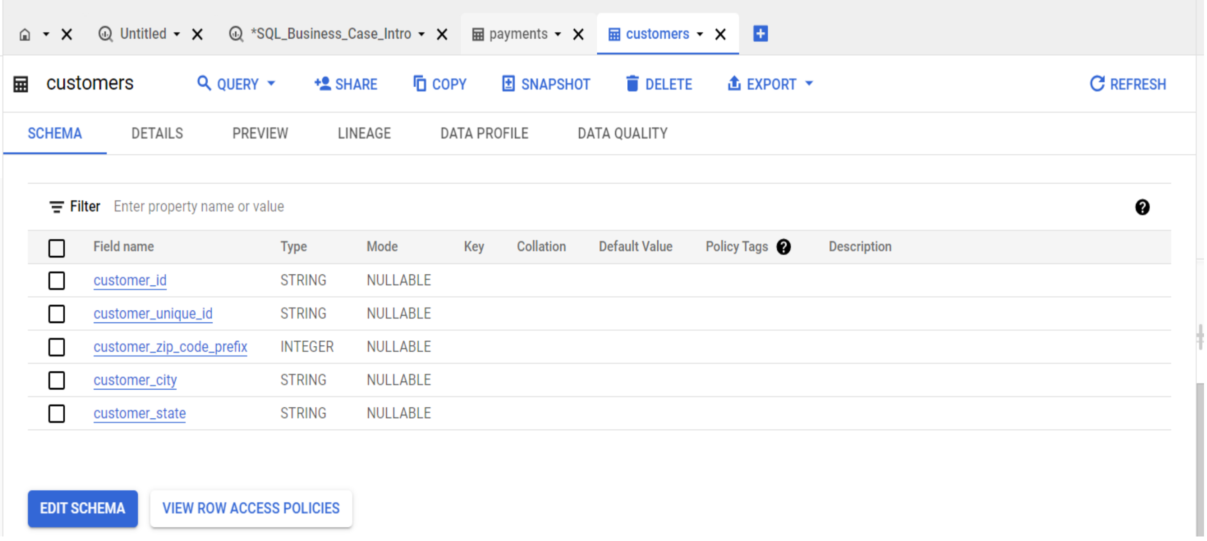
1.1: Importing the dataset

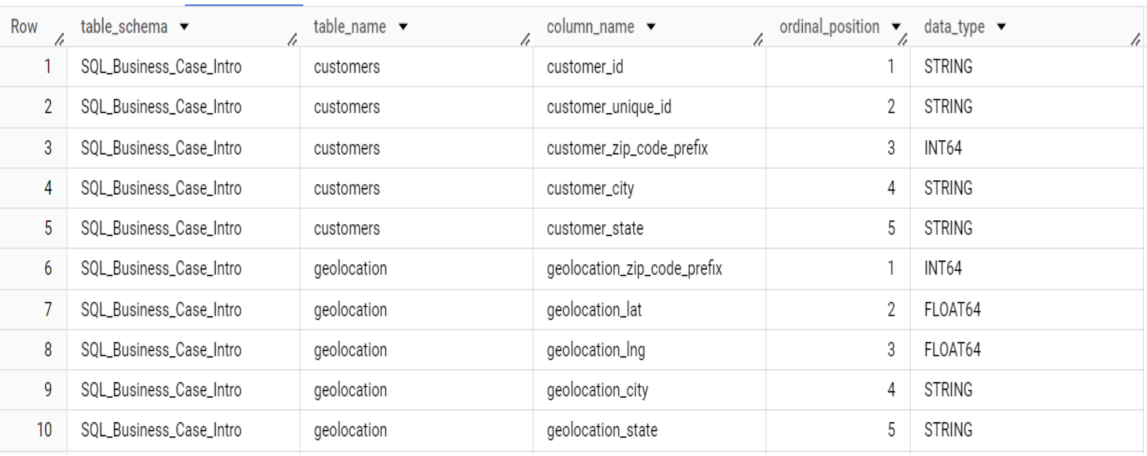
- A project "ayushsql2023" was created.
- A dataset "ayushsql2023.SQL_Business_Case_Intro" was created under the project.
- The csv files were downloaded from the **No index entries found.**location given and uploaded in the dataset using SQL_Business_Case_Intro -- Create Table -- Create Table from "Upload" and auto-detect schema was checked.
- For getting the datatypes of all the columns of customer tables, there are two steps:
  1.1.1. First is by clicking on table customers and selecting the schema tab.:



☐	Field name	Type	Mode	Key	Collation	Default Value	Policy Tags	Description
☐	customer_id	STRING	NULLABLE					
☐	customer_unique_id	STRING	NULLABLE					
☐	customer_zip_code_prefix	INTEGER	NULLABLE					
☐	customer_city	STRING	NULLABLE					
☐	customer_state	STRING	NULLABLE					

1.1.2. Second is by running the below query:

```
select table_schema, table_name, column_name,ordinal_position, data_type
from ayushsql2023.SQL_Business_Case_Intro.INFORMATION_SCHEMA.COLUMNS
order by table_name, ordinal_position
```

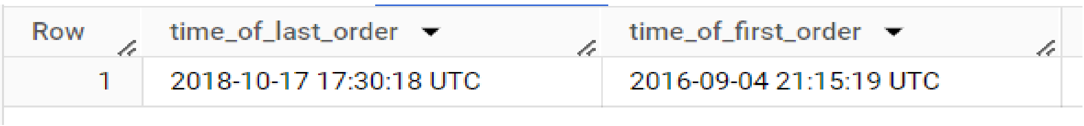


Row	table_schema	table_name	column_name	ordinal_position	data_type
1	SQL_Business_Case_Intro	customers	customer_id	1	STRING
2	SQL_Business_Case_Intro	customers	customer_unique_id	2	STRING
3	SQL_Business_Case_Intro	customers	customer_zip_code_prefix	3	INT64
4	SQL_Business_Case_Intro	customers	customer_city	4	STRING
5	SQL_Business_Case_Intro	customers	customer_state	5	STRING
6	SQL_Business_Case_Intro	geolocation	geolocation_zip_code_prefix	1	INT64
7	SQL_Business_Case_Intro	geolocation	geolocation_lat	2	FLOAT64
8	SQL_Business_Case_Intro	geolocation	geolocation_lng	3	FLOAT64
9	SQL_Business_Case_Intro	geolocation	geolocation_city	4	STRING
10	SQL_Business_Case_Intro	geolocation	geolocation_state	5	STRING

1.2: Getting the time range between which the orders were placed:

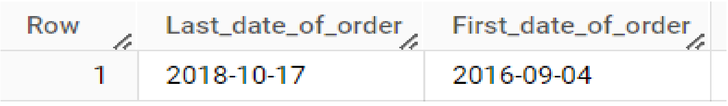
- Following query was run for getting the range of dates during which order was placed:

```
select max(order_purchase_timestamp) time_of_last_order ,
min(order_purchase_timestamp) time_of_first_order
from `SQL_Business_Case_Intro.orders`
```



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

```
select max(date(order_purchase_timestamp)) Last_date_of_order ,
min(date(order_purchase_timestamp)) First_date_of_order
from `SQL_Business_Case_Intro.orders`
```



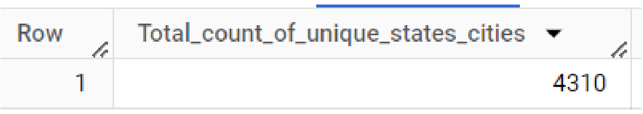
Row	Last_date_of_order	First_date_of_order
1	2018-10-17	2016-09-04

1.3: Getting the count of Cities & States of Customers who ordered during the given period:

- Following query was run for getting the count of unique Cities and States of Customers who ordered from orders, customers and geo-locations table:

  Method of join:

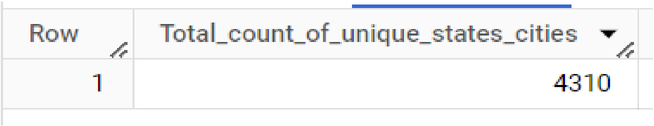
```
select count(*) Total_count_of_unique_states_cities
from (select distinct b.customer_city, b.customer_state
from `SQL_Business_Case_Intro.orders` a
left join `SQL_Business_Case_Intro.customers` b
on a.customer_id = b.customer_id) c ;
```



Row	Total_count_of_unique_states_cities
1	4310

  Method of subquery:

```
select count(*) Total_count_of_unique_states_cities
from (select distinct customer_city, customer_state
from `SQL_Business_Case_Intro.customers`
where customer_id in (select customer_id from `SQL_Business_Case_Intro.orders`));
```



Row	Total_count_of_unique_states_cities
1	4310

2.1: Analyzing the trend in the no. of orders placed over the past years:

- Following query was run for analyzing the trend from orders column:

  ```
  select  Year, count(distinct order_id) Ord_count
  from (select *, extract (year from order_purchase_timestamp ) Year from
  `SQL_Business_Case_Intro.orders`) b
  group by Year
  order by Year
  ```

  The following result was obtained:

Row	**Year**	**Ord_count**
1	2016	329
2	2017	45101
3	2018	54011

Observation: The number of orders has increased from 2017 to 2018. We cannot confirm for the year 2016 as the dataset is available only from 4 September onwards. Hence majority of yearly data for 2016 is not available.

2.2: Analyzing the monthly seasonality in the no. of orders being placed:

- Following query was run for analyzing the monthly seasonality in terms of no. of Orders being placed:

  ```
  With CTE as
  (select *, extract (month from order_purchase_timestamp) Month
  from `SQL_Business_Case_Intro.orders`)
  select Month, count(distinct order_id) Order_count
  from CTE
  group by Month
  order by Month
  ```

  The following output was generated:

Row	**Month**	**Order_count**
1	1	8069
2	2	8508
3	3	9893
4	4	9343
5	5	10573
6	6	9412
7	7	10318

Row	**Month**	**Order_count**
8	8	10843
9	9	4305
10	10	4959
11	11	7544
12	12	5674

Observation: As seen from the table above, the count of orders is high and is generally increasing in Months from January till August. There is a significant drop in orders in Months from September till December.

2.3: Analyzing the time when Brazilian Customers place their orders:

- Following query was run:

```
with CTE2 as
(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
'Mornings'
when extract(hour from order_purchase_timestamp) between 13 and 18 Then
'Afternoon'
Else 'Night'
end Time_of_day
from `SQL_Business_Case_Intro.orders`
)
select Time_of_day, count(distinct order_id) Ord_count
from CTE2
group by Time_of_day
order by Ord_Count;
```

The following output was received:

Row	**Time_of_day**	**Ord_count**
1	Dawn	5242
2	Mornings	27733
3	Night	28331
4	Afternoon	38135

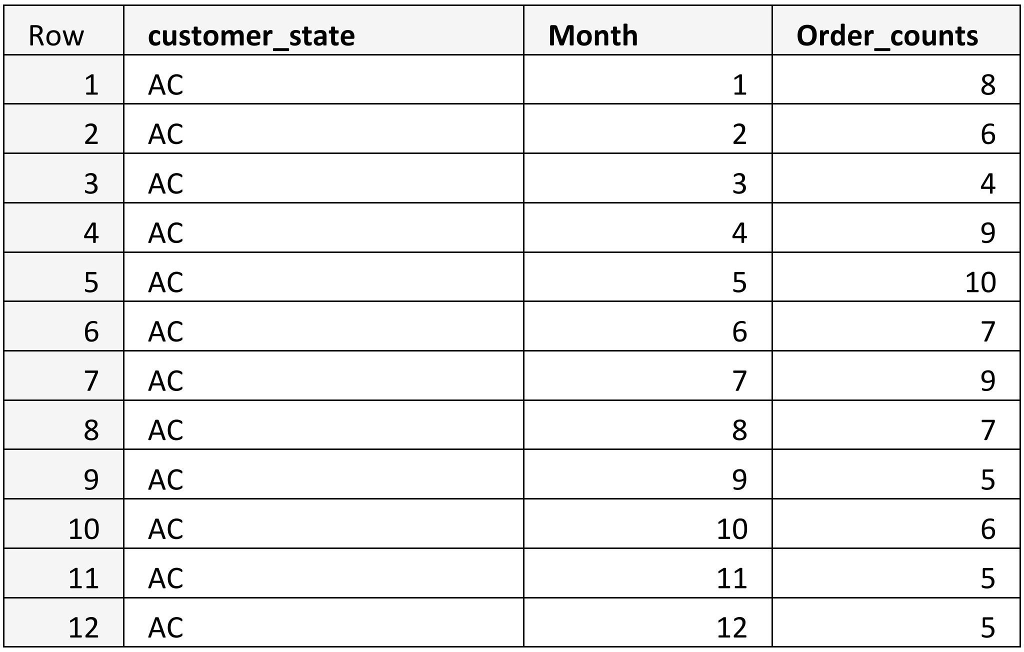
Observation: As seen from the table, the Brazilian place the orders mostly in afternoon and less likely during Dawn. The count of orders increases from Dawn till Afternoon and then drops at Night.

3.1: Getting the month on month no. of orders placed in each state:

- Following query was run:

```
with CTE3 as
(select a.*, b.* except(customer_id)
from
(select * , extract (month from order_purchase_timestamp) Month
from `SQL_Business_Case_Intro.orders`) a
left join `SQL_Business_Case_Intro.customers` b
on a.customer_id = b.customer_id)
select customer_state, Month, count(distinct order_id) Order_counts
from CTE3
group by customer_state, Month
order by customer_state, Month;
```

Following result (**Note: only results for Customer_state "AC" shown in the image below. The total row count in the table was 322)** was observed:



Row	customer_state	Month	Order_counts
1	AC	1	8
2	AC	2	6
3	AC	3	4
4	AC	4	9
5	AC	5	10
6	AC	6	7
7	AC	7	9
8	AC	8	7
9	AC	9	5
10	AC	10	6
11	AC	11	5
12	AC	12	5

3.2: Understanding the Distribution of Customers across all States:

- Following query was run:

```
with CTE4 as
(select a.*, b.* except(customer_id)
from
`SQL_Business_Case_Intro.orders` a
left join `SQL_Business_Case_Intro.customers` b
on a.customer_id = b.customer_id)
select customer_state, round(count(distinct customer_id)*100/(select count(*) from
CTE4),2) customer_count_percent
from CTE4
group by customer_state
order by customer_count_percent desc, customer_state
```

The first 10 cities with highest percentage of Customers is shown below (**Note: The above query gave a percentage of customers from all the 27 districts. Only 10 shown in the below image)**:

Row	**customer_state**	**customer_count_percent**
1	SP	41.98
2	RJ	12.92
3	MG	11.7
4	RS	5.5
5	PR	5.07
6	SC	3.66
7	BA	3.4
8	DF	2.15
9	ES	2.04
10	GO	2.03

4.1: Get the % increase in the cost of orders from year 2017 to 2018 (include months between Jan to Aug only).

- Following query was run for calculating increase in cost of orders between Jan to Aug from the years 2017 to 2018:

```
with CTE6 as
(select a.*, b.* except (order_id), extract(year from order_purchase_timestamp)
Year
from `SQL_Business_Case_Intro.orders` a
left join `SQL_Business_Case_Intro.payments` b
using(order_id)
where extract(month from order_purchase_timestamp) between 1 and 8
order by extract(month from order_purchase_timestamp) asc)
select round(((select sum(payment_value)
from CTE6
where Year = 2018) -
(select sum(payment_value)
from CTE6
where Year = 2017))*100/(select sum(payment_value)
from CTE6
where Year = 2017),2) Percent_Increase_in_sales;
```

Following output was generated:

Row	Percent_Increase_in_sales
1	136.98

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

- Following query was run

  Note: The below query will not give the correct Avg_order_price for each state, because in CTE7, the same order_id will have multiple lines in payments table. So, when average is taken while grouping by customer_state, each of the lines will be counted separately resulting in lower Avg_order_price than actual.

```
with CTE7 as
(select a.*, b.* except(order_id), c.* except(customer_id)
from `SQL_Business_Case_Intro.orders` a
left join `SQL_Business_Case_Intro.payments` b
using(order_id)
left join `SQL_Business_Case_Intro.customers` c
using(customer_id))
select customer_state, round(sum(payment_value),2) Total_order_price,
round(avg(payment_value),2) Avg_order_price
from CTE7
group by customer_state
order by Total_order_price desc, Avg_order_price desc;
```

Following output (only 10 results shown below) was received, ordered by Total Price in desc order and average price in desc order for each state for above query:

Row	customer_state	Total_order_price	Avg_order_price
1	SP	5998226.96	137.5
2	RJ	2144379.69	158.53
3	MG	1872257.26	154.71
4	RS	890898.54	157.18
5	PR	811156.38	154.15
6	SC	623086.43	165.98
7	BA	616645.82	170.82
8	DF	355141.08	161.13
9	GO	350092.31	165.76
10	ES	325967.55	154.71

For getting the actual Total_order_price and Avg_order_price for each state, Following query is used:

```
with CTE8 as
(select a.order_id,a.customer_id, b.payment_value, c.customer_state,
sum(payment_value) over(partition by customer_state) Total_state_order_price,
sum(payment_value) over(partition by order_id) Ind_order_price
from `SQL_Business_Case_Intro.orders` a
left join `SQL_Business_Case_Intro.payments` b
using(order_id)
left join `SQL_Business_Case_Intro.customers` c
using(customer_id)),
CTE9 as
(select distinct customer_state,Total_state_order_price, Ind_order_price
from CTE8 )
select customer_state, avg(Total_state_order_price) Total_order_price,
round(avg(Ind_order_price),2) Average_order_price
from CTE9
group by customer_state
order by Total_order_price desc, Average_order_price desc;
```

In the above query, Windows function is used for getting sum of payment_value over customer state (Total_state_order_price) and sum of payment_value over order_id (Ind_order_price) in CTE8. After that distinct customer_state, Ind_order_price and Total_state_order_price are extracted from CTE8 in CTE9. The average of both these values grouped by customer_state gives the correct result. Following result (top 10 shown below) is obtained:

Row	**customer_state**	**Total_order_price**	**Average_order_price**
1	SP	5998226.96	215.97
2	RJ	2144379.69	213.58
3	MG	1872257.26	199.52
4	RS	890898.54	194.54
5	PR	811156.38	189.52
6	SC	623086.43	197.6
7	BA	616645.82	209.98
8	DF	355141.08	184.02
9	GO	350092.31	189.86
10	ES	325967.55	177.84

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

- Following query was run for extracting Total Freight and Average Freight Price for each customer_state:

```
with CTE10 as
(select c.customer_state, a.order_id,sum(b.freight_value) over(partition by
c.customer_state) Total_freight_price,
sum(b.freight_value) over(partition by a.order_id) Ind_freight_price
from `SQL_Business_Case_Intro.orders` a
left join `SQL_Business_Case_Intro.order_items`b
using(order_id)
left join `SQL_Business_Case_Intro.customers` c
using(customer_id)),
CTE11 as
(select distinct *
from CTE10
order by customer_state)
select customer_state, round(avg(Total_freight_price),2) Total_freight_price,
round(avg(Ind_freight_price),2) Avg_Order_freight_price
from CTE11
group by customer_state
order by Total_freight_price desc, Avg_Order_freight_price desc ;
```

Following result ordered by Total_freight_price in descending order and Avg_Order_freight_price in descending order for each state (only top 10 shown) was obtained.:

Row	customer_state	Total_freight_price	Avg_Order_freight_price
1	SP	718723.07	17.37
2	RJ	305589.31	23.95
3	MG	270853.46	23.46
4	RS	135522.74	24.95
5	PR	117851.68	23.58
6	BA	100156.68	29.83
7	SC	89660.26	24.82
8	PE	59449.66	36.07
9	GO	53114.98	26.46
10	DF	50625.5	23.82

5.1: Calculate the delivery time and the difference in days between the estimated and actual delivery date of an order:

- Following query was run for calculating delivery time and difference between the estimated and actual delivery date of an order:

```
select order_id,
date_diff(order_delivered_customer_date,order_purchase_timestamp, day)
time_to_deliver,
date_diff(order_estimated_delivery_date, order_delivered_customer_date, day)
diff_estimated_delivery
from `SQL_Business_Case_Intro.orders`;
```

Following unsorted result (top 10) was obtained:

Row	**order_id**	**time_to_deliver**	**diff_estimated_delivery**
1	1950d777989f6a877539f53795b4c3c3	30	-12
2	2c45c33d2f9cb8ff8b1c86cc28c11c30	30	28
3	65d1e226dfaeb8cdc42f665422522d14	35	16
4	635c894d068ac37e6e03dc54eccb6189	30	1
5	3b97562c3aee8bdedcb5c2e45a50d5e1	32	0
6	68f47f50f04c4cb6774570cfde3a9aa7	29	1
7	276e9ec344d3bf029ff83a161c6b3ce9	43	-4
8	54e1a3c2b97fb0809da548a59f64c813	40	-4
9	fd04fa4105ee8045f6a0139ca5b49f27	37	-1
10	302bb8109d097a9fc6e9cefc5917d1f3	33	-5

5.2: Calculate the Total & Average value of order freight for each state.

- Following query was run for finding out the top 5 states with the highest & lowest average freight value (Granularity level: Customer_state, Order_id, Product_id for calculating average):

```
with CTE12 as
(select c.customer_state, a.order_id, b.freight_value
from `SQL_Business_Case_Intro.orders` a
left join `SQL_Business_Case_Intro.order_items` b
using(order_id)
left join `SQL_Business_Case_Intro.customers` c
using(customer_id))
(select customer_state, round(avg(freight_value),2) Avg_freight_value
from CTE12
group by customer_state
order by Avg_freight_value desc
limit 5)
union all
(select customer_state, round(avg(freight_value),2) Avg_freight_value
from CTE12
group by customer_state
order by Avg_freight_value asc
limit 5)
```

Following result was obtained (the first 5 rows represent highest 5 and the last 5 rows represent states with lowest average freight values):

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

- Following query was run for finding out the top 5 states with the highest & lowest average freight value (Granularity level: Customer_state, Order_id for calculating average):

```
with CTE13 as
(select c.customer_state, a.order_id, b.freight_value, round(sum(b.freight_value)
over(partition by a.order_id), 2) Tot_freight_value
from `SQL_Business_Case_Intro.orders` a
left join `SQL_Business_Case_Intro.order_items` b
using(order_id)
left join `SQL_Business_Case_Intro.customers` c
using(customer_id))
(select customer_state, round(avg(Tot_freight_value),2) Avg_freight_value
from CTE13
group by customer_state
order by Avg_freight_value desc
limit 5)
union all
(select customer_state, round(avg(Tot_freight_value),2) Avg_freight_value
from CTE13
group by customer_state
order by Avg_freight_value
limit 5);
```

Following result was obtained (the first 5 rows represent highest 5 and the last 5 rows represent states with lowest average freight values):

Row	**customer_state**	**Avg_freight_value**
1	PB	68.04
2	AC	54.45
3	RR	54.19
4	RO	50.95
5	MA	50.72
6	SP	20.91
7	DF	27.36
8	MG	27.53
9	ES	27.77
10	RJ	28.45

5.3: Calculate the Total & Average value of order freight for each state.

- Following query was run for calculating delivery time and difference between the estimated and actual delivery date of an order:

```
with CTE14 as
(select b.customer_state,a.order_id,date_diff(a.order_delivered_customer_date,
a.order_purchase_timestamp, day) day_diff
from `SQL_Business_Case_Intro.orders` a
left join `SQL_Business_Case_Intro.customers` b
using(customer_id)
)
(select customer_state, round(avg(day_diff),2) Avg_delivery_time
from CTE14
group by customer_state
order by Avg_delivery_time desc
limit 5)
union all
(select customer_state, round(avg(day_diff),2) Avg_delivery_time
from CTE14
group by customer_state
order by Avg_delivery_time
limit 5);
```

Following result (only top 10 shown below) was obtained:

Row	**customer_state**	**Avg_delivery_time**
1	RR	28.98
2	AP	26.73
3	AM	25.99

Row	**customer_state**	**Avg_delivery_time**
4	AL	24.04
5	PA	23.32
6	SP	8.3
7	PR	11.53
8	MG	11.54
9	DF	12.51
10	SC	14.48

5.4: Calculate the Total & Average value of order freight for each state.

- Following query was run for calculating delivery time and difference between the estimated and actual delivery date of an order:

```
with CTE15 as
(select b.customer_state,a.order_id,date_diff(a.order_estimated_delivery_date
,a.order_delivered_customer_date, day) day_diff
from `SQL_Business_Case_Intro.orders` a
left join `SQL_Business_Case_Intro.customers` b
using(customer_id)
)
select customer_state Fast_delivery_customer_state, round(avg(day_diff),2)
Avg_delivery_time
from CTE15
group by customer_state
order by Avg_delivery_time desc
limit 5;
```

Following result (only top 10 shown below) was obtained:

Row	**Fast_delivery_customer_state**	**Avg_delivery_time**
1	AC	19.76
2	RO	19.13
3	AP	18.73
4	AM	18.61
5	RR	16.41

6.1: Calculate the Total & Average value of order freight for each state.

- There are two ways of looking at it:
  Case 1: Granularity level at Month and Payment Type. Here, Year of the purchase is not taken into account and the order count is just the total no. of orders placed in a particular month for all years combined.

```
with CTE16 as
```

```
(select distinct a.order_id, a.order_purchase_timestamp,
ifnull(b.payment_type,'not_defined') payment_type
from `SQL_Business_Case_Intro.orders` a
left join `SQL_Business_Case_Intro.payments` b
using(order_id)
)
select payment_type,extract(month from order_purchase_timestamp)
Month,  count(order_id) Ord_count
from CTE16
group by payment_type, Month
order by payment_type, Month;
```

Following result (only top 10 shown below) was obtained:

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

Case 2: Granularity level at Payment Type, Year and Month. Here, Year of the purchase is taken into account and the order count is just the total no. of orders placed in a particular month of a particular year:

```
with CTE17 as
(select distinct a.order_id, a.order_purchase_timestamp,
ifnull(b.payment_type,'not_defined') payment_type
from `SQL_Business_Case_Intro.orders` a
left join `SQL_Business_Case_Intro.payments` b
using(order_id)
)
select payment_type,extract(month from order_purchase_timestamp) Month,extract(year
from order_purchase_timestamp) Year, count(order_id) Ord_count
from CTE17
group by payment_type,Year, Month
order by payment_type, Year, Month
```

Following result (only top 10 shown below) was obtained:

Row	**payment_type**	**Month**	**Year**	**Ord_count**

Row	payment_type	Month	Year	Ord_count
1	UPI	10	2016	63
2	UPI	1	2017	197
3	UPI	2	2017	398
4	UPI	3	2017	590
5	UPI	4	2017	496
6	UPI	5	2017	772
7	UPI	6	2017	707
8	UPI	7	2017	845
9	UPI	8	2017	938
10	UPI	9	2017	903

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

- Following query was run for finding the no. of orders on the basis of the payment installments that have been paid:

```
with CTE18 as
(select order_id, sum(payment_installments) Total_installment
from `SQL_Business_Case_Intro.payments`
group by order_id)
select Total_installment, count(order_id) Order_count
from CTE18
group by Total_installment
order by Total_installment;
```

Following result (only top 10 shown below) was obtained:

Row	Total_installment	Order_count
1	0	2
2	1	46264
3	2	13605
4	3	10709
5	4	7223
6	5	5295
7	6	3967
8	7	1689
9	8	4239
10	9	693