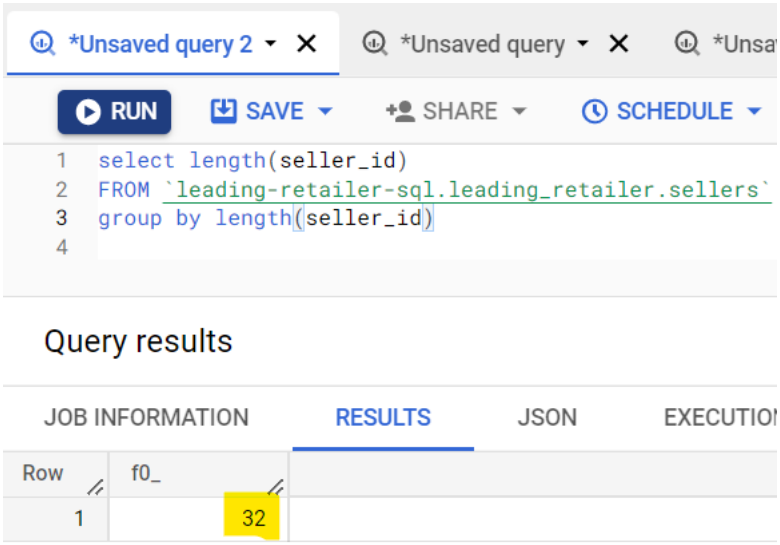
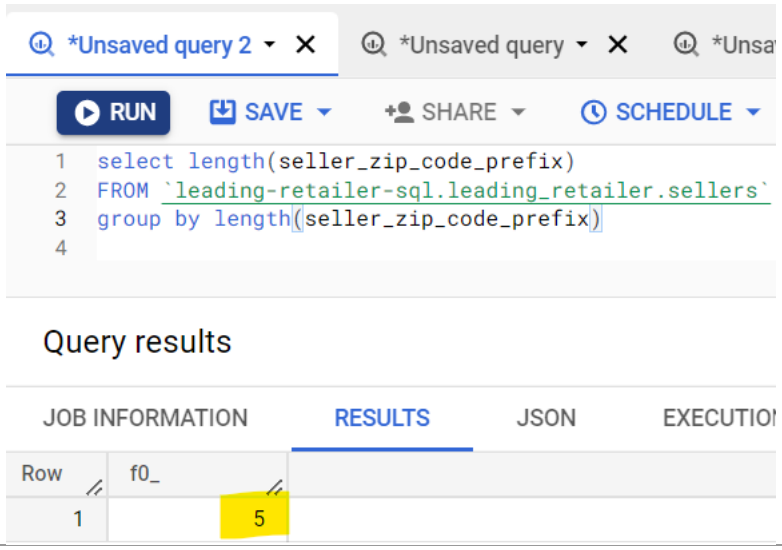
Columns	Data Type
product_id	char(32) 
product_category	varchar(48) 
product_name_length	tinyint

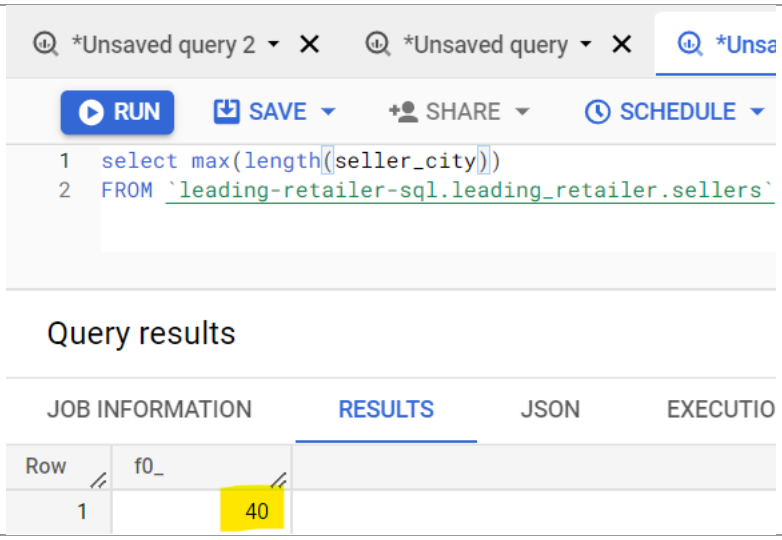
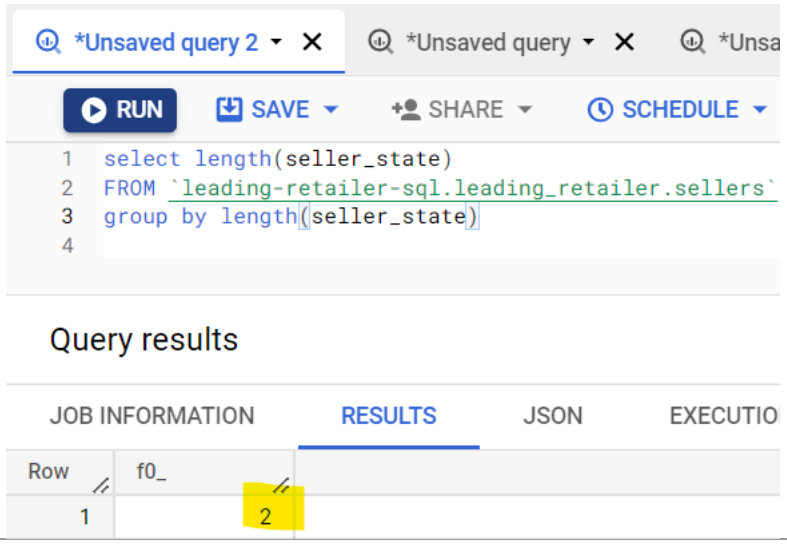
	
product_description_length	smallint 
product_photos_qty	tinyint 
product_weight_g	smallint

	
product_length_cm	tinyint 
product_height_cm	tinyint 
product_width_cm	tinyint

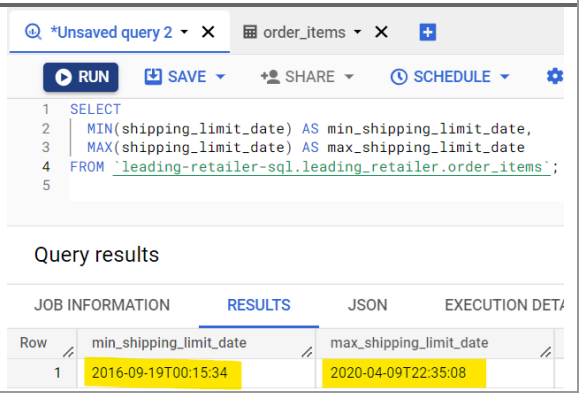
sellers

Columns	Data Type
seller_id	char(32) 
seller_zip_code_prefix	char(5) 
seller_city	varchar(40)

	
seller_state	char(2) 

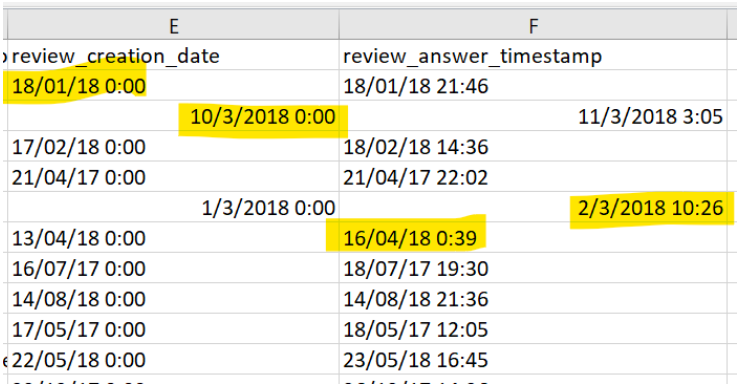
2. **Time period for which the data is given**

order_items (2016 to 2020)

Field	Query	Result
shipping_limit_date	SELECT MIN(shipping_limit_date) AS min_shipping_limit_date, MAX(shipping_limit_date) AS max_shipping_limit_date FROM `leading-retailer-sql.leading_retailer.order_items`;	

order_reviews (2016 to 2018)

The datetime data in this table was not clean (dates were in two different formats), so BigQuery was interpreting this wrong: -



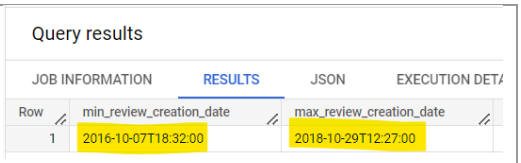
E	F
review_creation_date	review_answer_timestamp
18/01/18 0:00	18/01/18 21:46
10/3/2018 0:00	11/3/2018 3:05
17/02/18 0:00	18/02/18 14:36
21/04/17 0:00	21/04/17 22:02
1/3/2018 0:00	2/3/2018 10:26
13/04/18 0:00	16/04/18 0:39
16/07/17 0:00	18/07/17 19:30
14/08/18 0:00	14/08/18 21:36
17/05/17 0:00	18/05/17 12:05
22/05/18 0:00	23/05/18 16:45

First cleaning the data and storing in a view –

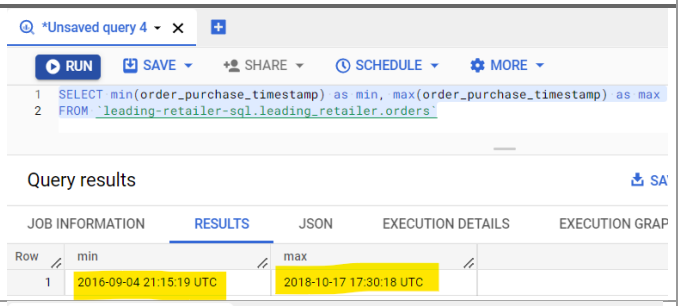
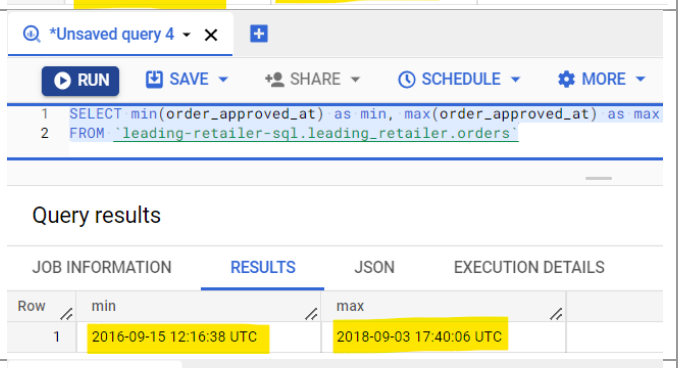
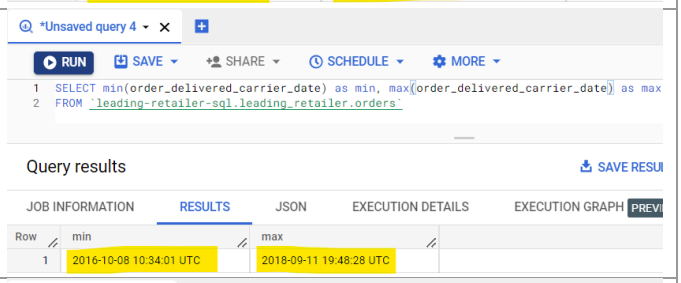
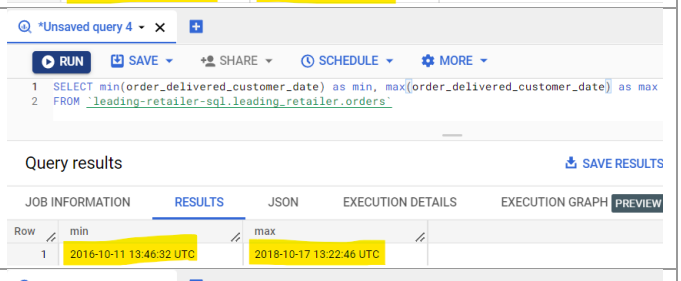
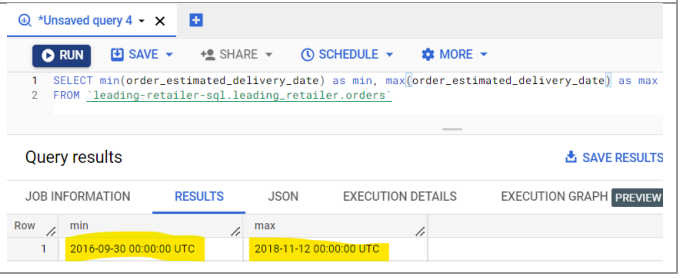
```
CREATE VIEW `leading-retailer-sql.leading_retailer.order_reviews_updated` AS
SELECT review_id,
    order_id,
    review_score,
    review_comment_title,
    review_creation_date,
    case
    when substring(string(review_creation_date), 0, 2) = '00'
    then PARSE_DATETIME('%y-%m-%d %H:%M:%S', ltrim(string(datetime(review_creation_date)),
'00'))
    else datetime(review_creation_date)
    end as review_creation_date_updated,
    review_answer_timestamp,
    case
    when substring(string(review_answer_timestamp), 0, 2) = '00'
    then PARSE_DATETIME('%y-%m-%d %H:%M:%S',
ltrim(string(datetime(review_answer_timestamp)), '00'))
    else datetime(review_answer_timestamp)
    end as review_answer_timestamp_updated,
FROM `leading-retailer-sql.leading_retailer.order_reviews`;
```

Using this view to comment on minimum and maximum datetimes -

Field	Query	Result
review_creation_date_updated	SELECT MIN(review_creation_date_updated) AS min_review_creation_date, MAX(review_creation_date_updated) AS max_review_creation_date FROM `leading-retailer-sql.leading_retailer.order_reviews_updated`;	Query results JOB INFORMATION RESULTS JSON EXECUTION DETA Row min_review_creation_date max_review_creation_date 1 2016-10-02T00:00:00 2018-08-31T00:00:00

review_answer_timestamp_updated	SELECT MIN(review_answer_timestamp_updated) AS min_review_creation_date, MAX(review_answer_timestamp_updated) AS max_review_creation_date FROM `leading-retailer-sql.leading_retailer.order_reviews_updated`;	

orders (2016 to 2018)

Field	Query	Result
order_purchase_timestamp	SELECT min(order_purchase_timestamp) as min, max(order_purchase_timestamp) as max FROM `leading-retailer-sql.leading_retailer.orders`;	
order_approved_at	SELECT min(order_approved_at) as min, max(order_approved_at) as max FROM `leading-retailer-sql.leading_retailer.orders`;	
order_delivered_carrier_date	SELECT min(order_delivered_carrier_date) as min, max(order_delivered_carrier_date) as max FROM `leading-retailer-sql.leading_retailer.orders`;	
order_delivered_customer_date	SELECT min(order_delivered_customer_date) as min, max(order_delivered_customer_date) as max FROM `leading-retailer-sql.leading_retailer.orders`;	
order_estimated_delivery_date	SELECT min(order_estimated_delivery_date) as min, max(order_estimated_delivery_date) as max FROM `leading-retailer-sql.leading_retailer.orders`;	

3. **Cities and States of customers ordered during the given period**

```
SELECT c.customer_city, c.customer_state
FROM `leading-retailer-sql.leading_retailer.orders` as o
JOIN `leading-retailer-sql.leading_retailer.customers` as c
ON o.customer_id = c.customer_id
WHERE EXTRACT(YEAR FROM o.order_purchase_timestamp) BETWEEN 2016 AND 2018
GROUP BY c.customer_city, c.customer_state;
```

2. **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?**

- Yearly trends –
    - Trend of orders

```
SELECT *
FROM (
   SELECT
      EXTRACT(YEAR FROM order_purchase_timestamp) AS year,
      COUNT(DISTINCT order_id) AS orders_per_year
   FROM `leading-retailer-sql.leading_retailer.orders`
   WHERE EXTRACT(YEAR FROM order_purchase_timestamp) BETWEEN 2016 AND
2018
   GROUP BY EXTRACT(YEAR FROM order_purchase_timestamp)
) AS x
ORDER BY x.year;
```

Query results

JOB INFORMATION | RESULTS

Row	year	orders_per_year
1	2016	329
2	2017	45101
3	2018	54011

    - Trend of customers who order

```
SELECT *
FROM (
   SELECT
      EXTRACT(YEAR FROM o.order_purchase_timestamp) AS year,
      COUNT(DISTINCT c.customer_unique_id) AS count_of_customers_per_year
```

```
    FROM `leading-retailer-sql.leading_retailer.customers` AS c
    -- joining so as to assure the data is between 2016 and 2018
    JOIN `leading-retailer-sql.leading_retailer.orders` AS o
    ON c.customer_id = o.customer_id
    WHERE EXTRACT(YEAR FROM o.order_purchase_timestamp) BETWEEN 2016 AND
2018
    GROUP BY  EXTRACT(YEAR FROM o.order_purchase_timestamp)
) AS x
ORDER BY x.year;
```

Query results

JOB INFORMATION | RESULTS | JSON

Row	year	count_of_customers_per_year
1	2016	326
2	2017	43713
3	2018	52749

- Quarterly trends
  - Trend of orders

```
SELECT *
FROM (
    SELECT
        EXTRACT(QUARTER FROM order_purchase_timestamp) AS quarter,
        COUNT(DISTINCT order_id) AS orders_per_quarter
    FROM `leading-retailer-sql.leading_retailer.orders`
    WHERE EXTRACT(YEAR FROM order_purchase_timestamp) BETWEEN 2016 AND
2018
    GROUP BY EXTRACT(QUARTER FROM order_purchase_timestamp)
) AS x
ORDER BY x.quarter;
```

Query results

JOB INFORMATION | RESULTS | JSON

Row	quarter	orders_per_quarter
1	1	26470
2	2	29328
3	3	25466
4	4	18177

- Trend of customers who order

```
SELECT  *
FROM (
    SELECT
        EXTRACT(QUARTER FROM o.order_purchase_timestamp) AS quarter,
        COUNT(DISTINCT c.customer_unique_id) AS
count_of_customers_per_quarter
    FROM `leading-retailer-sql.leading_retailer.customers` AS c
    -- joining so as to assure the data is between 2016 and 2018
    JOIN `leading-retailer-sql.leading_retailer.orders` AS o
    ON c.customer_id = o.customer_id
    WHERE EXTRACT(YEAR FROM o.order_purchase_timestamp) BETWEEN 2016 AND
2018
    GROUP BY EXTRACT(QUARTER FROM o.order_purchase_timestamp)
) AS x
ORDER BY x.quarter;
```

Query results

JOB INFORMATION | RESULTS

Row	quarter	count_of_custor
1	1	25880
2	2	28801
3	3	24941
4	4	17830

- Monthly trends
    - Trend of orders

```
SELECT x.month, x.orders_per_month
FROM (
    SELECT
        FORMAT_DATETIME("%B", DATETIME(order_purchase_timestamp)) as month,
        EXTRACT(MONTH FROM order_purchase_timestamp) AS month_number,
        COUNT(DISTINCT order_id) AS orders_per_month
    FROM `leading-retailer-sql.leading_retailer.orders`
    WHERE EXTRACT(YEAR FROM order_purchase_timestamp) BETWEEN 2016 AND
2018
    GROUP BY
```

```
        FORMAT_DATETIME("%B", DATETIME(order_purchase_timestamp)),
        EXTRACT(MONTH FROM order_purchase_timestamp)
) AS x
ORDER BY x.month_number;
```

Query results

JOB INFORMATION | RESULTS | JSON

Row	month	orders_per_month
1	January	8069
2	February	8508
3	March	9893
4	April	9343
5	May	10573
6	June	9412
7	July	10318
8	August	10843
9	September	4305
10	October	4959
11	November	7544
12	December	5674

- Trend of customers who order

```
SELECT  x.month, x.count_of_customers_per_month
FROM (
   SELECT
      FORMAT_DATETIME("%B", DATETIME(o.order_purchase_timestamp)) AS
month,
      EXTRACT(MONTH FROM o.order_purchase_timestamp) AS month_number,
      COUNT(DISTINCT c.customer_unique_id) AS count_of_customers_per_month
   FROM `leading-retailer-sql.leading_retailer.customers` AS c
   -- joining so as to assure the data is between 2016 and 2018
   JOIN `leading-retailer-sql.leading_retailer.orders` AS o
   ON c.customer_id = o.customer_id
   WHERE EXTRACT(YEAR FROM o.order_purchase_timestamp) BETWEEN 2016 AND
2018
   GROUP BY
      FORMAT_DATETIME("%B", DATETIME(o.order_purchase_timestamp)),
      EXTRACT(MONTH FROM o.order_purchase_timestamp)
) AS x
ORDER BY x.month_number;
```

Query results

JOB INFORMATION | RESULTS | JSON

Row	month	count_of_custor
1	January	7925
2	February	8321
3	March	9751
4	April	9252
5	May	10430
6	June	9302
7	July	10171
8	August	10699
9	September	4230
10	October	4886
11	November	7430
12	December	5604

From above analysis we see: -

- There is an increasing yearly trend in number of orders, where there was an abrupt increase from 2016 to 2017.
- Based on orders monthly trend we see: -
  - Maximum orders were made in month of May, July, and August.
  - Minimum orders were places in month of September, October, and December.
  - Moderate order count is observed in moth of January, February, and November.
  - For all other moths except January, September, October, November, and December the count number of orders places is greater than average number of orders i.e., **8286.74**

To increase count of orders in these months (January, September, October, November, and December) special discount can be given during these months.

2. **What time do Brazilian customers tend to buy (Dawn, Morning, Afternoon or Night)?**

   According to the data, Brazilian customers tend to buy most during “Afternoon”. The trend goes like this – Afternoon > Night > Morning > Dawn.

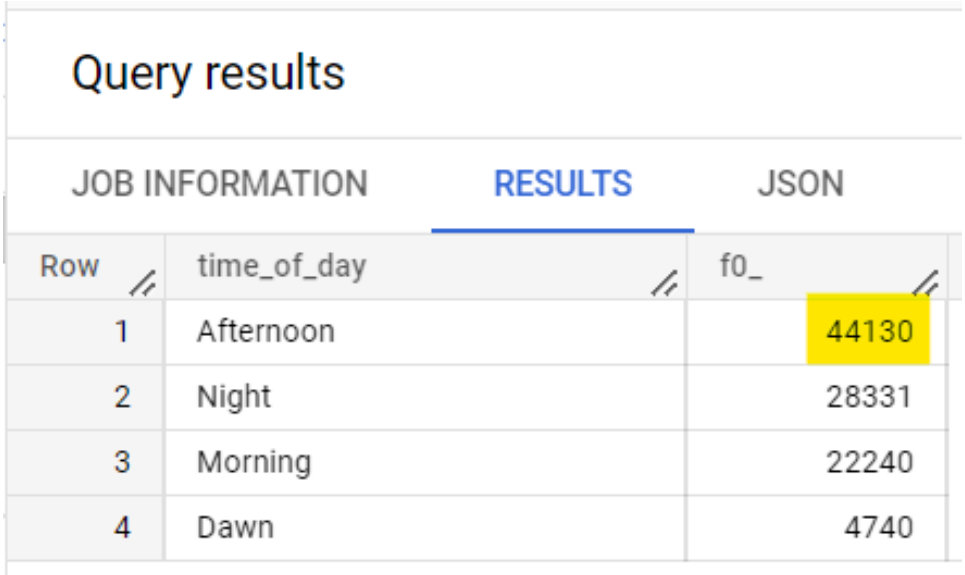
   To improve further the company can: -

- Send out notifications to users, about new products, drop in prices etc., during Afternoon.
- Start a sale during Night, Morning, or Dawn and give notification about this upcoming sale during Afternoon. So as to increase order count during these time of days as well.

The following query justifies this statement: -

```
WITH date_and_time_of_day AS (
 SELECT order_id,
    order_purchase_timestamp,
    CASE
    WHEN EXTRACT(TIME FROM order_purchase_timestamp) BETWEEN '00:00:00' AND '05:59:59'
    THEN 'Dawn'
    WHEN EXTRACT(TIME FROM order_purchase_timestamp) BETWEEN '06:00:00' AND '11:59:59'
    THEN 'Morning'
    WHEN EXTRACT(TIME FROM order_purchase_timestamp) BETWEEN '12:00:00' AND '18:59:59'
    THEN 'Afternoon'
    ELSE 'Night'
    END AS time_of_day
 FROM `leading-retailer-sql.leading_retailer.orders`
 WHERE EXTRACT(YEAR FROM order_purchase_timestamp) BETWEEN 2016 AND 2018
 -- extracting discrete orders only, since for 1 order there can be multiple records
 -- and we just need to analyse the time when order was placed
 GROUP BY order_id, order_purchase_timestamp
)

SELECT time_of_day, count(*)
FROM date_and_time_of_day
GROUP BY time_of_day
ORDER BY COUNT(*) DESC;
```



Row	time_of_day	f0_
1	Afternoon	44130
2	Night	28331
3	Morning	22240
4	Dawn	4740

3. **Evolution of E-commerce orders in the Brazil region:**
    1. **Get month on month orders by states**

The monthly orders per states can be extracted using following query -

```
WITH orders_per_month_per_state AS (
 SELECT
     FORMAT_DATETIME("%B", DATETIME(o.order_purchase_timestamp)) AS
month_of_purchase_name,
     EXTRACT(MONTH FROM o.order_purchase_timestamp) AS month_of_purchase_number,
     c.customer_state,
     COUNT(DISTINCT o.order_id) AS number_of_orders
 FROM `leading-retailer-sql.leading_retailer.orders` as o
 JOIN `leading-retailer-sql.leading_retailer.customers` as c
 ON o.customer_id = c.customer_id
 WHERE EXTRACT(YEAR FROM o.order_purchase_timestamp) BETWEEN 2016 AND 2018
 GROUP BY
     FORMAT_DATETIME("%B", DATETIME(o.order_purchase_timestamp)),
     EXTRACT(MONTH FROM o.order_purchase_timestamp),
     c.customer_state
)

SELECT customer_state, month_of_purchase_name, number_of_orders
FROM orders_per_month_per_state
ORDER BY customer_state, month_of_purchase_number;
```

Query results

JOB INFORMATION | RESULTS | JSON | EXECUTION DETAILS | EXEC

Row	customer_state	month_of_purchase_name	number_of_orde
1	AC	January	8
2	AC	February	6
3	AC	March	4
4	AC	April	9
5	AC	May	10
6	AC	June	7
7	AC	July	9
8	AC	August	7
9	AC	September	5
10	AC	October	6

2. **Distribution of customers across the states in Brazil**

```
SELECT
    c.customer_state,
    COUNT(DISTINCT c.customer_id) AS number_of_customers
FROM `leading-retailer-sql.leading_retailer.orders` as o
```

```
-- joining to ensure that we are taking only orders between 2016 and 2018
JOIN `leading-retailer-sql.leading_retailer.customers` as c
ON o.customer_id = c.customer_id
WHERE EXTRACT(YEAR FROM o.order_purchase_timestamp) BETWEEN 2016 AND 2018
GROUP BY c.customer_state
ORDER BY COUNT(DISTINCT c.customer_id) DESC;
```

Query results

JOB INFORMATION | RESULTS | JSON

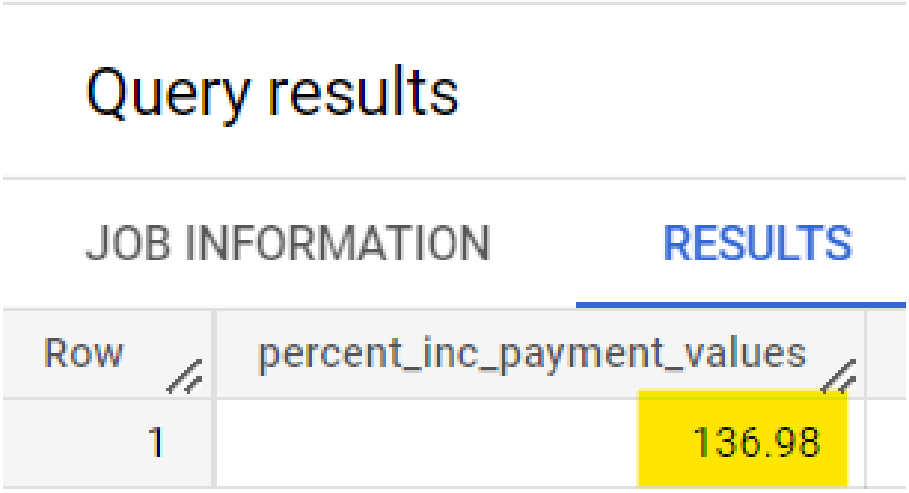
Row	customer_state	number_of_cust
1	SP	41746
2	RJ	12852
3	MG	11635
4	RS	5466
5	PR	5045
6	SC	3637
7	BA	3380
8	DF	2140
9	ES	2033
10	GO	2020

4. **Impact on Economy: Analyze the money movement 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) - You can use "payment_value" column in payments table**
        i. Yearly analysis: - We see an increase of **136.98%** in cost of orders between Jan to Aug 2017 and Jan to Aug 2018.

```
WITH order_purchases_2017 AS (
    SELECT
        EXTRACT(YEAR FROM o.order_purchase_timestamp) AS year,
        SUM(p.payment_value) AS total_payment_value
    FROM `leading-retailer-sql.leading_retailer.orders` as o
    JOIN `leading-retailer-sql.leading_retailer.payments` as p
    ON o.order_id = p.order_id
    WHERE EXTRACT(DATE FROM o.order_purchase_timestamp) BETWEEN '2017-01-01'
AND '2017-08-31'
    GROUP BY EXTRACT(YEAR FROM o.order_purchase_timestamp)
), order_purchases_2018 AS (
    SELECT
        EXTRACT(YEAR FROM o.order_purchase_timestamp) AS year,
        SUM(p.payment_value) AS total_payment_value
    FROM `leading-retailer-sql.leading_retailer.orders` as o
    JOIN `leading-retailer-sql.leading_retailer.payments` as p
    ON o.order_id = p.order_id
    WHERE EXTRACT(DATE FROM o.order_purchase_timestamp) BETWEEN '2018-01-01'
AND '2018-08-31'
```

```
    GROUP BY EXTRACT(YEAR FROM o.order_purchase_timestamp)
)

SELECT ROUND(((b.total_payment_value -
a.total_payment_value)/a.total_payment_value) * 100, 2) AS
percent_inc_payment_values
FROM order_purchases_2017 AS a, order_purchases_2018 AS b;
```



ii. Yearly analysis per payment_type: - There is an **increase** in cost of orders for each payment types between Jan to Aug 2017 and Jan to Aug 2018. **Out of which we see greatest increase in payments done via debit card.**

The company can give special discounts on payment via UPI, voucher, or credit card.

```
WITH order_purchases_2017 AS (
    SELECT
        EXTRACT(YEAR FROM o.order_purchase_timestamp) AS year,
        p.payment_type,
        SUM(p.payment_value) AS total_payment_value
    FROM `leading-retailer-sql.leading_retailer.orders` as o
    JOIN `leading-retailer-sql.leading_retailer.payments` as p
    ON o.order_id = p.order_id
    WHERE EXTRACT(DATE FROM o.order_purchase_timestamp) BETWEEN '2017-01-01'
AND '2017-08-31'
    GROUP BY EXTRACT(YEAR FROM o.order_purchase_timestamp), p.payment_type
), order_purchases_2018 AS (
    SELECT
        EXTRACT(YEAR FROM o.order_purchase_timestamp) AS year,
        p.payment_type,
        SUM(p.payment_value) AS total_payment_value
    FROM `leading-retailer-sql.leading_retailer.orders` as o
    JOIN `leading-retailer-sql.leading_retailer.payments` as p
    ON o.order_id = p.order_id
    WHERE EXTRACT(DATE FROM o.order_purchase_timestamp) BETWEEN '2018-01-01'
AND '2018-08-31'
    GROUP BY EXTRACT(YEAR FROM o.order_purchase_timestamp), p.payment_type
)
```

```
SELECT a.payment_type, ROUND(((b.total_payment_value -
a.total_payment_value)/a.total_payment_value) * 100, 2) AS
percent_inc_payment_values
FROM order_purchases_2017 AS a
JOIN order_purchases_2018 AS b
ON a.payment_type = b.payment_type;
```

Query results

JOB INFORMATION | RESULTS | JSON

Row	payment_type	percent_inc_pay
1	voucher	113.12
2	credit_card	142.88
3	UPI	99.81
4	debit_card	791.45

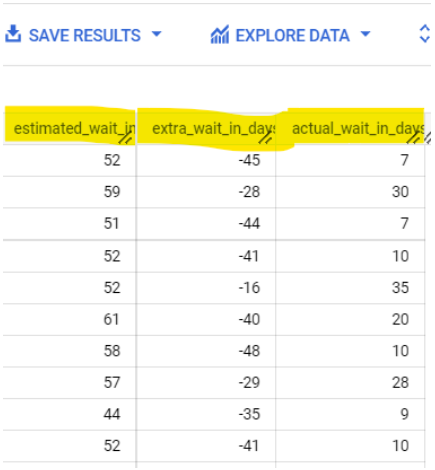
2. **Mean & Sum of price and freight value by customer state**

```
SELECT
    c.customer_state,
    ROUND(AVG(oi.price), 2) AS mean_price,
    ROUND(SUM(oi.price), 2) AS sum_price,
    ROUND(AVG(oi.freight_value), 2) AS mean_freight_value,
    ROUND(SUM(oi.freight_value), 2) AS sum_freight_value
FROM `leading-retailer-sql.leading_retailer.customers` AS c
JOIN `leading-retailer-sql.leading_retailer.orders` AS o
ON c.customer_id = o.customer_id
JOIN `leading-retailer-sql.leading_retailer.order_items` AS oi
ON o.order_id = oi.order_id
WHERE EXTRACT(YEAR FROM o.order_purchase_timestamp) BETWEEN 2016 AND 2018
GROUP BY c.customer_state;
```

5. **Analysis on sales, freight and delivery time**
    1. **Calculate days between purchasing, delivering and estimated delivery**
        i. For orders which have order_delivered_customer_date

```
SELECT
    order_id,
    order_status,
    order_purchase_timestamp,
    order_estimated_delivery_date,
    order_delivered_customer_date,
    DATE_DIFF(order_estimated_delivery_date, order_purchase_timestamp, DAY) AS
estimated_wait_in_days,
```

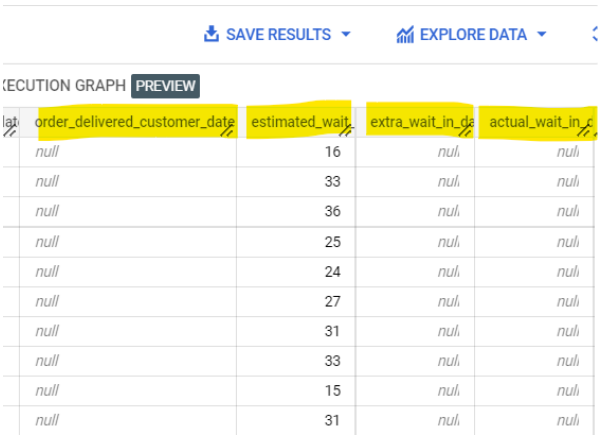
```
    DATE_DIFF(order_delivered_customer_date, order_estimated_delivery_date, DAY)
AS extra_wait_in_days,
    DATE_DIFF(order_delivered_customer_date, order_purchase_timestamp, DAY) AS
actual_wait_in_days
FROM `leading-retailer-sql.leading_retailer.orders`
WHERE EXTRACT(YEAR FROM order_purchase_timestamp) BETWEEN 2016 AND 2018
AND order_delivered_customer_date is not null;
```



estimated_wait_in	extra_wait_in_days	actual_wait_in_days
52	-45	7
59	-28	30
51	-44	7
52	-41	10
52	-16	35
61	-40	20
58	-48	10
57	-29	28
44	-35	9
52	-41	10

ii. For orders which do not have order_delivered_customer_date

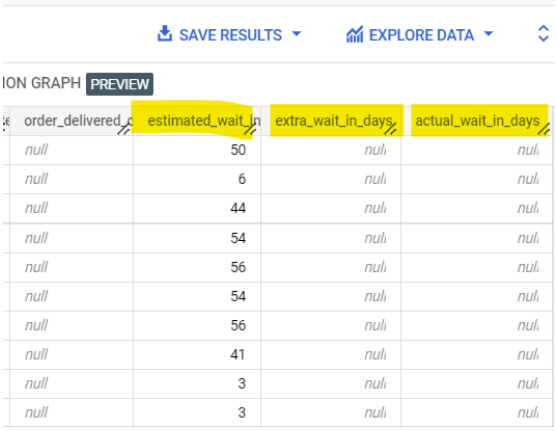
```
SELECT
    order_id,
    order_status,
    order_purchase_timestamp,
    order_estimated_delivery_date,
    order_delivered_customer_date,
    DATE_DIFF(order_estimated_delivery_date, order_purchase_timestamp, DAY) AS
estimated_wait_in_days,
    DATE_DIFF(order_delivered_customer_date, order_estimated_delivery_date, DAY)
AS extra_wait_in_days,
    DATE_DIFF(order_delivered_customer_date, order_purchase_timestamp, DAY) AS
actual_wait_in_days
FROM `leading-retailer-sql.leading_retailer.orders`
WHERE EXTRACT(YEAR FROM order_purchase_timestamp) BETWEEN 2016 AND 2018
AND order_delivered_customer_date is null;
```



order_delivered_customer_date	estimated_wait_	extra_wait_in_da	actual_wait_in_d
null	16	null	null
null	33	null	null
null	36	null	null
null	25	null	null
null	24	null	null
null	27	null	null
null	31	null	null
null	33	null	null
null	15	null	null
null	31	null	null

iii. For overall orders

```
SELECT
    order_id,
    order_status,
    order_purchase_timestamp,
    order_estimated_delivery_date,
    order_delivered_customer_date,
    DATE_DIFF(order_estimated_delivery_date, order_purchase_timestamp, DAY) AS
estimated_wait_in_days,
    DATE_DIFF(order_delivered_customer_date, order_estimated_delivery_date, DAY)
AS extra_wait_in_days,
    DATE_DIFF(order_delivered_customer_date, order_purchase_timestamp, DAY) AS
actual_wait_in_days
FROM `leading-retailer-sql.leading_retailer.orders`
WHERE EXTRACT(YEAR FROM order_purchase_timestamp) BETWEEN 2016 AND 2018;
```



order_delivered_c	estimated_wait_in	extra_wait_in_days	actual_wait_in_days
null	50	null	null
null	6	null	null
null	44	null	null
null	54	null	null
null	56	null	null
null	54	null	null
null	56	null	null
null	41	null	null
null	3	null	null
null	3	null	null

From the analysis above and below mentioned further analysis we see that:-

- Maximum extra wait time (i.e., when order get delivered after estimated date order_delivered_customer_date - order_estimated_delivery_date) is **188 days**.

```
SELECT
    order_id,
    order_status,
    order_purchase_timestamp,
    order_estimated_delivery_date,
    order_delivered_customer_date,
    DATE_DIFF(order_estimated_delivery_date, order_purchase_timestamp, DAY) AS
estimated_wait_in_days,
    DATE_DIFF(order_delivered_customer_date, order_estimated_delivery_date, DAY) AS
extra_wait_in_days,
    DATE_DIFF(order_delivered_customer_date, order_purchase_timestamp, DAY) AS
actual_wait_in_days
FROM `leading-retailer-sql.leading_retailer.orders`
WHERE EXTRACT(YEAR FROM order_purchase_timestamp) BETWEEN 2016 AND 2018
ORDER BY DATE_DIFF(order_delivered_customer_date, order_estimated_delivery_date,
DAY) DESC;
```

SAVE RESULTS EXPLORE DATA

EW

estimated_wait	extra_wait_in_days	actual_wait_in_d
19	188	208
28	181	209
15	175	191
22	167	189
28	166	194
30	165	195
25	162	187
32	161	194
13	161	175
28	159	188

- Most of the orders were delivered on or before time (i.e., order_delivered_customer_date <= order_estimated_delivery_date)

The company can advertise about how their orders are delivered mostly on or before estimated delivery date.

```
WITH order_delay AS (
    SELECT
        SIGN(DATE_DIFF(order_delivered_customer_date, order_estimated_delivery_date,
DAY)) AS delivered_before_or_after_estimated_date_sign,
        COUNT(*) AS delivery_delay
    FROM `leading-retailer-sql.leading_retailer.orders`
    WHERE EXTRACT(YEAR FROM order_purchase_timestamp) BETWEEN 2016 AND 2018
    AND DATE_DIFF(order_delivered_customer_date, order_estimated_delivery_date, DAY)
IS NOT NULL
    GROUP BY SIGN(DATE_DIFF(order_delivered_customer_date,
order_estimated_delivery_date, DAY))
)

SELECT x.delivered_before_or_after_estimated_date, SUM(delivery_delay)
FROM (
    SELECT IF(delivered_before_or_after_estimated_date_sign IN (0, -1),
                'On time', 'Late') AS delivered_before_or_after_estimated_date,
            delivery_delay
    FROM order_delay
) AS x
GROUP BY x.delivered_before_or_after_estimated_date;
```

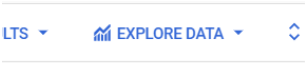
Query results

JOB INFORMATION RESULTS JSON

Row	delivered_before_or_after_estimate	f0_
1	On time	89941
2	Late	6535

2. **Find time_to_delivery & diff_estimated_delivery. Formula for the same given below:**
    i. **time_to_delivery = order_purchase_timestamp-order_delivered_customer_date**
    ii. **diff_estimated_delivery = order_estimated_delivery_date-order_delivered_customer_date**

```
SELECT
    order_id,
    order_status,
    order_purchase_timestamp,
    order_estimated_delivery_date,
    order_delivered_customer_date,
    DATE_DIFF(order_purchase_timestamp, order_delivered_customer_date, DAY) AS time_to_delivery,
    DATE_DIFF(order_estimated_delivery_date, order_delivered_customer_date, DAY) AS diff_estimated_delivery
FROM `leading-retailer-sql.leading_retailer.orders`
WHERE EXTRACT(YEAR FROM order_purchase_timestamp) BETWEEN 2016 AND 2018;
```



time_to_delivery	diff_estimated_delivery
-30	-12
-30	28
-35	16
-30	1
-32	0
-29	1
-43	-4
-40	-4
-37	-1
-33	-5

3. **Group data by state, take mean of freight_value, time_to_delivery, diff_estimated_delivery**

```
SELECT
    c.customer_state,
    ROUND(AVG(oi.freight_value), 2) AS mean_freight_value,
    ROUND(AVG(DATE_DIFF(order_purchase_timestamp, order_delivered_customer_date, DAY) )) AS mean_time_to_delivery,
    ROUND(AVG(DATE_DIFF(order_estimated_delivery_date, order_delivered_customer_date, DAY))) AS mean_diff_estimated_delivery
FROM `leading-retailer-sql.leading_retailer.customers` AS c
JOIN `leading-retailer-sql.leading_retailer.orders` AS o
ON c.customer_id = o.customer_id
JOIN `leading-retailer-sql.leading_retailer.order_items` AS oi
ON o.order_id = oi.order_id
WHERE EXTRACT(YEAR FROM o.order_purchase_timestamp) BETWEEN 2016 AND 2018
GROUP BY c.customer_state;
```

Query results

JOB INFORMATION | RESULTS | JSON | EXECUTION DETAILS | EXECUTION GRAPH PREVIEW

Row	customer_state	mean_freight_value	mean_time_to_delivery	mean_diff_estimated_delivery
1	MT	28.17	-18.0	14.0
2	MA	38.26	-21.0	9.0
3	AL	35.84	-24.0	8.0
4	SP	15.15	-8.0	10.0
5	MG	20.63	-12.0	12.0
6	PE	32.92	-18.0	13.0
7	RJ	20.96	-15.0	11.0
8	DF	21.04	-13.0	11.0
9	RS	21.74	-15.0	13.0
10	SE	36.65	-21.0	9.0

4. **Sort the data to get the following:**
    i. **Top 5 states with highest/lowest average freight value - sort in desc/asc limit 5**

Highest average freight –

```
SELECT
    c.customer_state,
    ROUND(AVG(oi.freight_value), 2) AS mean_freight_value,
    ROUND(AVG(DATE_DIFF(order_purchase_timestamp,
order_delivered_customer_date, DAY) )) AS mean_time_to_delivery,
    ROUND(AVG(DATE_DIFF(order_estimated_delivery_date,
order_delivered_customer_date, DAY))) AS mean_diff_estimated_delivery
FROM `leading-retailer-sql.leading_retailer.customers` AS c
JOIN `leading-retailer-sql.leading_retailer.orders` AS o
ON c.customer_id = o.customer_id
JOIN `leading-retailer-sql.leading_retailer.order_items` AS oi
ON o.order_id = oi.order_id
WHERE EXTRACT(YEAR FROM o.order_purchase_timestamp) BETWEEN 2016 AND 2018
GROUP BY c.customer_state
ORDER BY AVG(oi.freight_value) DESC
LIMIT 5;
```

Query results

JOB INFORMATION | RESULTS | JSON | EXECUTION DETAILS | EXECUTION GRAPH PREVIEW

Row	customer_state	mean_freight_value	mean_time_to_delivery	mean_diff_estimated_delivery
1	RR	42.98	-28.0	17.0
2	PB	42.72	-20.0	12.0
3	RO	41.07	-19.0	19.0
4	AC	40.07	-20.0	20.0
5	PI	39.15	-19.0	11.0

Lowest average freight –

```
SELECT
    c.customer_state,
    ROUND(AVG(oi.freight_value), 2) AS mean_freight_value,
    ROUND(AVG(DATE_DIFF(order_purchase_timestamp,
order_delivered_customer_date, DAY) )) AS mean_time_to_delivery,
    ROUND(AVG(DATE_DIFF(order_estimated_delivery_date,
order_delivered_customer_date, DAY))) AS mean_diff_estimated_delivery
FROM `leading-retailer-sql.leading_retailer.customers` AS c
JOIN `leading-retailer-sql.leading_retailer.orders` AS o
ON c.customer_id = o.customer_id
JOIN `leading-retailer-sql.leading_retailer.order_items` AS oi
ON o.order_id = oi.order_id
WHERE EXTRACT(YEAR FROM o.order_purchase_timestamp) BETWEEN 2016 AND 2018
GROUP BY c.customer_state
ORDER BY AVG(oi.freight_value)
LIMIT 5;
```

Query results

SAVE RES

JOB INFORMATION | RESULTS | JSON | EXECUTION DETAILS | EXECUTION GRAPH PREVIEW

Row	customer_state	mean_freight_value	mean_time_to_delivery	mean_diff_estimated_delivery
1	SP	15.15	-8.0	10.0
2	PR	20.53	-11.0	13.0
3	MG	20.63	-12.0	12.0
4	RJ	20.96	-15.0	11.0
5	DF	21.04	-13.0	11.0

ii. Top 5 states with highest/lowest average time to delivery

Highest average time of delivery –

```
SELECT
    c.customer_state,
    ROUND(AVG(oi.freight_value), 2) AS mean_freight_value,
    ROUND(AVG(DATE_DIFF(order_purchase_timestamp,
order_delivered_customer_date, DAY))) AS mean_time_to_delivery,
    ROUND(AVG(DATE_DIFF(order_estimated_delivery_date,
order_delivered_customer_date, DAY))) AS mean_diff_estimated_delivery
FROM `leading-retailer-sql.leading_retailer.customers` AS c
JOIN `leading-retailer-sql.leading_retailer.orders` AS o
ON c.customer_id = o.customer_id
JOIN `leading-retailer-sql.leading_retailer.order_items` AS oi
ON o.order_id = oi.order_id
WHERE EXTRACT(YEAR FROM o.order_purchase_timestamp) BETWEEN 2016 AND 2018
GROUP BY c.customer_state
ORDER BY AVG(DATE_DIFF(order_purchase_timestamp,
order_delivered_customer_date, DAY)) DESC
LIMIT 5;
```

Query results

SAVE

JOB INFORMATION | RESULTS | JSON | EXECUTION DETAILS | EXECUTION GRAPH PREVIEW

Row	customer_state	mean_freight_va	mean_time_to_delivery	mean_diff_estimated_delivery
1	SP	15.15	-8.0	10.0
2	PR	20.53	-11.0	13.0
3	MG	20.63	-12.0	12.0
4	DF	21.04	-13.0	11.0
5	SC	21.47	-15.0	11.0

Lowest average time of delivery –

```
SELECT
    c.customer_state,
    ROUND(AVG(oi.freight_value), 2) AS mean_freight_value,
    ROUND(AVG(DATE_DIFF(order_purchase_timestamp,
order_delivered_customer_date, DAY))) AS mean_time_to_delivery,
    ROUND(AVG(DATE_DIFF(order_estimated_delivery_date,
order_delivered_customer_date, DAY))) AS mean_diff_estimated_delivery
FROM `leading-retailer-sql.leading_retailer.customers` AS c
JOIN `leading-retailer-sql.leading_retailer.orders` AS o
ON c.customer_id = o.customer_id
JOIN `leading-retailer-sql.leading_retailer.order_items` AS oi
ON o.order_id = oi.order_id
WHERE EXTRACT(YEAR FROM o.order_purchase_timestamp) BETWEEN 2016 AND 2018
GROUP BY c.customer_state
ORDER BY AVG(DATE_DIFF(order_purchase_timestamp,
order_delivered_customer_date, DAY))
LIMIT 5;
```

Query results

SAVE RESULTS

JOB INFORMATION | RESULTS | JSON | EXECUTION DETAILS | EXECUTION GRAPH PREVIEW

Row	customer_state	mean_freight_value	mean_time_to_delivery	mean_diff_estimated_delivery
1	RR	42.98	-28.0	17.0
2	AP	34.01	-28.0	17.0
3	AM	33.21	-26.0	19.0
4	AL	35.84	-24.0	8.0
5	PA	35.83	-23.0	13.0

iii. Top 5 states where delivery is really fast/ not so fast compared to estimated date

Top 5 fastest deliveries –

```
SELECT
    c.customer_state,
    ROUND(AVG(oi.freight_value), 2) AS mean_freight_value,
    ROUND(AVG(DATE_DIFF(order_purchase_timestamp,
order_delivered_customer_date, DAY))) AS mean_time_to_delivery,
```

```
    ROUND(AVG(DATE_DIFF(order_estimated_delivery_date,
order_delivered_customer_date, DAY))) AS mean_diff_estimated_delivery
FROM `leading-retailer-sql.leading_retailer.customers` AS c
JOIN `leading-retailer-sql.leading_retailer.orders` AS o
ON c.customer_id = o.customer_id
JOIN `leading-retailer-sql.leading_retailer.order_items` AS oi
ON o.order_id = oi.order_id
WHERE EXTRACT(YEAR FROM o.order_purchase_timestamp) BETWEEN 2016 AND 2018
GROUP BY c.customer_state
ORDER BY AVG(DATE_DIFF(order_estimated_delivery_date,
order_delivered_customer_date, DAY)) DESC
LIMIT 5;
```

Query results

SAVE RESUL

JOB INFORMATION | RESULTS | JSON | EXECUTION DETAILS | EXECUTION GRAPH PREVIEW

Row	customer_state	mean_freight_value	mean_time_to_delivery	mean_diff_estimated_delivery
1	AC	40.07	-20.0	20.0
2	RO	41.07	-19.0	19.0
3	AM	33.21	-26.0	19.0
4	AP	34.01	-28.0	17.0
5	RR	42.98	-28.0	17.0

Top 5 slowest deliveries –

```
SELECT
    c.customer_state,
    ROUND(AVG(oi.freight_value), 2) AS mean_freight_value,
    ROUND(AVG(DATE_DIFF(order_purchase_timestamp,
order_delivered_customer_date, DAY))) AS mean_time_to_delivery,
    ROUND(AVG(DATE_DIFF(order_estimated_delivery_date,
order_delivered_customer_date, DAY))) AS mean_diff_estimated_delivery
FROM `leading-retailer-sql.leading_retailer.customers` AS c
JOIN `leading-retailer-sql.leading_retailer.orders` AS o
ON c.customer_id = o.customer_id
JOIN `leading-retailer-sql.leading_retailer.order_items` AS oi
ON o.order_id = oi.order_id
WHERE EXTRACT(YEAR FROM o.order_purchase_timestamp) BETWEEN 2016 AND 2018
GROUP BY c.customer_state
ORDER BY AVG(DATE_DIFF(order_estimated_delivery_date,
order_delivered_customer_date, DAY))
LIMIT 5;
```

Query results

SAVE RESULTS

JOB INFORMATION | RESULTS | JSON | EXECUTION DETAILS | EXECUTION GRAPH PREVIEW

Row	customer_state	mean_freight_value	mean_time_to_delivery	mean_diff_estimated_delivery
1	AL	35.84	-24.0	8.0
2	MA	38.26	-21.0	9.0
3	SE	36.65	-21.0	9.0
4	ES	22.06	-15.0	10.0
5	BA	26.36	-19.0	10.0

6. **Payment type analysis:**
    1. **Month over Month count of orders for different payment types**

```
WITH orders_per_month_per_payment_types AS (
 SELECT
     FORMAT_DATETIME("%B", DATETIME(o.order_purchase_timestamp)) AS
month_of_purchase_name,
     EXTRACT(MONTH FROM o.order_purchase_timestamp) AS month_of_purchase_number,
     p.payment_type,
     COUNT(DISTINCT o.order_id) AS number_of_orders
 FROM `leading-retailer-sql.leading_retailer.orders` as o
 JOIN `leading-retailer-sql.leading_retailer.payments` as p
 ON o.order_id = p.order_id
 WHERE EXTRACT(YEAR FROM o.order_purchase_timestamp) BETWEEN 2016 AND 2018
 GROUP BY
     FORMAT_DATETIME("%B", DATETIME(o.order_purchase_timestamp)),
     EXTRACT(MONTH FROM o.order_purchase_timestamp),
     p.payment_type
)

SELECT payment_type, month_of_purchase_name, number_of_orders
FROM orders_per_month_per_payment_types
ORDER BY UPPER(payment_type), month_of_purchase_number;
```

Query results

SA

JOB INFORMATION | RESULTS | JSON | EXECUTION DETAILS | EXE

Row	payment_type	month_of_purchase_name	number_of_orde
1	credit_card	January	6093
2	credit_card	February	6582
3	credit_card	March	7682
4	credit_card	April	7276
5	credit_card	May	8308
6	credit_card	June	7248
7	credit_card	July	7810
8	credit_card	August	8235
9	credit_card	September	3277
10	credit_card	October	3763

2. Count of orders based on the no. of payment installments

```
SELECT
    p.payment_installments,
    COUNT(DISTINCT o.order_id) AS number_of_orders
FROM `leading-retailer-sql.leading_retailer.orders` as o
-- joining to ensure that we are taking only orders between 2016 and 2018
JOIN `leading-retailer-sql.leading_retailer.payments` as p
ON o.order_id = p.order_id
WHERE EXTRACT(YEAR FROM o.order_purchase_timestamp) BETWEEN 2016 AND 2018
GROUP BY p.payment_installments
ORDER BY p.payment_installments;
```

Query results

JOB INFORMATION | RESULTS | JSON | EXECUTION

Row	payment_installments	number_of_orders
1	0	2
2	1	49060
3	2	12389
4	3	10443
5	4	7088
6	5	5234
7	6	3916
8	7	1623
9	8	4253
10	9	644

Except for the payment_installments = 0, we see a decrease in number_of_orders as the payment_installments increase.

Offer discount and/or less EMI on higher number of installments.