TARGET SQL PROJECT

Apr 18, 2023

1)

[a]

```
SELECT column_name,data_type from sqlproject.INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME='ORDERS'
```

Query results

JOB INFORMATION | RESULTS | JSON | EXECUTION DETAILS

Row	column_name	data_type
1	order_id	STRING
2	customer_id	STRING
3	order_status	STRING
4	order_purchase_timestamp	TIMESTAMP
5	order_approved_at	TIMESTAMP
6	order_delivered_carrier_date	TIMESTAMP
7	order_delivered_customer_date	TIMESTAMP
8	order_estimated_delivery_date	TIMESTAMP

DATA TYPES OF COLUMNS IN A TABLE:

ORDERS Table:

ORDERS | QUERY | SHARE | COPY | SNAPSHO

SCHEMA | DETAILS | PREVIEW | LINEAGE

Filter Enter property name or value

Field name	Type	Mode	Collatio
order_id	STRING	NULLABLE	
customer_id	STRING	NULLABLE	
order_status	STRING	NULLABLE	
order_purchase_timestamp	TIMESTAMP	NULLABLE	
order_approved_at	TIMESTAMP	NULLABLE	
order_delivered_carrier_date	TIMESTAMP	NULLABLE	
order_delivered_customer_date	TIMESTAMP	NULLABLE	
order_estimated_delivery_date	TIMESTAMP	NULLABLE	

CONCLUSION:
The Data Type of columns, order_id is *STRING*, customer_id is *STRING*, order_status is *STRING*, order_purchases_timestamp is *TIMESTAMP*, order_approved_at is *TIMESTAMP*, order_delivered_carrier_date is *TIMESTAMP* , order_delivered_customer_date is *TIMESTAMP*, order_estimate_delivery_date is *TIMESTAMP*.

ORDER ITEMS Table :

Order Items | QUERY | SHARE | COPY

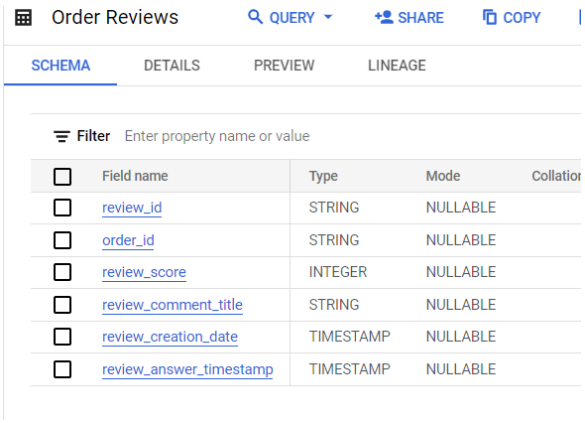
SCHEMA | DETAILS | PREVIEW | LINEAGE

Filter Enter property name or value

Field name	Type	Mode	Collation
order_id	STRING	NULLABLE	
order_item_id	INTEGER	NULLABLE	
product_id	STRING	NULLABLE	
seller_id	STRING	NULLABLE	
shipping_limit_date	TIMESTAMP	NULLABLE	
price	FLOAT	NULLABLE	
freight_value	FLOAT	NULLABLE	

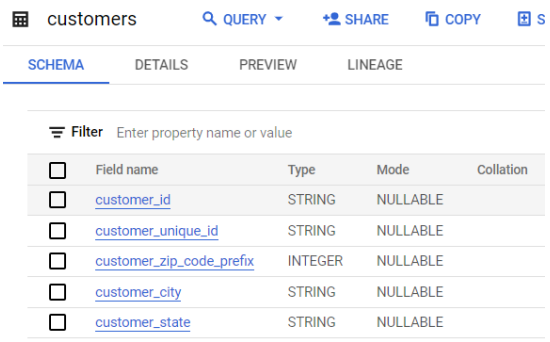
CONCLUSION:
The Data Type of columns, order_id is *STRING*, order_item_id is *INTEGER*, productid is *STRING*, seller_id is *STRING*, shipping_limit_date is *TIMESTAMP*, price is *FLOAT*, freight_value is *FLOAT*.

ORDER_REVIEWS Table :



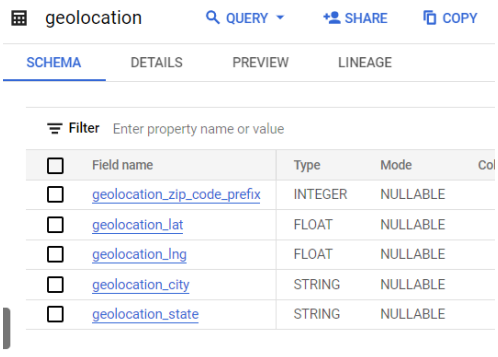
CONCLUSION:
The Data Type of the columns, review_id is *STRING*, order_id is *STRING*, review_score is *INTEGER*, review_comment_titleis *STRING*, review_creation_date is *TIMESTAMP*, review_answer_timestamp is *TIMESTAMP*.

CUSTOMERS Table:



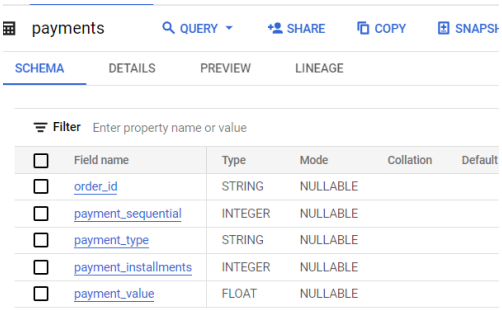
CONCLUSION:
The Data Type of the columns customer_id is *STRING*, customer_unique_id is *STRING*, customer_zip_code_prefix is *INTEGER*, customer_id is *STRING*, customer_state is *STRING*.

GEOLOCATION Table:



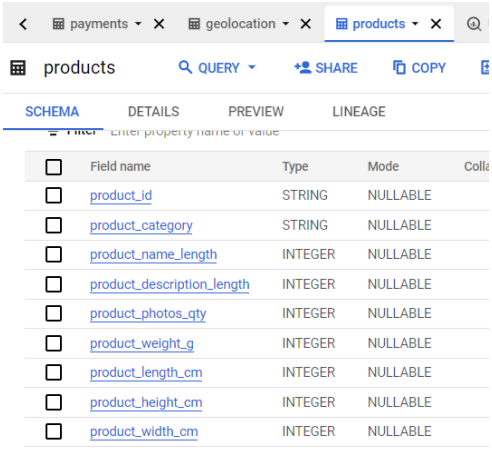
CONCLUSION:
The Data Type of the columns geolocation_zip_code_prefix is an *INTEGER*, geolocation_lat is *FLOAT*, geolocation_lng is *FLOAT*, geolocation_city is *STRING*, geolocation_state is *STRING*.

PAYMENTS Table:



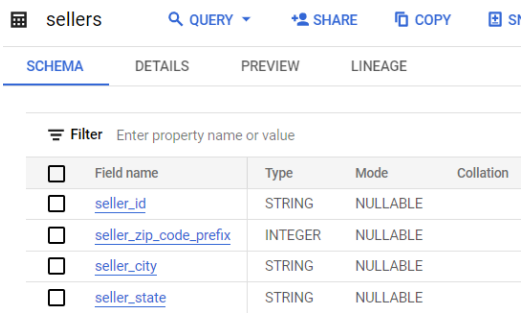
The Data Type of the columns order_id is *STRING*, payment_sequential is an *INTEGER*, payment_type is *STRING*, payment_installments is an *INTEGER*, payment_value is *FLOAT*.

PRODUCTS Table:



COMCLUSION:
The Data Types of columns product_id is *STRING*, product_category is *STRING*, product_name_length is an *INTEGER*, product_description_length is an *INTEGER*, product_photos_qty is an *INTEGER*, product_weight_g is an *INTEGER*, product_length_cm is an *INTEGER*, product_height_cm is an *INTEGER*, product_width_cm is an *INTEGER*.

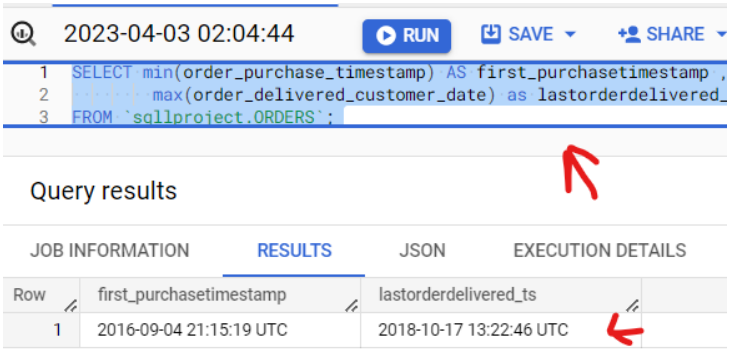
SELLERS Table:



CONCLUSION:
The data type of the column seller_id is *STRING*, seller_zip_code_prefix is *INTEGER*, seller_city is *STRING*, seller_state is *STRING*.

[b] Time period for which data is given is:

```
SELECT min(order_purchase_timestamp) AS first_purchasetimestamp ,
      max(order_delivered_customer_date) as lastorderdelivered_ts
FROM `sqllproject.ORDERS`;
```



CONCLUSION:

Time period is technically in between ***first_purchasetimestamp*** and ***lastorderdelevered_ts.***

What is meant by that is frankly speaking time period is in between first time purchase order took place and the last order that was delivered in the first case, in the second case we have taken the max and min of all the time related condition.

[c] CITIES and STATES of customers during the given period:

```
SELECT DISTINCT cust.customer_city, cust.customer_state
FROM `sqllproject.customers` cust
JOIN `sqllproject.ORDERS` o ON cust.customer_id = o.customer_id
WHERE o.order_delivered_customer_date BETWEEN order_purchase_timestamp and order_estimated_delivery_date
```

or

```
SELECT DISTINCT customer_city, customer_state
FROM `sqllproject.customers`
```

is also more than sufficient.

JOB INFORMATION | RESULTS | JSON | EXECUTION DETAILS

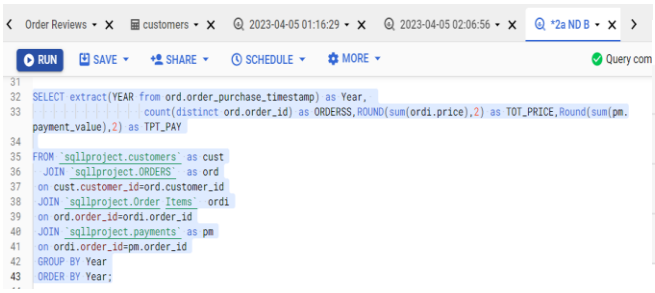
Row	customer_city	customer_state
1	acu	RN
2	acu	RN
3	acu	RN
4	ico	CE
5	ico	CE
6	ico	CE
7	ico	CE
8	ico	CE
9	ico	CE

THESE are few customer cities and states.

(2)

[a] To find the growing trend in e commerce query is given below:

```
SELECT extract(YEAR from ord.order_purchase_timestamp) as Year,  count(distinct ord.order_id) as ORDERSS,sum(ordi.price) as TOT_PRICE,sum(pm.payment_value)
as TPT_PAY
FROM
`sqllproject.ORDERS`  as ord

JOIN `sqllproject.Order Items`  ordi
on ord.order_id=ordi.order_id
JOIN `sqllproject.payments` as pm
on ordi.order_id=pm.order_id
GROUP BY Year
Order By Year
```



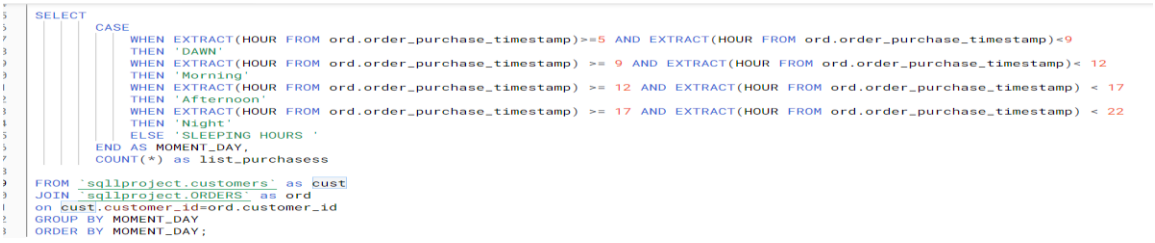
```
SELECT extract(YEAR from ord.order_purchase_timestamp) as Year,
                count(distinct ord.order_id) as ORDERSS,ROUND(sum(ordi.price),2) as TOT_PRICE,Round(sum(pm.
payment_value),2) as TPT_PAY

FROM `sqllproject.customers` as cust
 JOIN `sqllproject.ORDERS` as ord
 on cust.customer_id=ord.customer_id
 JOIN `sqllproject.Order Items` ordi
 on ord.order_id=ordi.order_id
 JOIN `sqllproject.payments` as pm
 on ordi.order_id=pm.order_id
 GROUP BY Year
 ORDER BY Year;
```

Row	Year	ORDERSS	TOT_PRICE	TPT_PAY
1	2016	311	51085.52	74281.72
2	2017	44579	6483893.86	9160941.9
3	2018	53775	7674135.96	11072911.09

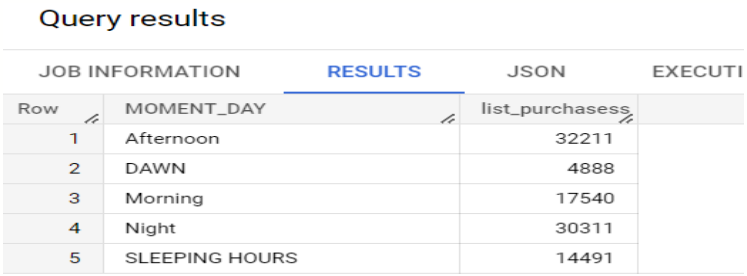
CONCLUSION: The fact that you can see the increase in distinct orders from the column orderss proves that there is a growing trend on e commerce.

[B] SELECT
CASE
WHEN EXTRACT(HOUR FROM ord.order_purchase_timestamp)>=5 AND EXTRACT(HOUR FROM ord.order_purchase_timestamp)<9
THEN 'DAWN'
WHEN EXTRACT(HOUR FROM ord.order_purchase_timestamp) >= 9 AND EXTRACT(HOUR FROM ord.order_purchase_timestamp) < 12
THEN 'Morning'
WHEN EXTRACT(HOUR FROM ord.order_purchase_timestamp) >= 12 AND EXTRACT(HOUR FROM ord.order_purchase_timestamp) < 17
THEN 'Afternoon'
WHEN EXTRACT(HOUR FROM ord.order_purchase_timestamp) >= 17 AND EXTRACT(HOUR FROM ord.order_purchase_timestamp) < 22
THEN 'Night'
ELSE 'SLEEPING HOURS '
END AS MOMENT_DAY,
COUNT(*) as list_purchasess

FROM `sqllproject.customers` as cust
JOIN `sqllproject.ORDERS` as ord
on cust.customer_id=ord.customer_id
GROUP BY MOMENT_DAY
ORDER BY MOMENT_DAY;



```
SELECT
        CASE
            WHEN EXTRACT(HOUR FROM ord.order_purchase_timestamp)>=5 AND EXTRACT(HOUR FROM ord.order_purchase_timestamp)<9
            THEN 'DAWN'
            WHEN EXTRACT(HOUR FROM ord.order_purchase_timestamp) >= 9 AND EXTRACT(HOUR FROM ord.order_purchase_timestamp)< 12
            THEN 'Morning'
            WHEN EXTRACT(HOUR FROM ord.order_purchase_timestamp) >= 12 AND EXTRACT(HOUR FROM ord.order_purchase_timestamp) < 17
            THEN 'Afternoon'
            WHEN EXTRACT(HOUR FROM ord.order_purchase_timestamp) >= 17 AND EXTRACT(HOUR FROM ord.order_purchase_timestamp) < 22
            THEN 'Night'
            ELSE 'SLEEPING HOURS '
        END AS MOMENT_DAY,
        COUNT(*) as list_purchasess

FROM `sqllproject.customers` as cust
JOIN `sqllproject.ORDERS` as ord
on cust.customer_id=ord.customer_id
GROUP BY MOMENT_DAY
ORDER BY MOMENT_DAY;
```



Query results

JOB INFORMATION | RESULTS | JSON | EXECUTI

Row	MOMENT_DAY	list_purchasess
1	Afternoon	32211
2	DAWN	4888
3	Morning	17540
4	Night	30311
5	SLEEPING HOURS	14491

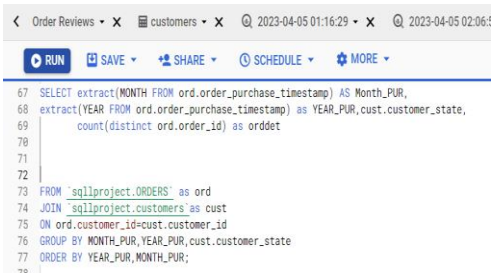
CONCLUSION: PURCHASES are maximum in the afternoon and minimum at the dawn.
RECOMMENDATION:
Product ads and launch should be in the afternoon more often because different users can come across it.

(3)

[a] MONTH ON MONTH orders by states

SELECT extract(MONTH FROM ord.order_purchase_timestamp) AS Month_PUR,
extract(YEAR FROM ord.order_purchase_timestamp) as YEAR_PUR,cust.customer_state,
count(distinct ord.order_id) as orddet

```
FROM `sqllproject.ORDERS` as ord
JOIN `sqllproject.customers`as cust
ON ord.customer_id=cust.customer_id
GROUP BY MONTH_PUR,YEAR_PUR,cust.customer_state
ORDER BY YEAR_PUR,MONTH_PUR;
```



Query results

JOB INFORMATION | RESULTS | JSON | EXECUTION DETAILS | EXECUTION

Row	Month_PUR	YEAR_PUR	customer_state	orddet
1	9	2016	RR	1
2	9	2016	RS	1
3	9	2016	SP	2
4	10	2016	SP	113
5	10	2016	RS	24
6	10	2016	RJ	56
7	10	2016	MT	3
8	10	2016	GO	9
9	10	2016	MG	40

Results per page:

CONCLUSION: TABLE SHOWS **MONTH ON MONTH** ORDERS by STATES.

[b]

DISTRIBUTION OF CUSTOMERS ACROSS THE STATES IS :

```
SELECT count(customer_id),customer_state
FROM `sqllproject.customers`
GROUP BY customer_state
```

```
SELECT count(customer_id),customer_state
 FROM `sqllproject.customers`
 GROUP BY customer_state
```

Query results

SAVE R

JOB INFORMATION | RESULTS | JSON | EXECUTION DETAILS | EXECUTION GRAF

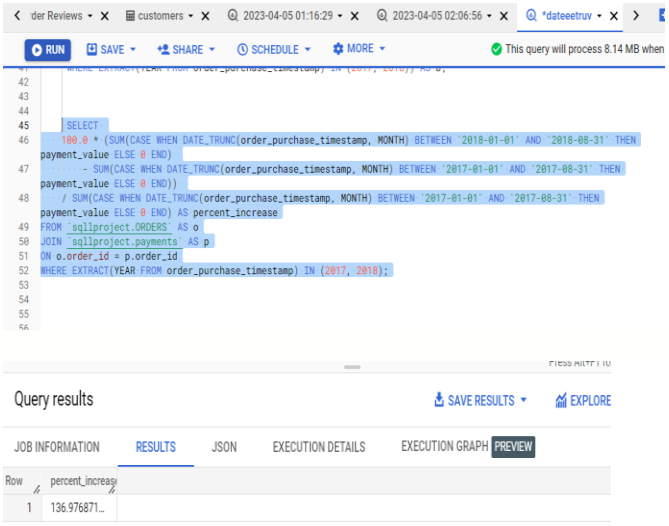
Row	f0_	customer_state
1	485	RN
2	1336	CE
3	5466	RS
4	3637	SC
5	41746	SP
6	11635	MG
7	3380	BA
8	12852	RJ

Results per page: 50

CONCLUSION:
THESE ARE THE LIST OF CUSTOMERS SPREAD ACROSS THE STATE.
SP has the highest distribution

4[a]

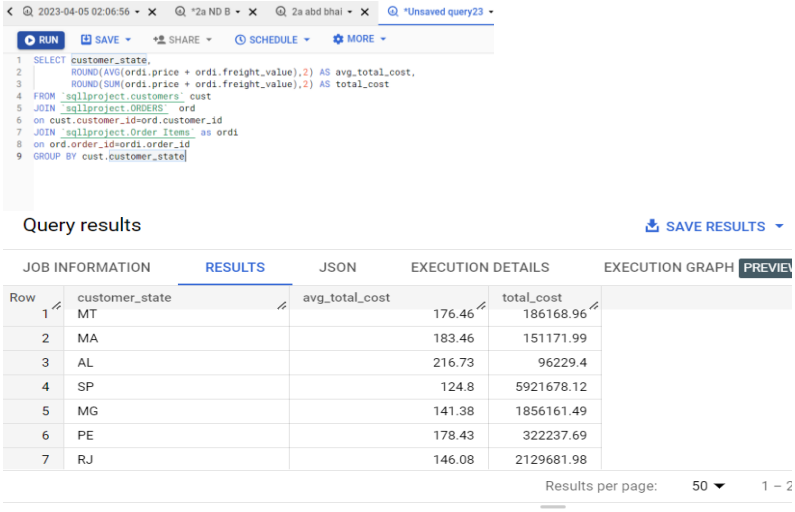
```
SELECT
  100.0 * (SUM(CASE WHEN DATE_TRUNC(order_purchase_timestamp, MONTH) BETWEEN '2018-01-01' AND '2018-08-31' THEN payment_value ELSE 0 END)
    - SUM(CASE WHEN DATE_TRUNC(order_purchase_timestamp, MONTH) BETWEEN '2017-01-01' AND '2017-08-31' THEN payment_value ELSE 0 END))
  / SUM(CASE WHEN DATE_TRUNC(order_purchase_timestamp, MONTH) BETWEEN '2017-01-01' AND '2017-08-
31' THEN payment_value ELSE 0 END) AS percent_increase
FROM `sqllproject.ORDERS` AS o
JOIN `sqllproject.payments` AS p
ON o.order_id = p.order_id
WHERE EXTRACT(YEAR FROM order_purchase_timestamp) IN (2017, 2018);
```



CONCLUSION: The percentage increase is 136.9.

[b]

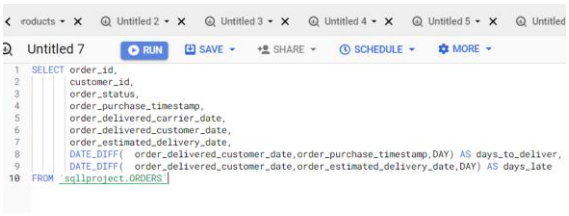
```
SELECT customer_state,
     ROUND(AVG(ordi.price + ordi.freight_value),2) AS avg_total_cost,
     ROUND(SUM(ordi.price + ordi.freight_value),2) AS total_cost
FROM `sqllproject.customers` cust
JOIN `sqllproject.ORDERS`  ord
on cust.customer_id=ord.customer_id
JOIN `sqllproject.Order Items` as ordi
on ord.order_id=ordi.order_id
GROUP BY cust.customer_state
```

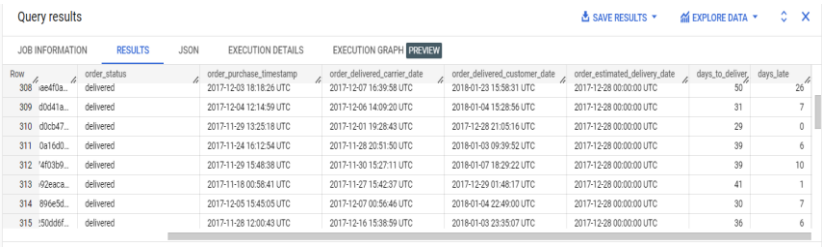


CONCLUSION: mean =avg(cost+freight),sum=sum(price+freight).
FOR each state it will be shown.

5 (1) SELECT order_id,

customer_id,
order_status,
order_purchase_timestamp,
order_delivered_carrier_date,
order_delivered_customer_date,
order_estimated_delivery_date,
DATE_DIFF(order_delivered_customer_date,order_purchase_timestamp,DAY) AS days_to_deliver,
DATE_DIFF(order_delivered_customer_date,order_estimated_delivery_date,DAY) AS days_late
FROM `sqllproject.ORDERS

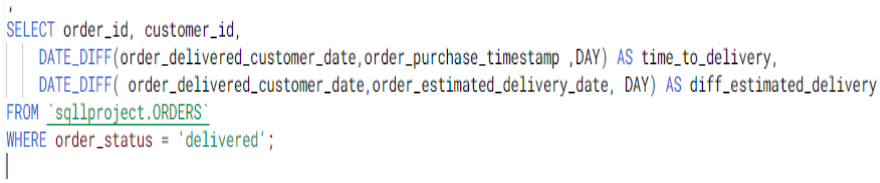




CONCLUSION :
days_to_deliver shows the time between purchases date and estimated delivery date.
days_late shows whether the ordered was delivered within the estimated delivery date.

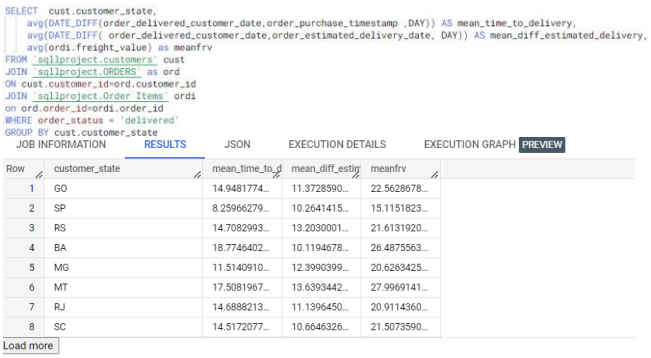
5[2]

```
SELECT order_id, customer_id,
   DATE_DIFF(order_delivered_customer_date,order_purchase_timestamp ,DAY) AS time_to_delivery,
   DATE_DIFF( order_delivered_customer_date,order_estimated_delivery_date, DAY) AS diff_estimated_delivery
FROM `sqllproject.ORDERS`
WHERE order_status = 'delivered';
```



CONCLUSION:
IF diff_estimated_delivery is in negative then order arrived before the estimated time.

5[3]
```
SELECT  cust.customer_state,
   avg(DATE_DIFF(order_delivered_customer_date,order_purchase_timestamp ,DAY)) AS mean_time_to_      delivery,
   avg(DATE_DIFF( order_delivered_customer_date,order_estimated_delivery_date, DAY)) AS mean_diff_estimated_delivery,
   avg(ordi.freight_value) as meanfrv
   FROM `sqllproject.customers` cust
   JOIN `sqllproject.ORDERS` as ord
   ON cust.customer_id=ord.customer_id
   JOIN `sqllproject.Order Items` ordi
   on ord.order_id=ordi.order_id
   WHERE order_status = 'delivered'
   GROUP BY cust.customer_state
```

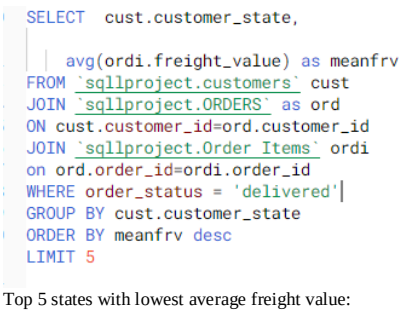


Row	customer_state	mean_time_to_d	mean_diff_estim	meanfrv
1	GO	14.9481774...	11.3728590...	22.5628678...
2	SP	8.25966279...	10.2641415...	15.1151823...
3	RS	14.7082993...	13.2030001...	21.6131920...
4	BA	18.7746402...	10.1194678...	26.4875563...
5	MG	11.5140910...	12.3990399...	20.6263425...
6	MT	17.5081967...	13.6393442...	27.9969141...
7	RJ	14.6888213...	11.1396450...	20.9114360...
8	SC	14.5172077...	10.6646326...	21.5073590...

Load more

5[5] Top 5 states with highest average freight value :

```
SELECT  cust.customer_state,

   avg(ordi.freight_value) as meanfrv
FROM `sqllproject.customers` cust
JOIN `sqllproject.ORDERS` as ord
ON cust.customer_id=ord.customer_id
JOIN `sqllproject.Order Items` ordi
on ord.order_id=ordi.order_id
WHERE order_status = 'delivered'
GROUP BY cust.customer_state
ORDER BY meanfrv desc
LIMIT 5
```



```
SELECT  cust.customer_state,

   avg(ordi.freight_value) as meanfrv
FROM `sqllproject.customers` cust
JOIN `sqllproject.ORDERS` as ord
ON cust.customer_id=ord.customer_id
JOIN `sqllproject.Order Items` ordi
on ord.order_id=ordi.order_id
WHERE order_status = 'delivered'
GROUP BY cust.customer_state
ORDER BY meanfrv desc
LIMIT 5
```

Row	customer_state	meanfrv
1	PB	43.0916894...
2	RR	43.0880434...
3	RO	41.3305494...
4	AC	40.0479120...
5	PI	39.1150860...

Top 5 states with lowest average freight value:

```
SELECT  cust.customer_state,

   avg(ordi.freight_value) as meanfrv
FROM `sqllproject.customers` cust
JOIN `sqllproject.ORDERS` as ord
ON cust.customer_id=ord.customer_id
JOIN `sqllproject.Order Items` ordi
on ord.order_id=ordi.order_id
WHERE order_status = 'delivered'
GROUP BY cust.customer_state
ORDER BY meanfrv
LIMIT 5
```

Query results

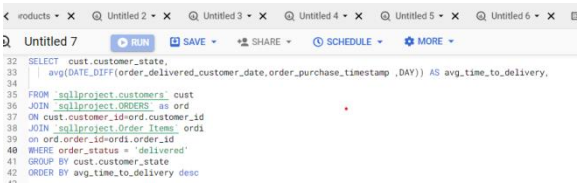
JOB INFORMATION | RESULTS | JSON | EXECUTI

Row	customer_state	meanfrv
1	SP	15.1151823...
2	PR	20.4718162...
3	MG	20.6263425...
4	RJ	20.9114360...
5	DF	21.0721613...

RECOMMENDATIONS: IF the freight value is to high, you might want to consider warehouses in different places to make it easy.

5[6]HIGHEST AVERAGE TIME TO DELIVERY:

```
SELECT  cust.customer_state,
   avg(DATE_DIFF(order_delivered_customer_date,order_purchase_timestamp ,DAY)) AS avg_time_to_delivery,

FROM `sqllproject.customers` cust
JOIN `sqllproject.ORDERS` as ord
ON cust.customer_id=ord.customer_id
JOIN `sqllproject.Order Items` ordi
on ord.order_id=ordi.order_id
WHERE order_status = 'delivered'
GROUP BY cust.customer_state
ORDER BY avg_time_to_delivery desc
Limit 5
```



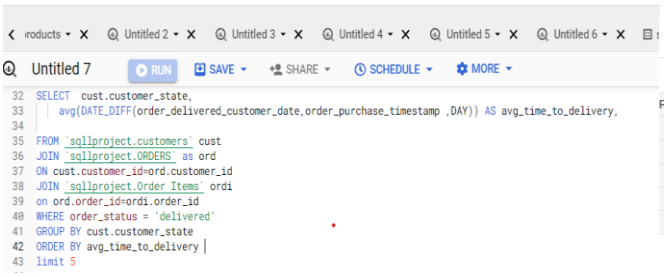
Query results

JOB INFORMATION | RESULTS | JSON | EXECUTION D

Row	customer_state	avg_time_to_deli
1	RR	27.8260869...
2	AP	27.7530864...
3	AM	25.9631901...
4	AL	23.9929742...
5	PA	23.3017077...

LOWEST AVERAGE TIME TO DELIVERY:

```
SELECT  cust.customer_state,
   avg(DATE_DIFF(order_delivered_customer_date,order_purchase_timestamp ,DAY)) AS avg_time_to_delivery,

FROM `sqllproject.customers` cust
JOIN `sqllproject.ORDERS` as ord
ON cust.customer_id=ord.customer_id
JOIN `sqllproject.Order Items` ordi
on ord.order_id=ordi.order_id
WHERE order_status = 'delivered'
GROUP BY cust.customer_state
ORDER BY avg_time_to_delivery
limit 5
```



Query results

JOB INFORMATION | RESULTS | JSON | EXECUTION DETAILS

Row	customer_state	avg_time_to_deli
1	SP	8.25966279...
2	PR	11.4807930...
3	MG	11.5140910...
4	DF	12.5014861...
5	SC	14.5172077...

RECOMMENDATION:
IF the delivery is slow:
Communication with customers: It is important for companies to communicate with customers about delays or delivery issues. They must provide new information about thei r orders and be clear about any challenges they will face.

Improve logistics: Companies can improve delivery by improving delivery processes, partnering with trusted carriers, and using tracking systems to ensure products are deliv ered on time.

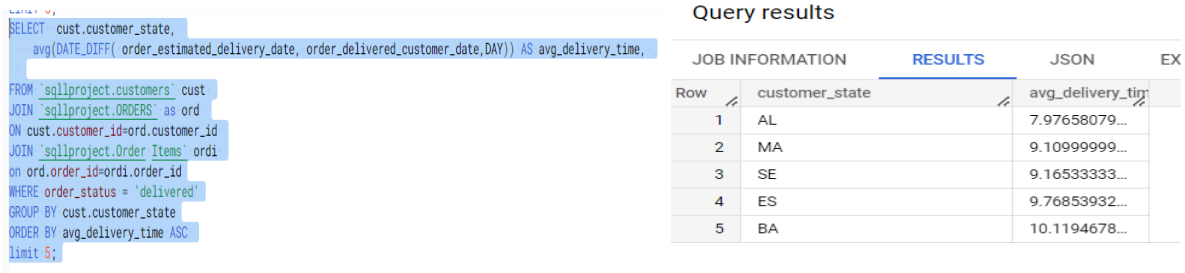
Offer Alternative Shipping Options: If businesses experience delays while using the shipping process, they may consider offering customers alternative shipping options such as expedited shipping or same-day shipping.

Service or Return: If the customer experiences a delay or problem with delivery, the company may wish to offer a refund or refund to resolve the issue.

This helps increase customer satisfaction and loyalty.

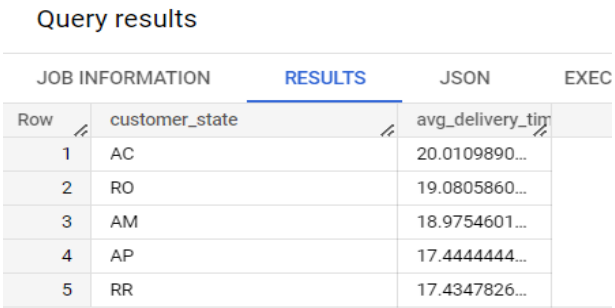
5[7]
Top 5 states where delivery is really fast compared to estimated date:

```
SELECT  cust.customer_state,
    avg(DATE_DIFF( order_estimated_delivery_date, order_delivered_customer_date,DAY)) AS avg_delivery_time,

FROM `sqllproject.customers` cust
JOIN `sqllproject.ORDERS` as ord
ON cust.customer_id=ord.customer_id
JOIN `sqllproject.Order Items` ordi
on ord.order_id=ordi.order_id
WHERE order_status = 'delivered'
GROUP BY cust.customer_state
ORDER BY avg_delivery_time ASC
limit 5;
```



NOT SO FAST:

```
SELECT  cust.customer_state,
    avg(DATE_DIFF( order_estimated_delivery_date, order_delivered_customer_date,DAY)) AS avg_delivery_time,

FROM `sqllproject.customers` cust
JOIN `sqllproject.ORDERS` as ord
ON cust.customer_id=ord.customer_id
JOIN `sqllproject.Order Items` ordi
on ord.order_id=ordi.order_id
WHERE order_status = 'delivered'
GROUP BY cust.customer_state
ORDER BY avg_delivery_time DESC
limit 5;
```



(6)
[a] MONTH OVER MONTH :

```
SELECT *,
      order_count-LAG(order_count,1,0) OVER (PARTITION BY payment_type,year Order By month) as monthomonth
      FROM
(SELECT EXTRACT(YEAR from order_purchase_timestamp) as YEAR,
  EXTRACT(MONTH FROM order_purchase_timestamp) AS month,
  p.payment_type,
  COUNT(DISTINCT o.order_id) AS order_count
FROM
  `sqllproject.ORDERS` o
  JOIN `sqllproject.payments` p ON o.order_id = p.order_id
GROUP BY
  1,2,3) as lol
ORDER BY
  1, 2,3
```

Query results

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

Row	YEAR	month	payment_type	order_count	monthomonth
3	2016	10	credit_card	253	250
4	2016	10	debit_card	2	2
5	2016	10	voucher	11	11
6	2016	12	credit_card	1	-252
7	2017	1	UPI	197	197
8	2017	1	credit_card	582	582
9	2017	1	debit_card	9	9
10	2017	1	voucher	33	33
11	2017	2	UPI	398	201
12	2017	2	credit_card	1347	765

CONCLUSION:
THE month over month is shown by monthomonth with respect to different years.

6[b] COUNT of ORDERS based on no. of payment installments:

```
SELECT
  pay.payment_installments,
  COUNT(DISTINCT ordi.order_id) AS OrderCount
FROM
  `sqllproject.ORDERS` ordi
  JOIN `sqllproject.payments` pay ON ordi.order_id = pay.order_id
GROUP BY
  pay.payment_installments
ORDER BY
  pay.payment_installments
```

5 RUN SAVE SHARE SCHEDULE

```
  1, 2
;
SELECT
  pay.payment_installments,
  COUNT(DISTINCT ordi.order_id) AS OrderCount
FROM
  `sqllproject.ORDERS` ordi
  JOIN `sqllproject.payments` pay ON ordi.order_id = pay.order_id
GROUP BY
  pay.payment_installments
ORDER BY
  pay.payment_installments
```

Query results

JOB INFORMATION | RESULTS | JSON | E

Row	payment_installr	OrderCount
2	1	49060
3	2	12389
4	3	10443
5	4	7088
6	5	5234
7	6	3916
8	7	1623
9	8	4253

CONCLUSION: BASED on each payment installments the count of orders has been shown above.

(7) The actionable insights are shown in the conclusion of each questions.

(8)RECOMMENDATION GIVEN AFTER FEW QUESTIONS. However here are my overall analysis:
Based on the analysis of the e-commerce dataset for Brazil, there are several recommendations that can be made to improve the company's operations:

Improve the logistics. The analysis reveals that there are some issues with delivery times in certain regions, so the company could consider optimizing its shipping processes or partnering with reliable shipping carriers to improve its delivery times.
Focus on marketing during peak months. The analysis suggests that there are some seasonality trends in e-commerce sales in Brazil, with peaks during specific months. The company could focus on marketing during these peak months to capitalize on the increased demand.
Improve estimated delivery times. The analysis reveals that there are some issues with estimated delivery times not being met, which could lead to dissatisfied customers. By improving the accuracy of its estimated delivery times, the company could improve customer satisfaction and loyalty.